Privacy Preserving and fully-Distributed Identity Management Systems

Mathieu Gestin

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *INFO*

Par
Mathieu Gestin

Privacy Preserving and fully Distributed Identity Management Systems

Thèse présentée et soutenue à Rennes, le 18 Décembre 2024
Unité de recherche : Inria

Rapporteurs avant soutenance :

Pascal FELBER — Full professor, Université de Neuchâtel
Maurice HERLIHY — Full professor, Brown University

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du jury doit être revue pour s'assurer qu'elle est conforme et devra être répercutée sur la couverture de thèse

Président :	*(à préciser après la soutenance)*	
Examinateurs :	Olivier BARAIS	Professeur, Université de Rennes
	Silvia BONOMI	Associate Professor, Saapienza University di Roma
Dir. de thèse :	Davide FREY	Chargé de recherche, Centre Inria de l'université de Rennes

"Le point idéal de la pénalité aujourd'hui serait la discipline indéfinie : un interrogatoire qui n'aurait pas de terme, une enquête qui se prolongerait sans limite dans une observation minutieuse et toujours plus analytique, un jugement qui serait en même temps la constitution d'un dossier jamais clos, la douceur calculée d'une peine qui serait entrelacée à la curiosité acharnée d'un examen, une procédure qui serait à la fois la mesure permanente d'un écart par rapport à une norme inaccessible et le mouvement asymptotique qui contraint à la rejoindre à l'infini."

Surveiller et Punir, Michel Foucault.

"[...] Nous avions le sentiment que nous avions le devoir de faire quelque chose, alors qu'en réalité il n'y avait plus rien que nous puissions faire [...]."

Hommage à la Catalogne, George Orwell.

Acknowledgement

Cette thèse est le fruit de l'ensemble des travaux que j'ai pu réaliser durant les quatre années passées à l'Inria. Elle n'aurait pas été possible sans la collaboration de mon directeur de Thèse, Davide Frey, que je remercie particulièrement. Je remercie aussi mes deux co-doctorants, Timothé Albouy et Arthur Rauch, avec qui nous avons traversé cette épreuve. Les discussions que nous avons eues, scientifiques ou non, ont permit d'aboutir au présent document. Je remercie aussi mes co-autheurs, François Taïani, pour sa précisions, Michel Raynal, pour son savoir, Guillaume Piolle et Daniel Bosk, pour m'avoir mis sur la voie. Sans oublier Emmanuelle Anceaume, Antonio Fernandez Anto, Chryssis Georgiou, Nicolas Nicolaou et Julang Wang. Je souhaite bonne chance à tous les doctorants de l'équipe WIDE, pour lesquels ce sera bientôt leur tour. Et j'adresse également ma reconnaissance aux autres membres de l'équipe, Virginie, et tous les autres membres, permanent ou non, avec qui j'ai partagé tout ou partie de mon temps. Je tiens aussi à remercier tous les membres de l'équipe Crypto de Bern, pour leur accueil à l'été 2023.

Ces quatre ans de travail n'auraient pas été possibles sans le soutien de ma famille, Hervé, Catherine, Juliette, Danielle et Madeleine. Je remercie particulièrement Aïcha, sans qui la rédaction de ce manuscrit aurait été laborieuse. Enfin, je n'oublie pas tous mes amis, de Quimper de Brest ou de Rennes, qui ont fait des temps durs des moments de bonheur.

Table of Contents

Résumé en Français

Cette thèse porte sur les systèmes de gestion d'identités numériques distribués et respectant la vie privée. Elle s'inscrit dans un contexte de forte demande de respect de la vie privée des personnes en général, et plus précisément, dans le contexte des méthodes d'autorisation et d'authentification. Cette demande provient aussi bien des citoyens que des états eux-mêmes (ou des unions d'états).

Autorisation et authentification

Un système de gestion d'identité, qu'il soit numérique ou non, a deux buts : l'authentification et l'autorisation. Ces deux mécanismes, l'autorisation et l'authentification, sont à la base de tout système de contrôle d'accès.

L'autorisation est un processus en deux phases. La première consiste à définir une liste de caractéristiques qu'une entité, le sujet, doit posséder pour permettre à une seconde entité, le porteur, d'accéder à un service. La seconde consiste à vérifier que le sujet possède en effet ces caractéristiques avant de permettre au porteur d'accéder au service. L'entité qui maintient ce processus d'autorisation est appelée le fournisseur de service.

Le sujet peut être un individu, un ordinateur, un objet, une organisation, une entreprise, ou toute autre entité qui peut être définie par ses caractéristiques propres. Le rôle du porteur peut lui aussi être assuré par plusieurs types d'entités : un individu, le représentant d'une entreprise, d'un état, d'une organisation, un ordinateur, un processus, etc...

Pour permettre l'autorisation, le porteur doit prouver au fournisseur de service que le sujet est bien défini par une caractéristique donnée. Par exemple, un club d'escalade n'autorise que les personnes sachant assurer un grimpeur à accéder à un mur d'escalade dont ils ont la gestion. Le sujet (qui est aussi le porteur dans notre cas) que nous nommerons Alice, affirme qu'elle est capable d'assurer un grimpeur en toute sécurité. Elle devra prouver cette affirmation en assurant un grimpeur du club devant une des personnes de ce même club. Elle va donc prouver sa capacité à assurer à un membre du club, Marc. Par extension, étant donné que les membres se font *confiance* entre eux, Alice prouvera sa capacité à assurer à l'ensemble des membres du club. Nous voyons ici apparaître une notion cruciale pour tout système de gestion d'identité, la confiance. Quand un porteur apporte une preuve d'une caractéristique à un fournisseur de service M, il va affirmer qu'un sujet possède cette caractéristique, et que quelqu'un auquel M fait confiance a certifié que le sujet possède bien cette caractéristique. En d'autres termes, le

porteur affirme quelque chose, mais le fournisseur de service n'acceptera cette affirmation que si une tierce partie auquel il fait confiance a vérifié que le sujet en question est bien décrit par cette caractéristique. Nous appelons cette tierce partie le fournisseur d'identité, ou l'émetteur d'identité (ou simplement l'émetteur). Dans notre exemple, Marc est l'émetteur.[1] Il se fait confiance à lui-même. Il sait donc qu'Alice peut assurer un grimpeur. Par transitivité, tous les autres membres du club qui font confiance à Marc et qui savent que Marc a vérifié les capacités d'Alice savent que le sujet est en capacité d'assurer un grimpeur.

Un problème persiste tout de même. Admettons que Marc et Alice n'étaient que deux quand Marc a vérifié la capacité d'Alice à assurer un grimpeur. Dans ce cas, Marc doit certifier auprès des autres membres du club qu'Alice a cette capacité. Pour ce faire, Marc peut utiliser deux méthodes. Soit il présente Alice à tous les autres membres du club en personne. Sinon, il trouve une méthode pour notifier à l'ensemble des membres du groupe qu'Alice peut assurer un grimpeur. Cette méthode peut être par l'envoie d'un message à tous les membres du club, ou par l'émission d'un document certifiant la capacité d'Alice. Dans ce cas, les autres membres, quand ils vont voir Alice, vont devoir vérifier que la personne qui se présente pour accéder à la salle est bien la Alice dont Marc à certifier les capacités. C'est le problème de l'authentification.

L'authentification est donc le fait de vérifier qu'un porteur est légitime à prouver qu'un sujet possède des caractéristiques spécifiques. Dis autrement, l'authentification est l'action de vérifier un lien entre le porteur et le sujet. Si le porteur et le sujet sont confondus, alors l'authentification est l'action de vérifier que le sujet est bien celui qu'il affirme être, et que les preuves des caractéristiques qu'il prétend posséder lui sont bien associées. L'authentification peut être réalisée de différentes manières. En utilisant une carte d'identité française, l'authentification est réalisée en comparant les informations de la carte d'identité (âge, taille, couleurs d'yeux, photo) au porteur.

Dans notre exemple, l'authentification peut être réalisée de différentes manières. La première a déjà été exposée. Marc peut être présent pour authentifier Alice auprès des autres membres du club. La seconde méthode consiste pour Marc à signer un document attestant que la personne qui porte ce papier est Alice. Une troisième méthode serait de laisser Marc divulguer un secret à Alice. Quand Alice se présente au club sans Marc, elle révèle ce secret aux membres du club présents à cet instant. Tous ces exemples ont des limites. Le premier implique que Marc doit présenter Alice à tous les membres du groupe, le second implique qu'Alice peut prêter ce papier à quelqu'un d'autre, voire qu'Alice peut le photocopier et le distribuer à toutes ses amies, et le dernier implique qu'Alice peut divulguer publiquement le secret.

Dans la suite de cette thèse, nous utilisons des méthodes de cryptographie à clé publique pour résoudre le problème de l'authentification. Marc annonce à tous les membres du club qu'Alice sait assurer un grimpeur, et que sa clé publique est pk. Quand Alice se présente au club ensuite,

1. Dans l'exemple, Marc est à la fois émetteur *et* fournisseur de service.

elle s'authentifie en prouvant sa connaissance de la clé secrète sk associée à pk.

Une chose reste à spécifier ici, nous avons vu qu'en fonction du contexte, une entité peut changer de rôle. Notamment, le fournisseur de service et l'émetteur peuvent se confondre. De plus, en continuant d'utiliser notre exemple, dans le futur, quand Alice sera une membre respectée du groupe, elle pourra elle-même certifier qu'un nouvel adhérent sait assurer un grimpeur. Dans ce cas, Alice sera à la fois sujet, fournisseuse de service et émettrice.

En revanche, nous n'avons pas expliqué pourquoi nous avions besoin de séparer le sujet du porteur. Il peut être nécessaire qu'une autorisation dépende d'un sujet qui n'est pas le porteur. Dans notre exemple, Alice aurait pu prouver qu'elle est membre du club car elle possède la signature de Marc. Néanmoins, si certaines personnes ne connaissent pas Marc, alors Alice peut utiliser le certificat qui certifie que Marc est membre du club pour prouver qu'Alice à bien une signature provenant d'un membre. Dans ce cas, Alice utilise un certificat dont le sujet est Marc. Elle est donc la porteuse du certificat de Marc.

Dans la suite, nous utiliserons la notion d'élément d'identité en lieu et place du qualificatif "caractéristique". Un élément d'identité étant la description d'une particularité (caractéristique) d'un individu qui permet de le distinguer d'autres individus d'un groupe donné. De plus, nous utiliserons aussi le terme de porteur et d'"utilisateur" de façon identique. En effet, le but du porteur d'identité est d'accéder à un service afin de l'utiliser.

Un autre point important qui reviendra dans la suite de cette thèse est le fait qu'une d'identité dépend d'un contexte. Un porteur pourra changer d'élément d'identité, ou de type d'élément d'identité utilisé en fonction du contexte dans lequel il évolue. Dans notre exemple, si Alice va dans un nouveau club d'escalade, le certificat que Marc lui a émis certifiant qu'elle sait assurer n'aura pas de valeur si les personnes de ce nouveau club ne connaissent pas Marc. Dans ce cas, elle devra prouver à nouveau ses capacités d'assurage à une personne de ce nouveau club, disons Jeanne. Dans ce nouveau club, c'est le certificat délivré par Jeanne qui sera considéré. De plus, en fonction du contexte, les identités peuvent changer. Par exemple, Alice peut être connue sous un pseudonyme dans un premier club et sous un autre pseudonyme dans un second club.

Nous avons donc vu les acteurs principaux d'un système de gestion d'identité : l'émetteur, le porteur, le sujet, et le fournisseur de service. Nous avons aussi exploré les deux mécanismes principaux de l'identification : l'autorisation et l'authentification. Enfin, nous avons vu deux notions clés, l'affirmation de possession d'un élément d'identité, et la preuve de possession de ce même élément d'identité.

Nous allons maintenant nous intéresser aux méthodes utilisées pour créer un système de gestion de l'identité numérique répondant à différentes contraintes de versatilité, de confiance, et de respect de la vie privée.

Systèmes de gestion de l'identité numérique

Comme nous le verrons dans le chapitre 3, la philosophie des systèmes de gestion de l'identité numérique a évolué en même temps que l'évolution de l'informatique. Nous différencions ici la philosophie d'un système de gestion d'identité des techniques utilisées pour l'implémenter. La philosophie donne les buts à atteindre pour un système de gestion de l'identité. Les techniques mises en oeuvre ne sont là que pour remplir au mieux les critères définis par la philosophie sous jacente.

Cette séparation entre philosophie de gestion de l'identité et technique de gestion de l'identité est propre à cette thèse. Comme nous allons le voir par la suite, cette terminologie permet de classifier avec précision, et donc de comparer, les différentes propositions et implémentations. D'après nos recherches, nous sommes les seuls à faire cette différence.

Cette thèse s'intéresse à la plus récente des philosophies concernant les systèmes de gestion de l'identité : les identités auto-souveraines, ou Self Sovereign Identity (SSI) comme elles ont été nommées en premier lieu. Cette dénomination a été popularisée par Cristopher Allen en 2016 [1]. Dans cet article, l'auteur revient sur les différentes philosophies qui ont été proposées jusqu'à 2016, en proposant une classification. Il les définit de la manière suivante, en partant de la plus ancienne, et moins versatile, à la plus récente :

— **Le modèle en silo.** Cette philosophie prédate toutes les autres [2]. Un utilisateur se présente à un fournisseur de service. Il crée un compte chez ce fournisseur. Si des informations supplémentaires sont nécessaires, le fournisseur va les vérifier lui-même (par exemple, via la vérification de la carte d'identité de l'utilisateur pour certifier son nom, sa date de naissance et son lieu de naissance). Quand l'utilisateur réutilise le service, il utilise les informations qui lui permettent de s'authentifier comme le propriétaire de ce compte (en général, en utilisant un nom d'utilisateur et un mot de passe). Si l'utilisateur se présente à un nouveau fournisseur de service, il devra réitérer tout le processus.
Les limites de cette philosophie sont évidentes. Premièrement, un compte et les autorisations associées, ne sont valides que pour un service donné. Cela implique qu'un porteur devra prouver son identité à une multitude de services. Chaque service stockera ces informations. Cela augmente la surface de fuites de données et d'attaque. De plus, cela implique que l'utilisateur doit faire confiance à ces différents services pour sécuriser suffisamment leurs serveurs et pour ne pas partager ces informations. De plus, l'utilisateur doit mémoriser un ensemble de paires (mot de passe, nom d'utilisateur). Ce qui est compliqué si chaque mot de passe est différent, ou propice à des attaques avec un fort impact si tous les mots de passes sont identiques.

— **Le modèle fédéré.** Cette philosophie est une évolution du modèle en silo. Comme pour le précédent modèle, l'utilisateur s'enregistre chez une entité, qui joue ici le rôle de fournisseur d'identité. L'utilisateur enregistre aussi ses éléments d'identité. La différence

étant que le compte créé peut être utilisé pour s'authentifier et accéder à différents services mis en place par une entité dépendant du fournisseur de service. Cette philosophie permet à l'utilisateur de créer moins de comptes, et donc de limiter les problèmes listés plus haut. En revanche, ces problèmes existent toujours. En effet, un compte n'est valable que pour une entité fédérée donnée qui gère différents services. De plus, le fournisseur de service gagne en pouvoir de nuisance. Il obtient plus d'information sur l'utilisateur, qui aura plus de mal à cacher son activité s'il utilise un unique compte sur différents services.

— **Le modèle centré sur l'utilisateur.** Le modèle centré sur l'utilisateur ressemble beaucoup au modèle fédéré. De la même manière, l'utilisateur crée un compte chez un fournisseur d'identité. Il associe à ce compte des éléments d'identités qui peuvent être vérifiés par le fournisseur d'identité. La seule différence avec le modèle présenté précédemment est que ces éléments d'identité peuvent être utilisés pour s'authentifier et être autorisé par un fournisseur de service qui ne dépend pas du fournisseur d'identité. Pour ce faire, le fournisseur de service doit simplement faire confiance au fournisseur d'identité (et mettre en place un outil permettant aux deux services de communiquer). Comparé au modèle précédent, ce modèle permet à l'utilisateur de réduire grandement le nombre de comptes qu'il crée. Par exemple, si tous les services que l'utilisateur utilise sont gérés par des fournisseurs de service qui font confiance à un unique fournisseur d'identité, alors l'utilisateur pourra ne créer qu'un seul compte chez ce fournisseur d'identité. Cela permet de réduire grandement le nombre de comptes que l'utilisateur doit créer, la surface d'attaque d'un potentiel attaquant, et le nombre de fournisseur de service auxquels l'utilisateur doit faire confiance.

  En revanche, les limites du modèle fédéré sont exacerbées dans le modèle centré sur l'utilisateur. Si le fournisseur d'identité par lequel passe l'utilisateur cesse de fonctionner, tous les services qui lui sont associés et que l'utilisateur utilisait lui seront dorénavant inaccessibles. De plus, si le fournisseur d'identité veut tracer l'activité d'un utilisateur, il aura d'autant plus de pouvoir que toutes les authentifications de cet utilisateur passeront par lui.

  Dans la suite de cette thèse, nous ne différencierons pas le modèle fédéré du modèle centré sur l'utilisateur, en cela que leur fonctionnement est proche, et que les limites de ces deux modèles sont similaires. La seule différence étant le degré avec lequel ces limites peuvent impacter l'utilisateur.

— **Le modèle d'Identités Auto Souveraine.** Cette philosophie à pour but de donner à l'utilisateur le contrôle de ses éléments d'identité et des preuves de ces éléments. L'idée est de se rapprocher du modèle de la preuve d'identité physique tel que celui de la carte d'identité. Une carte d'identité est donnée à son porteur/sujet par un fournisseur d'identité (en général un état). L'utilisateur n'aura ensuite plus besoin d'interagir avec

le fournisseur d'identité jusqu'à l'expiration de sa carte. Il stocke lui-même ses éléments d'identité dans son portefeuille. Il peut choisir de la présenter ou non à un fournisseur de service, et il sait que le fournisseur d'identité ne peut pas utiliser ses éléments d'identités sans son consentement étant donné qu'il ne possède pas de copie de la carte. L'identité auto-souveraines est la transposition de ce modèle au monde numérique. L'utilisateur doit être à tout moment en contrôle de ses éléments d'identité. Il doit consentir de façon éclairée pour que ces derniers soient utilisés. Il doit aussi pouvoir présenter des preuves d'élément d'identité à tout moment, et donc ne pas dépendre d'une entité unique qui pourrait cesser son activité (pour des raisons économiques, techniques, de censure etc...). Plus précisément, Cristopher Allen [1] présente dix points que n'importe quel système de gestion d'identité devrait maximiser pour pouvoir atteindre le statut d'"identité auto-souveraine".

1. *Existence.* Les éléments d'identité et les preuves d'identités des utilisateurs doivent avoir une existence indépendante. Cela implique qu'ils ne doivent pas dépendre d'un unique système, d'une unique norme ou d'une unique implémentation technique. Le système de gestion d'identité ne doit pas être limitant pour l'utilisateur.

2. *Contrôle.* L'utilisateur doit contrôler ses éléments d'identité. Ils sont les seuls à pouvoir demander l'émission de preuve à propos de leur identité, à pouvoir les mettre à jour, à pouvoir les présenter. Ils peuvent aussi choisir de ne pas les révéler. De plus, ils doivent être tenus au courant de tout traitement relatif à leurs éléments d'identité.

3. *Accès.* Les utilisateurs doivent avoir accès à leurs éléments d'identité. Ils ne peuvent dépendre d'un acteur qui pourrait potentiellement cesser son activité. Ils doivent pouvoir à tout moment récupérer ces éléments. Cela implique aussi qu'ils doivent avoir connaissance de toutes les affirmations et les preuves concernant leurs identités.

4. *Transparence.* Les algorithmes et les systèmes implémentant des systèmes de gestion de l'identité doivent être transparents. C'est-à-dire que les implémentations techniques, aussi bien que la gestion de ces systèmes doivent pouvoir être analysé par tous les acteurs les utilisant. Les décisions prisent doivent l'être selon une politique définie à l'avance. Les résultats de ces décisions doivent être publics. Les algorithmes et les systèmes utilisés doivent être accessibles à tous, leurs sources doivent être publiques.

5. *Persistance.* Les identités des utilisateurs doivent être utilisables dans la durée, et dans l'idéal tout le long de la vie de l'utilisateur. Cela implique à nouveau que ces identités, et les éléments d'identité attenants ne doivent pas être dépendants d'une entité qui pourrait cesser son activité, ni d'une technologie donnée qui pourrait évoluer ou ne plus être utilisée. De plus, les contraintes de sécurités évoluant au rythme de l'évolution du matériel informatique, les systèmes permettant de sécuriser et de prouver les

éléments d'identité d'un utilisateur doivent pouvoir être mis à jour, notamment les outils cryptographiques utilisés, et la taille des clés permettant leur sécurisation.

6. *Transportabilité.* Les éléments d'identités et preuves d'identités d'un porteur doivent être transportables. Ils ne doivent pas être seulement valides s'ils sont stockés chez une unique entité, ou s'ils sont utilisés dans le contexte d'un système unique.
7. *Interoperabilité.* Les éléments et preuves d'identités doivent être utilisables dans le plus grand nombre de contextes possibles. Aucune barrière ne devrait exister, qu'elle soit technologique, administrative, ou politique, justifiant le fait qu'une preuve d'identité soit valide dans un contexte mais pas dans un autre. Cette propriété implique donc une coopération des différents acteurs de l'écosystème, aussi bien au niveau technique que politique et organisationnel. [2]
8. *Consentement.* Les utilisateurs doivent consentir à la présentation et à l'utilisation de leurs éléments d'identité. Premièrement, cela implique que l'utilisateur doit être tenu informé des différentes utilisations de ses éléments d'identités. Deuxièmement, ce consentement doit être éclairé. C'est à dire que l'utilisateur doit comprendre les implications de la divulgation d'éléments d'identités à un fournisseur de service. Il ne suffit donc pas de demander à un utilisateur de signer un contrat dont le but est l'obfuscation des volontés réelles du fournisseur de service pour valider le consentement de l'utilisateur.
9. *Minimisation.* La présentation d'éléments d'identité concernant un utilisateur doit être réduite à leur strict minimum. Quand un utilisateur accède à un service, il ne doit révéler que les informations absolument nécessaires au fonctionnement de ce service. Cela implique aussi que les solutions techniques utilisées pour prouver un élément d'identité ne devraient pas révéler plus d'information que nécessaire (c.f. chapitre 5). Par exemple, si un utilisateur veut prouver qu'il a plus de 18 ans, alors, le fournisseur de service ne doit apprendre que cette information. L'âge exact de l'utilisateur ne doit pas être révélé.
10. *Protection.* Les droits des utilisateurs doivent être protégés. Les algorithmes utilisés, aussi bien que les politiques de vie privée implémentées par ces algorithmes doivent protéger l'utilisateur.

Ces dix points à maximiser sont le résultat de discussions qui commencèrent au début des

2. Ce dernier point doit néanmoins être limité par la notion de confiance et de souveraineté nationale. On ne peut pas demander à un acteur local d'accepter une preuve venant d'un autre acteur local à l'autre bout du monde, si ce dernier ne possède pas une certification de son état qu'il est en capacité d'émettre une telle preuve. De plus, dans certains contextes, seule une preuve venant d'un pays spécifique peut être prise en compte. Par exemple, pour certifier des capacités d'un individu à réaliser une tâche, nous utilisons des diplômes. Or, tous les diplômes ne se valent pas. Il n'est pas possible de forcer un état A à accepter un diplôme émit par un état B alors que les conditions d'obtention de ce dernier sont moins contraignantes dans le pays B que dans le pays A.

années 2000 (c.f. chapitre 3). Cette classification est globalement consensuelle. Certaines évolutions ont été proposées [3], mais l'essence des identités auto-souveraines reste la même au travers des définitions.

Plusieurs solutions techniques ont été proposées afin de créer un système de gestion d'identité respectant les principes des identités auto-souveraines. Elles reposent toutes sur un principe fondamental, la distribution. Le but de la distribution est de proposer des solutions techniques permettant de respecter les principes des identités auto-souveraines tout en ne dépendant pas d'une unique entité ou d'un ou d'un petit nombre d'entités.

Nous considérons ici deux grandes familles. Chacune de ces familles à ses avantages et ses défauts, mais elles permettent toutes les deux de répondre aux attentes édictées par Allen. La différence entre ces deux familles repose sur la façon de gérer les propriétés annexes des systèmes de gestion d'identité, c'est-à-dire les propriétés n'étant pas le cœur de l'autorisation.

En effet, la distribution vient de l'utilisation d'un mécanisme distribué pour l'émission, le stockage et la présentation des preuves d'identité. En revanche, rien n'empêche d'utiliser un serveur central jouant le rôle de tierce partie de confiance pour tous les besoins annexes. C'est le fonctionnement des systèmes de gestion d'identité *partiellement distribués*. Les systèmes de gestion d'identité *totalement distribués*, en revanche, ne repose sur aucune tierce partie de confiance.

Plus formellement, et comme nous le disions précédemment, la première famille est la famille des systèmes de gestion d'identité *partiellement* distribués. Ces systèmes ne requièrent pas l'interaction avec l'émetteur d'identité lors de la présentation d'une preuve d'identité par l'utilisateur. En revanche, certaines fonctionnalités du système peuvent dysfonctionner suite à la cessation d'activité d'une ou d'un petit nombre d'entités. Ces fonctionnalités peuvent être l'authentification, la révocation d'un certificat, ou le récupération de clés après une perte ou un vol.

La seconde famille est la famille des systèmes de gestion d'identité *totallement* distribués. Comme les précédents, ces systèmes ne requièrent pas d'interaction avec l'émetteur lors de la présentation d'une preuve d'identité par l'utilisateur. En revanche, la continuité du fonctionnement des fonctionnalités ne doit pas dépendre d'une unique entité ou d'un petit nombre d'entités. Plus précisément, un système de gestion d'identité totalement distribué est défini pour un paramètre t tel que le fonctionnement du système n'est pas impacté par l'arrêt de l'activité de t entités.

Cette définition implique que le fonctionnement des différents composants du système doit être distribué entre différents processus ou ordinateurs, mais il implique aussi que ces composants doivent être distribués entre différentes entités indépendantes. Nous ne considérons pas, par exemple, qu'une entreprise qui gère n ordinateurs et implémente un système de gestion d'identité distribué qui est résiliant à t fautes parmi ces n ordinateurs soit un système totalement distribué. En effet, si l'entreprise fait faillite, le système ne sera plus opérationnel. Une seule entité failli et le système n'est plus utilisable.

Un point important qui va guider les différentes propositions de cette thèse et de tout système se réclamant des SSIs est la minimalisation. Cette propriété de la philosophie de Allen implique que, lors d'une présentation d'un élément d'identité, l'utilisateur ne doit révéler d'autres informations que celles escomptées. Parmi les informations auxiliaires que l'utilisateur doit protéger, nous devons considérer les identifiants uniques. Parmi eux, il y a les adresses IP, les clés publiques, les différentes URIs nécessaire à l'authentification. De plus, les signatures utilisées ne peuvent être uniques. En effet, l'utilisation répétée d'une même signature, identifiable et inchangée, permet de tracer les différents usages de l'utilisateur. Ce traçage permettrait à un adversaire d'obtenir des informations supplémentaires sur les habitudes de l'utilisateur. Il permettrait aussi de réaliser des *timing attacks*. Des inférences sur l'identité d'une personne réalisées en recoupant différentes opérations, et leurs proximités temporelles.

Pour respecter la minimalisation, les systèmes doivent donc limiter, ou supprimer ce type d'identifiants. Pour cette raison, nous rajoutons le qualificatif de "respect de la vie privée" aux systèmes se réclamant des SSIs.

Dans la suite de cette thèse, nous nous intéresserons principalement aux systèmes de gestion d'identité totalement distribués, bien que certains des résultats que nous présentons s'appliquent aussi aux systèmes partiellement distribués.

Les composants d'un système de gestion d'identité respectant la vie privée totalement distribué

Nous présentons ici les différents outils techniques utilisés pour permettre d'implémenter un système de gestion d'identité totalement distribué respectant la vie privée. Ces outils, s'ils sont utilisés correctement, permettent de se rapprocher de la philosophie de Allen. En revanche, il est important de se rendre compte que ces considérations techniques (qui sont le sujet de cette thèse) ne respectent pas à eux seuls les caractéristiques d'une SSI. Une SSI dépend à la fois d'outils techniques et de considérations organisationnelles. Ce commentaire est la raison principale qui nous a conduits à marquer une différence entre la philosophie que nous voulons suivre (SSI), et le système technique que nous explorons (système de gestion d'identité totalement distribué respectant la vie privée).

Les outils que nous présentons s'inscrivent dans un modèle théorique construit en couche à la manière du modèle *Open Systems Interconnection* (OSI) : c'est le modèle Trust over IP (ToIP) [4]. Ce modèle est constitué de quatre couches. La première couche est un registre public appelé Distributed IDentity (DID) network. Cette couche sert à stocker publiquement les informations nécessaires au fonctionnement des couches supérieures. Ces informations sont représentées sous la forme de documents appelés DID documents. Ils sont identifiés à l'aide d'URIs appelés DID. La seconde couche est un protocole de communication sur réseau hautement asynchrone entre processus qui peuvent se déconnecter, ce protocole est appelé DIDComm. Il fait usage des

DID et DID documents pour permettre l'échange de messages. La troisième couche est le cœur du système de gestion d'identité. C'est le mécanisme d'authentification et d'autorisation. La quatrième et dernière couche est une couche organisationnelle, la couche de gouvernance. Son but est de définir les buts du système de gestion d'identité, d'auditer les implémentations des couches inférieures, et d'en certifier les composant. Dans cette thèse, nous nous intéresserons principalement à la première et à la troisième couche.

Explorons d'abord le premier prérequis d'un système de gestion d'identité respectant la vie privée : l'autorisation. C'est l'élément le plus important de la troisième couche ToIP, et de tout système de gestion d'identité distribué. Pour permettre l'autorisation, un système de gestion d'identité distribué doit permettre à l'utilisateur d'exprimer et de présenter des affirmations sur son identité à plusieurs vérificateurs. Ces affirmations et ces preuves doivent être stockées par l'utilisateur lui-même pour permettre de respecter au mieux les propriétés de contrôle, d'accès et de consentement. De plus, les vérificateurs avec lesquels l'utilisateur peut interagir ne sont pas prédéfinis. Les affirmations et les preuves stockées par l'utilisateur ne peuvent donc pas dépendre des vérificateurs qui vont les vérifier. Malgré tout, les vérificateurs doivent faire confiance à la preuve soutenant l'affirmation, ou, plus précisément, ils doivent faire confiance à l'entité qui a créé cette preuve. Ils doivent aussi être assurés que la preuve (et le sujet de la preuve) n'a pas été modifiée entre le moment de son émission et le moment de sa présentation. Finalement, le dernier point important est le respect de la minimalisation. Les affirmations concernant l'identité d'un utilisateur doivent être limitées au strict minimum nécessaire au fonctionnement du service. L'utilisateur doit donc soit avoir une preuve par affirmation potentielle (par exemple, pour la date de naissance de l'utilisateur, une preuve qu'il a plus de 16 ans, une preuve qu'il a plus de 18 ans, une preuve qu'il a plus de 21 ans etc...), ou il doit pouvoir dériver des preuves restreintes à partir d'une preuve globale (pour l'exemple de l'âge, l'utilisateur n'aurait besoin que d'une preuve de sa date de naissance de laquelle il pourrait dériver toutes les autres preuves nécessaires). De plus, et comme exprimé précédemment, un identifiant traçable lors de plusieurs présentations peut compromettre la minimisation. Les preuves des affirmations d'identités ne peuvent donc pas contenir ce genre d'identifiants (ou ils ne doivent pas être révélés au vérificateur). De plus, les preuves en elles-mêmes ne doivent pas constituer un identifiant unique, elles doivent don être modifiables, ou plus précisément, "aléatoirisables" (nous utiliserons l'anglicisme "randomisable"). De façon générale, nous appelons certificat vérifiable (ou Verifiable Credential, VC) une preuve (associée à une affirmation) qui remplie ces prérequis. [3]

Un outil remplissant tous ces prérequis existe, il s'agit des certificats anonymes (ou Anonymous Credentials, AC). Un certificat anonyme est un type de signature cryptographique, introduit par David Chaum en 1985 [6], qui, en plus des propriétés de signatures, doit vérifier

3. Les prérequis présentés ici sont plus contraignants que les prérequis des VCs tels que standardisés par la W3C [5]. Nous préférons notre définition car elle permet de plus se rapprocher de la vision de Allen.

trois propriétés supplémentaires : signatures à l'aveugle, randomisabilité, et preuve à divulgation nulle de connaissance de signature. Une signature à l'aveugle est un schéma de signature qui permet à l'émetteur (le signataire) de signer un message qu'il ne connaît pas. Il signe un engagement cryptographique envers le message final. Le receveur de la signature peut ensuite "ouvrir" l'engagement, se retrouvant alors avec une signature valide du message original. Ce faisant, l'utilisateur peut utiliser la signature sans qu'une éventuelle entente entre l'émetteur et le vérificateur de cette signature ne permette d'apporter des informations supplémentaires sur l'utilisateur. La randomisabilité est le fait de pouvoir ajouter des nombres aléatoires à la signature sans en modifier le contenu ni la validité. Cela permet de rendre les différentes présentations d'une même signature *non-chainable*. C'est-à-dire qu'un vérificateur qui reçoit plusieurs signatures sur le même message ne peut pas déterminer si elles proviennent du même utilisateur ou de différents utilisateurs ayant reçu une signature sur le même attribut. En d'autres termes, tous les utilisateurs recevant une signature sur le même attribut et par le même émetteur constituent un *ensemble d'anonymat* pour cet attribut particulier. La présentation de leur attribut les rend indistinguable parmi l'ensemble des individus qui ont un certificat signé par le même émetteur et sur le même attribut.

Nous devons nous arrêter un moment sur un point important. La distribution des systèmes de gestion d'identités distribués vient principalement de ces certificats anonymes. En effet, l'utilisation de certificats anonymes permet à un utilisateur d'affirmer et de prouver des éléments d'identités sans interagir avec l'émetteur. Tous ces éléments d'identité et leurs preuves associées sont stockés directement par l'utilisateur. Aucun service centralisateur n'est nécessaire pour réaliser ces actions. Ce point a parfois été mal compris par certains, notamment U-Port [7], sur lequel nous reviendrons plus tard. Une blockchain ou tout autre registre distribué n'est donc pas nécessaire à la création d'un système de gestion d'identité distribué respectant la vie privée. Tout du moins, ce type de registre n'est pas nécessaire pour créer un système de gestion d'identité *partiellement* distribué respectant la vie privée. Les registres distribués peuvent néanmoins être nécessaires pour permettre d'implémenter certaines fonctionnalités annexes de façon distribuée. Les plus importantes de ces fonctionnalités sont les suivantes : l'authentification forte de différents appareils, l'authentification des émetteurs et des vérificateurs, la découverte de schémas de certificats ou la révocation de certificats.

Ces fonctionnalités annexes ne sont donc pas strictement nécessaires pour prouver les éléments d'identité d'un utilisateur. Elles le sont en revanche pour résoudre des problèmes de confiance et d'utilisabilité. Par exemple, l'authentification d'un certificat peut se faire à l'aide d'une paire clé publique/clé secrète intégré à tous les certificats anonymes d'un utilisateur. La preuve de la connaissance de la clé secrète permet de s'assurer que la personne qui utilise le certificat est bien la personne à qui il a été délivré. En revanche, pour l'utilisateur, cela implique que chaque certificat doit partager la même clé secrète. Si cette clé secrète est perdue, l'utili-

sateur perd alors l'accès à tous ses certificats, il doit donc demander une réémissions à tous les fournisseurs d'identité qui lui avaient émis des certificats. De plus, si l'utilisateur veut pouvoir utiliser ses certificats depuis plusieurs appareils (par exemple, depuis son smartphone et son ordinateur personnel), il doit partager sa clé secrète avec ses différents appareils. Ce partage est généralement considéré comme étant une mauvaise pratique en termes de sécurité. Il faut donc un système permettant d'autoriser différents appareils pour un utilisateur donné. Étant donné que nous nous intéressons aux systèmes de gestion d'identités *totallement* distribués, ce système permettant les fonctionnalités annexes d'un système de gestion d'identité ne peut dépendre d'une entité ou d'un petit nombre d'entités (c.f. définition d'un système de gestion d'identité *totallement distribué*). Nous devons donc utiliser un type de registre distribué pour implémenter ces fonctionnalités. [4]

Ce registre distribué doit permettre aux différents acteurs de publier les éléments nécessaires aux fonctionnalités annexes du système de gestion d'identité. [5] Étant donné qu'il n'existe pas d'entité centralisatrice dans le système, chaque entité publie ses propres informations. Chaque entité à donc un "espace réservé" sur le registre. Cet espace est appelé DID document, et est référencé à l'aide d'un Distributed IDentifier (DID). Le format du DID et du DID document est standardisé par la W3C [8]. Le DID est un URI qui peut être résolu en son DID document. Théoriquement, l'association entre un DID et son DID document doit être unique. Le DID est au format suivant : *did :[method] :[identifier]*. La méthode *[method]* est l'identifiant du registre qui permet la résolution du DID, et l'identifiant *[identifier]* est l'identifiant particulier du DID document dans le contexte du registre *[method]*. Une méthode doit spécifer comment créer, mettre à jour, supprimer, et lire un DID document (le CRUD). Au moment de l'écriture de cette thèse, la W3C avait enregistré 193 méthodes de résolution de DID [9].

Le DID document est notamment utilisé durant la présentation d'une preuve d'identité pour l'authentification de l'appareil utilisant le certificat, pour vérifier la non-révocation du certificat, et pour permettre à l'utilisateur d'authentifier le vérificateur. Cette utilisation ne doit généralement pas modifier le registre. Lors d'une présentation, le registre est utilisé en lecture seule, et les preuves sur ce registre sont conduites entre l'utilisateur et le vérificateur. Cela permet de respecter la propriété de minimalisation. Si une présentation laisse une trace sur le registre, il est alors possible pour des acteurs malveillants de tracer l'activité de l'utilisateur.

Le troisième composant d'un système de gestion d'identité distribué respectant la vie privée est le système de communication pair à pair. Un standard de la W3C se basant sur les DIDs et

4. La caractérisation de ces registres est l'une des contribution de cette thèse. Nous reviendrons dessus dans la suite de ce document.

5. Il est important de voir que, pour permettre au système de respecter la vie privée, et parce que le registre doit être publiquement accessible, *aucune information personnelle, ni aucune information permettant d'inférer des informations personnelles à propos d'un individu ne doit être publiée sur le registre !* Cette remarque ne s'applique pas aux entreprises, états, organisation etc... Car nous ne considérons la protection des données personnelles que pour les individus.

les DID documents a été proposé en ce sens, le protocole DIDComm [10]. Nous ne reviendrons pas en détail sur ce protocole, étant donné que nos contributions font l'hypothèse qu'un tel protocole de communication existe et qu'il respecte la vie privée. C'est-à-dire qu'il ne révèle pas d'information supplémentaire sur l'identité de l'utilisateur, ou sur son activité.

Le cas U-Port et l'utilisation de registres distribués comme unique facteur de distribution

Une question qui est longtemps restée en suspens concernant les systèmes de gestion de l'identité distribué respectant la vie privée est la question de la nécessité d'une Blockchain. Certains ont cru que la blockchain pouvait être suffisante pour permettre d'implémenter un système de gestion d'identités distribué. Le cas le plus notable est celui de U-Port [7]. L'idée de ce système était de stocker tous les éléments d'identité et preuves d'identités sur la blockchain (Ethereum dans leur cas), ou en utilisant un service IPFS. Les éléments d'identités et leurs preuves associées pouvaient être chiffrés. Ce modèle pose un problème en termes de vie privée. Les informations publiées sur la blockchain ou sur le système IPFS sont publiques, bien qu'elles soient chiffrées. Chaque participant au système peut inspecter les différentes émissions d'éléments d'identité concernant un utilisateur. Cet attaquant ne peut pas nécessairement découvrir quels sont les éléments d'identité de l'utilisateur, mais il peut obtenir des méta-informations sur les usages et habitudes de ce dernier. Avec des connaissances auxiliaires, le système proposé par U-Port permettait donc des attaques à fort impact, par exemple en utilisant des *timing attacks*.

Suivant ces critiques, la proposition de U-Port n'est plus supportée aujourd'hui. Les ressources ont été redirigées vers un nouveau projet, Veramo [11], qui est un environnement plus classique permettant d'implémenter un système de gestion d'identité distribué agnostique du registre distribué utilisé.

Plus généralement, la question de l'utilité d'une blockchain a fait couler beaucoup d'encre depuis 2017. Les premiers à réfléchir à l'utilité d'un registre distribué sont les personnes travaillant sur le projet Sovrin [12]. Sovrin est un projet de registre distribué servant à supporter des systèmes de gestion d'identités totalement distribués. Dans un document [12], ils décrivent les différents usages qu'ils réservent à leur registre distribué. Ils attestent notamment que le registre ne devrait contenir aucunes informations personnelles concernant les individus, qu'elles soient chiffrées ou non.

Un second article intéressant sur le sujet a été publié par Jolocom en 2021 [13]. Cet article revient sur ce qui fait la base de la décentralisation dans les systèmes de gestion d'identité distribuée (l'utilisation de VCs possédés par l'utilisateur). De façon étonnante, cet article n'est plus disponible sur le site de Jolocom, mais nous pouvons encore le lire en utilisant web archive. Finalement, un article de 2022 revient lui aussi sur la différence entre système de gestion d'identité distribué et blockchains [14]. L'une des contributions de cet article est que l'auteur revient

sur une dizaine d'implémentations de système de gestion d'identité distribué et analyse leur utilisation de la blockchain. De façon intéressante, on se rend compte que la plupart des projets utilisant une blockchain comme unique facteur de distribution ont cessé d'être développés.

Une des contributions importantes de cette thèse est la caractérisation des besoins minimums qu'un registre distribué doit remplir afin de faire fonctionner un système de gestion d'identité distribué. Plutôt que de se demander si une blockchain" est nécessaire, nous allons plus loin et nous caractérisons de façon fine les besoins en synchronisation de tels registres. Comme nous le verrons ensuite, bien qu'un registre soit nécessaire, il n'a pas nécessairement besoin d'être distribué (dans le cas d'un système de gestion d'identité *partiellement* distribué). Si le registre est en effet distribué, les contraintes quand à la synchronisation du système sont bien moins fortes que celles offertes par une blockchain. En effet, une blockchain ordonne totalement toutes les opérations du système. Cela à un coût très important en termes de latence, de calcul de fonctions cryptographique, etc... Un système de gestion de l'identité distribué à besoin de garanties de synchronicité assez faible, que nous caractérisons précisément dans les chapitres 6 et 7. De fait, si un registre distribué est la technologie choisie pour implémenter un système de gestion d'identité, ce registre peut être implémenté de façon économe. Une abstraction optimale pour ce genre de cas d'usages est présentée en chapitre 8. L'abstraction présentée dans ce chapitre peut être utilisée pour d'autres cas d'usage dont les besoins de synchronicité sont faibles. Il est intéressant de remarquer que c'est le cas de la majorité des applications distribuées, que ce soit pour les services de nommage, le contrôle d'accès, le transfert d'actif, etc...

Contributions

D'un point de vue général, dans cette thèse, nous nous intéressons aux différents outils permettant de créer un système de gestion d'identité totalement distribué respectant la vie privée. Nous nous intéressons donc aux techniques distribuées permettant de maximiser la vie privée des utilisateurs, tout en améliorant la facilité d'usage de ces techniques.

Le but est de répondre à la question suivante : quels sont les outils permettant d'implémenter un système de gestion d'identité totalement distribué et respectant totalement la vie privée requérant le minimum de synchronisation entre les participants du système.

Les contributions de cette thèse sont décomposées en deux grands thèmes. La contribution du premier thème se concentre sur la couche 3 du modèle ToIP, et plus spécifiquement une amélioration de l'état de l'art concernant le respect de la vie privée des certificats anonymes.

Les contributions du deuxième thème se concentrent sur la couche une et trois (sur l'interaction entre ces deux couches) du modèle ToIP. Ces contributions sont des analyses et des propositions de systèmes interagissant avec le registre distribué de la couche une et le mécanisme de présentation de certificats anonymes de la couche trois. Le but de ces contributions est de

comprendre formellement le fonctionnement d'un registre distribué servant à implémenter un système de gestion de l'identité respectant la vie privée totalement distribuée et d'utiliser ces analyses pour proposer des implémentations efficaces qui répondent aux défis spécifiques liés aux systèmes de gestion de l'identité.

La contribution du premier thème est présentée dans le chapitre 5. Cette contribution prend comme point de départ une constatation simple, bien que les constructions de certificats anonymes se targuent de respecter le principe de minimalisation, la connaissance, pour le vérificateur, de la clé publique de l'émetteur d'un certificat révèle des informations supplémentaires sur l'utilisateur. En effet, la connaissance de la clé publique de l'émetteur est nécessaire pour tout schéma de signature cryptographique. Or, la connaissance de cette clé publique implique la connaissance de l'identité de l'émetteur. Cette connaissance peut révéler des informations supplémentaires sur l'utilisateur. Nous donnons deux exemples pour motiver cette affirmation Tout d'abord, un émetteur est souvent une entité locale (en terme géographique). Un utilisateur requérant l'émission de certificats concernant son état civil se tournera vers la mairie de son lieu de résidence. Dans ce cas, quand le vérificateur vérifie un élément d'identité relatif à l'état civil de l'utilisateur, il apprendra aussi son lieu de résidence, qui est (avec une grande probabilité) la commune émettrice du certificat. De façon plus dérangeante encore, si un vérificateur et un émetteur entrent échangent des informations, et si le vérificateur possède des informations complémentaires sur l'utilisateur [6], ces informations complémentaires peuvent permettre d'identifier de façon unique l'utilisateur dans la base de données de l'émetteur. Cela implique qu'il est possible pour l'émetteur et le vérificateur de tracer les usages de l'utilisateur, mais aussi que le vérificateur peut obtenir toutes les informations que l'émetteur possède à propos de l'utilisateur.

Au-delà de l'atteinte à la vie privée de l'utilisateur et à la propriété de minimalisation, ce genre d'attaque à un impact plus grand lié à la méconnaissance des utilisateurs. En effet, un utilisateur se pensant protégé par les propriétés du certificat anonyme fera peu attention et révélera plus d'information le concernant qu'un utilisateur n'utilisant pas de certificats anonymes.

Afin d'empêcher ce type d'attaque, nous proposons un schéma de certificat anonyme dont l'émetteur du certificat est caché au vérificateur. Plus précisément, avant une vérification, le vérificateur publie la liste des émetteurs auxquels il fait confiance. L'utilisateur va ensuite utiliser cette liste pour randomiser la clé publique de l'émetteur de son certificat, tout en construisant une preuve que la clé originale (non randomisée) à laquelle est associée la clé randomisée appartient bien à un émetteur présent dans la liste des émetteurs de confiance du vérificateur. L'utilisateur présente ensuite cette clé randomisée, son certificat randomisé et la preuve décrite précédemment au vérificateur. Le vérificateur peut donc vérifier qu'il fait confiance à l'émetteur du certificat, et que cet émetteur a bien signé le certificat, sans apprendre l'identité exacte de cet émetteur. Ce

6. Ces informations complémentaires peuvent être obtenues facilement dans le cadre d'une vérification "dans la vie réelle". Dans ce cas, le vérificateur peut simplement enregistrer les caractéristiques physiques de l'utilisateur et les comparer à celles enregistrées dans la base de données de l'émetteur.

type de certificat anonyme permet donc de garder les propriétés de confiances liées aux certificats anonymes classiques, tout en évitant les attaques présentées précédemment. Ce travail a été publié à la conférence Privacy Enhancing Technologies 2022 (PETs) [15].

Le deuxième thème regroupe quatre contributions, deux contributions majeures et deux contributions mineures.

La première contribution mineure, présentée dans le chapitre 7, est une analyse du problème de nommage, central dans les systèmes de gestion d'identités. Cette analyse revient sur le problème du triangle de Zooko [16]. Ce problème est la conjecture d'un trilemme. Zooko, dans un poste de blog de 2001, réfléchit à l'attribution de noms dans un espace de nom (*namespace*) donné. Nous considérons ici un nom comme étant une chaîne de caractères. Zooko établit trois propriétés désirables pour un nom, la sécurité, la distribution et la compréhensibilité pour un humain. La propriété de sécurité est informellement définie comme le fait qu'un nom ne peut être utilisé que par son possesseur légitime. Nous utilisons ici le terme utilisé comme le fait de prouver la possession du nom. Dans le contexte des systèmes de gestion de l'identité distribués, l'utilisation serait le fait de pouvoir modifier un DID document associé à un DID.

La distribution est le fait pour différent processus de partager la même association entre un nom et une ressource. En d'autres termes, un nom caractérise la même chose pour tous les membres d'un système donné.

La propriété de compréhensibilité pour un humain est la plus complexe à définir. Sa définition originelle était le fait que le nom pouvait facilement être manipulé par un être humain. Nous lui préférons la définition de humainement choisissable. C'est-à-dire que le système laisse le choix aux utilisateurs de choisir le nom qu'ils veulent utiliser. Le trilemme de Zooko conjecture que les trois propriétés ne peuvent être atteintes en même temps.

Beaucoup d'incompréhensions autour de ce trilemme sont apparues après qu'il a été exprimé. Par exemple, la page Wikipedia relative à ce problème dit qu'il existe des implémentations réfutant la conjecture. De la même manière, un article de blog parut en 2011 et écrit par Aaron Swartz [17] montre comment le trilemme de Zooko peut être résolu. Comme nous le verrons, ces différentes propositions ne résolvent pas le problème, mais le contournent. Elles le font en associant deux types de noms, des pointeurs et des surnoms. Nous revenons plus en détail sur ce type de solutions dans le chapitre 7. Finalement, nous explorons aussi un dernier point intéressant. Dans un papier de Mühle, Grüner, Gayvoronskaya et Meinel [18], il est dit qu'une blockchain est nécessaire pour pouvoir contourner le trilemme de Zooko. Nous proposons d'explorer ce point et d'y répondre formellement.

Notre contribution est en deux parties. Nous formalisons le problème du triangle de Zooko et prouvons formellement l'impossibilité qui n'était jusque la qu'une conjecture. Nous formalisons ensuite les différentes manières de contourner le problème, et nous analysons leurs besoins en termes de synchronisation. Cette dernière analyse permet de répondre à la question du besoin

ou non d'une blockchain (ou plus généralement d'un algorithme de consensus) pour implémenter un tel système de nommage.

La seconde contribution mineure, présentée en chapitre 9, est la proposition d'une solution au problème d'autorisation d'utilisation d'un même VC depuis différents appareils. En effet, ce problème est encore non résolu à ce jour dans le contexte des systèmes de gestion d'identité distribué. Ce système d'autorisation doit répondre à différentes contraintes. Premièrement, il doit permettre à un utilisateur, le porteur d'un VC, d'utiliser ce VC depuis différents appareils qu'il contrôle. Par exemple, son ordinateur de bureau et son téléphone portable. De plus, bien que le VC en lui-même doive être partageable entre ces différents appareils, les éléments nécessaires à l'autorisation de cet appareil ne devraient pas être copiés et partagés. Plus précisément, si cette autorisation passe par la connaissance d'une clé secrète, alors cette clé secrète doit être unique à chaque appareil. Enfin, toutes ces opérations doivent respecter la minimalisation. En d'autres termes, ils doivent cacher l'identité de l'utilisateur. Nous reviendrons plus en détail sur les solutions proposées dans l'état de l'art lié à ce problème, mais jusqu'à aujourd'hui, une seule solution remplis en effet ces critères. Cette solution est celle proposée par Hyperledger Aries [19]. Le problème de cette solution est qu'aucun document finalisé n'a été publié. Seul un brouillon existe. De plus, ce brouillon laisse à penser que la solution proposée utilise des méthodes cryptographiques coûteuses, dures à implémenter, et dont la validité est complexe à vérifier en l'état. Nous proposons donc une solution à ce problème, basé sur des techniques simples à mettre en œuvre et éprouvées, notamment des preuves à divulgation nulles de connaissance simple (en termes de complexité des propositions prouvées) et un schéma de certificat anonyme à seuil [20]. Nous utilisons ces outils pour construire un nouveau système d'authentification distribué appelé Anonymous Agreement Proof. Nous présentons cette solution en chapitre 9, expliquons comment elle doit être utilisée, et nous prouvons sa validité.

La première contribution majeure du deuxième thème, présentée en chapitre 6, étudie le pouvoir de synchronisation de deux types objets distribués : les AllowLists et les DenyLists. Cette étude est conduite au regard de la hiérarchie du consensus de Herlihy. Ces objets sont d'abord formalisés sous la forme d'objets distribués. Cette formalisation sous cette forme est, selon nous, la première de ce type. Bien que les deux objets aient des spécifications proches, leurs consensus number sont foncièrement différents. L'AllowList peut être implémentée sans consensus parmi les processus du système (son consensus number est de 1) alors que la DenyList requiert un consensus parmi un sous-ensemble spécifique des processus.

Ces objets permettent d'implémenter différentes fonctionnalités des systèmes de gestion d'identité distribués respectant la vie privée. Notamment, ces objets permettent de modéliser le processus de preuve de non-révocation d'un certificat anonyme et le processus de gestion de multiples appareils présenté au paragraphe précédent. Ces deux processus étant les deux principaux algorithmes distribués nécessaires au bon fonctionnement d'un système de gestion de l'identité

distribué. Ces résultats nous permettent donc d'en apprendre plus sur le besoin en synchronisation des systèmes de gestion de l'identité distribué en général. Plus précisément, cette partie de l'étude permet de définir que la présentation d'un certificat ne nécessite de synchronisation que entre l'utilisateur et le vérificateur. Les processus maintenant le registre distribué n'ont pas besoin de se synchroniser pour cette étape. De plus, la même étude nous permet de définir que seuls les appareils aillant le droit de modifier un DID document donné ont besoin de se synchroniser pour réaliser cette modification. En conclusion, pour implémenter les propriétés que nous avons mise en évidence dans cette introduction, et à l'exclusion du problème du nommage, cette étude nous permet de dire qu'un système de gestion d'identité distribué n'a jamais besoin de consensus au niveau de tous les processus du système, mais seulement des consensus locaux entre des sous-ensembles du système. Ce travail a été présenté dans la conférence DISC 2023 [21].

La seconde et dernière contribution majeure présentée en chapitre 8 est la proposition d'un algorithme de coopération permettant de réaliser des consensus locaux de façon efficace, en réduisant au maximum les besoins de synchronisation des processus. Ce travail est basé sur d'une nouvelle abstraction : la Coopération Adaptée au Contexte (*Context Adaptive Cooperation* en anglais, CAC). Cette nouvelle abstraction cherche à se rapprocher de la frontière de la calculabilité du consensus, sans la franchir. En effet, un résultat bien connu [22] prouve qu'il est impossible de résoudre le problème du consensus distribué de façon déterministe dans un réseau asynchrone. Dans ce contexte, le CAC est une abstraction à la lisière entre le Broadcast Fiable Byzantin (*Byzatine Reliable Broadcast* en anglais, BRB) et le consensus.

Ces deux abstractions, tout comme le CAC, permettent à un ou plusieurs processus de *proposer* des valeurs, c'est-à-dire de les distribuer aux autres processus du système. Ensuite, ces valeurs peuvent être acceptées par les autres processus du système en fonction de conditions sur le type d'accord entre les processus qui est recherché. Dans tous les cas, une valeur acceptée par un processus correct sera acceptée par tous les processus corrects du système.

Comme le BRB, le CAC permet à des processus de se mettre d'accord sur les valeurs proposées par les processus du système. Tous les processus acceptent les mêmes valeurs mais pas nécessairement dans le même ordre. Si un processus ne suit pas le protocole, alors la (ou les) valeur qu'il propose ne sera pas prise en compte par les processus du système. En revanche, le BRB ne permet pas aux processus de proposer des valeurs concurrentes. Il est possible de voir cette proposition de deux manières différentes. La première est de dire que le BRB ne permet qu'à un seul processus de proposer une valeur. Ce processus est prédéterminé par l'abstraction. Dans ce cas, il est possible d'utiliser des numéros de séquence pour chaque valeur proposés, permettant de les ordonner totalement et pour tous les processus corrects du système. De fait, les valeurs proposées ne peuvent entrer en concurrence car elles sont ordonnées. Le second est de dire que le BRB ne peut qu'implémenter des objets distribués dont la validité ne dépend pas

de l'ordre des opérations. C'est par exemple le cas des systèmes de transfert d'actif aux comptes non partagés [23, 24].

Au contraire, le CAC permet, comme le consensus, à différents processus de proposer des valeurs concurrentes. Toutes les valeurs ne sont pas nécessairement acceptées par les processus corrects. L'abstraction garantie seulement que, si un processus correct propose une valeur, alors au moins une valeur sera acceptée. Les valeurs sont acceptées séquentiellement par les processus du système, mais chaque processus accepte les valeurs dans un ordre différent. De plus, l'abstraction propose un oracle imparfait à chaque processus. Cet oracle informe le processus sur les valeurs qu'il pourra accepter dans le futur. L'oracle est imparfait dans le sens qu'il peut prédire de façon erronée qu'une valeur sera acceptée alors qu'elle ne le sera jamais. En revanche, une valeur non prédite par l'oracle ne pourra jamais être délivrée. En ce sens, l'imperfection de cet oracle est la différence majeure entre le consensus et le CAC. En effet, et de façon triviale, un oracle parfait permettrait d'implémenter le consensus, et impliquerait l'utilisation d'une certaine forme de synchronie pour notre système. En revanche, nous proposons une implémentation du CAC dans un système parfaitement asynchrone et en présence de fautes Byzantines.

Nous montrerons que malgré cette imperfection, le CAC permet d'implémenter de façon efficace des objets distribués utile pour les systèmes de gestion de l'identité distribuée. Notamment, un algorithme de consensus n'ayant besoin que de la synchronisation d'un sous-ensemble de processus du système, et un système de nommage permettant de réduire l'entropie des noms attribués comparés aux autres systèmes de nommage asynchrone. Ce travail est en cours de soumission.

Finalement, dans le chapitre 10, nous utilisons les différents outils présentés dans cette thèse pour proposer un modèle de système de gestion de l'identité totalement distribué qui respecte totalement la vie privée et qui requière un minimum de synchronisation parmi ses participants malgré la présence de fautes Byzantines.

PUBLICATIONS

This chapter lists the different papers that have been published or are currently being published during this thesis. Published papers have been published in peer-reviewed international conferences. [7]

Published papers

- **Daniel Bosk, Davide Frey, Mathieu Gestin, and Guillaume. Hidden issuer anonymous credential. Proceedings on Privacy Enhancing Technologies, 2022, vol. 2022, p. 571-607.**
- **Davide Frey, Mathieu Gestin, and Michel Raynal. The Synchronization Power (Consensus Number) of Access-Control Objects: the Case of AllowList and DenyList. In : 37th International Symposium on Distributed Computing. 2023.**

Papers in submition

- **Thimothé Albouy, Davide Frey, Mathieu Gestin, Michel Raynal, and François Taïani. Context Adaptive Cooperation.**
  arXiv preprint arXiv:2311.08776, 2023.
- Thimothé Albouy, Emmanuelle Anceaume, Davide Frey, Mathieu Gestin, Michel Raynal, and François Taïani. Asynchronous BFT Asset Transfer: Quasi-Anonymous, Light, and Consensus-Free. arXiv preprint arXiv:2405.18072, 2024.
- Thimothé Albouy, Antonio Fernández Anta, Chryssis Georgiou, Mathieu Gestin, Nicolas Nicolaou, and Junlang Wang. AMECOS: A Modular Event-based Framework for Concurrent Object Specification. arXiv preprint arXiv:2405.10057, 2024.
- Jayamine Alupotha, Mathieu Gestin, and Christian Cachin. Nopenena Untraceable Payments: Defeating Graph Analysis with Small Decoy Sets. Cryptology ePrint Archive, 2024.

7. Papers that are referred to in this thesis are highlighted in bold.

Chapter 1

Introduction

This thesis is about Privacy Preserving Distributed Identity Management Systems (PPDIMSs). This subject is of particular interest as privacy preservation becomes a popular subject. The privacy requirements of Identity Management Systems (IMSs) is particularly important as identity is related to sensitive information. The need for PPDIMS solutions comes from citizens as well as from states.

1.1 Authorisation and authentication

An IMS, be it digital or not, has two goals: authentication and authorization. These two mechanisms are the common ground for any access control mechanism.

The authorization is a two-phase mechanism. The first phase defines a set of attributes that must define an entity, the *subject*, in order for a second entity, the *holder*, to access a service. The second phase consists of verifying that these attributes indeed define the subject before letting the holder access the service. The entity that maintains and manages this mechanism is called the *service provider*. The subject can be an individual, a computer, an object, an organization, a company, or any other entity that can be defined by its own characteristics. The holder role can also be taken by different entities: an individual, the legal representative of a company, a state, a computer, etc...

To be authorized, the holder must prove to the service provider that the subject is indeed defined by specific attributes. For example, a rock climbing club only accepts new members if they know how to belay. Let Alice be the subject in this example (Alice is also the holder in this case). Alice claims that she can belay a climber. She needs to prove this claim to the club members by belaying a climber in front of Marc, a club member. Therefore, these members will validate Alice's ability to belay. Club members *trust* themselves. Hence, when Alice proves to Marc that she can belay, she also proves to other members that she knows how to belay. This notion of trust is central to any IMS. When a holder proves an attribute of a subject to a service provider M, she will claim that this attribute indeed describes the subject and that someone who M trusts certified this claim. In other words, the holder produces a claim about subject's attributes, but the service provider only trusts it if a trusted third party certifies that these attributes define the subject. We call this third party the *identity provider* or the *issuer*.

In our example, the issuer is Marc. [1] He trusts himself. Therefore, he knows Alice can belay. By transitivity, all the members of the club who trust Marc and know that Marc verified Alice's ability to belay know that Alice has this ability.

An issue still needs to be addressed in our example. If we assume Marc and Alice were alone when Marc verified Alice's ability to belay, then Marc must find a way to notify the other club members that Alice has this ability. In this case, Marc must notify the other club members of Alice's ability. To do so, Marc can use one of two methods. He can either introduce Alice in person to all the other members of the club, or he can find a way to notify the other members that Alice has the ability to belay. For example, he can send a message to the club members or issue a document that certifies that he has verified Alice's ability to belay. In this case, when the other club members meet Alice, they must verify that the person in front of them is the person Marc was talking about. This is the authentication problem.

Authentication is the process of verifying that a holder is allowed to prove that a subject possesses specific characteristics. Put differently, authentication is the action of verifying a link between the holder and the subject. If the subject is also the holder, then authentication is the action of verifying that the subject is indeed who they claim to be. Authentication can be carried out in various ways. For instance, when using a national identity card, authentication is done by comparing the information on the card (age, height, eye color, photo) with the holder.

In our example, authentication can be performed in several ways. The first method has already been presented. Marc can be present to authenticate Alice to other members of the club. A second method is for Marc to sign a document attesting that the person who carries this document is Alice. A third method would be for Marc to share a secret with Alice. When Alice shows up at the club without Marc, she reveals this secret to the present club members. Each example has specific limitations. The first method requires Marc to introduce Alice to all the group members. The second method implies that Alice could lend this paper to someone else or even copy it and distribute it to her friends. The third method implies that Alice could publicly reveal the secret.

In the remainder of this thesis, we use public-key cryptography methods to address the authentication problem. Marc announces to the club members that Alice knows how to belay a climber and that her public key is pk. When Alice shows up at the club later, she authenticates by proving her knowledge of the secret key sk associated with pk.

One thing remains to be specified here: depending on the context, the role of an entity may change. Specifically, the service provider and the issuer may be the same. Additionally, with our example, in the future, when Alice becomes a respected member of the group, she may certify that a new member knows how to belay a climber. In this case, Alice will be at the same time a subject, a service provider, and an issuer.

1. In the example, Marc is at the same time issuer *and* service provider.

Furthermore, we did not explain why we needed to distinguish between the subject and the holder. It may be necessary for a holder to prove attributes about a third party. In our example, Alice could prove she is a club member because Marc certified it. However, if some club members do not know Marc, then Alice can use the certificate that certifies that Marc is a club member to prove that she indeed has a signature from a club member. In this case, Alice uses a certificate where Marc is the subject. She is, therefore, the holder of Marc's certificate. However, she is not the subject of this certificate.

In the remainder of this work, we will also use the term "identity element" to refer to an "attribute" of an individual. An identity element is the description of an individual's particular feature (attribute) that allows them to be distinguished from others in a given group. Furthermore, we will also use the terms "holder" and "user"" interchangeably. Indeed, the holder's goal is to access a service to *use* it.

Another critical point revisited throughout this thesis is that identity is contextual. The identity elements of a user may change depending on the context in which they are operating. In our example, if Alice joins a new climbing club, the certificate issued by Marc certifying that she knows how to belay will not be valid for the members of the new club who do not know Marc. In this case, Alice will have to prove her belaying skills again to a person from the new club. Let us call this person Jeanne. In this new club, the certificate issued by Jeanne will be considered.

We have thus seen the main actors of an IMS: the issuer, the holder, the subject, and the service provider. We have also explored the core mechanisms of identification: authorization and authentication. Finally, we have covered two key concepts: the claim of possession of an identity element and the proof certifying that this identity element describes the subject. We will now focus on the methods used to create digital IMSs that meet various constraints of versatility, trust, and privacy preservation.

1.2 Digital Identity Management System

As we will see in Chapter 3, the philosophy of digital identity management systems has evolved alongside the evolution of computer science. Here, we differentiate between the philosophy of an identity management system and the techniques used to implement it. The philosophy defines the philosophical goals an identity management system needs to achieve, whereas the implementation techniques are meant to fit the criteria defined by the underlying philosophy.

This separation between identity management philosophy and identity management techniques is unique to this thesis. As we will see later, this terminology allows us to classify precisely and, therefore, formally compare different types of implementations. To the best of our knowledge, we are the only ones making this distinction.

This thesis focuses on the most recent philosophy concerning identity management systems: Self-Sovereign Identities (SSI). This terminology was popularized by Christopher Allen in 2016 [1]. In his article, the author reviews the different philosophies that had been proposed until 2016, offering a classification. He defines them as follows, starting with the oldest and least versatile to the most recent:

— **The siloed model.** This philosophy predates all others [2]. A user presents himself to a service provider and creates an account in that provider's database. If additional information is required, the provider verifies it themselves (for example, by checking the user's identity card to certify their name, date of birth, and place of birth). When the user reuses the service, they use a credential to authenticate themselves as the account owner (usually using a username and password). When accessing a new service, the user must repeat the entire process.
The limitations of this philosophy are obvious. First, an account and associated permissions are only valid for a given service. Hence, the user has to prove their identity to multiple services. Each service stores this information, increasing the risk of data breaches and attacks. Moreover, the user must trust these different services to adequately secure their servers and not share sensitive information with third parties. Additionally, the user must remember a set of pairs (username, password) for each account created. It can either be cumbersome if each password is different or prone to high-impact attacks if all the passwords are the same.

— **The federated model.** This philosophy is an evolution of the siloed model. As in the previous model, the user registers with an entity, playing the role of an identity provider. The user also registers their identity elements. Unlike the siloed model, the created account can be used to authenticate and access various services provided by an entity dependent on the identity provider. This philosophy allows the user to create fewer accounts, thus limiting the problems mentioned earlier. However, these problems still exist. Indeed, an account is only valid for a given federated entity that manages different services. Additionally, the identity provider gains more influence, acquiring more information about the user. Hence, the identity provider can easily trace the user's activity.

— **The user-centric model.** The user-centric model is close to the federated model. Similarly, the user creates an account with an identity provider. They associate identity elements to this account by providing proofs to the identity provider. The only difference with the previously presented model is that these identity elements can be used by the user to authenticate and to be authorized to a service provider that does not depend on the identity provider. However, similarly to the federated model, the service provider must communicate with the identity provider to authorize and authenticate the user. To do this, the service provider must trust the identity provider (and implement a tool

that allows the two services to communicate). Compared to the previous model, the user-centric model allows users to reduce the number of accounts they create significantly. For example, suppose all the services the user utilizes are managed by service providers that trust a single identity provider. In that case, the user may only need to create one account with that identity provider. This greatly reduces the number of accounts the user must create, the potential attack surface, and the number of service providers the user must trust.

However, the limitations of the federated model are exacerbated in the user-centric model. If the user relies on a service provider that ceases its activity, all the associated services the user was using will now become inaccessible. Additionally, if the identity provider wishes to track a user's activity, they will have even more control since all the user's authentications will go through them.

In the remainder of this thesis, we will not differentiate between the federated and user-centric models, as they work similarly. Therefore, the limitations of both models affect users in similar ways. The only difference is the degree to which these limitations can impact the user.

— **The Self-Sovereign Identity model.** This philosophy aims to give the user control over their identity elements and the proofs of these elements. The idea is to get closer to the physical identity proof model, such as an identity card. An identity card is issued to its holder by an identity provider (usually a government). The user will not need to interact with the identity provider again until the card expires. They store their identity elements in their wallet. They can choose whether to present them to a service provider, knowing that the identity provider cannot use their identity proofs without their consent as they do not hold a copy of the card. Self-sovereign identities are the digital transposition of this model. The user must always be in control of their identity elements, consent to their use in an informed manner, and be able to present proofs of identity elements at any time, meaning they do not depend on a single entity that could cease operations (for economic, technical, or censorship reasons). More precisely, Christopher Allen [1] presents ten points that any identity management system should maximize to achieve "self-sovereign identity" status:

1. *Existence.* Users' identity elements and proofs of identity must have an independent existence. This implies that they should not depend on a single system, standard, or technical implementation.

2. *Control.* The user must control their identity elements and the proofs of their identity elements. They is the only one who can request the issuance of proofs about their identity, update them, or present them. They can also choose not to reveal them. Furthermore, they must be informed of any processing related to their identity elements.

3. *Access.* Users must have access to their identity elements. They cannot depend on a provider that could potentially go out of business. They should be able to retrieve these elements at any time. This also implies they must know all assertions and proofs concerning their identities.
4. *Transparency.* The algorithms and systems implementing the IMS must be transparent. This means that the technical implementations and the management of these systems must be open to analysis by all stakeholders. Decisions made must follow predefined policies, and the outcomes of these decisions should be public. The algorithms and systems used should be accessible to everyone, with their sources being made public.
5. *Persistence.* Users' identity element and proof of identity elements must be usable over time, ideally throughout the user's lifetime. This implies again that these identity elements and the associated proofs should not depend on an entity that could cease its activities or on a given technology that may evolve or become obsolete. Additionally, as security constraints evolve with computing advancements, systems that secure and prove the user's identity elements must be upgradable, especially the cryptographic tools used and the key sizes employed to secure them.
6. *Portability.* A user's identity elements and proofs of identity must be portable. They should not only be valid if stored by a single entity or used within the context of a unique system.
7. *Interoperability.* Identity elements and proofs must be usable in as many contexts as possible. There should be no technological, administrative, or political barriers justifying that an identity element's proof is valid in one context but not in another. This property thus implies cooperation among various actors in the ecosystem, both technically, politically and organizationally. [2]
8. *Consent.* Users must consent to the presentation and use of their identity elements. First, this implies that the user must be informed of the various uses of their identity elements. Second, this consent must be informed, meaning that the user must understand the implications of disclosing identity elements to a service provider.
9. *Minimalization.* The presentation of the identity elements of a user must be reduced to a strict minimum. Users should reveal only the information strictly necessary for a

2. This last point, however, must be limited by the concept of trust and national sovereignty. A local actor cannot be expected to accept a proof from another local actor on the other side of the world if the latter does not possess certification from its own country that it is authorized to issue such proof. Additionally, in some contexts, only proof from a specific country may be accepted. For example, we use diplomas when certifying an individual's ability to perform a task. However, not all diplomas are equivalent. One cannot force a country A to accept a diploma issued by country B if the conditions for obtaining the diploma in country B are less restrictive than in country A.

service to work when they access it. This also implies that the technical solutions used to prove an identity element should not reveal more information than necessary (see chapter 5). For example, if a user wants to prove they are over 18, the service provider should only learn that information. The user's exact age should not be revealed.

10. *Protection.* Users' rights must be protected. The algorithms used and the privacy policies implemented by these algorithms must protect the user.

These ten points to maximize results from discussions that began in the early 2000s (see chapter 3). This classification is largely consensual. Some evolutions have been proposed [3], but the essence of self-sovereign identities remains the same across definitions.

Multiple technical solutions have been proposed to create an IMS that respects the principles of self-sovereign identities. All of them are based on a fundamental principle: distribution. The goal of the distribution is to propose technical solutions that respect the principles of self-sovereign identities while avoiding reliance on a single entity or a small number of entities.

This thesis considers two main families of Distributed IMS (DIMS): *partially distributed* IMS and *fully distributed* IMS. Each of these families has its own advantages and drawbacks. However, both of them achieve the principles stated by Allen. The difference between these two families lies in how they handle the auxiliary features of identity management systems, *i.e.*, properties that are not the core of authorization.

Indeed, the distribution of a DIMS mainly comes from the use of a distributed mechanism to issue, store, and present identity claims and proofs. However, nothing prevents the use of a central server acting as a trusted third party for auxiliary features. This is how *partially distributed* IMS work. In contrast, *fully distributed* IMS do not rely on any trusted third party.

More formally,, the first family is the family of *partially* distributed IMS (pDIMS). These systems do not require interaction with the issuer when the user presents proofs of their identity elements. However, some system functionalities may cease to work if one or a few entities stop operating. These functionalities may include authentication, certificate revocation, or key recovery after a loss or a theft.

The second family is the family of *fully* distributed IMS (fDIMS). Similarly to the previous family, this family does not require interaction with the issuer when the user presents proof of their identity elements. However, the continuity of system functionalities should not depend on a single entity or a small number of entities. More specifically, a fDIMS is defined for a parameter t, such that the system's operation is not impacted by the cessation of activity of t entities.

This definition implies that various system components must be distributed across different processes or computers. It also implies that these components must be distributed among different independent entities. For example, we do not consider a company managing n computers and implementing a DIMS resilient to t failures among these n computers to be a fully distributed system. Indeed, if the company goes bankrupt, the system will no longer work. If a single entity

fails, the system becomes unusable.

An important point that will guide the various proposals of this thesis and any system claiming to be based on SSIs is minimalization. This property, central to Allen's philosophy, implies that when presenting an identity element, the user should not reveal any unexpected information. Among the auxiliary information that the user must protect, we must consider unique identifiers. These include IP addresses, public keys, and various URIs required for authentication. Additionally, the signatures used to prove an identity element should not be unique. Reusing the same identifiable and unchanged signature allows for tracking the different activities of a user. This tracking could enable an adversary to gather additional information about the user's habits and even carry out *timing attacks*—making inferences about a person's identity by cross-referencing different operations and their temporal proximity. To respect minimalization, systems must, therefore, eliminate such identifiers. For this reason, we add the qualifier "privacy-preserving" to implementations claiming to be SSIs. In the following sections, we refer to Privacy Preserving fully distributed Identity Management Systems (PPfDIMS) when discussing this kind of system.

In the rest of this thesis, we will mainly focus on PPfDIMSs, although some of the results we present also apply to pDIMS.

1.2.1 Components of a Privacy Preserving fully Distributed Identity Management Systems

This section presents the various technical tools used to implement a PPfDIMS. When used accordingly, these tools can implement a system that aligns closely with Allen's philosophy. However, it is crucial to recognize that these technical considerations (the focus of this thesis) only partially ensure compliance with the Self-Sovereign Identity (SSI) philosophy. An SSI depends on both technical tools and organizational considerations. This distinction is why we differentiate between the philosophy we aim to follow (SSI) and the technical system we are exploring (PPfDIMS).

The tools we present can be classified into layers, similarly to the *Open Systems Interconnection* (OSI) model for networks. This is the Trust over IP (ToIP) model [4], which consists of four layers. The first layer is a public registry called a Distributed IDentity (DID) network, which stores the necessary information for the higher layers. This information is represented as documents known as DID documents, identified by URIs called Distributed IDentifiers (DIDs). The second layer is a communication protocol for highly asynchronous network with processes that can disconnect. This protocol is called DIDComm. It uses DIDs and DID documents to exchange messages. The third layer is the core of the identity management system; it represents the authentication and authorization mechanisms. The fourth and final layer is an organizational layer: the governance layer. Its role is to define the identity management system's goals, audit the lower layers' implementations, and certify their components. This thesis mainly focuses on

the first and third layers.

The first component of a PPfDIMS explored is the authorization mechanism. This is the most important mechanism of the third layer of the ToIP model and of any IMS. To enable authorization, a DIMS must allow users to present claims and proofs about their identity to multiple verifiers. The user should store these claims and proofs, thus ensuring the control, access, and consent properties of Allen's philosophy. Moreover, the verifiers with whom the user interacts are not defined before the issuance of the proofs of identity. Hence, the assertions and proofs stored by the user cannot depend on the verifiers that will verify them. Nevertheless, verifiers must trust the proofs handed by the users. More precisely, the verifiers must trust the entities that created the proofs. They must also be assured that the proof (and the claim certified by the proof) has not been altered between its issuance and presentation. Finally, it is essential to respect minimalization. Claims regarding a user's identity elements must be limited to the minimum information necessary for the service to function. The user should either have a proof for each potential assertion (for example, a proof that they are over 16, over 18, over 21, etc.), or they should be able to derive restricted statements from a global proof (such as proving different ages based on a proof of their date of birth). Additionally, as previously mentioned, a traceable identifier across multiple presentations can compromise minimalization. Therefore, the proofs must not contain such identifiers (or they must not be revealed to the verifiers). Moreover, the proofs themselves must not be unique identifiers. They must be modifiable, or more precisely, "randomizable." In general, we refer to a proof (associated with an assertion) that meets these requirements as a Verifiable Credential (VC). [3]

A tool that satisfies each requirement already exists: anonymous credential (AC). An anonymous credential is a type of cryptographic signature introduced by David Chaum in 1985 [6], that, in addition to digital signature properties, must verify three additional properties: blind signatures, randomizability, and zero-knowledge proof of knowledge of a signature. A blind signature is a scheme where the issuer (the signer) signs a message they do not know. They sign a cryptographic commitment to the final message. The signature recipient can then "open" the commitment, resulting in a valid signature on the original message. This allows the user to use the signature without enabling potential collusion between the issuer and verifier to gather additional information about the user. Randomizability is the ability to add random values to the signature without altering its content or validity. This makes different presentations of the same signature unlinkable. In other words, a verifier receiving multiple signatures on the same message cannot determine whether they come from the same user or different users who received signatures on the same attributes. All users receiving a signature on the same attribute from the same issuer thus form an anonymity set for that specific attribute. Presenting their attribute

3. The requirements presented here are more restrictive than those for VCs as standardized by the W3C [5]. We prefer our definition as it aligns more closely with Allen's vision.

makes them indistinguishable from those with a certificate signed by the same issuer on the same attribute. Zero-knowledge proof of knowledge of a signature makes it possible for the user to prove they know the message signed by an AC without revealing it. The proof can then be re-used to prove additional statements. For example, if the user has an AC certifying their date of birth, they can hide this information while convincing the verifier that they are at least 18 years old.

It is essential to emphasize an important point relative to the distribution of DIMSs. The distribution of a DIMS mainly arises from these anonymous credentials. The use of anonymous credentials allows a user to claim and prove identity elements without interacting with the issuer. All identity elements and associated proofs are stored directly by the user. No central service is required to perform these actions (the issuance, the presentation, and the verification). This point has sometimes been misunderstood, this is the case for U-Port [7], which we will revisit later. A blockchain or any other distributed ledger is not necessary to create a PPDIMS, or at least not for a privacy-preserving *partially* distributed IMS. However, distributed ledgers, *i.e.*, a distributed algorithm that makes it possible to publish data with specific coherence properties, may be required to implement certain auxiliary features in a distributed manner. Among those functionalities, we can cite: multi-device authorization, strong user authentication, issuer and verifier authentication, certificate scheme discovery, and credential revocation.

Therefore, these auxiliary features are optional to prove a user's identity elements. However, they are essential for addressing trust and usability issues. For instance, a certificate can be authenticated using a public/private key pair embedded in each user's anonymous credential. Proving knowledge of the private key ensures that the person using the certificate is the one to whom it was issued. However, it implies that each credential must share the same private key. If this private key is lost, the user loses access to all their credentials and must request the reissuance of each of them. Furthermore, suppose the user wants to use their credentials across multiple devices (*e.g.*, a smartphone and a personal computer). In that case, they must share their private key between devices, which is generally considered poor security practice. Therefore, a system allowing the authorization of different devices for a given user is needed. Since we are interested in *fully* distributed identity management systems, this system cannot depend on a single entity or a small number of entities (per our definition of a *fully distributed* identity management system). Therefore, and as we show in Chapter 6, we must use some kind of distributed ledger to implement these functionalities. [4]

This distributed ledger must allow different actors to publish the elements necessary to enable the auxiliary features of the PPfDIMS. [5] Since there is no centralizing entity in the system, each

4. Characterizing these ledgers is one of the contributions of this thesis.

5. It is important to note that, to ensure privacy and because the registry must be publicly accessible, *no personal information or any information that could be used to infer personal details about an individual should be published on the registry!* This remark does not apply to businesses, governments, organizations, etc., as we only

entity publishes its own information. Each entity thus has access to a "reserved space" on the ledger, called a DID document, referenced by a Distributed IDentifier (DID). The format of DIDs and DID documents is standardized by the W3C [8]. A DID is a URI that can be resolved into its DID document. In theory, the association between a DID and its DID document must be unique. A DID has the format: *did:[method]:[identifier]*. The *[method]* field is the identifier of the ledger that allows DID resolution, and *[identifier]* is the specific identifier of the DID document within the context of the *[method]* ledger. A method must specify how to create, update, delete, and read a DID document (CRUD operations). At the time of writing, the W3C had registered 193 DID resolution methods [9].

DID documents are used during the presentation of identity proofs to authenticate the device from which the user connects, verify that the credential has not been revoked, and allow the user to authenticate the verifier. Generally, this usage does not modify the ledger. During a presentation, the ledger is used in a read-only mode. This ensures the minimization property. Malicious actors could track the user's activity if a presentation leaves a trace on the ledger.

A peer-to-peer communication system is the third component of a privacy-respecting distributed identity management system. A W3C standard based on DIDs and DID documents has been proposed for this purpose: the DIDComm protocol [10]. We will not discuss this protocol in detail, as our contributions assume that such a communication protocol exists and respects privacy. In other words, it does not reveal additional information about the user's identity or activity.

1.2.2 The U-Port case and the use of distributed ledger as the unique source of distribution

The question of whether a blockchain is required to implement a PPfDIMS has long been debated. Some believed that blockchain could be sufficient as the unique distribution factor to implement PPfDIMSs. U-Port [7] is the most notable example of this belief. The idea behind this system was to store all identity elements and identity proofs on a blockchain (Ethereum in this case) or using an IPFS service. Identity elements and their associated proofs could be encrypted. However, this model raises privacy concerns. Information published on the blockchain or the IPFS system is public, even if encrypted. Any participant in the system can inspect the various identity element emissions related to a user. While the attacker may not necessarily discover the user's identity elements, they can obtain metadata about their usage and habits. With additional auxiliary knowledge, timing attacks could be performed, allowing for high-impact attacks.

Following these criticisms, U-Port's proposal is no longer supported. Resources have been redirected toward a new project, Veramo [11]. Veramo is a more conventional framework that allows implementing a distributed identity management system agnostic to the distributed ledger

consider personal data protection for individuals.

used.

More generally, the usefulness of blockchain has generated significant debate since 2017. The first to reflect on the utility of a distributed ledger were people working on the Sovrin project [12]. Sovrin is a distributed ledger project designed to support PPfDIMS. In a document [12], they describe the various uses they envision for their distributed ledger. Notably, they assert that the ledger should not contain any personal information, whether encrypted or not.

Another interesting article on the subject was published by Jolocom in 2021 [13]. This article revisits the foundations of distribution in DIMSs. They assess the fact that PPDIMSs should not necessarily use a Distributed ledger. Finally, a 2022 article also revisits the difference between DIMSs and blockchains [14]. One of the contributions of this article is that the author reviews ten implementations of DIMS and analyzes their use of blockchain. Interestingly, most projects relying on blockchain as their unique distribution factor have ceased development.

One of the key contributions of this thesis is the characterization of the minimum requirements that a distributed ledger must meet to operate a PPfDIMS. Instead of asking whether a blockchain is "necessary", we go further by precisely characterizing the synchronization requirements of such ledgers. As we will see later, while a ledger is necessary, it does not necessarily need to be distributed (in the case of a *partially* distributed identity management system). If the ledger is indeed distributed, the synchronization constraints of the system are much weaker than those offered by a blockchain. A blockchain completely orders all system operations, which has a high cost in terms of latency, cryptographic function computation, etc. In contrast, we prove that a PPfDIMS only requires weak synchronization guarantees, which we characterize in detail in Chapters 6 and 7. Therefore, if a distributed ledger is chosen to implement an identity management system, it can be implemented economically. Chapter 8 presents an optimal abstraction for such use cases. The abstraction presented in this chapter can be used for other use cases where synchronization requirements are low. Interestingly, this applies to most distributed applications, such as naming services, access control, and asset transfers. Finally, an implementation based on those results is presented in Chapter 10.

1.3 Contributions

In this thesis, we explore the various tools required to create a PPfDIMS. We focus on distributed techniques aimed at maximizing user privacy while enhancing the usability of these systems. The goal is to answer the following question: What tools enable the implementation of a fully decentralized IMS that fully preserves user's privacy while minimizing the synchronization required between system participants? The contributions of this thesis are divided into two main themes. The contribution of the first theme focuses on Layer 3 of the ToIP model. Specifically, this contribution is an improvement to the state-of-the-art concerning privacy guarantees of

anonymous credentials. The contributions of the second theme focus on the interactions between layers 1 and 3 of the ToIP model. These contributions include analyses and proposals for systems that interact with the distributed ledger of Layer 1 and the anonymous credential presentation mechanism of Layer 3. The goal is to formally understand how a distributed ledger can be used to implement a PPfDIMS and to use these analyses to propose efficient implementations that address specific challenges related to identity management systems.

The contribution of the first theme is presented in Chapter 5. This contribution starts from a simple observation: while anonymous credentials schemes claim to enforce the principle of minimalization, the verifier's knowledge of the public key of the issuer of a credential reveals non-expected additional information about the user. Indeed, knowledge of the issuer's public key is necessary for any cryptographic signature scheme. However, knowing this public key implies knowing the identity of the issuer, which can disclose additional information about the user. We provide two examples to illustrate this point. First, an issuer is often a local entity (geographically). A user requesting the issuance of a credential related to their civil status will ask the town hall of their place of residence. In this case, when the verifier checks an identity element related to the user's civil status, they will also learn the user's place of residence, which is (with high probability) the same as the municipality issuing the certificate. Even more troubling, if the verifier and issuer collude, and if the verifier possesses additional information about the user,[6] then this additional information can allow the verifier to uniquely identify the user in the issuer's database. This implies that both the issuer and the verifier can track the user's activities. Beyond the privacy violation and the breach of the minimalization principle, such an attack has a greater impact due to the users' ignorance. Indeed, a user who believes the properties of an anonymous credential protect them will be less cautious and reveal more information about themselves than a user who does not use an anonymous credential.

To prevent this attack, we propose an anonymous credential scheme where the issuer is hidden from the verifier. More precisely, before verification, the verifier publishes a list of trusted issuers. The user then uses this list to randomize the public key of their credential issuer while constructing a proof that the original (non-randomized) key associated with the randomized key belongs to an issuer in the verifier's list of trusted issuers. The user then presents this randomized key, their randomized credential, and the aforementioned proof to the verifier. The verifier can thus verify that they trust the issuer of the credential and that the issuer has indeed signed the credential. However, the verifier does not learn the exact identity of the issuer. This type of anonymous credential exhibits the trust properties of classic anonymous credentials while avoiding the attacks previously mentioned. This work was published at the Privacy Enhancing Technologies 2022 conference (PETs) [15].

6. This additional information can be easily obtained in the context of "real-life" verifications. In this case, the verifier can record the user's physical characteristics and compare them to those stored in the issuer's database.

The contributions of the second theme encompass four contributions, two major and two minor.

The first minor contribution, presented in Chapter 7, analyzes the naming problem. The naming problem is a central issue for PPfDIMSs. This analysis revisits the Zooko's Triangle problem [16]. In a 2001 blog post, Zooko analyzed the assignment of names within a given namespace, where a name is considered as a string of characters. Zooko identifies three desirable properties for a name: security, decentralization, and human-meaningfulness. The security property is informally defined as the ability for a name to be used exclusively by its legitimate owner, where "used" refers to proving ownership of the name. In the context of decentralized identity management systems, this means being able to modify a DID document associated with a DID. The decentralization property refers to different processes sharing the same association between a name and a resource. In other words, a name characterizes the same entity for all members of a given system. The human-meaningful property is the most complex to define. Its original definition implies that a human could easily manipulate the name. We prefer the definition of "humanly-choosable", meaning the system allows users to choose the name they wish to use. Zooko's trilemma conjectures that these three properties cannot be achieved simultaneously. Confusion arose regarding this trilemma since it was first presented. For example, the Wikipedia page on the subject claims that there are implementations that refute the conjecture. Similarly, a 2011 blog post by Aaron Swartz [17] claims that it is possible to solve Zooko's trilemma. However, as we show, these proposed solutions do not resolve the problem but rather circumvent it by associating two types of names: pointers and nicknames. We discuss these solutions in more detail in Chapter 7. Lastly, we explore an interesting point raised in a paper by Mühle, Grüner, Gayvoronskaya, and Meinel [18], which suggests that a blockchain is necessary to circumvent Zooko's trilemma. We propose to formally explore and address this claim.

Our contribution is twofold. First, we formalize the Zooko Triangle problem and provide a formal proof of the impossibility, which was previously only a conjecture. We then formalize the different methods for circumventing the problem and analyze their synchronization requirements. This analysis allows us to answer whether a blockchain (or, more generally, a consensus algorithm) is needed to implement such a naming system.

The second minor contribution, presented in Chapter 9, proposes a solution to the problem of authorizing the use of the same Verifiable Credential (VC) across different devices. This issue remains unresolved in the context of PPfDIMSs. The multi-device authorization system must satisfy several constraints. First, it must allow a user, the holder of a VC, to use it across multiple devices he controls, such as a desktop computer and a mobile phone. Additionally, although the VC itself must be shareable across these devices, the elements necessary to authenticate a device should not be copied and shared. Specifically, if the authorization relies on a secret key, that key must be unique to each device. Finally, all these operations must comply with the principle of

minimalization, meaning they should hide the user's identity.

We review the existing state-of-the-art solutions related to this problem. To this day, only one solution appears to meet these criteria: the one proposed by Hyperledger Aries [19]. However, the problem with this solution is that no finalized document has been published, only a draft exists. Moreover, this draft suggests that the proposed solution relies on computationally expensive cryptographic methods, which are difficult to implement and whose validity is complex to verify in their current form. Therefore, we propose a solution to this problem based on simple, well-established techniques, including zero-knowledge proofs and a threshold anonymous credential scheme [20]. We designed a new distributed authentication system called Anonymous Agreement Proof using these tools. We present this solution in Chapter 9, explain how it should be used, and provide a proof of its validity.

The first major contribution of the second theme, presented in Chapter 6, investigates the synchronization power of two types of distributed objects: AllowLists and DenyLists. This study is conducted under the lens of Herlihy's consensus number hierarchy. These objects are first formalized as distributed objects. Although the specifications of both objects are closely related, their consensus numbers are fundamentally different. The AllowList can be implemented without requiring consensus among the system's processes (its consensus number is 1), whereas the DenyList requires consensus among a specific subset of processes.

These objects enable the implementation of various features of PPfDIMSs. Notably, they can be used to model the revocation of an anonymous credential or the multi-device authorization mechanism discussed in the previous paragraphs. Allowlists and Denylists are the main distributed components of a PPfDIMS. Hence, our findings provide greater insight into the synchronization requirements of such systems. More specifically, this study reveals that presenting a credential only requires synchronization between the user and the verifier. The processes maintaining the distributed ledger do not need to synchronize for this step. Additionally, the study shows that only devices authorized to modify a given DID document need to synchronize to make such changes. In conclusion, excluding the naming problem, this study allows us to assert that a PPfDIMS never requires consensus among all processes but only a localized consensus between a subset of processes in the system. This work was presented at the DISC 2023 conference [21].

The second and final major contribution, presented in Chapter 8, proposes a cooperation algorithm that can be used to achieve efficient local consensuses while minimizing synchronization requirements. This work is based on a new abstraction: the Context Adaptive Cooperation (CAC). This abstraction aims to approach the boundary of consensus computability without crossing it. Indeed, a well-known result [22] proves that it is impossible to deterministically solve the distributed consensus problem in an asynchronous network. In this context, CAC is an abstraction at the edge between Byzantine Reliable Broadcast (BRB) [25] and consensus. Like BRB, this abstraction can be implemented in a fully asynchronous network, albeit the pres-

ence of faults. However, like consensus and unlike BRB, this abstraction allows the processes to propose conflicting values. However, unlike consensus, values are accepted sequentially by the system's processes, but each process may accept them in a different order. Additionally, CAC provides each process with an imperfect oracle. This oracle informs the processes about values they may accept in the future. The oracle is imperfect because it may incorrectly predict that a value will be accepted. However, a value not predicted by the oracle will never be accepted. In this sense, the imperfection of this oracle is the main difference between consensus and CAC. A perfect oracle would trivially implement consensus, requiring some form of synchrony in the system. In contrast, we propose a CAC implementation in a fully asynchronous system with Byzantine faults. Despite this imperfection, we show that CAC allows for efficient implementation of distributed algorithms that are useful for PPfDIMSs. Specifically, it enables a consensus algorithm that only requires synchronization of a subset of the system's processes, as well as a naming algorithm that reduces the entropy of assigned names compared to other asynchronous naming systems. This work is currently under submission.

1.4 Description of the chapters

The remainder of this thesis is as follows. Chapter 2 presents the main components required to build a functional PPfDIMS from a high-level point of view. This chapter also characterizes those components.

Chapter 3 presents the state of the art in terms of digital Identity Management Systems. This chapter uses a historical approach. It begins in the 80s with the advent of the siloed model and ends with the latest developments in terms of PPfDIMS.

Chapter 4 presents the different distributed models used throughout the thesis and the main cryptographic and distributed algorithm considered or used in this document.

Chapter 5 presents the Hidden Issuer Anonymous Credential scheme. The chapter presents the motivation for this new anonymous credential scheme, a formal definition of the scheme, an instantiation of the algorithms, and a comparison to a concurrent scheme [26].

Chapter 6 studies access control in distributed systems from a formal point of view. AllowList and DenyList, two access control objects, are formally defined. Then, their consensus number is studied [27]. Finally, the knowledge of the consensus number of those objects is used to analyze the theoretical need for a PPfDIMS in terms of synchronization between the different processes of the system.

Chapter 7 analyzes a second important distributed feature of PPfDIMS: naming algorithms. Naming algorithms are used to associate a resource with a name. In PPfDIMSs, those algorithms are of particular interest as they are used to create DIDs and DID documents. This chapter studies the consensus number of those algorithms. Furthermore, it formally proves Zooko's im-

possibility, which states that a name cannot simultaneously be secure, distributed, and human-meaningful. This proof leads to categorizing the different types of names used in a distributed system.

Chapter 8 presents the Context Adaptive Cooperation (CAC) abstraction, a distributed abstraction that can be used as a building block to solve low contetion problems. This abstraction is used in the remainder of this thesis as problems relative to identity management have low contention probability. Hence, this abstraction, which can be implemented deterministically in a fully asynchronous system, can help improve the efficiency of PPfDIMSs. In Chapter 8, the CAC abstraction is formally defined, two algorithms that implement the abstraction are presented, and examples of usages of the abstraction in low contention problems are provided. The first proposed example is a naming algorithm, and the second is a consensus algorithm that reduces the synchronization requirements between processes of the system.

In Chapter 9, a solution to the multi-device authorization problem is presented. This problem can be stated as follows: In a fully distributed system, how can a user be authorized to prove their identity from multiple devices while maintaining a high level of security, privacy, and usability? It is based on the analysis of AllowList and DenyList conducted in Chapter 6 and on a specific type of anonymous credentials, threshold anonymous credentials. In this chapter, the problem of multi-device authorization is formally defined. Then, the Anonymous Agreement Proof abstraction is formally presented and used to solve the multi-device authorization problem.

Chapter 10 summarizes the technical developments of the thesis. It uses the results of chapters 5 to 9 to propose a high-level implementation of a PPfDIMS. The idea of this section is to provide a guideline for future work to develop a fully distributed and fully privacy-preserving implementation of a PPfDIMS that has minimal synchronization requirements between the different processes of the system. It considers each requirement exposed in Chapter 2.

Chapter 11 presents a personal point of view of the political and philosophical impacts that the large-scale deployment of a PPfDIMS such as the one presented in Chapter 10 could have on individuals.

Finally, Chapter 11 concludes this thesis. It presents possible evolutions of the different works presented in this thesis. Furthermore, it assesses subjects not addressed by this thesis relative to PPfDIMS.

Chapter 2

Main components of a Privacy Preserving Identity Management System

This chapter summarizes the different components required to build a fully functional and usable Privacy-Preserving and fully-Distributed Identity Management System (PPfDIMS). The first and most important component is a signature scheme respecting user privacy. As explained in Chapter 1, a distributed identity management system could be implemented only using such a signature scheme. However, many auxiliary features are required to make this scheme fully functional and usable. Those features are required to hope that, someday, individuals choose to use DIMSs as a day-to-day identification mechanism.

Each contribution of this thesis tackles one or multiple features presented in this chapter. They mainly focus on the computational requirements and the efficiency of those features. In Chapter 10, we use the different developments of this thesis to propose a high-level implementation guideline for a full-fledged PPfDIMS that proposes all the features listed in this chapter.

2.1 Privacy Preserving signature scheme

The most important component of a PPfDIMS is a way for an identity provider to issue proof that a specific identity element describes a user. As explained in Chapter 1, this proof is called a Verifiable Credential (VC). To fulfill the requirements of a DIMS, VCs must comply with different requirements. First, VCs must be stored by the user, and a VC's presentation to a service provider should not require interaction with its issuer. Second, the system's evolution must not impact VC's validity. If an actor enters or leaves the system after the VC's issuance, this VC should still be valid. Third, the VC should be linked to its issuer. Indeed, if it is not, a self-issued credential would have the same value as a credential issued by a state. Hence, the verification of a VC should only be valid if a trusted issuer issued it. Third, a VC must be linked to the claimed identity element. It is to say that the VC's holder should not be able to make a verifier believe that a VC certifying a claim c certifies a claim $c' \neq c$. Finally, the proof must

preserve the user's privacy. Using the minimalization definition, privacy preservation implies that the issuance, storage, and presentation of a VC do not leak information not intended by the user. Therefore, the VC must not constitute a unique identifier that could be used to track the user across multiple presentations. This property is called *unlinkability*. Furthermore, the user must be able to *selectively disclose* information. When a user is required to prove a specific identity element, they should not prove anything else. For example, if they want to prove they are over 18, they should not give their date of birth. These two properties, unlinkability and selective, disclosure make it possible for a VC to respect the minimalization property stated by Allen [1]. An additional privacy property can be required; it is the *issuer indistinguishability* property. When a user presents a VC to a verifier, disclosing the issuer's identity can leak extra information, thus breaking the minimalization property. For example, suppose a VC proving the date of birth of a user is only issued by the town authority in which they lives. In that case, the disclosure of the identity of the issuer discloses unintended information about the user's place of residence. Issuer indistinguishability can, in this case, enforce minimalization.

There exists one type of cryptographic signature scheme that meets all the requirements stated above, *anonymous credentials* (ACs) [6]. Extended details about this scheme and its different flavors and properties are presented in Chapter 5. An Anonymous Credential scheme is a signature scheme with additional properties. First, it is randomizable. Random numbers can be added to a signature to enable *unlinkability*. Second, like any other secure cryptographic signature scheme, it supports the Existential Unforgeability under the Chosen Message Attack (EUF-CMA) property. This property means that the content of a signature cannot be modified by the signature's holder, even if this holder can choose signed messages. Most AC schemes provide Zero Knowledge Proof (ZKP) of signature, meaning that the holder of a signature can prove statements about signed messages without revealing the actual message, thus enabling selective disclosure. Furthermore, AC does not require the issuer's participation during a signature presentation, and the validity of the signature does not depend on the existing issuers or verifiers of the system (unlike other signature schemes such as ring signatures [28] or group signatures [29]). Like other signature schemes, the verification of the signature requires the use of the issuer's public key, thus ensuring that the signature originates from him. Therefore, an AC scheme meets all the requirements stated in the previous paragraph, except for the issuer indistinguishability property. Chapter 5 presents an AC scheme that provides this additional property while maintaining the "trusted issuer" property (the fact that the verifier knows they trusts the issuer of the credential). Other researchers have explored the same problem concurrently, it is the case of Bobolz et al. [26], Connolly et al. [30] and Sanders and Traore [31].

2.2 Informationnal features

Actors of an IMS need to publicly share information about their own identity. In this section, we give a list of the information that may be required to be published. This list is not exhaustive. Furthermore, the information published can consist of identifying data. Hence, and due to the minimalization property, this feature should not be used to publish information about a user whose privacy must be protected. This section, therefore, applies only to actors that assume the issuer or the verifier role.

2.2.1 Information published by an issuer

First, an issuer can publish information about the information it certifies. This publication has several goals. For users, it allows them to automatically detect which issuer can certify a specific identity element, thus allowing them to directly contact the right issuers. It provides the same information for the verifier as for the user, allowing them to identify the issuers they can trust to certify specific identity elements. It also makes it possible for verifiers to identify some fraud attempts. Indeed, suppose an adversary steals or gains control over an issuer's secret key and tries to issue a credential about an identity element it is not supposed to. In that case, this malicious behavior can be automatically detected.

Second, the issuer can publish its service endpoints. This information is used to know how to contact the issuer. It will usually be an Ip address, or a URL, but depending on the network layer, it can be another address that makes it possible to contact the issuer reliably. A unique issuer may publish different endpoints for different services. For example, one specific endpoint can be used for each type of credential issued. Furthermore, some issuers may propose specific services, like automatic refresh of credentials as explained in the W3C draft [5]. Specific endpoints may identify those services.

Third, issuers may need to publish important information, such as the protocols they use and the cryptographic primitives they support. This information helps users determine whether they can transact with the issuer. For the verifier, it is also helpful to know which issuers they trust use which cryptographic primitive, thus allowing them to update their verification methods accordingly.

2.2.2 Information published by a verifier

The information published by a verifier is similar to that published by an issuer. First, a verifier can publish service endpoints. Similarly to the issuer, a verifier may publish different endpoints for each service it maintains. They can also publish the protocols they support and the cryptographic algorithms they accept. This information is used to help the user determine whether they can perform a presentation with this verifier.

The only difference between an issuer and a verifier is that a verifier may publish the list of issuers it trusts. This list can be augmented with which type of credential is accepted from which issuer. This information lets the user know if they have the required credentials to prove their identity to the verifier. If they do not, the user knows to whom they can request the issuance of a new credential.

2.3 Key management features

Key management is one of the main requirements of any public key-based protocol. First, and because PPfDIMS are supposed to be used by many individuals, it is highly probable that some of them will lose their secret keys at some point. Therefore, a key recovery mechanism must exist. This mechanism can take multiple forms, but it should not contradict the control, access, and consent properties stated by Allen [1] (cf Chapter 1). Furthermore, a key loss should not revoke access to the user's credentials or DIDs. Otherwise, usability would be decreased as some credential issuance may require heavy administrative formalities. Hence, the key recovery mechanism must be built to give the user back control over all its proofs of identity elements while ensuring that no adversary can abuse this mechanism to gain unintended control over one's credentials.

A second important key management feature is the key rotation mechanism. This mechanism is widely adopted as a best practice to undermine cryptanalysis attempts [32]. Like the key recovery mechanism, this key rotation mechanism should not impact usability. A key rotation should not revoke a user their access to their proofs of identity elements nor their DIDs and DID documents.

We analyse the theory behind these problems in Chapter 6. Then, we propose an implementation of a PPfDIMS that implements these key management features in Chapter 10.

2.4 Strong and versatile authentication features

The main authorization mechanism of a PPfDIMS is its AC scheme. However, it is not enough for an individual to prove the possession of an AC to prove that this AC was issued to him. Indeed, an AC can be stolen. The most obvious way of doing this is using a verifier that stores a credential and uses it as its own. Therefore, a PPfDIMS needs to provide a strong authentication method. By strong, we mean that the authentication mechanism must, with high probability, prove that a credential holder is its rightful owner. Furthermore, this authentication mechanism should not impact usability. That is, it should not restrict legitimate usages. One legitimate usage is that a user should be able to use its VCs from different devices. For example, let us assume a user owns a computer and a cellphone; then, it should be able to use its VCs from

both of them. However, it is generally considered a bad practice to share a key among different devices. Hence, each device should possess its own authentication means. Furthermore, using the requirements exposed in Section 2.3, a key loss or a key rotation should impact user access to its VCs. Hence, the authentication mechanism cannot be directly linked to an element "hard-coded" in a VC. For example, the linked data type authentication method proposed by the AnonCred project [33] does not respect the usability requirements exposed in this section. We propose in Chapter 9 the first efficient, strong, and versatile authentication method for PPfDIMS.

The authorized devices should be revocable for this mechanism to be interoperable with the key management feature. A manager for a specific VC should be able to grant or revoke the right to present a VC at any time.

2.5 Revocation features

In Section 2.3, we tackled the problem of a user's losing control over their secret keys. In this section, we add a new requirement: how to deal with a loss of control over a VC. In this case, VCs must be revoked. A revoked credential cannot be used in an authorization process. Revocation can emanate from the VC's holder and the VC's issuer.

VC holders may want to revoke their credentials in multiple cases. The first case is if a credential gets stolen or the user gets evidence that their VC is misused. The second case is if the user's identity evolves; for example, if they change their place of residence, then they may want to revoke older and outdated credentials.

On the issuer's side, if it gets evidence that a user misuses a credential it issued, it must be able to revoke it. Furthermore, if an issuer loses control of one of its issuing secret keys, it may want to revoke all the credentials issued with this key (in addition to other countermeasures). The theory of revocation in distributed systems is conducted in Chapter 6 and these results are used to implement a PPfDIMS with revocation capabilities in Chapter 10.

2.6 DID-capable ledger and naming system

Except for the Anonymous Credential scheme, all the requirements exposed in the previous sections require some form of publication of information. The information published must be highly available even if one actor has a technical issue. For example, if the services of an issuer go down for a short period, then, in a PPfDIMS, its revocation list should still be available. [1] Hence, there should exist some form of ledger capable of implementing each feature stated in this chapter. We assume in this thesis that this ledger mainly stores DIDs and DID documents [8]. As we can see, most of the additional features a PPfDIMS requires depend on a specific

1. This statement may not be true with a partially distributed identity management system, see the different definitions given in Chapter 1.

actor. Actors mainly publish information about themselves. Therefore, DID documents, which belong to and describe individual entities, can be used for most of these features. The additional requirements are summarized in Chapter 10.

Furthermore, DID-capable ledgers need to allow actors to create DIDs. Therefore, we must study how individuals can create DIDs and the associated DID documents. This study is conducted in Chapter 7.

Hence, a DID-capable ledger is enough to implement all the requirements stated in this chapter. However, the DID standard is broad. A DID-capable ledger can take many forms. In section Chapter 10, we use results from Chapters 6, 7, 8, and 9 to characterize this ledger.

2.7 Accountability feature

In this chapter, we proposed features to address misusage, theft, or loss of control over a secret key or a VC. However, we did not say how to identify these misusages. Even though we do not study this feature in this thesis, it may be required to be implemented. State authorities will likely require this feature. This accountability feature should enable authorized actors to detect malicious behaviors and de-anonymize the actor who acted maliciously. This feature may be required to allow actors to build proof that some individuals behaved fraudulently. Therefore, it may be necessary to hold this individual responsible in court. However, this feature also directly contradicts Allen's minimalization property. Furthermore, if a malicious actor is able to use this accountability feature to decrease users' privacy, then individuals will no longer trust the system and will not use it.

This thesis is about privacy preserving IMS, hence, we do not interest ourselves in the accountability feature, which could reduce privacy guarantees given to system users. However, other works [34, 35] have already tackled this point. We invite the interested reader to refer to them for a more in-depth comprehension of the topic.

Chapter 3

State of the art

This chapter provides an understanding of the historical developments of privacy-preserving Distributed Identity Management Systems (PPDIMSs) and the latest developments in this domain. This study of state-of-the-art focuses on two different but complementary sets of works: academic works published in conferences, articles, and workshops, and non-academic works, such as industry developments, whitepapers, blog posts, standards, and documentation of frameworks.

This thesis is not a historical thesis. Hence, the method used is not historical and has to be taken as it is. It is to say, as an overview of the historical development of digital IMSs. We do not claim to be exhaustive. This section is based on the work of Allen [1] that paved the way for a new paradigm in the Identity Management Systems world.

State of the art related to each topic studied in this thesis are incorporated in the relevant chapters; namely, we study developments related to anonymous credentials in Chapter 5, and state-of-the-art distributed ledger technologies in Chapter 6 and Chapter 8. We also focus on the state-of-the-art authorization methods related to DIMS in Chapter 9.

3.1 The early days of Identity Management

Access control is highly correlated with the creation of computers. The first type of identity management system was developed to access shared computers. With the creation of the Compatible Time-Sharing System [2], each user had an account created by the manager. This account was used to access shared computer resources. This identification mechanism is a simple siloed example where an issuer (the manager) provides proof (an account) to access a service (computer's resources). This model was then reused in many other use cases. On the Internet, the primary authentication method for accessing websites is by registering a username and password. The problem with this method is its high impracticality for large-scale usage. First, the user has to remember multiple account information for the different services they access. Second, each user has to prove their identity to the service provider (which is also the identity provider in this case). Suppose the user has to prove an extra identity element. For example, if the user is a university student, they have to prove they are a student to each new service provider requiring this information. They must find a way to present their physical credentials, proving they are a university student. The service provider, on the other hand, has to verify each user's

credentials. This protocol is slow and error-prone. Moreover, this method increases the surface of potential attacks. User's identity elements are scattered and stored throughout the different service providers. Hence, it is a considerable threat to user's privacy. Finally, with this type of siloed identity management model, users have no control over their identity elements. They do not know what information is stored by the service provider. When the service provider verifies the user's credentials, it could just store the important information, such as the fact that the user is a university student. However, it could also store extra personal information related to the user and written on the user's credentials.

In 1988, some of these issues were addressed with the first X.500 standard [36]. This standard aimed to provide a unified way to implement directory services. The important section of this specification, which is still used nowadays, is the X.509 standard. It was first created to securely authenticate clients who accessed the directory. It is based on public key cryptography for authentication, where an X.509 certificate is issued by a trusted party to a user, and this user then proves it knows a secret key associated with the public key presented in the certificate. The certificate holder is identified by a Distinguished Name (DN), which is supposed to identify this subject uniquely. This standard was later used to authenticate any service provider or user online. The interesting point is that this standard makes it possible to transfer trust. An identity provider can issue an X.509 certificate whose DN is x to someone. Then, suppose this person presents this credential to a third party that trusts the certificate's issuer. In that case, this third party will know that x identifies the user because it trusts the issuer in verifying this property of the user.

A second trust-based authentication means was developed in the meantime, Pretty Good Privacy (PGP) [37]. It was first released in 1991. It is an encryption mechanism that also provides authentication. This mechanism uses e-mail addresses as identifiers. PGP, like X.509, also makes it possible to transfer trust. Alice can prove she trusts Bob by signing his PGP public key. Hence, if Charlie, who trusts Alice, receives a signature from Bob, and Alice signs Bob's public key, then Charlie can be confident that Bob is indeed who he claims to be. This trust transitivity mechanism comes with a confidence level. This confidence level allows Alice to state to what point she is confident in the fact that Bob is who he claims to be.

These mechanisms (X.509 and PGP) can be considered authentication methods. Though it is important to highlight them as the two first authentication methods that provide trust transitivity, they do not imply any identity management philosophy. They can be used to implement siloed, federated, or distributed Identity Management Systems.

In 1997 [38], Ellison was the first to question the nature of online identities and how to prove one's identity claim. Ellison's paper reviews the existing solutions of the time, *i.e.*, PGP and the X.500 family, and assesses their limitations. Ellison's paper proposes several use cases and protocols to identify a subject online. However, the most interesting part of Ellison's work is the

reflection given by the author about identity and the definition of an entity on the Internet. He also comes back to a notion that we will explore later in this thesis: the notion of subjectivity of identifiers. He assesses the fact that an identifier is only valid in a given namespace. He further links the contextuality of names to the contextuality of the subjects' identities. Indeed, identity is a contextual concept, and a given subject may have multiple identities depending on the context. For example, Alice can be known under her name for her employer, but on a forum, she can be known under a pseudonym.

Ellison's paper seems to have had little influence on the domain until Kim Cameron wrote the "The laws of identity" in 2005 [39]. Nonetheless, from 1997 to 2005, some developments have been made regarding IMSs. We can talk about the Microsoft Passport initiative, an IMS developed in 1999 by Microsoft to allow users to authenticate and be authorized on multiple websites with a single identity provider. It was the first federated IMS under Allen's classification [1]. This IMS was even used by non-Microsoft-related companies such as eBay. However, the system was shut down in 2005 because of its several flaws. Among them was the tremendous control of Microsoft over user identity elements and the lack of interoperability with other systems and service providers [40, 41]. This initiative has to be noticed as the first IMS to associate accounts with services rather than with people.

In the early 2000s, multiple groups were created to question identification on the Internet. These groups elaborated the first works that later paved the way for Allen's Self-Sovereign Identity [1]. The first group to be created around this question is the Liberty Alliance, in response to Microsoft Passport's initiative [42]. Sun Microsystems created this consortium. It grouped multiple companies [1] that aimed at building a truly federated IMS. This consortium also focused on user privacy and security. Their main achievement is the creation of interoperable standards for Single-Sign-Ons (SSOs) that are still used nowadays, for example, the SAML standard.

A second initiative of the time is XNS.org's proposal [43]. Even though we are left with little information about this initiative, it seems the goal was to provide an interoperable single sign-on service for federated identity.

A third interesting initiative at the time was the Augmented Social Network report [44]. It is the first notion of a trust layer for the Internet. The notions explored in this work are really interesting, as they are the basis of modern PPDIMSs: interoperability, persistence, control, and self-organization (or distribution). Interestingly, the authors of this work assess the contextual aspect of identities and trust. Unlike previous works, they also assess that identity management is not only about commercial relationships. As we will see in the following sections, this is a major consideration when discussing IMS. Furthermore, this paper criticizes the Liberty Alliance and Microsoft Passport, as they are built by and for private companies to increase commercial

1. For example, Apache Software Group, NTT DoCoMo, Nokia, VISA, RSA Security, Real Networks, BankAmerica, Vodaphone, ...

growth, thus, going away from the Internet's original spirit, which was to bring people together on an equal and open basis. The paper also assesses a problem the EU's GDPR will address later. If public authorities do not control how private companies manage personal data, then these private companies have the liberty to use (and abuse) their prominent position. We will raise this question again in Chapter 11. Even with privacy guarantees, any IMS can be misused if users are unaware of privacy risks and if the public authority does not regulate (and verifies) the usage of this IMS. However, this work has some issues. The "persistence" notion the work relied on was based on a unique global identifier. The goal was to efficiently identify people and share knowledge with them. However, this implies that individuals are uniquely identifiable across multiple domains, thus greatly reducing their privacy. Nowadays, when we talk about persistence, we mean that the IMS can still work if one or more actors cease their activity or have technical issues.

In 2005, following the Liberty Alliance initiative, the OpenId standard [45, 46] was developed. This standard is now used as one of the main technologies to build federated IMS. For example, the latest evolution of this standard, OpenId Connect, is used by the French state's Single Sign On service, FranceConnect [47].

The first step toward the "Self-Sovereign Identity" philosophy, which is now considered the best practice in terms of IMS, was formalized in 2005 by Kim Cameron through two papers, one published under his name and the second under Microsoft's name. Cameron was responsible for Microsoft's IMS and was against the passport initiative as it removed control of subjects over their identity elements. The first paper [39] aims to provide objective "laws of identities". The idea, which can be criticized, is to express identity through axioms. Those axioms should enable the creation of an identity metasystem, *i.e.*, a trust layer on the Internet, or, in other words, a set of standards and interoperable technologies that make it possible to build multiple IMSs that can work together. The axioms proposed in this paper are user control and consent over its identity elements, minimal disclosure, only justifiable parties can access identity elements, unlinkability of identity elements, pluralism of operators and technologies, and human-centric designs. As we can see, those axioms will later be used by Allen to build its ten laws of Self-Sovereign Identity. The only flaw of those axioms is that they are not minimal, *e.g.*, the justifiable parties and the minimal disclosure axioms are very similar. On the other hand, this work is the first to clearly introduce a subject's identity as a set of claims about its individuality, claims that can be evaluated and certified by third parties.

The second paper written by Cameron [48] is the vision of Microsoft related to the "laws of identity", or how they aim to implement such an identity metasystem. The answer to this challenge will be the discontinuation of Microsoft Passport, and the creation of InfoCard [49], a federated identity system aiming to enhance user control and consent. The idea was for users to possess ID cards on their personal computers. When requesting the presentation of an identity

element, users should have been able to choose between different ID cards depending on the context. Unlike Passport, Microsoft was not supposed to be the only identity provider with InfoCard, thus qualifying as a true federated IMS. This system stopped working in 2010 and was replaced by the development of U-Port, a blockchain-based IMS (whose development has also stopped).

The early IMS developments presented in this section have one point in common: they all propose some form of federation in terms of identity management. None of them proposed the implementation of an actual distributed IMS philosophy, even though the work on the augmented social network [44] and the work of Kim Cameron [39, 48] was the first step towards SSI.

Those early works related to federated identities culminated in Europe in 2014 with the electronic IDentification And Digital Services (eIDAS) 1.0 regulation [50]. The regulation aimed to propose digital identification as a means of interaction between European governments (or private companies acting on behalf of European governments) and European citizens. This digital identification service aimed to facilitate administrative work for citizens and let them use a unique identification system in their own country and other European countries. This regulation has already laid the basis for digital IMS. However, decentralization and privacy were not central in this early proposal. The main point of this regulation was the regulation of identification technologies used by each country. The two most interesting points in this regulation regarding our subject is the identification of three "levels of assurance" (LoA) [2]. Where a higher level of assurance means that the claimed identity element is more likely to describe the user, or, more precisely, the probability of fraud is reduced. The second point is the interest of the regulation for interoperability of IMSs. The regulation proposed multiple tools for adopting interoperable IMS; namely, they made it mandatory for states to accept identification means proposed by other states of the European Union. The regulation also assesses the need for accountability, revocation, transparency (regarding the technology used and requirements), technological neutrality, and standardization. France connect [47] can be seen as an heir of this regulation.

However, as stated earlier, the eIDAS 1.0 regulation did not consider decentralization and user privacy. Furthermore, the regulation mainly focused on administrative services, thus reducing the overall adoption of the proposed IMSs. A new regulation, eIDAS 2.0, is currently being developed to tackle these issues.

3.2 The rise of Self Sovereign Identity

Until 2010, most IMS propositions focused on improving users' control over their identity elements. Although this is an important topic, it is not sufficient to see a wide adoption of digital IMS. One of the most important issues to address when managing identity elements is providing

2. Level of Assurance was already a recommendation of the NIST before 2009 [51]

users with privacy. This becomes even more important as the amount of information shared increases. Academic research and industry propositions have slowly assessed and addressed this problem.

The first evolution of IMSs came with U-Port [7]. This proposal is interesting as it is the first IMS proposal that escaped the federated model. The idea behind U-Port is to use blockchains to store and certify identity claims. As we will see later, this way of storing and proving claims threatens user privacy. It is even worse than the federated model. In the federated model, some identity providers have extra information and control over the subject's information. Meanwhile, with U-Port, all the system's actors have information about the issuance of credentials, the information in the credentials, etc. Hence, even though U-Port was an interesting experiment, the idea was intrinsectingly flawed. Due to those restrictions, the U-Port development has stopped.

The actual evolution of IMSs was around the middle of the 2010s with the creation of Sovrin and the formalization of Self-Sovereign Identity by Allen [1]. The latter has already been described in this thesis. It is, without a doubt, the most cited article about PPDIMS. Most works in this area now base their development around the ten laws of identity exposed in the introduction of this thesis. As we saw in this chapter, those ten laws were not "invented" by Allen. They were an (excellent) synthesis of more than 15 years of reflections on the identity problem on the Internet.

Davie et al. proposed one of the most important formalizations of PPDIMSs in 2019 [4]. We already talked about this proposal in the introduction of this thesis. The purpose of Davie et al.'s paper is to formalize the existence of four independent layers that make it possible to implement a DIMS. The first layer is the DID network layer. It is used to share public information about the actors of the scheme, *e.g.*, public keys, revocation lists, authorization lists. As explained in the introduction, the goal of this layer is to provide auxiliary information to the layers on top of it. It makes it possible to implement highly asynchronous peer-to-peer communication methods (layer 2) and to provide efficient authorization and revocation methods for the presentation of credentials (layer 3). The fourth layer is an organizational layer. The idea is that layers 1 to 3 are technical layers. However, identity and, by extension, identity verification is a highly subjective, non-technical concept. Hence, layer 4's goal is to ensure that implementations of layers 1 to 3 actually fulfill their goals. However, the goal of layer 4 is also political. It has to ensure that the IMSs implemented do not harm users by censuring them or by revealing their identity elements in a non-informed way to malicious actors.

This Trust over IP stack is still the paradigm used nowadays. For example, initiatives like Hyperledger separate their work following this formalization (layer 1 is managed by Hyperledger Aries, and layer 3 is managed by AnonCred). In the same way, the recommendations of the W3C also follow this paradigm. They proposed three different standards that focus each on a different layer (layer 1: DID standard [8], layer 2: DIDComm standard [10], layer 3: VC standard [5]).

On the technical side, in the early days of PPfDIMS, one of the most important project was Sovrin [52]. It seems to be the first project to understand that the distribution in a DIMS comes from anonymous credentials and that the distributed ledger should only be used for auxiliary functionalities [12].

The Hyperledger foundation [53] provides tools for blockchain technologies. Some of those tools are focused on DIMS. Among them, we can cite Fabric, a general-purpose distributed ledger, AnonCred, a Verifiable Credential implementation that is compatible with most of the privacy requirements stated by Allen, Aries, which is used to manage the interface between layers 1 and 3 of the ToIp stack. Finally, hyperledger recently promoted a new project, Identitus, whose goal is to integrate the other identity-related projects of Hyperledger to provide a complete toolbox for developers. Hyperledger has been present in the PPDIMS ecosystem from its early days and has, without a doubt, influenced the trajectory of standards and beliefs of the industry toward a more privacy-preserving and distributed paradigm.

In the same way as in the precedent section, this new identity management paradigm is now being assessed by multiple laws and regulations. We can talk about the swiss regulation [54]. Furthermore, a new regulation, eIDAS 2.0 [55], is currently being developed at the European level. The main improvement of this regulation compared to eIDAS 1.0 is to propose to European citizens a digital wallet that will make it possible to store verifiable claims that are presentable online and offline. This tends to suggest that anonymous credentials will be used. Therefore, it is possible that eIDAS 2.0 will lead to systems that fulfill (to some extent) Allen's principles.

Non-governmental efforts have been made to improve the interoperability of DIMS, thus creating a step toward the global adoption of DIMSs. Those efforts have been supported through the W3C organization. Three standards have been proposed. The first standard concerns Distributed IDentifiers (DIDs) [8]. A DID is a unique identifier that links an individual to its VCs. It also enables many distributed and privacy-preserving features, such as multi-device capabilities, credential revocation, actor identification, actor endpoint identification, user unicity, and accountability. To enable those features, each DID is associated with a DID document. The format of a DID is "did:[method]:[identifier]", where "[method]" is the way the DID must be resolved into its DID document. The documentation of a method must also describe how a DID document can be created, destroyed and modified. "[identifier]" is the specific identifier of a DID document in the context of the method. Ideally, each identifier should be unique in the context of a specific method.

The second standard proposed by the W3C is a standard about Verifiable credentials [5]. It enumerates the different requirements of verifiable credentials, the format of the signed messages, and the format of the signatures. This standard also provides several use cases and best practices for security/privacy regarding identity verification.

The third and last standard proposed by the W3C concerns the DIDComm protocol [10]. It

is a two-party communication protocol that makes it possible to route identification messages in a secure and anonymous way using ledgers and DIDs.

To the best of our knowledge, there exists no standardization effort in terms of a distributed ledger. This domain evolves fast as new theoretical research and implementations are proposed monthly. In this thesis, we clarify theoretical requirements regarding distributed ledgers for PPfDIMSs.

3.3 Digital identity and the blockchain scam era

In the previous sections, we discussed papers and industrial projects that proposed new ideas and frameworks for PPfDIMSs. However, this ecosystem is also filled with short-lived speculation projects. This section presents some of those projects and uses them as one of the motivations of this thesis.

After the advent of Bitcoin in 2009 [56], many actors tried to legitimate their blockchains by adding specific properties or goals to their algorithms. Some of those blockchains brought new perspectives, such as the case of Ethereum [57], or other protocols like Zerocash [58], which is an anonymous asset transfer protocol built on top of the Bitcoin blockchain. However, most of them bring little to no advantages compared to other protocols. We can find thousands of cryptocurrencies by looking at any cryptocurrency ranking website (*e.g.*, [59]). Thus, actors used other sources of distinction to become noticeable in this jungle. We can talk about the Non-Fungible Token trend, but to some extent, digital identity also suffered from this gold rush. We can cite projects like Jolocom [60] (ceased activity in 2020), SelfKey [61], which seems to only use their distributed ledger for a cryptocurrency that is used to incentivize actors, EverId [62], which also ceased its activity, and promoted a distributed currency. As we can see, many of those projects were built around a cryptocurrency rather than to provide a PPDIMS. The existence of a cryptocurrency associated with a DIMS project provides an easy way to analyze its utility. If a DIMS project proposes a cryptocurrency along with its core features, it means (with high probability) that it is built to speculate and will not last. To dig deeper in this direction, we could ask ourselves why a DIMS framework would need its own cryptocurrency. First, a cryptocurrency is not a money. Using Marx's [63] definition of money, for an asset to become money, it needs to fulfill three properties: it must be a unit of account, a store of value, and a medium of exchange. To the best of our knowledge, most cryptocurrencies (if not all) do not fulfill any of these properties. Furthermore, according to David Graeber [64], a money only exists and is valuable if an individual can pay taxes with it. Therefore, a cryptocurrency cannot be considered as a money. It has to be considered as an asset. Thus, using cryptocurrency to build a DIMS can only lead to one pitfall: speculation. Moreover, even though expenses will necessarily be associated with the maintenance of a DIMS, the retribution can be done

using sovereign currencies. [3] Thus ensuring stability of prices and ease of access for all actors to liquidity. Therefore, even though this may not be a general rule if a DIMS project creates its own cryptocurrency, it should be considered with maximum precautions.

3.4 The 193 DID methods

Blockchains and consensus-based ledgers provide strong guarantees, at the cost of a high latency, high computation overhead, and high space and message complexity. Hence, this tool should be used only when strictly required. As an answer to this statement, some DIMS frameworks proposed identification methods without requiring consensus algorithms. We study them to see how they balance utility and efficiency.

A good method to have an overview of the DIMS ecosystem, and to understand how quickly it evolved is to look at the W3C's DID specification document [9]. It lists all known DID methods that have been registered to the W3C. Each DID method listed in the W3C's list does not necessarly corespond to one project, but the association is close. This list has 193 entries. The goal of this section is not an exhaustive exploration of all of them. But we will discuss here some interesting ones.

The list of DID methods gives information that is of particular interest: the registry used by each DID method. However, as we will see by analyzing those different methods, not all are used to implement DIMSs. Some only use the possibilities offered by the DID standard to solve other problems, *e.g.*, key sharing. Most of the other methods use blockchain or full-fledged consensus for all their on-ledger operations. On June 4, 2024, we counted 161 out of 193 methods that explicitly require a blockchain or a consensus-based ledger to work. Some of these methods build their own blockchains, and others use existing ones (most of them use Bitcoin, Hyperledger Fabric, or Ethereum). Among the methods that do not explicitly require blockchain, we identified five different categories. [4]

1. **Server-based methods.** The first category groups the four methods that use a central server as the ledger. These methods are comparable to blockchain-based methods as they provide total ordering and strong consistency.
2. **Ledger-free methods.** The second category does not require a ledger at all. This category comprises the methods whose DID only represents the public key of a secret/public

3. By sovereign, we mean a money printed and managed by a state, or a union of states, with a monetary policy.

4. An interesting DID method we do not discuss in this list is the did:snail method [65]. This method is built as a joke. The resolution of a DID document using this method is done through (IRL) mail. The DID is basically the address where the subject lives. Hence, to resolve the document, one has to send a request to this individual using a stamped envelope. The subject of the DID answers the request similarly, posting the DID document in a mailbox. Aside from the joke, this method tackles a concern that we formulate in Chapter 11. Why use digital IMSs in the first place, and do they empower individuals?

key cryptosystem. For example, the did:key method takes the DID itself as a cryptographic public key, *e.g.*, if the did is "did:key:abcdefghij", then the did document is considered to be "abcdefghij", where this string is considered as a public key. Other methods have a similar purpose, *e.g.*, "did:jwk", "did:pkh", "did:self". This category is interesting as the DID document is self-contained in the DID. However, these methods are comparable to a key exchange and do not provide authentication and authorization means.

Several methods are similar to this category but add interesting features. The first one is the "did:peer" method [66]. This method does not require a ledger and has multiple usages or setups. Two setups, method 0 and method 2, are equivalent to did:key. [5] Two other setups, method 1 and method 3, reference the signature (or the hash of a signature for method 3) of a DID document to authenticate it. The final setup, method 4, uses a method 3 DID and associates it with a second DID, which represents an encoding of the whole DID document. Hence, this fourth method provides a self-contained DID. Compared to the other "did:key"-like methods, the "did:peer" method provides more versatility, as it makes it possible to share a DID document. However, this added versatility can be achieved with the "did:key" method by signing a DID document using the key associated with the DID. Hence, it does not provide additional features in terms of authorization or authentication.

The second method that is similar to the "did:key"-like methods is the "did:keri" method, and its KERI protocol [67]. This method resolves a specific type of cryptographic key, Key Event Receipt Infrastructure (KERI). The KERI protocol is used to create and derive public keys. Compared to previous protocols, it is possible to rotate keys deterministically using a specific derivation algorithm. This method was initially proposed to provide a cryptographic scheme that makes it possible to unite the DID methods, thus creating a link between DID methods rather than adding an additional one. However, this is the only extra feature that this method provides. Furthermore, let us assume k_1 is the first key created by an actor, and k_2 is its second key, derived from k_1. Then, when k_2 is used, k_1 must be revoked. Otherwise, impersonation or double-spending-like attacks could be conducted. Furthermore, this revocation requires synchronization between the verification processes (c.f. Chapter 6), which the "did:keri" method does not provide. Hence, this protocol should be backed by a revocation-capable ledger to be fully functional. The protocol is still a good solution to the key loss and key rotation problems. This problem must be assessed in any public-key-based cryptographic scheme, and it is often a bottleneck in many distributed systems where no central authority can help a user recover

5. A DID "did:key:azeaz" can be translated to "did:peer:0azeaz" where 0 is the method identifier, and "azeaz" is the public key.

from a key loss.

3. **DNS-based methods.** The third category contains the methods that resolve DIDs using a DNS-like network. These methods resolve DIDs without using distributed ledgers. The "did:web" method resolves a DID directly using the DNS. The DID is, therefore, a domain name or a web address, that should point to a DID document. Other methods work similarly, the "did:psqr", "did:dns", "did:onion" and "did:webs" methods. These methods are similar to the "did:web" method, but they add additional features. For example, the "did:onion" method resolves the DIDs as an onion address over the Tor network. Another method that must be cited in this category is the "did:gns" method, which is part of the GNU Name System (GNS) [68]. GNS aims to provide a privacy-preserving and distributed DNS-like resolver. To this end, the "did:gns" method is used to identify entities in the network. The resolution is done through a Distributed Hash Table [69]. Hence, this method is not part of a DIMS but of a naming service. Furthermore, their names are not human-choosable (c.f., Chapter 7), thus making it possible to implement it without consensus. Similarly, the "did:ssb" method [70] uses the secure scuttlebutt network, a fault-tolerant, eventually consistent ledger for social media. The "did:ssb" method is used to identify feeds of a given user. The resolution of the DID will, therefore, output the feed, which is considered to be the DID document. We consider it a DNS-like method as it uses an existing resolution method, but it uses the DID as a new type of identifier for the given resource.
4. **IPFS-based methods.** The fifth category comprises Inter Planetary File System-based (IPFS) [71] methods. In the same way as with the previous category, the projects solely based on IPFS (*e.g.*, "did:schema") do not provide auxiliary features. On the other hand, some of these methods (*e.g.*, "did:ipid") use or propose to use a blockchain anchor to provide additional security and usability features.
5. **The Holochain method.** The fourth category comprises only one method, the "did:holo" method, used by the Holochain DIMS [72]. The idea of this category is to provide a complete DIMS without the use of consensus nor strongly consistent ledgers (such as Byzantine Reliable Broadcast (BRB) based ledgers [73]). Even though documentation about this scheme is scarce, we can say that they do not require consensus for their DIMS. However, they do not propose auxiliary features such as credential revocation, account recovery, or multi-device capabilities. They associate each person's identity with a unique device, thus reducing all the synchronization problems we will study in the following of this thesis to a local problem. However, as stated before, this DIMS lacks essential features to be usable. Holochain's documentation assesses these problems. However, as we will see in the following, they may be unable to overcome them without an in-depth modification of their protocol.

Most existing DIMSs that propose auxiliary features (revocation, multi-device capabilities, identification of actors, etc.) use consensus-based blockchains or central servers as their main storage framework. We highlighted some protocols that try to free themselves from blockchain tyranny. However, to the best of our knowledge, all of them solve the problem by removing features from their systems and, thus, by reducing usability. The problem with these initiatives is that they don't base their constructions on a solid theoretical basis. This assessment leads to the following question: Can a privacy-preserving, fully distributed identity management system be implemented without using a consensus algorithm as the main communication primitive while keeping all required auxiliary features? This is one of the two questions answered in this thesis by thoroughly studying the different blocks required to build a DIMS and by providing tools to solve open questions.

3.5 Our contributions: Privacy Preserving and fully-Distributed Identity Management Systems

From the overview we presented, the reader can see that we lack important theoretical basis to build efficient PPfDIMSs. First, most implemented frameworks do not respect privacy, as they use JSON Web Token (JWT) rather than Anonymous Credentials. Furthermore, even implementations that use ACs do not use their more privacy-preserving versions: Hidden Issuer Anonymous Credential. This type of AC, presented in Chapter 5, aims to completely protect user privacy when presenting ACs. Indeed, the implemented frameworks reveal a lot of information about users when they present their identity element. These privacy issues are explored and solved in Chapter 5. On the other hand, and as stated in the previous section, almost all DIMS projects use blockchains. Blockchains, and more precisely, consensus algorithms, have several flaws. They have huge storage overhead (linear with the number of transactions for most of the blockchains), require synchronization between their participants, have high latencies, and usually restrict parallel processing of operations. To the best of our knowledge, this thesis is the first to thoroughly study the need for synchronization in DIMSs and propose techniques to reduce this need to its minimum. Thus increasing efficiency in terms of storage, message complexity, and latency. These works are explored in Chapter 6, Chapter 7, Chapter 8, and Chapter 9. Those different contributions are then summarized in Chapter 10.

Chapter 4

Model and building blocks

This chapter presents the different models considered in the different chapters of this thesis.

This thesis is comprised of two parts, the first part is relative to Anonymous Credentials, and uses a cryptographic approach.

The second part studies the requirements of PPfDIMSs in term of distributed systems along with new techniques to reach those requirements while minimizing synchronization between processes. The study of the requirements of a PPfDIMS (Chapter 6 and Chapter 7) foccusses on synchronization needs between processes of the system. Hence, the main metric we will use is the consensus number [27] of the different objects used to build a PPfDIMS. The consensus number of a distributed object is formally defined in a shared memory model. However, the goal of this thesis is to present a theoretical framework for PPfDIMS in the message passing model. Hence, in this chapter, we present those two different model.

4.1 Distributed-Systems Notions and Definitions

A distributed system is a set of processes that communicate using a medium, either shared memory or message passing. In this thesis, we use two different distributed system models, the shared memory model is used to study the consensus number (*i.e.*, the synchronization requirements) of access control in distributed systems, and the message passing model is used to propose new ways to build efficient distributed ledgers in the context of PPfDIMS.

4.1.1 Shared memory model

Let Π be a set of n asynchronous sequential crash-prone processes $p_1, \cdots, p_n$. Sequential means that each process invokes one operation of its own algorithm at a time. We assume the local processing time to be instantaneous, but the system is asynchronous. This means that non-local operations can take a finite but arbitrarily long time and that the relative speeds between the clocks of the different processes are unknown. Finally, processes are crash-prone: any number of processes can prematurely and definitely halt their executions. A process that crashes is called *faulty*. Otherwise, it is called *correct*. The system is eponymous: a unique positive integer identifies each process, and this identifier is known to all other processes.

Communication Processes communicate via shared objects of type T. Each operation on a shared object is associated with two *events*: an *invocation* and a *response*. An object type T is defined by a tuple (Q, Q_0, O, R, Δ), where Q is a set of states, $Q_0 \subseteq Q$ is the set of initial states, O is the set of operations a process can use to access this object, R is the set of responses to these operations, and $\Delta \subseteq \Pi \times Q \times O \times R \times Q$ is the transition function defining how a process can access and modify an object.

Histories and Linearizability A *history* [74] is a sequence of invocations and responses in the execution of an algorithm.

An invocation with no matching response in a history, H, is called a *pending* invocation. A *sequential history* is one where the first event is an invocation, and each invocation—except possibly the last one—is immediately followed by the associated response. A sub-history is a sub-sequence of events in a history. A process sub-history $H|p_i$ of a history H is a sub-sequence of all the events in H whose associated process is p_i. Given an object x, we can similarly define the object sub-history $H|x$. Two histories H and H' are equivalent if $H|p_i = H'|p_i$, $\forall i \in \{1, \cdots, n\}$.

In Chapters 6 and 7, we define the specification of a shared object, x, as the set of all the allowed sub-histories, $H|x$. We talk about a sequential specification if all the histories in this set are sequential. A *legal history* is a history H in which, for all objects x_i of this history, $H|x_i$ belongs to the specification of x_i. The completion $\bar{H}$ of a history H is obtained by extending all the pending invocations in H with the associated matching responses. A history H induces an irreflexive partial order $<_H$ on operations, *i.e.*, $op_0 <_H op_1$ if the response to the operation op_0 precedes the invocation of operation op_1. A history is sequential if $<_H$ is a total order. The algorithm executed by a correct process is *wait-free* if it always terminates after a finite number of steps. A history H is linearizable if a completion $\bar{H}$ of H is equivalent to some legal sequential history S and $<_H \subseteq <_S$.

Consensus number The consensus number of an object of type T (noted cons(T)) is the largest n such that it is possible to wait-free implement a consensus object from atomic read/write registers and objects of type T in a system of n processes. If an object of type T makes it possible to wait-free implement a consensus object in a system of any number of processes, we say the consensus number of this object is ∞. Herlihy [27] proved the following well-known theorem.

Theorem 4.1. Let X and Y be two atomic objects type such that cons(X) = m and cons(Y) = n, and $m < n$. There is no wait-free implementation of an object of type Y from objects of type X and read/write registers in a system of more than m processes.

In Chapter 6 and Chapter 7, we will determine the consensus number of the distributed objects using Atomic Snapshot objects and consensus objects in a set of k processes. A Single Writer Multi Reader (SWMR) [75] Atomic Snapshot object is an array of fixed size, which

supports two operations: Snapshot and Update. The Snapshot() operation allows a process p_i to read the whole array in one atomic operation. The Update(v, i) operation allows a process p_i to write the value v in the i-th position of the array. Afek et al. showed that a SWMR Snapshot object can be wait-free implemented from read/write registers [75], *i.e.*, this object type has consensus number 1. In this thesis, we assume that all Atomic Snapshot objects used are SWMR. A consensus object provides processes with a single one-shot operation Propose. When a process p_i invokes Propose(v) it proposes v. This invocation returns a *decided* value such that the following three properties are satisfied.

— *Validity.* If a correct process decides value v, then v was proposed by some process;
— *Agreement.* No two correct processes decide differently; and
— *Termination.* Every correct process eventually decides.

A k-consensus object is a consensus object accessed by at most k processes. From the $k+1$th time it is accessed, it returns $\perp$.

4.1.2 Message passing model

The message passing model is the main model considered throughout this thesis. We refine the model for the specific needs of PPfDIMSs. More precisely, we consider two different type of processes. Processes in a set Π who maintain a distributed ledger and which will be refered as nodes, and processes who use the ledger as clients. The second type of processes are either users, issuers or verifiers. Moreover, from section Chapter 9 onward, we consider that a user may own multiple devices. Hence, we do not consider users as single processes.

We consider an asynchronous message-passing system, where processes communicates using authenticated, asynchronous, reliable, point to point channels. "Authenticated" means that a process that receives a message knows its sender, and no malicious adversary can impersonate a correct sender. "Asynchronous" means that each message can be delayed an arbitrary finite long time, and that processes can take an arbitrary but finite long time to process an incoming message. "Reliable" means that no message is dropped, nor modified after it is sent.

We consider a distributed ledger $\mathcal{L}$ that is implemented by a set Π of processes that run a distributed algorithm. Some processes in Π may be Byzantine, *i.e.*, they can arbitrarily deviate from their prescribed algorithm. We consider t to be the maximum number of Byzantine processes in Π, where $t < n$. Generally, we consider $n \geq 3t + 1$. In specific cases, we will adjust this threshold for efficiency gains.

We also consider a set of issuers $\mathcal{I}$ and a set of verifiers $\mathcal{V}$. Each verifier $v \in \mathcal{V}$ is associated to a finite set of trusted issuers $T_v \in \mathcal{I}^{k_v}$, where k_v is an arbitrary integer. Finally, we consider a set of devices $\mathcal{D}$.

We consider that each device $d \in \mathcal{D}$ is associated to a secret key $\mathsf{sk}_d \in \mathcal{SK}_{\mathcal{D}}$, and each issuer $I \in \mathcal{I}$ is associated to a secret key $\mathsf{sk}_I \in \mathcal{SK}_{\mathcal{I}}$. Secret keys are assumed to be known only by the

process it is associated to, and never shared. Each secret key is associated to a public key. Unlike secret keys, public keys can be widely shared. Processes in Π (and their public keys) are known by all the other processes in the system, whereas processes in $\mathcal{I}, \mathcal{V}$ or $\mathcal{D}$ may not be known. For simplicity sake, we assume a public/secret key of any actor can be used for any cryptographic scheme. In reality, each actor should possess a secret/public key pair for each cryptographic protocol.

In this thesis, the word *message* refers to messages sent by the algorithm on the network level to implement an abstraction, they are sent and received. The word *value*, on the other hand, refers to the payloads disseminated at the user level by the abstractions, they are proposed and accepted (or decided in the case of consensus).

Finally, the CAC abstraction (presented in Chapter 8) uses a best-effort (unreliable) broadcast abstraction, noted be_broadcast, as an underlying communication primitive. An invocation of be_broadcast MSG by a correct process p_i sends the same message MSG to all processes in Π. We say that messages are "be-broadcast" and "received".

4.2 Distributed building blocks

In the followings, we will refer to two classical distribute abstractions: the consensus abstraction [22] and the Byzantine Reliable Broadcast abstraction [73]. We present both abstractions in this section.

4.2.1 Consensus

Consensus is a cooperation abstraction that allows a set of processes to agree on one of the values proposed by one of them (or a default value in some specific scenarios). Consensus offers one operation Propose and one callback Decide and is defined by the following four properties.

- *Termination.* If all processes are correct and a process decides a value v, then v was proposed by some process.
- *Agreement.* No two correct processes decide different values.
- *Integrity.* A correct process decides at most one value.
- *Termination.* If a correct process proposes a value v, then all correct processes eventually decide some value (not necessarily v).

Note that the termination property differs slightly from the usual definition of consensus: we do not always guarantee termination, only when some correct process proposes a value.

This problem is well known to be unsolvable in an asynchronous system in the presence of faulty processes [22]. Algorithms that do solve consensus need to circumvent this impossibility, either by assuming some sort of synchronization, or by reducing weakening the termination property, such that the correct processes decide only with high probability.

4.2.2 Byzantine Reliable Broadcast

The Byzantine Reliable Broadcast (BRB) abstraction [73] is a broadcast abstraction. If a correct process broadacst a value, then this value is delivered by all the other correct processes. If a Byzantine process proposes a value, then, either this value is not delivered by the correct processes, or it is delivered by *all* the correct processes. Importantly, processes deliver values in different order. Hence, this abstraction is strictly less powerfull than the consensus abstraction. Formally, the BRB abstraction has one operation Broadcast and one callback Deliver, they fulfill the following properties [24]:

— *Validity.* If a correct process p_i delivers a message from a correct process p_j, then p_j broadcast it;
— *Integrity.* A correct process p_i delivers at most one message from sender p_j.
— *Termination-1.* If a correct process broadcasts a message, it delivers it.
— *Termination-2.* If a correct process delivers a message from a (correct or faulty) process p_j , then all correct processes deliver it.

4.3 Cryptographic Notions and Definitions

This section introduces some notation and the definitions of the main cryptographic building blocks used in this thesis.

— Let $\mathbb{G}$ be a group. We note by $\langle g \rangle = \mathbb{G}$ the fact that g is the generator of $\mathbb{G}$, and we note $1_{\mathbb{G}}$ the neutral element of $\mathbb{G}$.
— Let $\mathbb{Z}$ be a set, we note by $a \leftarrow_{\$} \mathbb{Z}$ the fact that a is chosen uniformly at random in the set $\mathbb{Z}$.

Hash Function The different works of this thesis rely on cryptographic hash functions [76].

Definition 4.1. A hash function is a map H which takes an element of arbitrary size $A \in \mathbb{Z}$ and outputs an element B of fixed size p. A cryptographic hash function has two additional properties:

— *Pre-image resistance.* given $a = \mathsf{H}(b)$, no probabilistic polynomial time (PPT) adversary can find b with non-negligible probability; and
— *Collision resistance.* no PPT adversary can find, with non-negligible probability, $x \in \mathbb{Z}$ and $y \in \mathbb{Z}$ such that $\mathsf{H}(x) = \mathsf{H}(y)$ and $x \neq y$.

Bilinear Pairing This thesis (namely Chapter 5) uses bilinear pairings for some of its develpements.

Definition 4.2. Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, $\mathbb{G}_T = \langle g_T \rangle$, three groups of prime order p. A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that has the bilinearity property. This means that

$e(g_1^a, g_2) = e(g_1, g_2)^a = e(g_1, g_2^a)$, for all $a \in \{0, \cdots, p-1\}$;

To be efficiently used in cryptography we also require this map to be non-degenerate and efficiently computable:

— *Non degeneracy* $\forall A \in \mathbb{G}_1, A \neq 1_{\mathbb{G}_1}, \exists B \in \mathbb{G}_2$ such that $e(A, B) \neq 1_{\mathbb{G}_T}$ and $\forall B \in \mathbb{G}_2, B \neq 1_{\mathbb{G}_2}, \exists A \in \mathbb{G}_1$ such that $e(A, B) \neq 1_{\mathbb{G}_T}$;

— *Efficient computability.* there is an efficient algorithm to compute e.

Furthermore, we can classify bilinear pairings in three types [77]. In our construction, we will use a type-3 bilinear pairing. Type-3 bilinear pairings are pairings where $\mathbb{G}_1 \neq \mathbb{G}_2 \neq \mathbb{G}_T$ and there is no efficiently computable morphism $\psi_1 : \mathbb{G}_2 \to \mathbb{G}_1$ or $\psi_2 : \mathbb{G}_1 \to \mathbb{G}_2$.

Zero-Knowledge Proof To protect the privacy of their data, the actors of an IMS often need to prove knowledge of values, while keeping them secret. A Zero-Knowledge Proof (ZKP) system is a tool that matches this requirement by allowing a prover to prove to a verifier that it knows a secret statement, without revealing it.

A Zero Knowledge Proof (ZKP) system is an interactive protocol —i.e., multiple messages must be exchanged between the prover and the verifier — that allows a prover to prove some boolean statement about a value x to a verifier without leaking any information about x. Formally, let $\mathcal{L}$ be a language in the complexity class $\mathcal{NP}$, and let (x, w) be a tuple. There exist a polynomial time algorithm $\mathcal{A}$ such that if $x \in \mathcal{L}$, then $\mathcal{A}(x, w) = 1$. $\mathcal{A}$ outputs 0 otherwise. We call w the witness. A ZKP system is a system of five algorithms that allows a prover to provide a proof π. π proves that a value x fulfills the relation $x \in \mathcal{L}$, where $\mathcal{L}$ is a language of the complexity class $\mathcal{NP}$. Furthermore, π is indistinguishable from another valid proof π^* built using an unbounded algorithm $\mathcal{S}$ that does not takes x as input. In other words, the verifier can verify that the Prover knows a value in $\mathcal{L}$, but it cannot learn this value. We note $\mathcal{ZKP}_{\mathcal{L}}$ the set of possible ZKP which refers to the language $\mathcal{L}$.

In this thesis, we will use the convenient notation of ZKP protocols introduced by Camenisch and Stadler [78]. The aim of the notation is to explain what is proven, and what is hidden, in a compact way. For example, the notation for a ZKP about the knowledge of the exponent α in the expression $y = g^\alpha$ is

$$ZKPoK\{(\alpha) : y = g^\alpha\} \tag{4.1}$$

Where the elements between parentheses (α) are the elements known only by the prover, and the equations on the right explain how the prover commits to those elements. The letters on the left define what proof is being conducted. Here, ZKPoK means Zero Knowledge Proof of Knowledge, which explains that the prover will prove knowledge of an element. We will also use ZKPoE, which is a Zero Knowledge Proof of Equality. The goal of a ZKPoE is to prove that different elements have the same exponent. Chaum and Pedersen presented a version with only two elements in 1993 [79], but this construction can be generalized to any number of elements.

We will use the prefix "NI-" to refer to non-interactive proofs. An instantiation of NI-ZKPoK in the random oracle model was described by Fiat and Shamir [80], and later modified and proved secure by Bernhard, Pereira, and Warinschi [81].

Signature Scheme A digital signature [76] can be seen as a proof that a given message m was certified by an issuer. Here, we give a formal definition for a digital signature scheme:

Definition 4.3. A digital signature scheme Σ is defined as a tuple of four algorithms (**Setup, KeyGen, Sign, Verify**).

1. $(pp) \leftarrow$ **Setup**(λ) : On input of a security parameter λ, the algorithm outputs pp, a description of public parameters whose security level depends on λ;
2. $(\mathsf{sk}, \mathsf{pk}) \leftarrow$ **KeyGen**(pp): on input of a setup parameter pp, the algorithm computes a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$;
3. $(\sigma) \leftarrow$ **Sign**(pp, m, sk): on input of a setup parameter pp, a message m, and a secret key sk, the algorithm outputs a signature σ; and
4. $\{0, 1\} \leftarrow$ **Verify**$(pp, m, \mathsf{pk}, \sigma)$: on input of a setup parameter pp, a message m, a public key pk, and a signature σ, the algorithm outputs 1 if σ is a valid signature on the message m by the secret key associated with pk, and 0 otherwise.

Security Notion: A signature is secure if no adversary can create a signature in the name of some other issuer, and if no adversary can modify the previously signed message without invalidating the signature. The standard security notion for a signature scheme is the Existential Unforgeability under Chosen Message Attack (EUF-CMA) [82]. This notion states that the scheme is resistant against an adversary that can make an arbitrary number of queries to a signing oracle.

Blind signatures A *blind signature* [83] is a digital signature that make it possible to for the recipient of the signature to hide to the issuer the message m it signs using a commitment. The issuer signs the commitment and, when the recipient received the signature, it can open the commitment, thus ending up with a signature on the message m that can be verified with the classical Verify algorithm. A blind signature scheme is a digital signature scheme with three additional operations, $\mathsf{CommitMessage}, \mathsf{SignCommitment}$ and $\mathsf{UncommitSig}$ which are defined as:

— $(C) \leftarrow \mathsf{CommitMessage}(pp, m)$: on input of public parameters pp, and a message m, the algorithm, executed by the recipient of a signature, outputs C a commitment to the message m.

— $\sigma \leftarrow \mathsf{SignCommitment}(pp, C, \mathsf{sk})$: on input of public parameters pp, a commitment to a message C, and a secret key sk, the algorithm outputs a signature σ;

— $\sigma' \leftarrow \mathsf{UncommitSig}(pp, \sigma, m, C, \mathsf{pk})$: on input of public parameters pp, a message m, a commitment C to m, a public key pk and a signature on C σ, the algorithm executed by the recipient of a signature, outputs σ', a signature on the message m.

A signature on a commitment is equivalent to a signature on the actual message, *i.e.*,

$$\mathsf{Verify}(pp, m, \mathsf{pk}, \mathsf{UncommitSig}(pp, \mathsf{SignCommitment}(pp, \mathsf{CommitMessage}(pp, m, \mathsf{pk}), \mathsf{sk}), m, C, \mathsf{pk})) = \mathsf{Verify}(pp, m, \mathsf{pk}, \mathsf{Sign}(pp, m, \mathsf{sk}), m, C, \mathsf{pk})).$$

Unlinkable signatures Informally, the unlinkability property states that, when a user presents multiple randomized credentials to a verifier, the latter cannot tell whether they come from the same original signature or from two different signatures. Signature schemes that provide unlinkability are equipped with one additional algorithm $\mathsf{Randomize}$.

— $\sigma' \leftarrow \mathsf{Randomize}(pp, m, \mathsf{sk}, \sigma)$: on input of a setup parameter pp, a message m, a secret key sk, and a signature σ, the algorithm outputs a randomized signature σ'.

A randomized signature can be verified using the same verifying algorithm:

$$\mathsf{Verify}(pp, m, \mathsf{pk}, \mathsf{Randomize}(\sigma)) = \mathsf{Verify}(pp, m, \mathsf{pk}, \sigma).$$

The unlinkability property can be stated as follows:

Unlinkability: Let b be a random bit, *i.e.*, $b \leftarrow_{\$} \{0, 1\}$. Let $pp = \mathsf{Setup}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(pp)$. We choose a message m and compute the signature $\sigma = \mathsf{Sign}(pp, m, \mathsf{sk})$. Furthermore, σ' is $\mathsf{Sign}(pp, m, \mathsf{sk})$ if $b = 0$ and $\mathsf{Randomize}(pp, m, \mathsf{sk}, \sigma)$ if $b = 1$. A signature scheme is unlinkable if, for any PPT adversary $\mathcal{A}$ which knows $m, pp, \mathsf{pk}, \sigma$ and σ', the probability for $\mathcal{A}$ to output $b' = b$ is lesser than $\frac{1}{2} + \epsilon$, where ϵ is negligible.

Signature on vector of messages and zero knowledge proof of knowledge of signature It is often required to prove statement about signed messages, without revealing them. A zero knowledge proof of signature makes it possible to prove that the recipient of a signature knows a message m in a signature, without revealing the message. It can then compose this proof of knowledge with another statement. For example, it can prove that it knows the message m of a signature, and the value of m is greater than another value. We note a ZKP of signature on a message m as follows (using our simplified Camenisch Stadler notation). In this notation, we also prove that the message m respect some condition represented by the function f.

$$NI - ZKP\{(m) : \mathsf{Verify}(pp, m, \sigma, \mathsf{pk}) = 1 \wedge f(m) = 1\}.$$

Furthermore, it is also interesting to prove that multiple messages are signed by the same entity, and that those messages are linked. To do so, it is often required to have signatures on vector of messages. Hence, the recipient of a signature can prove that it knows a signature on a message m_1 and a message m_2, and that the signatures on those messages were produced at the same time.

Anonymous Credentials An anonymous credential (AC) scheme [84, 85, 86, 6] is a cryptographic signature scheme with three additional properties. Namely, unlinkability, zero knowledge proof of signature and signature on commitments (or blind signatures). Furthermore, most AC schemes makes it possible to sign vector of messages.

Issuer-indistinguishable threshold anonymous credential A specific type of anonymous credential scheme is the threshold anonymous credential scheme. It combines properties of threshold signatures [87] with properties of anonymous credentials. This scheme works as follows. Signers produce fragments of signature and send them to a recipient. The recipient wait until it receives ℓ distinct fragments $\sigma_1, \ldots, \sigma_\ell$ and aggregates them into one signature σ. This aggregated signature proves that the recipient knows ℓ signatures on a message m—or on a vector of messages M—from ℓ different issuers out of n authorized issuers. A verifier can verify that, indeed, σ is an aggregation of ℓ signatures from *distinct* issuers. However, the verifier cannot learn the identity of the issuers of the signatures that constitute σ, *i.e.*, it cannot know if the signature of a specific issuer was used to produce σ. In addition, a threshold anonymous credential scheme incorporates all the properties of an anonymous credential scheme, *i.e.*, unlinkability, zero knowledge proof of signature and blind signatures.

To achieve this goal, one must add two operations to a classical Anonymous Credential scheme, the AggKey and the $\mathsf{AggCred}$ operations. The AggKey is used to build a group key that will be used to verify the aggregated signature, and the $\mathsf{AggCred}$ operation is used to aggregate the credentials.

We say that a signature scheme is a (t, n)-threshold anonymous credential scheme if it fulfils the Anonymous Credential properties and two additional properties:

— (t, n)**-aggregation.** Let $AggKey = \mathsf{AggKey}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ be the aggregation of n keys, and let $Agg\text{-}\sigma$ be a signature. If $\mathsf{Verify}(pp, m, AggKey, Agg\text{-}\sigma) = 1$, then with high probability, $Agg\text{-}\sigma = \mathsf{AggCred}(\sigma_1, \ldots, \sigma_l)$, where, $\forall\ \sigma_i, \sigma_j \in \{\sigma_1, \ldots, \sigma_l\}, \exists \mathsf{pk}_a, \mathsf{pk}_b \in \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, $\mathsf{Verify}(pp, m, \sigma_i, \mathsf{pk}_a) = 1$ and $\mathsf{Verify}(pp, m, \sigma_j, \mathsf{pk}_b) = 1$ and $pk_a \neq pk_b$.

— **Issuer-indistinguishability.** Let an adversary $\mathcal{A}$ who is given $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$ a set of secret keys, $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ their associated public keys, a set of l signatures $\{\sigma_1, \ldots, \sigma_l\}$ such that $\forall i \in \{1, \ldots, l\}, \exists \mathsf{pk}_a \in \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}, \mathsf{Verify}(pp, m, \sigma_i, \mathsf{pk}_a) = 1$, an aggregated key $AggKey = \mathsf{AggKey}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and an aggregated signature $\mathsf{AggCred}$ such that

$\mathsf{Verify}(pp, m, AggKey, Agg\text{-}\sigma)$. The probability for $\mathcal{A}$ to determine if $\mathsf{AggCred}$ is the aggregation of $\{\sigma_1, \ldots, \sigma_l\}$ is negligible.

There exists one scheme that achieve those additionnal properties, Coconut [20]. To achieve this goal, Coconut combines the Pointcheval-Sanders signature scheme [84] with techniques inherited from the Shamir Secret Sharing [88].

Assumptions The signature scheme presented in Chapter 5 relies on several computationnal complexity assumptions. The first is the Discrete Logarithm (DL) assumption [89]. The Dl assumption is defined as:

Definition 4.4. Discrete Logarithm assumption Let $\mathbb{G} = \langle g \rangle$ be group of prime order n, let $a \in \mathbb{Z}_n$ be a scalar and $A = g^a$. Let $\mathcal{A}$ be a PPT adversary. The DL assumption holds if the probability for $\mathcal{A}$ to compute a given A and g is lesser or equal to ϵ, where epsilon is negligible.

The second assumption is the Computational Diffie Hellman Assumption (CDH) [89], whose definition is:

Definition 4.5. Computationnal Diffie Hellman assumption Let $\mathbb{G} = \langle g \rangle$ be group of prime order n, let $a, b \in \mathbb{Z}_n$ be two scalars and $A = g^a, B = g^b$. Let $\mathcal{A}$ be a PPT adversary. The CDH assumption holds if the probability for $\mathcal{A}$ to compute $C = g^{ab}$ given A, B and g is lesser or equal to ϵ, where epsilon is negligible.

The third assumption is the Decisional Diffie Hellman (DDH) assumption [89] defined as:

Definition 4.6. Decisional Diffie Hellman assumption Let $\mathbb{G} = \langle g \rangle$ be group of prime order n, let $a, b \in \mathbb{Z}_n$ be two scalars and $A = g^a, B = g^b$. Let $\mathcal{A}$ be a PPT adversary. Let $s \leftarrow_\$ \{0, 1\}$, if $s = 0$, then $C = g^{ab}$ and if $s = 1$ then $C \leftarrow_\$ \mathbb{G}$. The DDH assumption holds if the probability for $\mathcal{A}$ to output s' such that $s' = s$ given A, B, C and g is lesser or equal to $\frac{1}{2} + \epsilon$, where epsilon is negligible.

We also assume that there exists a type-3 bilinear pairing, e, defined on elliptic curves over a finite field and a cryptographic hash function, H.

4.4 Notations

We denote by $\langle v_1, ..., v_k \rangle$ the k-tuple containing the sequence of k values v_1 to v_k. The $\star$ symbol is used as the wildcard symbol (any value can be matched).

Chapter 5

A privacy preserving Anonymous Credential scheme for DIMS: Hidden Issuer Anonymous Credential

This chapter introduces the notion of hidden issuer anonymous credentials. It formally defines the problem of an Anonymous Credential scheme that respects the issuer-indistinguishabilty and the trusted issuer properties. Then, it proposes an implementation of the scheme based on a new cryptographic primitive: the Aggregator. Finally, a comparison with the state-of-the-art is conducted. It was written in collaboration with Daniel Bosk, Davide Frey and Guillaume Piolle. It was published in the Privacy Enhancing Technologies conference in 2022 [15].

5.1 introduction

Self Sovereign Identity (SSI) aims to provide users with a privacy-preserving manner to prove their identities.

In this chapter, we focus on two of the ten principles defined by Allen [1]: consent and minimization. Consent states that users must remain in control of their identities by agreeing to their usage by third parties. Minimization states that the system must only disclose minimal information about a user's identity, effectively protecting their privacy. SSI frameworks allow for versatile, user-centric, privacy-preserving IMSs. Many SSI implementations have appeared in recent years, from EL PASSO [90] to the more decentralized Sovrin implementation [52, 91]. Most of them protect privacy by relying on Anonymous Credential (AC).

An AC scheme, introduced by Chaum [6] in 1985, allows a user to prove his/her identity elements without disclosing any information other than the one he/she intends to disclose. It relies on the presence of three groups of actors: users, identity providers (or issuers), and service providers (or verifiers) as defined in Chapter 1. In this section, we use the following running example to explain our contributions: a user wants to access a bar. However, they need to prove

that they have received their Covid-19 vaccines and that they are of legal age to buy alcohol. If the bar organizes a concert, the user will also have to present a ticket.

The separation between issuers and verifiers is a key factor for privacy preservation in AC schemes. Establishing a proof of identity most likely requires access to numerous information items. For example, a user that wants to prove to be over 18 years old will typically provide an ID card with a picture, which is then verified in person or by matching the ID to a video or a photograph. An AC scheme allows the user to disclose all extra information (picture, date of birth, name or all other elements on the id card) only to the issuer. The issuer then provides the user with a certification of an identity attribute [92] (being over 18). This takes the form of a digital signature that the user can employ as a credential with the verifier in order to access its services.

However, the separation between the issuer and the verifier is not complete. A verifier can only accept a credential if it trusts the corresponding issuer, and by using a key produced by the issuer itself. This raises important privacy concerns. In the previous Covid-19 example, a verifier may exploit the identity of the issuer (a vaccination center in this case) to infer where the user lives. Similarly, in the case of a certificate stating that a user is 18 years old, knowledge of the issuer may help the verifier infer where the user was born. These pieces of information constitute partial identifiers, which, when combined, may lead to the full identification of a user. This situation can be even worse if the verifier and the issuer collude. Let us assume that a covid-19 vaccination certificate states the place and date of vaccination. If the user is the only person vaccinated at that time on that date, verifier and issuer together can easily identify the user. These privacy concerns can undermine the very objective of SSIs, violating minimization and consent.

When we wrote this paper before its publication at PETs, none of the existing existing implementations [52, 90, 93, 60], nor the AC instantiations [94, 95, 84, 34, 85] offered protection against these vulnerabilities in the absence of a trusted setup. A paper [26] that appeared concurrently with the preparation of our own proposes a scheme addressing the same goal in the presence of a trusted setup. We discuss this scheme and provide a comparison in Sections 5.9 and 5.3.

In this chapter, we take a different approach. We formalize the key features of hidden-issuer anonymous credentials into a novel cryptographic primitive and instantiate it into a solution that does not require a trusted setup. In doing so, we make four main contributions.

— We introduce the cryptographic *aggregator*, a new primitive that makes it possible to prove the set-membership of an element, without revealing it, and while only knowing a commitment to it. We formally define this new primitive and give a concrete instantiation that works in the absence of a trusted setup.

— We propose a novel credential scheme that provides issuer indistinguishability from the

verifier's point of view with an instantiation that does not require a trusted setup. Our scheme uses aggregators to hide the issuer of a given credential among a set of issuers that are trusted by the verifier. This allows the verifier to trust the information in the credential even if it does not know its precise issuer. This new scheme ensures that verifiers cannot use a credential to infer personal information that it does not already disclose. Our scheme supports non-transferable signatures and signatures-on-committed-message.

— We prove the security of our scheme. Specifically, we prove its soundness under the Computational Diffie-Hellman assumption, in the Random Oracle model. We also prove the indistinguishability of the issuer among the whole set of issuers trusted by one verifier under the Decisional Diffie-Hellman assumption.

— We provide an open-source implementation of our scheme in Rust and evaluate its performance.

5.2 Problem Statement

We aim to provide an AC that hides not only the identity of the user, but also that of the issuer, thereby reducing the associated privacy risks. In the rest of this section, we define the properties we seek for our Hidden-Issuer Anonymous-Credential scheme, before describing our adversary model.

Properties We focus on two of the principles enunciated by Allen [1]: minimization and consent. Existing credential schemes satisfy these principles only partially because, as we discussed, knowledge of the issuer can reveal personal information about the user. Specifically, they provide minimization and consent (i) by sharing only the identity elements that were selected by the user, (ii) by providing unlinkability, *i.e.*, by ensuring that multiple uses of the same credential cannot be linked together, and (iii) by providing a way for the issuer to sign committed message, *i.e.*, the issuer is able to sign a message without learning the actual content of this message.

In order to hide the identity of the issuer from the verifier, we introduce two additional properties:

— **Issuer indistinguishability** states that, given two credentials certifying the same types of messages, a verifier should not be able to say if the issuers of these credential are distinct or not.

— **Trusted issuer** states that the verifier should be certain that the credential it accepts has been issued by an issuer it trusts. This would be trivial in a classic Anonymous Credential scheme, but becomes relevant once we hide the identity of the issuer.

The credential should satisfy these properties while being used an arbitrary number of times (multi-show) and with an arbitrary number of different verifiers (multi-verifier). It should also

have an optional non-transferable-credential feature. To ensure usability, it should be efficiently computable with low-end modern computers for its three types of actors. Finally, the scheme should not require a trusted setup as this would either introduce a trusted third party or require complex distributed synchronization protocols [96, 97].

Need for a Decentralized Solution At first glance, it might appear that the above properties could be provided by a central anonymizer—possibly chosen by a decentralized leader-election algorithm—to which issuers would delegate the signature of all credentials. However, this solution would defeat the very point of a decentralized IMS, adding the (strong) hypothesis that all issuers and verifiers trust the same anonymizer, creating a single point of failure. A verifier would not be able to choose specific trusted issuers and would need to trust *all* the issuers the anonymizer represents. A centralized anonymizer would also complicate the issuance workflow. Let us consider a bar that verifies tickets for an event it organizes. The bar can trust multiple ticket resellers but it can also sell tickets directly to its clients. With a central anonymizer, the bar would thus need to register to the anonymizer responsible for ticket sales. This would create unnecessary overhead, particularly if the bar sells tickets only occasionally. Moreover, if the central anonymizer is responsible for a specific population, *e.g.*, a region or a country, its usage would still leak information about this region. A decentralized IMS using AC should instead (i) give verifiers the freedom to trust their preferred issuers, and (ii) give users the freedom to choose verifiers based on the issuers they trust.

Adversary Model We consider two adversaries: 1) A malicious verifier colludes with a subset or all of its trusted issuers. It has access to their secret keys, and to all previously issued credentials. It tries to reveal the identity of the owner or of the issuer of a credential. 2) A malicious user tries to use credentials on messages not signed by any trusted issuers. He can request multiple signatures from the signing oracles of each trusted issuer, and obtain all their public keys.

Colluding parties can share any information with each other. We consider the security of our system in an isolated world. The only potential flaws considered here are the ones within the scheme. We provide a formal adversary model in Section 5.6.1.

5.3 Related Work

Anonymous Credentials (AC) have been a well studied topic since they were introduced by Chaum [6] in 1985. This first work defines them as signatures certifying that an element was issued by a given entity, while ensuring unlinkability between each use of this signature by a user. The first efficient implementation is due to Camenisch and Lysyanskaya in 2001 [34]. This

work offers a cryptographic basis to build an efficiently computable signature, with the properties listed by Chaum. It also adds properties, in particular non transferability. Several papers enhanced this first implementation of Chaum's principles with more efficient schemes, including one from the same authors, in 2002 [98]. The first two ACs we cited above are based on RSA groups. Later evolution made it possible to work on elliptic-curve-based groups with bilinear pairings. These include the 2004 Camenisch and Lysyanskaya article [94], and the 2016 Pointcheval Sanders (PS) Short Randomizable Signature [84]. The latter raised our interest because, it contains few elements, and it does not contain trap-doors. Even though the AC schemes presented here require a third-party issuer, it is important to notice that, in some particular cases, such issuers are not needed. Decentralized anonymous credentials [99] propose an efficient scheme for cases such as the mitigation of sybil attacks or Direct Anonymous Attestation.

Other interesting signatures have been developed in the meantime, in particular, ring signatures [28], and ad hoc group signatures [100]. These signatures do not disclose the identity of the issuer of a credential. This property comes from the fact that the issuer is able to hide among a group of other potential issuers. To create such a signature, the issuer chooses a set of issuers. It signs a message using these issuers' public keys and its own secret key. It is then computationally impossible to find the original issuer. However, only the issuer can create such a ring, which makes ring signatures impractical for the user, who wants to be able to use their credential with numerous different verifiers, who do not trust the same set of issuers. An answer to this problem can be found in oblivious signatures [101], and more precisely in the Universal ring Signature (US) scheme [102]. The latter makes it possible for a user to recreate a ring signature from a simple signature, without knowledge of any secret key. Although this signature seems to meet our issuer indistinguishability requirement, it presents major drawbacks with respect to HIAC. First, it does not provide unlinkability: the signature in the US scheme is given to the verifier without randomization. Secondly, the US scheme does not offer collusion resistance: a verifier colluding with the issuer of a given credential can directly find the real issuer inside the produced ring, by making a comparison between the elements in the ring and the previously signed elements. Finally, the US scheme always exhibits linear complexity both in verification and in the ring-generation process that effectively hides the identity of the issuer. On the other hand, HIAC always runs verification in constant time, and only exhibits linear-time complexity for hiding the public key of the issuer when a user interacts with a given verifier for the first time.

During the writing of this paper, a new AC scheme was proposed by Bobolz et al. [26]. Their paper also offers an AC scheme with issuer indistinguishability. As shown in Table 5.10, their scheme provides anonymity and credential revocation by means of a revocation authority. However, it uses an aggregator-like construct that is less efficient than our own (**VerifierSetup** and **IntegrityVerification** lines of Table 5.4) and it employs Groth's signature scheme [103],

which requires a trusted setup—the value Y in the signature.

This construction has one advantage over the implementation proposed in this chapter, it provides efficient Zero Knowledge Proof of Knowledge of signatures. Because both our solution and theirs use bilinear pairings, and because our aggregator is more efficient, but their signature offers more properties, it seems interesting to explore the perspective of a HIAC scheme built using their signature scheme and our aggregator.

Our new aggregator primitive makes it possible to prove the set-membership of a value, without revealing it, and while knowing only a commitment to it. The state of the art in the domain of set-membership uses two different approaches. The first one is based on cryptographic accumulators [104, 105, 98, 106, 107]. These objects enable the aggregation of multiple elements in one object and the proof that a given element was indeed accumulated. The second way is more direct, without accumulators [108]. However, all these approaches require the prover to know the value it proves set-membership of. Our aggregator removes this need: the prover only needs to know a commitment to the value.

5.4 Overview

We propose a novel Anonymous Credential scheme that does not disclose the identity of a credential's issuer. Our scheme introduces a randomization process, run by the user, that hides the issuer of a credential among the set of issuers trusted by the verifier. To make this possible, each verifier publishes the list of its trusted issuers in the form of a set, represented by an *aggregator*, a new primitive, central to an HIAC scheme. The aggregator allows a user to prove that a randomized issuer's public key belongs to a given set, while not knowing the associated secret key, and while being able to use this key in a subsequent signature-verification process. This aggregator is independent from the rest of the signature scheme. The only interface between the aggregator and the signature is the randomized issuer's public key.

When the user wants to use a credential received from an issuer, they randomize the issuer's public key, while providing a proof that the issuer's key belongs to the verifier's aggregator. This randomize-and-prove process hides the identity of the issuer, while allowing the verifier to ensure that it belongs to its trusted list.

Our Hidden Issuer Anonymous Credential (HIAC) scheme combines a modified Pointcheval Sanders (PS) signature [84], and two Aggregator instances, one for each secret key used by PS. We use PS because it does not require a trusted setup, and it is trapdoor-free.

Notation	Correspondence	Set
$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$	Three groups of prime order p linked by a bilinear pairing e, such that $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$	
g_1, g_2, g_T	Generator of the groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ respectively	
p	Prime order of the groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$	$\mathbb{P}$
e	Bilinear pairing	$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
H	Collision-free one-way hash function	$\mathsf{H} : \mathbb{Z} \to \{0, \cdots, k\}$

Table 5.1 – General notations

5.5 Notations

We give a summary of the notations used in the aggregator scheme, and the hIAC scheme. A general notation table is given in Table 5.1, a summary of the notations used for the aggregator is given in Table 5.2, and a summary of the notations used in the HIAC scheme is given in Table 5.3.

Notation	Correspondence	Value	Set
x_i	Secret values aggregated in the aggregator		$\mathbb{Z}_p^*$
$\mathcal{S}$	Set of values aggregated in the aggregator	$\{g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}}\}$	$(\mathbb{Z}_p^*)^k$
C	Commitment to one element $s \in \mathcal{S}$	$g_2^{x_s}$	$\mathbb{G}_2$
sk	Secret set-membership verifcation key		$\mathbb{Z}_p^*$
W	Aggregator set-membership set of witnesses	$\{g_1^{\mathsf{sk}\frac{1}{x_j}}\}_{j=1}^k$	$\mathbb{G}_1$
$(W)_{(j)}$	j-th element of W	$g_1^{\mathsf{sk}\frac{1}{x_j}}$	$\mathbb{G}_1$
β	Aggregator integrity verification proof	NI-ZKPoE$\{(\mathsf{sk}) : \bigwedge_{j=1}^k (W)_{(j)} = (W_p)_{(j)}^{\mathsf{sk}}\}$	
r_1, r_2	Two randomizing elements used by the user to hide the values he commits to		$\mathbb{Z}_p^*$
C'	Randomized commitment to one element $s \in \mathcal{S}$	$g_2^{x_s r_1}$	$\mathbb{G}_2$
π_s	Proof that C' belongs to the set $\mathcal{S}_{\mathsf{comm}}$	$((W)'_{(l)} = (W)_{(l)}^{r_2}, h = g_1^{r_1 r_2})$	$(\mathbb{G}_1, \mathbb{G}_2)$

Table 5.2 – Aggregator notations

5.6 Formal Definitions

5.6.1 Hidden Issuer Anonymous Credential

A Hidden Issuer Anonymous Credential (HIAC) is an Anonymous Credential with the issuer-indistinguishability and trusted-issuer properties.

Definition 5.1. A Hidden Issuer Anonymous Credential scheme Σ is defined as a tuple of six algorithms (**Setup, IssuerKeygen, Sign, VerifierSetup, Randomize, VerifyRandomized**) defined as follows:

1. $(pp) \leftarrow$ **Setup**(λ): On input of a security parameter λ, the algorithm outputs pp, a description of public parameters whose security level depend on λ;

Notation	Correspondence	Value	Set
isk_i	i-th issuer secret key	(x_i, y_i)	$(\mathbb{Z}_p^*)^2$
ipk_i	i-th issuer public key	$(X^{(i)} = g_2^{x_i}, \bar{X_1^{(i)}} = g_1^{\frac{1}{x_i}}, Y_1^{(i)} = g_1^{y_i}, \bar{Y_1^{(i)}} = g_1^{\frac{1}{y_i}}, Y_2^{(i)} = g_2^{y_i})$	$(\mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2)$
m	Message certified by a credential		$\mathbb{Z}_p^*$
R_y	User's random secret element used to enhance issuer indistinguishability property of a credential. Used with only one credential.		$\mathbb{Z}_p^*$
R	User's secret random element used to hide R_y in the issuance protocol		$\mathbb{Z}_p^*$
u_1	User commitment to the i-th issuer key $Y_1^{(i)}$	$(Y_1^{(i)})^{R_y} g_1^R$	$\mathbb{G}_1$
u_2	User commitment to the i-th issuer key $Y_1^{(i)}$	$(Y_2^{(i)})^{R_y}$	$\mathbb{G}_2$
ϕ	Proof of knowledge of R_y and R in u_1	NI-ZKPoK$\{(R_y, R) : u_1 = (Y_1^{(i)})^{R_y} g_1^R\}$	
$(r_{(I,1)}, r_{(I,2)})$	I-th issuer's random secret elements used to enhance the security of the **VerifyRandomized** algorithm. Used only once.		$(\mathbb{Z}_p^*)^2$
σ^Δ	Outputted value by the issuer I after running the **Sign** algorithm. Depends on R.	$(h_1 = g_1^{r_{(I,1)}}, h_2 = g_1^{r_{(I,2)}}, \sigma_1^\Delta = g_1^{r_{(I,1)}(x_I + \mathsf{H}(m)(y_I + R))}, \sigma_2^\Delta = g_1^{r_{(I,2)}(x_I + \mathsf{H}(\mathsf{H}(m))(y_I + R))})$	$(\mathbb{G}_1)^4$
σ	User final credential. Does not depend on R.	$(h_1 = g_1^{r_{(I,1)}}, h_2 = g_1^{r_{(I,2)}}, \sigma_1 = g_1^{r_{(I,1)}(x_I + \mathsf{H}(m) y_I)}, \sigma_2^\Delta = g_1^{r_{(I,2)}(x_I + \mathsf{H}(\mathsf{H}(m)) y_I)})$	$(\mathbb{G}_1)^4$
Agg_x	Verifier's aggregator referring to the x values of the issuer's secret keys		
Agg_y	Verifier's aggregator referring to the y values of the issuer's secret keys		
aux_x	Verifier's aggregator's auxiliary values associated to Agg_x		
aux_y	Verifier's aggregator's auxiliary values associated to Agg_y		

Table 5.3 – Hidden issuer Anonymous Credential scheme notations

2. $(isk, ipk) \leftarrow$ **IssuerKeyGen**(pp): on input of pp, an issuer computes a secret/public key pair (isk, ipk);
3. $(\sigma) \leftarrow$ **Sign**(pp, m, isk): on input of pp, of a message m, and of its issuer secret key isk, an issuer outputs a signature σ;
4. $(\mathcal{S}, \mathsf{aux}_\mathcal{S}) \leftarrow$ **VerifierSetup**$(pp, \mathcal{S}')$: on input of pp, and of a set of k' issuers $\mathcal{S}'$, a verifier outputs $\mathcal{S}$ a selection of $k \leq k'$ trusted issuers in $\mathcal{S}'$. The algorithm also outputs auxiliary information $\mathsf{aux}_\mathcal{S} = (vpk, vsk)$, about $\mathcal{S}$; where vpk is public and vsk is secret.
5. $(ipk', \pi_{ipk'}, \sigma') \leftarrow$ **Randomize**$(pp,\ ipk,\ vpk,\ \mathcal{S},\ \sigma)$: on input of pp, of a signature σ, of a public key ipk of the issuer of σ, of a set of issuers $\mathcal{S}$, and of public auxiliary information vpk, a user:
   (a) Checks that vpk is built using elements from $\mathcal{S}$;

(b) Produces ipk', a randomized version of ipk;

(c) Produces a proof $\pi_{ipk'}$ that ipk' is an element of $\mathcal{S}$, using vpk;

(d) Produces a randomized signature σ' using σ, ipk', and $\pi_{ipk'}$.

The algorithm outputs ipk', $\pi_{ipk'}$, and σ'.

6. $\{0,1\} \leftarrow$ **VerifyRandomized**(pp, vsk, ipk', $\pi_{ipk'}$, σ', m): on input of pp, of verifier secret auxiliary information vsk, of a randomized issuer key ipk', of a proof $\pi_{ipk'}$, of a randomized signature σ', and of a message m, a verifier outputs 1 if the proof $\pi_{ipk'}$ is valid, and if σ' is a valid signature on message m, signed with the secret key associated with ipk'. The algorithm outputs 0 otherwise.

Security notions: We define three security notions: *correctness*, *issuer indistinguishability*, and *trusted issuer*. *Issuer indistinguishability* extends the unlinkability property of classical Anonymous Credential schemes.

Definition 5.2. (Correctness) A HIAC scheme is correct, if for any honestly built tuple $(pp, vsk, ipk', \pi_{ipk'},$
$\sigma', m)$, **VerifyRandomized**$(pp, vsk, ipk', \pi_{ipk'}, \sigma', m)$ always outputs 1.

Definition 5.3. (Issuer Indistinguishability)

We define the **IssuerIndistinguishGame**$(\mathcal{A}, \mathcal{C})$, for a challenger $\mathcal{C}$ that represents a user, and the actions preformed by the issuers, and an adversary $\mathcal{A}$ that represents a malicious verifier colluding with *all* its trusted issuers [1]:

— **Setup**: First, $\mathcal{C}$ runs the HIAC **Setup** algorithm, and two rounds of the **Keygen** algorithm to obtain two issuer key pairs $(\{ipk_1, ipk_2\}, \{isk_1, isk_2\})$. Second, $\mathcal{A}$ runs **VerifierSetup** to build a set $\mathcal{S}$ with the two issuers, and the associated auxiliary information (vpk, vsk). Third, $\mathcal{C}$ computes $\sigma_1 =$ **Sign**(pp, isk_1, m) and $\sigma_2 =$ **Sign**(pp, isk_2, m) on the same message $m \leftarrow_{\$} \mathbb{Z}_p$. $\mathcal{A}$ is given $(pp, \{ipk_1, ipk_2\}, \{isk_1, isk_2\}, \sigma_1, \sigma_2, m, \mathcal{S}, vpk, vsk)$. Finally, $\mathcal{C}$ chooses a random number $I \leftarrow_{\$} \{0,1\}$.

— **Queries**: $\mathcal{A}$ adaptively requests a finite number of randomized signatures. $\mathcal{C}$ answers each query by returning $(ipk'_I, \pi_{ipk'_I}, \sigma'_I) \leftarrow$ **Randomize**$(pp, ipk_I, vpk, \mathcal{S}, \sigma_I)$.

— **Output**: $\mathcal{A}$ eventually outputs $I' \in \{0,1\}$. The game outputs 1 if $I' = I$. The game outputs 0 otherwise.

A HIAC scheme is said to be Issuer-Indistinguishable iff, for a negligible ϵ, the probability for a PPT Adversary $\mathcal{A}$ to win the **IssuerIndistinguishGame** against a challenger $\mathcal{C}$ is:

$$\Pr[\textbf{IssuerIndistinguishGame}(\mathcal{A}, \mathcal{C}) = 1] \leq \frac{1}{2} + \epsilon$$

1. The formal definition of the adversary model only takes into account two trusted issuers. However, the privacy of the users relies on the size of the underlying anonymity set. Therefore, the size of a verifier's trusted issuer set should be large enough to hide the users.

Definition 5.4. (Trusted issuer) We define the **TrustedIssuerGame**$(\mathcal{A}, \mathcal{C})$, for a challenger $\mathcal{C}$ that represents a verifier and its trusted issuers, and an adversary $\mathcal{A}$ that represents a malicious user:

- **Setup**: $\mathcal{C}$ runs HIAC's **Setup**, **IssuerKeygen**, and **VerifierSetup** algorithms to obtain public parameters pp, k issuers' key pairs $\{(isk_i, ipk_i)\}_{i=1}^{k}$ and one verifier's information pair $(\mathcal{S}, \mathsf{aux}_{\mathcal{S}})$. $\mathcal{A}$ is given $\{(ipk_i)\}_{i=1}^{k}$, $\mathcal{S}$, and public information vpk.
- **Queries**: $\mathcal{A}$ adaptively requests signatures on at most n messages $m_1, ..., m_n$ to the k issuers. $\mathcal{C}$ answers each query by returning for each issuer $\{\sigma_{(i,j)} \leftarrow \textbf{Sign}(isk_j, m_i)\}_{i=1}^{n}$.
- **Output**: $\mathcal{A}$ eventually outputs a randomized message-signature pair $(m^*, (ipk^*, \pi^*_{ipk}, \sigma^*))$. The game outputs 1 if **VerifyRandomized**$(pp, vsk, ipk^*, \pi^*_{ipk}, m^*) = 1 \wedge ipk^* \neq \mathcal{S}$, and 0 otherwise.

An HIAC scheme is said to have the trusted-issuer property if, for a negligible value, ϵ, the probability for a Probabilistic Polynomial Time (PPT) Adversary $\mathcal{A}$ to win **TrustedIssuerGame** against a challenger $\mathcal{C}$ is:

$$\Pr[\textbf{TrustedIssuerGame}(\mathcal{A}, \mathcal{C}) = 1] \leq \epsilon$$

Definition 5.5. (Existential Unforgeability Against Chosen Message Attack) We define the **EUFCMAGame** $(\mathcal{A}, \mathcal{C})$, for a challenger $\mathcal{C}$ that represents a verifier and its trusted issuers, and an adversary $\mathcal{A}$ that represents a malicious user:

- **Setup**: $\mathcal{C}$ runs the HIAC's **Setup**, **IssuerKeygen**, and **VerifierSetup** algorithms to obtain the public parameters pp, k issuers' key pairs $\{(isk_i, ipk_i)\}_{i=1}^{k}$ and one verifier's information pair $(\mathcal{S}, \mathsf{aux}_{\mathcal{S}})$. $\mathcal{A}$ is given $\{(ipk_i)\}_{i=1}^{k}$, $\mathcal{S}$, and vpk.
- **Queries**: $\mathcal{A}$ adaptively requests signatures on at most n messages $m_1, ..., m_n$ from the k issuers. $\mathcal{C}$ answers each query by returning, for each issuer, $\{\{\sigma_{(i,j)} \leftarrow \textbf{Sign}(isk_j, m_i)\}_{i=1}^{n}\}_{j=1}^{k}$.
- **Output**: $\mathcal{A}$ eventually outputs a randomized message-signature pair $(m^*, (ipk^*, \pi^*_{ipk}, \sigma^*))$. The game outputs 1 if **VerifyRandomized**$(pp, vsk, ipk^*, \pi^*_{ipk}, m^*) = 1 \wedge m^* \neq m_i \, \forall i$, and 0 otherwise.

An HIAC scheme is said to have the EUF-CMA property if for a negligible value, ϵ, and M the space of possible messages, the probability for a PPT Adversary $\mathcal{A}$ to win the **EUFCMAGame** against a challenger $\mathcal{C}$ is:

$$\Pr[\textbf{EUFCMAGame}(\mathcal{A}, \mathcal{C}) = 1] \leq \frac{1}{M} + \epsilon$$

Definition 5.6. A Hidden Issuer Anonymous credential scheme is secure iff it achieves *correctness*, *EUF-CMA*, *issuer indistinguishability*, and *trusted issuer*.

5.6.2 Aggregator

To instantiate an HIAC scheme, we need an object which allows a verifier to commit to a set of values, and which offers a way for a user to prove that some value is indeed in the commitment set, without revealing this value, and while only knowing a commitment to this value. State-of-the-art primitives do not offer such a possibility. We therefore introduce a novel cryptographic object: the *aggregator*. In our HIAC scheme, the verifier builds the aggregator and uses it as a commitment to a set of trusted issuers. But we conjecture that other designs for an HIAC scheme would also need an aggregator or a similar object. An aggregator provides one witness for each trusted issuer. Each such witness can be used to prove that a specific issuer's public key (or a commitment to an issuer's secret key) is accumulated in the aggregator. The verifier is the only actor able to verify this set-membership proof. The user can further make this proof element indistinguishable by adding random elements to the witness and the issuer key. An aggregator is defined by the five following algorithms:

Definition 5.7.
— $(pp) \leftarrow \textbf{Setup}(\lambda)$: on input of setup parameter λ, the algorithm outputs pp, a description of public parameters whose security depends on λ;
— $(\mathsf{Agg}, \mathsf{aux}, \mathsf{sk}) \leftarrow \textbf{Gen}(pp, \mathcal{S})$: on input of pp and of a set $\mathcal{S}$, the algorithm outputs aggregator Agg, auxiliary information aux, and a secret verification key sk;
— $\{0,1\} \leftarrow \textbf{IntegrityVerification}(pp, \mathcal{S}, \mathsf{Agg}, \mathsf{aux})$: on input of pp, a set $\mathcal{S}$, aggregator Agg, and auxiliary parameters aux, the algorithm outputs 1 if Agg is valid, and if it aggregates all the elements in $\mathcal{S}$. The algorithm outputs 0 otherwise;
— $(C', \pi_s) \leftarrow \textbf{WitCreate}(pp, \mathsf{Agg}, \mathsf{aux}, C)$: on input of pp, aggregator Agg, auxiliary parameters aux, and a cryptographic commitment C to an element $s \in \mathcal{S}$, the algorithm outputs C' a new commitment to s, and a proof π_s which proves that C' is a commitment to an element aggregated in Agg.
— $\{0,1\} \leftarrow \textbf{Verify}(pp, \mathsf{Agg}, \mathsf{aux}, \mathsf{sk}, C', \pi_s)$: on input of pp, aggregator Agg, auxiliary parameters aux, a cryptographic commitment C' to an element $s \in \mathcal{S}$, a secret verification key sk, and a witness W_s, the algorithm outputs 1 if π_s is valid and 0 otherwise.

Security notions for a secure aggregator can be derived from the needs expressed in the HIAC formal definition. The first, *Collision-freedom*, states that the probability for an adversary to find a proof of set-membership for a non accumulated element is negligible. The trusted issuer property of the formal HIAC definition is derived from the collision-freedom property of its aggregator.

Definition 5.8. (Collision Freedom) We define the $\textbf{CollisionFreedomGame}(\mathcal{A}, \mathcal{C})$, for a challenger $\mathcal{C}$ that represents a verifier and its trusted issuers, and an adversary $\mathcal{A}$ that represents a malicious user:

— **Setup**: $\mathcal{C}$ runs the aggregator **Setup** algorithm, chooses a set $\mathcal{S}$, runs the **Gen** algo-

rithm to build a commitment to $\mathcal{S}$, and the associated verification key. $\mathcal{A}$ is given the aggregator's public information.

— **Output**: $\mathcal{A}$ outputs C^* a commitment to s^* and π_s^*. The game outputs 1 if $s^* \notin \mathcal{S}$, and **Verify**$(pp, \mathsf{Agg}, \mathsf{aux}, \mathsf{sk}, C^*, \pi_s^*) = 1$, or 0 otherwise.

An Aggregator is said to be element-indistinguishable if for a negligible ϵ, the probability for a PPT Adversary $\mathcal{A}$ can to win the **CollisionfreedomGame** is:

$$\Pr[\textbf{CollisionfreedomGame}(\mathcal{A}, \mathcal{C}) = 1] \leq \epsilon$$

The second security notion, *Element Indistinguishabity* states that, while the verifier running the **Verify** algorithm knows that commitment C' is associated with one of the elements of $\mathcal{S}$, it cannot learn the actual element committed to. The issuer-indistinguishability property of the HIAC definition is partially derived from the element-indistinguishability property of its aggregator.

Definition 5.9. (Element indistinguishability) We define the **ElementIndistinguishGame**$(\mathcal{A}, \mathcal{C})$, for a challenger $\mathcal{C}$ that represents a user, and the actions preformed by the issuers, and an adversary $\mathcal{A}$ that represents a malicious verifier:

— **Setup**: $\mathcal{C}$ runs the aggregator **Setup** algorithm, and chooses a set $\mathcal{S}$ with at least 2 elements. Then, $\mathcal{A}$ runs the **Gen** algorithm to build a commitment to the set $\mathcal{S}$, and the associated verification key. Finally, $\mathcal{C}$ chooses $s \in \mathcal{S}$. $\mathcal{A}$ is given the set of secret values aggregated in the aggregator, the commitment to the issuer set, and the secret verification key.

— **Queries**: $\mathcal{A}$ adaptively requests q randomized commitments to elements of $\mathcal{S}$, and the associated proofs. $\mathcal{C}$ answers each query by returning $(C', \pi_s) \leftarrow$ **WitCreate**$(pp, \mathsf{Agg}, \mathsf{aux}, C)$ a commitment to $s \in \mathcal{S}$.

— **Output**: $\mathcal{A}$ eventually outputs $s' \in \mathcal{S}$. The game outputs 1 if $s' = s$. The game outputs 0 otherwise.

An Aggregator is said to be element-indistinguishable if for a negligible value, ϵ, and k trusted issuers, the probability for a PPT Adversary $\mathcal{A}$ to win the **ElementIndistinguishGame** against a challenger $\mathcal{C}$ is:

$$\Pr[\textbf{ElementIndistinguishGame}(\mathcal{A}, \mathcal{C}) = 1] \leq \frac{1}{k} + \epsilon$$

The last security notion is *correctness*:

Definition 5.10. An aggregator scheme is correct iff, for any honestly built tuple $(pp, \mathsf{Agg}, \mathsf{aux}, \mathsf{sk}, C', \pi_s)$, the **Verify** algorithm always outputs 1.

Definition 5.11. An Aggregator is secure iff it achieves *correctness*, *collision-freedom*, and

element indistinguishability.

If this aggregator is embedded correctly into a Hidden Issuer Anonymous Credential scheme, then the *Trusted Issuer* property relies entirely on the aggregator's *Collision Freedom* property.

5.7 Instantiation

Next, we instantiate our aggregator object and our HIAC scheme. We start by describing a non-interactive version of our scheme. Then, we present our final, interactive version.

Our setup is trust free. None of the actors needs to trust the constructions made by the other actors. The integrity of the elements published by each actor can be verified algorithmically. In this section, we assume all issuers of the system issue the same types of messages. We explain how to verify this assumption in Section 5.8.2.

5.7.1 Non-Interactive HIAC

Aggregator

Our aggregator consists of five algorithms: **Setup**, **Gen**, **IntegrityVerification**, **WitCreate**, and **Verify**. Unlike in the formal definition of Section 5.6.2, we pass the random values r_1 and r_2 to **WitCreate**. This makes it possible to map elements of the set-membership proof to the actual randomized signature in the instantiation of our HIAC scheme. As we mentioned, the aggregator is independent from the HIAC scheme, and can be used with other signature schemes based on bilinear pairings to provide issuer indistinguishability.

Protocol 5.1.

1. $(pp) \leftarrow$**Setup**(λ): on input of a security parameter λ, the algorithm chooses three prime-order p groups with the associated generators $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T = \langle g_T \rangle$. The algorithm then chooses a type 3 bilinear pairing e such that $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. The algorithm outputs public parameters $pp = (p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_t, g_T, e)$.
2. $(\mathsf{Agg}, \mathsf{aux}, \mathsf{sk}, T_1) \leftarrow$**Gen**$(pp, \mathcal{S})$: on input of pp and of a set $\mathcal{S} = \{g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}}\}, x_i \in \mathbb{Z}_p, \forall i \in \mathbb{Z}_p^*$, $k \geq 2$, the algorithm computes a secret key $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_p$, an aggregator $\mathsf{Agg} = (\mathcal{S}, W)$, with witness set $W = \{(W)_{(j)} = g_1^{\mathsf{sk}\frac{1}{x_j}}\}_{j=1}^{k}$. The algorithm also computes some auxiliary information $\mathsf{aux} = \beta$, with $\beta = \mathrm{NI-ZKPoE}\{(\mathsf{sk}) : \bigwedge_{i=1}^{k}(W)_{(i)} = (\mathcal{S})_{(i)}^{\mathsf{sk}}\}$. The algorithm finally outputs $(\mathsf{Agg}, \mathsf{aux}, \mathsf{sk})$;
3. $\{0,1\} \leftarrow$**IntegrityVerification**$(pp, \mathcal{S}_{\mathsf{Comm}}, \mathsf{Agg}, \mathsf{aux})$: on input of pp, of a set $\mathcal{S}_{\mathsf{Comm}} = \{g_2^{x_1}, \cdots, g_2^{x_k}\}, x_i \in \mathbb{Z}_p, \forall i \in \{1, \cdots, k\}$, of an aggregator $\mathsf{Agg} = (\mathcal{S}, W)$, and of the auxiliary parameter β, the algorithm outputs 1 if β is valid and 0 otherwise;
4. $(C', \pi_s) \leftarrow$**WitCreate**$(pp, \mathsf{Agg}, \mathsf{aux}, C, r_1, r_2)$: on input of pp, of an aggregator $\mathsf{Agg} = (\mathcal{S}, W)$, of auxiliary parameters aux, of a cryptographic commitment $C = g_2^{x_l}, l \in$

$\{1, \cdots, k\}$, and of two randomly chosen numbers r_1, r_2, the algorithm computes $C' = C^{r_1}$, and a proof $\pi_s = ((W)'_{(l)}, h)$, with $(W)'_{(l)} = (W)^{r_2}_{(l)}$ and $h = g_2^{r_1 r_2}$; the algorithm outputs (C', π_s).

5. $\{0,1\} \leftarrow$**Verify**$(pp, \mathsf{sk}, C', \pi_s)$: on input of pp, of the secret aggregator's verification key sk, of a randomized commitment C', and of a proof $\pi_s = ((W)'_{(l)}, h)$, the algorithm outputs 1 if: $e((W)'_{(l)}, C') \stackrel{?}{=} e(g_1^{\mathsf{sk}}, h)$; and 0 otherwise.

Security Analysis

Theorem 5.1. The aggregator presented by Protocol 5.1 is correct.

Proof sketch 5.1. The proof of this theorem, in Appendix C.2, uses a correctly built element $e((W)'_{(i)}, C')$ to prove that it is equivalent to $e(g_T^{sk}, h)$.

Theorem 5.2. (Collision Freedom) The aggregator presented by Protocol 5.1 is collision free, under the CDH assumption (Definition 5.8). [2]

Remark In anticipation of the Hidden Issuer Anonymous Credential (HIAC) protocol, we will prove Theorem 5.2 with an extended version of the adversary model, in which the adversary has access to the extra values $g_1^{x_i}, \forall i \in \{1, \cdots, k\}$. Indeed, these values are accessible to the user in our instantiation of HIAC.

Proof sketch 5.2. The proof of Theorem 5.2, in Appendix C.3, shows that under the CDH assumption the secret value sk used in $e(g_T^{\mathsf{sk}}, h)$ forces the adversary to use exactly one witness to build its $(W)'_l$. Then it shows that C^* contains exactly one x_i. [3]

Remark The *collision-freedom* proof uses the fact that the value g_1^{sk} is known only to the verifier. But, if a given Aggregator was used multiple times, it would be possible for a malicious Issuer I to compute and publish the value $g_1^{\mathsf{sk}} = (W)^{x_I}_{(I)}$. For this reason, the non-interactive HIAC scheme computes a new aggregator at every transaction (cfr. Section 5.7.1).

Theorem 5.3. The scheme presented here is *Element Indistinguishable* (Definition 5.9).

Proof sketch 5.3. The proof of the Theorem 5.3, in Appendix C.4, proceeds by analyzing the elements the adversary is given. Either there is no linear relationship between these elements (which, under the DDH assumption, means that no adversary can decide which element is being proven to be in $\mathcal{S}$) or the elements are symmetrical (and thus they do not depend on the variables, or they are distributed uniformly at random in the group).

2. The *collision-freedom* proof uses the fact that the value g_1^{sk} is known only to the verifier. But, if a given Aggregator was used multiple times, it would be possible for a malicious Issuer I to compute and publish the value $g_1^{\mathsf{sk}} = (W)^{x_I}_{(I)}$. For this reason, the non-interactive HIAC scheme computes a new aggregator at every transaction (cfr. Section 5.7.1).

3. In anticipation of the Hidden Issuer Anonymous Credential (HIAC) protocol, we will prove Theorem 5.2 with an extended version of the adversary model, in which the adversary has access to the extra values $g_1^{x_i}, \forall i \in \{1, \cdots, k\}$.

Signature Scheme

Our Hidden Issuer Anonymous Credential (HIAC) scheme relies on a modified Pointcheval Sanders (PS) scheme [84] and uses six algorithms. The first three come from a classical signature scheme, namely **Setup** which produces the shared material, **IssuerKeygen** which allows an issuer to build a key pair, and **Sign** which allows an issuer to sign a credential. The three others provide the issuer-indistinguishability and trusted-issuer properties. The verifier runs the **VerifierSetup** algorithm to select a set of trusted issuers and generate the aggregator. In this non-interactive version of HIAC, the verifier needs to do this before each transaction to prevent malicious issuers from extracting the verifier's private key from the aggregator. Once it has the aggregator, the user runs the **Randomize** algorithm to randomize the credential and the associated issuer's key, and to create the associated proof of set-membership. Finally, the verifier runs **VerifyRandomized** to verify the proof and the authenticity of the randomized credential.

Protocol 5.2.

1. $(pp) \leftarrow$ **Setup**(λ): on input of a security parameter λ , the algorithm chooses three prime-order p groups with the associated generators $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T = \langle g_T \rangle$. The algorithm then chooses a type 3 bilinear pairing e such that $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Finally, it chooses a cryptographic hash function $\mathsf{H} : \mathbb{Z} \rightarrow \mathbb{Z}_p$. The algorithm outputs $pp = (p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_t, g_T, e, \mathsf{H})$.

2. $(isk_I, ipk_I) \leftarrow$ **IssuerKeygen**(pp): on input of pp, an issuer I computes a secret key $isk_I = (x_I, y_I)$, with $x_I, y_I \leftarrow_\$ \mathbb{Z}_p^*$ and the associated public key $ipk_I = (X^{(I)}, \bar{X}_1^{(I)}, Y_1^{(I)}, \bar{Y}_1^{(I)}, Y_2^{(I)}) = (g_2^{x_I}, g_1^{\frac{1}{x_I}}, g_1^{y_I}, g_1^{\frac{1}{y_I}}, g_2^{y_I})$. [4] The algorithm outputs (isk_I, ipk_I).

3. $(\sigma, R_y) \leftarrow$ **Sign**(pp, m, isk_I, u, ϕ): on input of pp, of a message m, of an issuer's secret key $isk_I = (x_I, y_I)$, of a value given by the user $u = (Y_1^{(I)})^{R_y} \cdot g_1^R$, where R_y and R are secret random values, and of ϕ, a ZKPoK of u, ϕ :NI-ZKPoK$\{(R_y R) : u = (Y_1^{(I)})^{R_y} \cdot g_1^R\}$, the issuer initially outputs a signature $\sigma^\Delta = (h_1, h_2, \sigma_1^\Delta, \sigma_2^\Delta)$, where $h_1 = g_1^{r_{(I,1)}}$, $h_2 = g_1^{r_{(I,2)}}$, $\sigma_1^\Delta = (g_1^{x_I} \cdot u^{\mathsf{H}(m)})^{r_{(I,1)}}$, $\sigma_2^\Delta = (g_1^{x_I} \cdot u^{\mathsf{H}(\mathsf{H}(m))})^{r_{(I,2)}}$, with $r_{(I,1)}, r_{(I,2)} \leftarrow_\$ \mathbb{Z}_p^*$. Then the user transforms the signature σ^Δ into $\sigma = (h_1, h_2, \sigma_1 = \sigma_1^\Delta \cdot h_1^{-R\mathsf{H}(m)}, \sigma_2 = \sigma_2^\Delta \cdot h_2^{-R\mathsf{H}(\mathsf{H}(m))})$. The user stores σ and R_y.

4. $(vpk, vsk) \leftarrow$ **VerifierSetup**$(pp, \{ipk_i\}_{i=1}^k)$: on input of pp and of trusted issuers' public keys $\{ipk_i\}_{i=1}^k$, the verifier commits to the issuer's secret keys x by building a new aggregator $(\mathsf{Agg}_x, \mathsf{sk}_x, \mathsf{aux}_x) = \text{Aggregator}\cdot\textbf{Gen}(pp, \mathcal{S}_x)$. Symmetrically, the verifier also commits to the issuer's secret key y by building another aggregator $(\mathsf{Agg}_y, \mathsf{sk}_y, \mathsf{aux}_y) = \text{Aggregator}\cdot\textbf{Gen}(pp, \mathcal{S}_y)$. The algorithm outputs $vsk = (\mathsf{sk}_x, \mathsf{sk}_y)$ and $vpk = (\mathsf{Agg}_x, \mathsf{aux}_x, \mathsf{Agg}_y, \mathsf{aux}_y)$.

4. In the PS version of the signature, $g_1^{\frac{1}{x_I}}$ and $g_1^{\frac{1}{y_I}}$ are not disclosed. We show in the proof section that, thanks to the CDH assumption, this modification does not impact the security of the signature.

5. $(X^{(I)'}, Y_2^{(I)'}, \pi_x, \pi_y, \sigma') \leftarrow$ **Randomize**(pp, ipk, vpk, $\mathcal{S}$, σ, R_y): on input of pp, of a signature $\sigma = (h_1, h_2, \sigma_1, \sigma_2)$, of a verifier public key $vpk = (\mathsf{Agg}_x, \mathsf{aux}_x, \mathsf{Agg}_y, \mathsf{aux}_y)$, and of a secret value R_y generated by the user in step 3, and associated with σ, this algorithm, run by a user:
   (a) Verifies if:

$$\text{Aggregator}\cdot\textbf{IntegrityVerification}(pp, \{X^{(j)}\}_{j=1}^{k}, \mathsf{Agg}_x, \mathsf{aux}_x) \stackrel{?}{=} 1$$

$$\text{Aggregator}\cdot\textbf{IntegrityVerification}(pp, \{Y_2^{(j)}\}_{j=1}^{k}, \mathsf{Agg}_y, \mathsf{aux}_y) \stackrel{?}{=} 1$$

   Otherwise, it aborts.
   (b) Chooses $r_u, r'_u, r''_u, r_{(u,x)}, r_{(u,y)} \leftarrow_{\$} \mathbb{Z}_p^*$
   (c) Computes :

$$(X^{(I)'}, \pi_x) = \text{Aggregator} \cdot \textbf{WitCreate}(pp, \mathsf{Agg}_x, \mathsf{aux}_x, r_u, r_{(u,x)})$$

$$(Y^{(I)'}, \pi_y) = \text{Aggregator} \cdot \textbf{WitCreate}(pp, \mathsf{Agg}_y, \mathsf{aux}_y, r_u R_y, r_{(u,y)})$$

   (d) Computes $\sigma' = (h'_1 = h_1^{r'_u}, h'_2 = h_2^{r''_u}, \sigma'_1 = \sigma_1^{r_u r'_u}, \sigma'_2 = \sigma_2^{r_u r''_u})$
   The algorithm outputs $(X^{(I)'}, Y_2^{(I)'}, \pi_x, \pi_y, \sigma')$.
6. $\{0,1\} \leftarrow$ **VerifyRandomized**(pp, σ', $X^{(I)'}$, $Y_2^{(I)'}$, π_x, π_y, m, vsk):[5] on input of pp, of commitments to an issuer's keys $X^{(I)'}, Y_2^{(I)'}$, of two proofs of set-membership π_x, π_y, of a randomized signature $\sigma' = (h'_1, h'_2, \sigma'_1, \sigma'_2)$, of a verifier secret key $vsk = (\mathsf{sk}_x, \mathsf{sk}_y)$, and of a message m, a verifier, outputs 1 if the proof of set-membership is honestly built, *i.e.*, if:
   $\text{Aggregator} \cdot \textbf{Verify}(pp, \mathsf{Agg}_x, \mathsf{aux_x}, \mathsf{sk}_x, X^{(I)'}, \pi_x) \stackrel{?}{=} 1$,
   $\text{Aggregator} \cdot \textbf{Verify}(pp, \mathsf{Agg}_y, \mathsf{aux_y}, \mathsf{sk}_y, Y_2^{(I)'}, \pi_y) \stackrel{?}{=} 1$,
   and the signature is correct, *i.e.*,
   — $e(h'_1, X^{(i)'} Y_2^{(i)'\mathsf{H}(m)}) \stackrel{?}{=} e(\sigma'_1, g_2)$
   — $e(h'_2, X^{(i)'} Y_2^{(i)'\mathsf{H}(\mathsf{H}(m))}) \stackrel{?}{=} e(\sigma'_2, g_2)$

5. The user can verify the integrity of the signature – without using **VerifyRandomized** – by verifying that $e(\sigma_1, g_2) \stackrel{?}{=} e(h_1, X^{(I)}(Y_2^{(I)})^{R_y \mathsf{H}(m)})$, $e(\sigma_2, g_2) \stackrel{?}{=} e(h_2, X^{(I)}(Y_2^{(I)})^{R_y \mathsf{H}(\mathsf{H}(m))})$, and $h_1 \neq 1_{\mathbb{G}_1}, h_2 \neq 1_{\mathbb{G}_1}$

— $h'_1 \neq 1_{\mathbb{G}_1}, h'_2 \neq 1_{\mathbb{G}_1}, \sigma'_1 \neq 1_{\mathbb{G}_1}, \sigma'_2 \neq 1_{\mathbb{G}_1}$;
the verifier outputs 0 otherwise.

Security Analysis The formal definition of HIAC gives us 4 security notions that our instantiation must achieve: EUF-CMA, unlinkability, issuer indistinguishability, trusted issuer. The trusted-issuer property comes from the collision freedom of the aggregator and is already proven.

Theorem 5.4. The **VerifyRandomized** algorithm is correct, *i.e.*, any honestly built signature will be accepted by the **VerifyRandomized** algorithm.

Proof sketch 5.4. The proof, in Appendix C.5, shows that a correctly built signature is equivalent to $e(h_1^{'}, X^{(i)'} Y_2^{(i)'\mathsf{H}(m)})$.

Theorem 5.5. The signature presented here has the EUF-CMA property.

Proof sketch 5.5. The proof, in Appendix C.6, shows that, under the CDH assumption, the user cannot find a way to inject malicious material into $Y_2'^{(I)}$, σ_1 and σ_2. First, the malicious user would need to know $g_1^{r_{(I,a)} y_I}, a \in \{1, 2\}$ to be able to modify the message a credential refers to. However, under the CDH assumption, this value cannot be found by the adversary. Second, the malicious user would need to tweak the commitment to $Y_2^{(I)}$. However, tweaking these values, so that they validate both of the signature's verification equations at the same time would imply inverting a hash function. This shows that the signature has the EUF-CMA property, under the CDH assumption, in the Random Oracle Model.

The random elements added by the user in **Randomize** ensure that the verifier cannot compare the commitment's elements with actual elements of its own key, and the elements of the signature with the formerly issued credentials. This provides issuer indistinguishability.

Theorem 5.6. The Hidden Issuer Anonymous Credential scheme presented here is *issuer indistinguishable*.

Proof sketch 5.6. The proof, in Appendix C.7 extends the one for Theorem 5.3: it compares two randomized signatures issued by two different issuers and shows that the differences between them are hidden by the random elements added by the user in the **Randomize** algorithm. Under the DDH assumption, the malicious verifier has no way to compare the signature it is given with any other signature as all its elements are hidden by a random number that is unknown to the verifier.

5.7.2 Interactive HIAC

The interactive version of our scheme removes the need for the verifier to run **VerifierSetup** and for the user to run Aggregator·**IntegrityVerification** as part of **Randomize** at each transaction. To achieve this, it relies on an interactive version of the aggregator instantiation.

Aggregator

Our interactive aggregator version modifies the **Gen**, and **WitCreate** algorithms in Protocol 5.1. It modifies the witnesses making sure that no malicious issuer can output the verifier's private key by computing $(W_{(l)})^{x_l} = g_1^{\mathsf{sk}}$. This implies that the user needs to interact with the verifier to randomize a witness, hence the interactive nature of the protocol.

- $(\mathsf{Agg}, \mathsf{aux}, \mathsf{sk}, T_1) \leftarrow$**Gen'**$(pp, \mathcal{S})$: on input of pp and of a set $\mathcal{S} = \{g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}}\}, x_i \in \mathbb{Z}_p, \forall i \in \mathbb{Z}_p^*$, $k \geq 2$, the algorithm computes a secret key $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_p$, an aggregator $\mathsf{Agg} = (\mathcal{S}, W)$, with $W = \{(W)_{(j)} = g_1^{\mathsf{sk}(1+\frac{1}{x_j})}\}_{j=1}^k$. The algorithm also computes some auxiliary information $\mathsf{aux} = \beta$, with:$\beta = \mathrm{NI-ZKPoE}\{(\mathsf{sk}) : \bigwedge_{i=1}^k (W)_{(i)} = ((\mathcal{S})_{(i)} \cdot g_1)^{\mathsf{sk}}\}$ The algorithm finally outputs $(\mathsf{Agg}, \mathsf{aux}, \mathsf{sk})$;
- $(C', \pi_s) \leftarrow$**WitCreate'**$(pp,\ \mathsf{Agg},\ \mathsf{aux},\ C,\ r_1,\ r_2)$: on input of pp, of an aggregator $\mathsf{Agg} = (\mathcal{S}, W)$, of auxiliary parameters aux, of a commitment $C = g_2^{x_l}, l \in \{1, \cdots, k\}$, and of two random numbers r_1, r_2, the algorithm computes $C' = C^{r_1}$, and a proof $\pi_s = ((W)'_{(l)}, h)$, with $(W)'_{(l)} =$**CommitRevealExchange**$((W)_{(l)})$ and $h = g_2^{r_1 r_2}$; the algorithm outputs (C', π_s). The result of **CommitRevealExchange** is sent to the verifier.

The algorithms use the **CommitRevealExchange** function in Figure 5.1. On input of a witness, the function allows user and verifier to compute a randomized witness interactively. We stress that this does not limit applicability as the presentation of a user's credential to a verifier is generally an interactive operation. We discuss the efficiency trade-off between interactive and non-interactive variants in Section 5.9.1.

User$(pp, \sigma, \mathsf{pk})$		**Verifier**(pp, sk)
$r_a, r_b, r_c \leftarrow_\$ \mathbb{Z}_p^*;\ (\mathsf{Comm}'(s), \pi_s, r_2) =$ **Randomize**(σ, pk)		$r_d \leftarrow_\$ \mathbb{Z}_p^*$
$(W)''_I = (W)_I^{r_2} g_1^{r_a};\ C_1 = g_1^{r_2 r_b};\ C_2 = g_1^{r_a r_c}$		
$\phi_1 = NI - ZKPoK\{(r_2 r_b, r_a r_c) : C_1 = g_1^{r_2 r_b} \wedge C_2 = g_1^{r_a r_c}\}$	$\xrightarrow{(\sigma', (W)''_I, C_1, C_2, \phi_1)}$	**VerifyZKP**(ϕ_1)
		$C'_1 = C_1^{\mathsf{sk} r_d}; C'_2 = C_2^{r_d}$
		$\phi_2 = NI - ZKPoK\{(\mathsf{sk}, r_d) :$
VerifyZKP(ϕ_2)	$\xrightarrow{(C'_1, C'_2, \phi_2)}$	$C'_1 = C_1^{\mathsf{sk} r_d} \wedge C'_2 = C_2^{r_d}\}$
$C''_1 = (C'_1)^{\frac{1}{r_b}} (C'_2)^{\frac{1}{r_c}}$	$\xrightarrow{C''_1}$	$(W)'_I = (W)''_I (C''_1)^{-\frac{1}{r_d}}$

Figure 5.1 – Commitment reveal exchange

Signature Scheme

The interactive aggregator can be directly used in the signature instantiation by replacing Aggregator.**Gen** and Aggregator.**WitCreate** by Aggregator.**Gen'** and Aggregator.**WitCreate'**. Thanks to these modifications, the verifier only needs to run **VerifierSetup** when its set of trusted issuers changes. Similarly, the user only needs to run Aggregator·**IntegrityVerification**

as part of the **Randomize** algorithm when interacting with a given verifier for the first time or when the verifier's aggregators or keys change.

Security Analysis All we need to show is that the interactive aggregator respects the same properties as the non-interactive one.

Lemma 5.1. The interactive version of the aggregator is collision-free, correct, and element indistinguishable.

Proof sketch 5.7. The proof, in appendix C.8, uses the original aggregator proofs presented in Appendix C.3 and Appendix C.4, and shows that the protocol presented in Figure 5.1 does not affect their validity.

5.8 Deployment

The protocols described in the previous sections fulfill the adversary model, and can be instantiated in real-world use cases. But an Identity Management System does not depend only on its signature scheme as trust between real-life actors cannot be enforced only by algorithms. In this section, we discuss how organizational solutions could improve the security of our scheme by ensuring that its assumptions are satisfied. It is important to note that these solutions are not trivial to implement, and that the privacy impact of an insecure implementation could be significant. A user that trusts the system to protect his/her privacy will tend to leak more information when the system's assumptions are not satisfied than a user who is aware of being in a "known-issuer" setup.

5.8.1 Credential and Aggregator Management

The formal definition of the issuer-indistinguishability property (Definition 5.3) assumes that all the verifier's trusted issuers issue credentials on the same types of messages. But in many cases, including in our concert-bar example, the verifier needs to verify multiple credential types. This can easily be solved by having multiple credentials(e.g. one for the ticket, one for the Covid-19 vaccine, and one for the legal age). The user can obtain these credentials either from the same issuer or from different issuers. In either case, each credential will be associated with a specific anonymity set when matched against a verifier's aggregator. To this end, the verifier—the bar in our example—should build a specific aggregator for each credential type. Note that a given issuer may well appear into multiple such aggregators if it issues credentials of multiple types.

5.8.2 Issuer Selection

Clearly, issuer indistinguishability only holds within a specific anonymity set. If the set is too small, or if the set contains issuers that do not issue credentials of the right type, the verifier may be able to associate a credential with a specific issuer. [6] To limit this kinds of attacks, the user's SSI client can integrate a conformity checker. The user should be able to input a desired security policy. The conformity checker can then verify that a verifier's aggregator contains a large enough number of issuers of the right type that satisfy the policy. The security policy can, for example, specify the geographical distribution of issuers or the organization running the issuing servers.

The checker should be able to access the relevant information for verifying the policy. Most SSI frameworks use a ledger that allows actors to register Decentralized IDentifier (DID) Documents [8]. These documents describe some characteristics of the actors of the framework, and are accessible by every one of them. To this end, the verifier provides the user with the public keys of the issuers aggregated in its aggregator. The conformity checker can verify that these keys correspond to those referenced in the aggregator. It can use them to retrieve the DIDs of the issuers from the ledger and use those to verify the security policy. If the checker detects that a verifier's aggregator does not satisfy the policy, it aborts the transaction and informs the user with a message similar to the one displayed by browsers in the event of an expired HTTPS certificate. [7] If an advanced user understands the associated risks, they can resume the transaction. If the checker detects that a verifier's aggregator does not Furthermore, the checker can detect more elaborated attacks. For example, a verifier could link multiple presentations from the same user via a side channel. In this case, the verifier could adapt its trusted-issuer set, by presenting a set from which it removes one new issuer at each new presentation. Eventually, the removed issuer will be the user's issuer. To detect and counter this attack, the checker can consider the trusted-issuer sets of all the transactions between its owner and the verifier in a specified time window, and check the validity of their intersection rather than the validity of the latest of them. The checker can then warn the user and block the transaction as soon as this intersection becomes too small.

This scheme does not require a central authority, but if one exists, the checker can report misbehaviors to it. For example, legislative power can produce norms involving sanctions if not respected by verifiers. Other central authorities can emerge in decentralized settings. For example, the SSI project Sovrin [52] is decentralized but uses a permissioned blockchain. In this case a central authority consisting of the permissioned nodes of the blockchain can, for instance,

6. We need to emphasize the fact that not only does the verifier need to build a consistent set of trusted issuers, but these issuers must also agree on credential templates and curve parameters.

7. The design and implementation of messages informing the user about privacy risks is out of the scope of this chapter. These messages will have a important role, as they will allow users to understand and mitigate privacy risks.

revoke the verifying rights of a misbehaving verifier by publishing a revocation list. A variety of other solutions are possible but their discussion is outside the scope of this chapter.

5.8.3 Issuer Acting as a Verifier

It is interesting to consider the case in which the verifier is also the issuer of a user's credential. In this case, issuer and verifier are automatically colluding. This makes the identity-inference threats highlighted in the introduction a lot more likely. In this case, Classical AC schemes only provide credential-multi-usage unlinkability. The issuer/verifier can identify whether it was the one that issued a given user's credential. This places the user in a relatively small anonymity set corresponding to all the users that received a credential from the issuer/verifier.

With our solution, the issuer/verifier cannot know whether a given credential originates from itself. This enlarges the user's anonymity set to all the users that received credentials from any of the issuers trusted by the issuer/verifier. Our scheme therefore increases user privacy even in this particularly challenging case.

5.9 Efficiency

We evaluate the efficiency of our implementation and compare it with that of other signature schemes. We first present a runtime-complexity analysis and compare our scheme with the Pointcheval-Sanders (PS) signature [84]. Then, we compare the asymptotic complexity of our scheme with that of PS and that of the Issuer-Hidden-Attribute-Based-Credential scheme (IHABC) [26]. Finally, we estimate the communication cost of our scheme.

5.9.1 Runtime Comparison

We start by comparing the runtime cost of our scheme with that of PS, highlighting the overhead resulting from the issuer-indistinguishability property. As both schemes are implemented using the same tools, this is the most accurate manner to identify the overhead of our scheme, regardless of implementation details. We start by comparing the non-interactive and interactive variants. Then we analyze the cost of each operation: issuer-key generation, verifier-key generation, signature issuance, and signature presentation.

We base our comparisons on an implementation in Rust (at: `https://gitlab.inria.fr/mgestin/rust_hidden_issuer_signature`). We use the BLS12-381 curve [109], a widely used curve for pairing-based cryptography with efficient computation time. BLS is a type-3-pairing-friendly curve with 256-bit-prime group size. This group size gives us a 128-bit AES security level. We ran our experiments on an i7-1185G7, 3.0 GHz CPU, with no multi-processor optimizations.

Non-Interactive vs Interactive Version. Section 5.7 presented two versions of our scheme: a

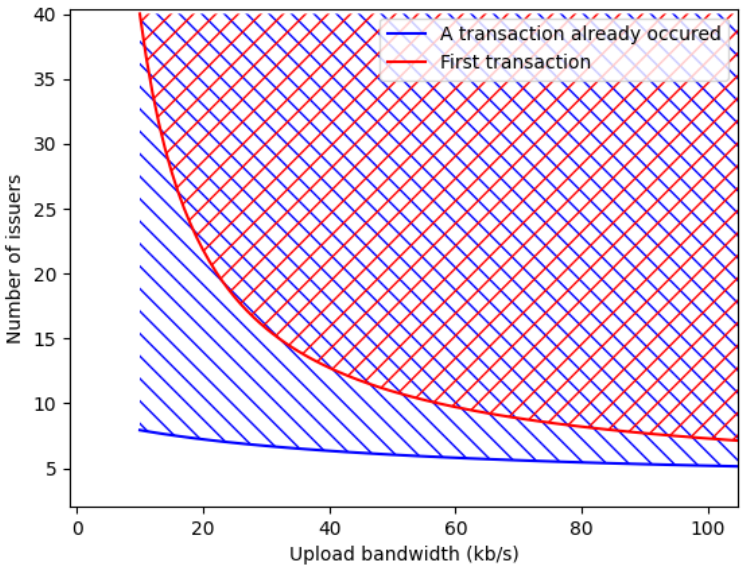


Figure 5.2 – Number of issuers beyond which the interactive scheme is more efficient than the non-interactive one based on the available bandwidth. The hatched areas represent the area where the interactive protocol is more efficient than the non-interactive one. The red curve and hatching from bottom left to top right consider a first-time interaction: both variants needs to verify the verifier's keys. The blue curve and hatching from top left to bottom right considers a subsequent transaction with the same verifier without a change in the aggregator or key: the interactive scheme does not need to verify the verifier's keys and aggregators as they are already present on the user's device.

non-interactive and an interactive one. The non-interactive version requires repeating **VerifierSetup** and **IntegrityVerification** at each credential-presentation transaction, which involves sending two aggregators whose size depends on the number of issuers. As a result, the interactive version becomes more and more interesting as the number of issuers in these aggregators increases. Figure 5.2 depicts, for a given value of available upload bandwidth, the number of issuers beyond which the interactive version is more efficient than the non-interactive one. For non-first-time interactions (in blue) the interactive protocol turns out to be more efficient as soon as there are more than a few issuers in the aggregator (7 issuers for 40 kbps of upload bandwidth). For first-time interactions (in red), the interactive protocol remains more efficient as long as the upload bandwidth is reasonable albeit for a slightly larger number of issuers (beyond 13 issuers for 40 kbps).

In scenarios where the presentation of credentials cannot be done interactively, it is necessary to run **VerifierSetup** and **IntegrityVerification** at each transaction. But even then, our experiments show that the running time of each of these two operations remains under 250 ms

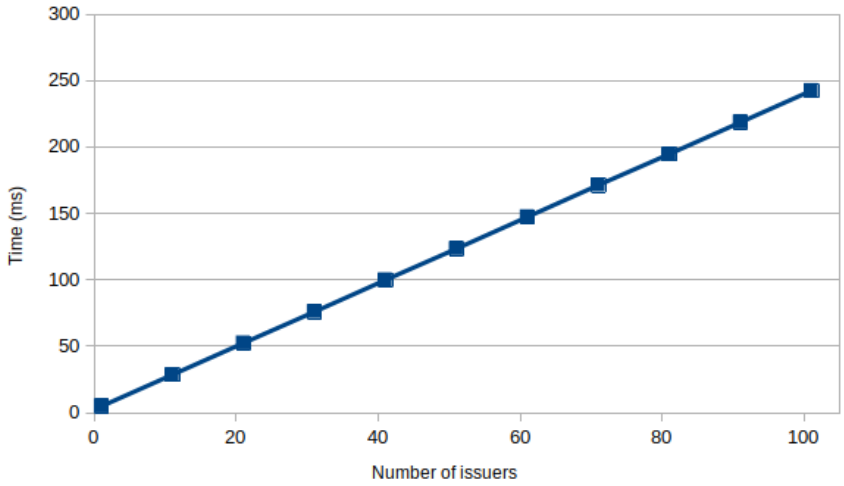


Figure 5.3 – Average time to build one verifier commitment (**VerifierSetup**) with the HIAC scheme as a function of the number of trusted issuers added to the aggregator.

with 100 trusted issuers. Moreover, in SSI systems the presentation of credentials is generally an interactive process, so using the interactive protocol appears natural. For this reasons, we concentrate on the interactive protocol in the following.

Issuer-Key Generation. The generation of an issuer key (**IssuerKeyGen**) takes approximately twice as long in our scheme (average: 4.17 ms) as in PS (average: 2.17 ms). This is because our scheme needs to compute 3 more elements: two to build a verifier key, and one to issue a signature. However, key generation is run only once in a long period of time. Therefore this difference is not a major drawback for our scheme.

Verifier-Key Generation. The verifier-key-generation algorithm is unique to the HIAC scheme. It is run by the verifier after all trusted issuers have created their keys and its performance depends on the number of issuers that are being added to the aggregator. Figure 5.3 shows that this relationship is linear. Nonetheless, in the interactive version of our scheme, this algorithm only needs to be run when the verifier's secret keys, sk, change. Moreover, since the dependence is on the number of added issuers, this update can actually be very efficient in a number of use cases.

Signature Issuance. The HIAC signature-issuance protocol is equivalent to that of PS. However, our scheme requires issuing the equivalent of two PS signatures, thus taking approximately twice as long to run (average: 9.7 ms) as in PS (average: 4.67 ms) on the issuer's side. In addition, our scheme requires the user to compute 6 exponentiations during the issuance process. This takes 4.91 ms on average on the user's side. In most of the use cases, a user only requests

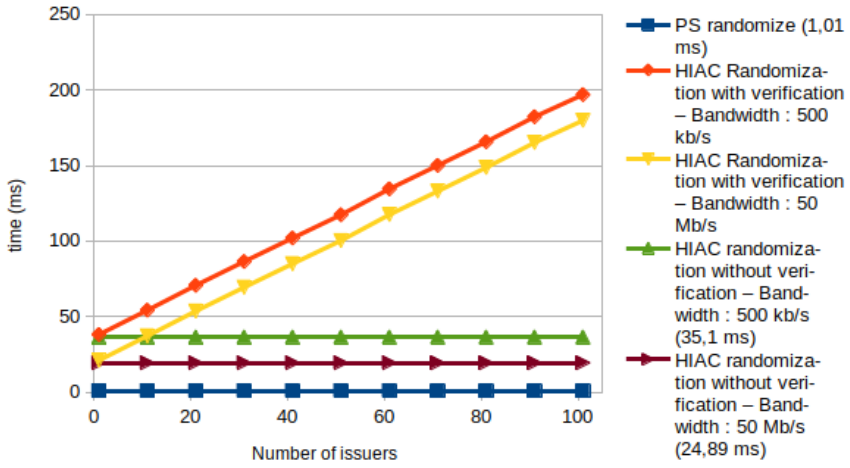


Figure 5.4 – Comparison of the running time to randomize a PS signature, and a HIAC Signature as a function of the number of issuers.

a few credentials, and thus this additional computation time remains negligible.

Signature Presentation. During the presentation of a signature, on the user side, both schemes run a randomization process. In the PS scheme, this randomization has a constant time complexity. With our scheme, there are two different cases. The first time the user interacts with the verifier, they must verify the integrity of the verifier's key. This verification has a linear time complexity. Each new transaction with the same verifier will rely on the first integrity verification, so new transactions only experience a constant time complexity for the user. Both scenarios depend on the available bandwidth to run the interactive protocol. Figure 5.4 compares these different cases with the PS randomization process.

On the verifier side, the verifier needs to compute the **VerifyRandomized** algorithm. The running time of this algorithm depends on network bandwidth. The comparison of the verification times of the PS scheme and of the HIAC scheme is presented in Figure 5.5. With a bandwidth of at least $2,5$ Mb/s, our verification algorithm is four times slower than that of PS.

General Remarks. The above analysis highlights that while the running time of most of the algorithms is longer than that of an equivalent Anonymous Credential scheme, interactive algorithms still exhibit efficient running times. In particular, the issuance and presentation of a credential only take tenths of milliseconds.

The only concern on the user side lies in the verification of the verifier's key and aggregator, which is linear in the number of trusted issuers, and which must be conducted with each unknown verifier. However, a verification of a subset of the verifier's aggregator may be sufficient if it is

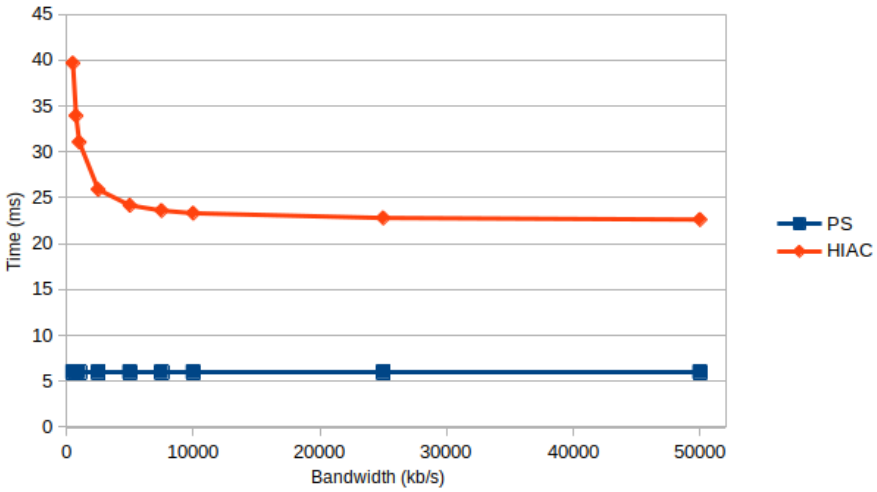


Figure 5.5 – Time to verify one credential with the PS scheme and the HIAC scheme, as a function of the Bandwidth.

enough to fulfill the requirements of the conformity checker discussed in Section 5.8.

From the point of view of issuers and verifiers, the overhead with respect to the PS scheme is lower (at most 5 times less efficient with a 2500 kb/s bandwidth). Thus the practicality of our scheme will depend on usage. If the verifier (respectively, issuer) needs to verify (respectively, issue) as many credentials per second as possible, the overhead becomes significant, but we expect our scheme to be used mostly in interactions that involve humans, which are less sensitive to slightly higher latency. Furthermore, in an SSI, the overall throughput in terms of transactions per second increases with the number of verifiers, unlike in centralized or blockchain-based services. In particular, with a bandwidth of 2500 kb/s, each new verifier added to the network adds a capacity of 38 transaction per second to the infrastructure. A service provider can thus easily add a server to augment its verification capacity.

Asymptotic Comparison

Table 5.4 [8] presents an asymptotic comparison of our signature scheme, with two other schemes. As mentioned, PS constitutes the basis which our scheme builds upon. The differences between our scheme and this building block highlight the real cost of issuer-indistinguishability. The recent Issuer-Hidden Attribute-Based Credential (IHABC) [26] achieves issuer-indistinguishability in a trusted-setup setting, using bilinear pairings, and Groth [103] signatures.

Table 5.4 compares the number of exponentiations and pairings required by our scheme, PS,

8. The BLS12-381 implementation we use gives us: 1 exponentiation in $\mathbb{G}_2 \approx 2$ exponentiation in $\mathbb{G}_1$ and 1 pairing operation ≈ 1.12 exponentiation in $\mathbb{G}_2$.

	HIAC	IHABC	PS
IssuerKeyGen	$3\mathbb{G}_1 + 2\mathbb{G}_2$	$1\mathbb{G}_2$	$2\mathbb{G}_2$
Sign (User Side)	$6\mathbb{G}_1$	$\emptyset$	$\emptyset$
Sign (Issuer Side)	$10\mathbb{G}_1$	$4\mathbb{G}_1 + 1\mathbb{G}_2$	$3\mathbb{G}_1$
VerifierSetup	$4k\mathbb{G}_1$	$(1+k)\mathbb{G}_1 + 4k\mathbb{G}_2$	$\emptyset$
IntegrityVerification	$4k\mathbb{G}_1$	$6k$ Pairing	$\emptyset$
Randomize	$26\mathbb{G}_1 + 2\mathbb{G}_2$	$10\mathbb{G}_1 + 6\mathbb{G}_2 + 6$ Pairing	$2\mathbb{G}_1$
VerifyRandomize	$20\mathbb{G}_1 + 2\mathbb{G}_2 + 8$ Pairing	$4\mathbb{G}_1 + 2\mathbb{G}_2 + 12$ Pairing	$1\mathbb{G}_2 + 2$ Pairing

Table 5.4 – Comparison of the number of exponentiations in $\mathbb{G}_1$, $\mathbb{G}_2$ and number of pairings required to run our scheme (HIAC)[26], the Issuer Hidden Attribute Based Credential (IHABC) without ZKP of knowledge signature, and the Pointcheval Sanders Signature Scheme (PS) [84], with k being the number of trusted issuers.

	HIAC	IHABC
IssuerKeyGen	9	2
Sign (Issuer Side)	10	6
VerifierSetup	40	90
IntegrityVerification	40	134.4
Randomize	30	35.44
VerifyRandomize	41.92	34.88

Table 5.5 – Comparison of the number of operations to run our scheme (HIAC), the Issuer Hidden Attribute Based Credential [26] (IHABC) without ZKP of signature. Operations in $\mathbb{G}_2$ and pairings operations are converted to $\mathbb{G}_1$. With 10 trusted issuers.

and IHABC, for the various operations. Table 5.5 expresses the asymptotic efficiency of our scheme, and the IHABC scheme in $\mathbb{G}_1$ equivalent operation, based on the approximation : 1 exponentiation in $\mathbb{G}_2 \approx 2$ exponentiation in $\mathbb{G}_1$ and 1 paring operation ≈ 1.12 exponentiation in $\mathbb{G}_2$.

The comparison with PS confirms the observations we made in Section 5.9.1 with respect to our implementation. The one with IHABC depends on whether we analyze the user, the verifer, or the issuer. On the user's side, the most frequent operations is **Randomize**. Albeit short, its running time is 15% faster for our scheme for 10 trusted issuers. Even if less frequent, the **IntegrityVerification** operation runs 3 times faster in our scheme than in IHABC.

On the verifier's side, the most frequent operation is **VerifyRandomized**. The IHABC implementation of this algorithm is 20% more efficient. In the interactive version we are considering, **VerifierSetup** is executed relatively rarely but it runs 4 times faster in our scheme than in IHABC. Finally, on the issuer's side, **Sign** runs twice as fast in IHABC as in our scheme, while **IssuerKeyGen** runs four times faster in IHABC. However, both operations are very quick to execute, so the difference remains very low in absolute value. Moreover, **IssuerKeyGen** is run only when the issuer starts or changes its keys. To conclude, our scheme offers better perfor-

mance for the user, and comparable or slightly worse performance for verifiers and issuers, but without requiring a trusted setup.

5.9.2 Communication Cost

This section studies the size and communication costs of our protocol, compared to the state of the art. The size of the data structures of our scheme are given in Table 5.6. With the BLS12-381 configuration [9], the elements of our signature scheme are relatively small. Except from the verifier key, they are comparable to a high or very-high strength RSA key (2048 - 4096 bits). The only large element to transfer is the verifier key, which contains multiple issuer keys, and the associated aggregator. However, if this key is static, users will save it in their device, and will only need to request it to a the verifier once. The size of the elements of the IHABC scheme are equivalent in most of the cases, except for the issuer public key, which is smaller in their case. Table 5.7 completes the picture by expressing the size of the elements of each signature scheme in bits, using the fact that, in BLS12-381 configuration, the size of the element are : $\mathbb{Z}_p$: 256 bits, $\mathbb{G}_1$: 384 bits, and $\mathbb{G}_2$: 768 bits. Table 5.8 uses the data from previous tables to estimate the communication costs of operations. This table is the source for Table 5.9.

	HIAC	IHABC	PS
Issuer public key	$3\mathbb{G}_1 + 2\mathbb{G}_2$	$1\mathbb{G}_2$	$2\mathbb{G}_2$
Issuer secret key	$2\mathbb{Z}_p$	$1\mathbb{Z}_p$	$2\mathbb{Z}_p$
Verifier public key	$2k\mathbb{Z}_k + 4k\mathbb{G}_1$	$(k+1)\mathbb{G}_1$ $+3k\mathbb{G}_2$	$\emptyset$
Verifier secret key	$2\mathbb{Z}_p$	$1\mathbb{Z}_p$	$\emptyset$
Signature	$1\mathbb{Z}_p + 4\mathbb{G}_1$	$2\mathbb{G}_1 + 1\mathbb{G}_2$	$2\mathbb{G}_1$
Randomized signature	$6\mathbb{G}_1 + 4\mathbb{G}_2$	$6\mathbb{Z}_p + 3\mathbb{G}_1$ $+4\mathbb{G}_2$	$2\mathbb{G}_1$

Table 5.6 – Asymptotic size of the keys and signatures of the HIAC scheme, PS scheme, and IHABC scheme.

Finally, we study the communication cost of our approach. Table 5.9 presents the communication costs expressed using the BLS12-381 group elements' size. The most expensive communication is related to the integrity-verification operation, in which the verifier needs to transfer its aggregators for verification by the user's device. Yet, the table shows that with 100 verifiers, this cost totals to only 25 kB. All other operations exhibit very low communication costs of less than 1 kB per transaction.

9. In the BLS12-381 configuration, the size of the element are : $\mathbb{Z}_p$: 256 bits, $\mathbb{G}_1$: 384 bits, and $\mathbb{G}_2$: 768 bits.

	HIAC (bits)	IHABC (bits)	PS (bits)
Issuer public key	2688	768	1536
Issuer secret key	512	256	512
Verifier public key	$256 \times 2k$ $+384 \times 6k$	$384 \times (k+1)$ $+768 \times 3k$	$\emptyset$
Verifier secret key	512	256	$\emptyset$
Signature	1792	1536	768
Randomized signature	4608	5760	768

Table 5.7 – Size of the keys and signatures of the HIAC scheme, PS scheme, and IHABC scheme, using BLS12-381 setup, with compressed elements.

	Issuer	User	Verifier
Sign	$4\mathbb{G}_1$	$2\mathbb{G}_1$	$\emptyset$
VerifierSetup	$4\mathbb{G}_1$	$\emptyset$	$\emptyset$
IntegrityVerification	$\emptyset$	$\emptyset$	$2k\mathbb{Z}_p + 4k\mathbb{G}_1$
Credential Presentation	$\emptyset$	$12\mathbb{G}_1 + 4\mathbb{G}_2$	$4\mathbb{G}_1$

Table 5.8 – Communication costs of each operation of the interactive HIAC scheme, for each issuers, users, and verifiers. Only upload costs are considerated. Credential presentation operation takes into account the transmission of the user's randomized credential and the commitment reveal exchange.

	Issuer	User	Verifier
Sign	1536	768	$\emptyset$
VerifierSetup	1536	$\emptyset$	$\emptyset$
IntegrityVerification	$\emptyset$	$\emptyset$	$2k \times 256 + 4k \times 384$
Credential Presentation	$\emptyset$	7680	1536

Table 5.9 – Communication costs of each operation of the interactive HIAC scheme, for each issuer, user, and verifier, expressed in bits, with k being the number of trusted issuers. Only upload costs are considerated, round trip times are not taken into account. The Credential Presentation operation takes into account the transmission of the user's randomized credential and the commitment reveal exchange.

5.10 Qualitative Comparison

We now consider the security properties of HIAC in comparison with those of other schemes as summarized in Table 5.10. The first property is issuer indistinguishability (hidden issuer). Apart from our scheme, only one other achieves it, Issuer Hidden Attribute Based Credential [26], but using a trusted setup. The second property is the non-transferability of credentials. Our scheme achieves it with the transformation presented in Appendix B. Camenisch and Lysyanskaya (CL01) [34] also propose a signature scheme with such a property. The third property to

analyze is the ability to sign a committed message (One-show). Our scheme achieves it in the same way as the PS scheme. Numerous signature schemes, like CL01 or CL04 [94], also have this property.

The fourth property relates to the fact that most AC signature schemes offer a way to prove the possession of a credential without revealing the associated message. Our scheme could implement this by adapting the zero-knowledge proof of a signature used by PS. However, the prover would need to prove a relationship between a message and its hash. Whereas this can be proven in zero knowledge, any implementation would be inefficient, because hash functions are based on boolean circuits, for which zero knowledge proofs are known to be inefficient. To achieve this property efficiently, we could replace the hash function with a low-degree polynomial fulfilling some conditions. We leave this as future work. Another way to achieve this property would be to use Groth's signature like IHABC, but this would come at the cost of a trusted setup as discussed in Section 5.3.

	HIAC	CL01	CL04	PS	IHABC
Hidden issuer	✓				✓
Non transferable	✓	✓			
One-show	✓	✓	✓		
ZKP of signature		✓	✓	✓	✓
Revocation			✓		✓

Table 5.10 – Qualitative comparison of different Anonymous Credential scheme. (HIAC: Hidden Issuer Anonymous Credential; CL01: hidden size group based Anonymous Credential scheme [34] ; CL04: bilinear pairing based Anonymous Credential scheme [94]; PS: Short randomizable Anonymous Credential scheme [84]; IHABC: Issuer Hidden Attribute Based Credential [26])

The fifth and last property often implemented by AC schemes is credential revocation. Designing a revocation mechanism while keeping issuer indistinguishability is non trivial. The most efficient way to achieve it in our scheme would be for the issuers to regularly publish a revocation accumulator, accumulating all revoked credentials. Users would later leverage this accumulator to prove that their credentials do not appear in the revocation list, as suggested by Camenisch et al. [110]. However, this can be rather complicated to achieve in a hidden-issuer context, and should be the subject of further research.

5.11 Conclusion and evolutions

In this chapter, we provided a formal definition of an Anonymous Credential scheme that hides the issuer of a credential inside the set of issuers trusted by a verifier. We gave an instantiation of this scheme, based on a new primitive called aggregator, and a modified Pointcheval Sanders signature scheme. This new Hidden Issuer Anonymous Credential (HIAC) enhances the

minimization principle of SSIs, and it improves the collusion resistance of Anonymous Credential schemes. It achieves EUF-CMA, unlinkability, issuer indistinguishability, and the trusted-issuer property; and it does not require a trusted-setup. The aggregator primitive can be used in other issuer-indistinguishable signature schemes, and its instantiation is interoperable with state-of-the-art signatures.

The work presented in this chapter was published in 2022, and since then, it has led to multiple evolutions that proposed more efficient schemes to enable the issuer's indistinguishability property. We can cite a work by Mir et al. [111] and a work by Sanders and Traoré [31]. When we studied the issuer indistinguishability property for Anonymous Credentials and published this paper at PETs 2022, we were the first (concurrently with IHABC [26]) to tackle this problem. Therefore, many efficiency problems were not addressed by our work nor by the work of Bobolz et al. [26]. If an implementer needs the issuer's indistinguishability property, we encourage them to choose one of the schemes that was proposed later. In that sense, the scheme proposed by Sanders and Traoré is particularly interesting as it also uses the PS signature scheme. The difference with the work presented in the present chapter is that their scheme modifies the PS scheme in a more controlled way, thus leading to a more efficient HIAC scheme where the size of the scheme's elements is significantly decreased. Moreover, unlike the scheme presented here or the IHABC scheme [26], the Sanders and Traoré scheme does not require the use of Zero Knowledge Proofs. Not only does it improve efficiency, but it also makes it possible to prove the security of their scheme in the standard model rather than in the Random Oracle Model.

To conclude, the issuer's indistinguishability property is essential for building a PPfDIMS. We presented the first (concurrently with Bobolz et al. [26]) scheme anonymous credential scheme that supports issuer indistinguishability. This work inspired others to build more efficient schemes. In Chapter 10, we use one of these evolutions proposed by Sanders and Traoré [31].

Chapter 6

Synchronization requirements for revocation, access control, and multi-device capability

This chapter formally studies AllowLists and DenyLists as distributed objects. In this chapter, those two objects are defined in the shared memory model. Then, their consensus number is analyzed. Those objects are the core of auxiliary features of PPfDIMSs. Hence, this analyses is used in Chapters 9 and 10 to build theoretically optimal (in term of synchronization requirements) auxiliary features for PPfDIMSs. It was written in collaboration with Davide Frey and Michel Raynal. It was published in the international Symposium on Distributed Computing (DISC) conference in 2023 [21].

6.1 Introduction

Chapter 3 studied different implementations of DIMSs. As we saw, most of them use a consensus algorithm or a blockchain as their main distributed component. However, this use of the blockchain as a swiss-army knife that can solve numerous distributed problems highlights a lack of understanding of the actual requirements of those problems. Because of these poor specifications, implementations of these applications are often sub-optimal.

This chapter thoroughly studies a class of problems widely used in distributed applications and provides a guideline to implement them with reasonable but sufficient tools.

Differently from the previous approaches, it aims to understand the amount of synchronization required between processes of a system to implement *specific* distributed objects. To achieve this goal it studies such objects under the lens of Herlihy's consensus number [27]. This parameter is inherently associated to shared memory distributed objects, and has no direct correspondence in the message passing environment. However, in some specific cases, this information is enough to provide a better understanding of the objects analyzed, and thus, to gain efficiency in the message passing implementations. For example, recent papers [112, 24] have shown that cryptocurrencies can be implemented without consensus and therefore without a blockchain. In

particular, Guerraoui et al. [112] show that k-asset transfer has a consensus number k where k is the number of processes that can withdraw currency from the same account [27]. Similarly, Alpos et al. [113] have studied the synchronization properties of ERC20 token smart contracts and shown that their consensus number varies over time as a result of changes in the set of processes that are approved to send tokens from the same account. These two results consider two forms of asset transfer: the classical one and the one implemented by the ERC20 token, which allows processes to dynamically authorize other processes. The consensus number of those objects depends on specific and well identified processes. From this study, it is possible to conclude that the consensus algorithms only need to be performed between those processes. Therefore, in these specific cases, the knowledge of the consensus number of an object can be directly used to implement more efficient message passing applications. Furthermore, even if this study uses a shared memory model, with crash prone processes, its results can be used to implement more efficient Byzantine resilient algorithm, in a message passing environment. This chapter proposes to extend this knowledge to a broader class of applications.

Indeed, the transfer of assets, be them cryptocurrencies or non-fungible tokens, does not constitute the only application in the Blockchain ecosystem. In particular, as previously indicated, a number of applications like e-voting [114], naming [115, 116], or Identity Management [52, 7] use Blockchain as a tool to implement some form of access control. This is often achieved by implementing two general-purpose objects: AllowLists and DenyLists. An AllowList provides an opt-in mechanism. A set of managers can maintain a list of authorized parties, namely the AllowList. To access a resource, a party (user) must prove the presence of an element associated with its identity in the AllowList. A DenyList provides instead an opt-out mechanism. In this case, the managers maintain a list of revoked elements, the DenyList. To access a resource, a party (user) must prove that no corresponding element has been added to the DenyList. In other words, AllowList and DenyList support, respectively, set-membership and set-non-membership proofs on a list of elements.

The proofs carried out by AllowList and DenyList objects often need to offer privacy guarantees. For example, the Sovrin privacy preserving Decentralized Identity-Management System (DIMS) [52] associates an AllowList [1] with each verifiable credential that contains the identifiers of the devices that can use this verifiable credential. When a device uses a credential with a verifier, it needs to prove that the identifier associated with it belongs to the AllowList. This proof must be done in zero knowledge, otherwise the verifier would learn the identity of the device, which in turn could serve as a pseudo-identifier for the user. For this reason, AllowList and DenyList objects support respectively a zero-knowledge proof of set membership or a zero-knowledge proof of set non-membership.

Albeit similar, the AllowList and DenyList objects differ significantly in the way they handle

1. In reality this is a variant that mixes AllowList and DenyList which we discuss in section 6.8.

the proving mechanism. In the case of an AllowList, no security risk appears if access to a resource is prohibited to a process, even if a manager did grant this right. As a result, a transient period in which a user is first allowed, then denied, and then allowed again to access a resource poses no problem. On the contrary, with a DenyList, being allowed access to a resource after being denied poses serious security problems. Hence, the DenyList object is defined with an additional anti-flickering property prohibiting those transient periods. This property is the main difference between an AllowList and a DenyList object and is the reason for their distinct consensus numbers.

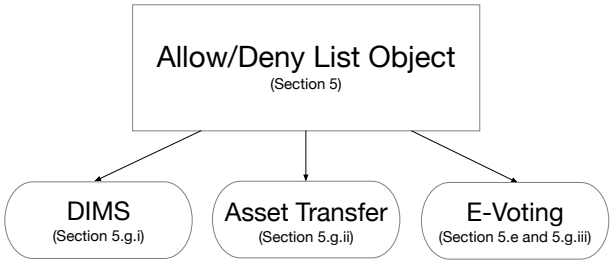
Existing systems [114, 115, 116, 52, 7] that employ AllowList and DenyList objects implement them on top of a heavy blockchain infrastructure, thereby requiring network-level consensus to modify their content. As already said, this chapter studies this difference under the lens of the consensus number [27]. It shows that (i) the consensus number of an AllowList object is 1, which means that an AllowList can be implemented without consensus; and that (ii) the consensus number of a DenyList is instead equal to the number of processes that can conduct prove operations on the DenyList, and that only these processes need to synchronize. Both data structures can therefore be implemented without relying on the network-level consensus provided by a blockchain, which opens the door to more efficient implementations of applications based on these data structures.

To summarize, this chapter presents the following three contributions.

1. It formally defines and studies AllowList and DenyList as distributed objects (Section 6.4).

2. It analyses the consensus number of these objects: it shows that the AllowList does not require synchronization between processes (Section 6.6), while the DenyList requires the synchronization of all the verifiers of its set-non-membership proofs (Section 6.7).

3. It uses these theoretical results to give intuitions on their optimal implementations. Namely the implementation of a DIMS, as well as of an e-vote system and an anonymous asset-transfer protocol (Section 6.9).



6.2 Related Works

Even though distributed consensus algorithms were already largely studied [117, 118, 119, 25], the rise of Ethereum—and the possibilities offered by its versatile smart contracts—led to new ideas to decentralized already known applications. Among those, e-vote and DIMS [91] are two examples.

Blockchains increased the interest in distributed versions of already existing algorithms. However, these systems are usually developed with little concern for the underlying theoretical basis they rely on. A great example is trustless money transfer protocols or crypto money. The underlying distributed asset-transfer object was never studied until recently. A theoretical study proved that a secure asset-transfer protocol does not need synchronicity between network nodes [112]. Prior to this work, all proposed schemes used a consensus protocol, which cannot be deterministically implemented in an asynchronous network [22]. The result is that many existing protocols could be replaced by more efficient, Bizantine Reliable Broadcast [25] based algorithms. This work leads to more efficient implementation proposal for money transfer protocol [24]. Alpos et al. then extended this study to the Ethereum ERC20 smart contracts [113]. This last paper focuses on the asset-transfer capability of smart contracts. Furthermore, the object described has a dynamic consensus number, which depends on the processes authorized to transfer money from a given account. Furthermore, this work and the one from Guerraoui et al. [112] both analyze a specific object that is not meant to be used to find the consensus number of other applications. In contrast, our work aims to be used as a generic tool to find the consensus number of numerous systems.

A recent paper published after the present work was presented at DISC 2023 studies the consensus number of Auditable Read/Write register objects. Those objects have a behavior similar to one of the DenyList objects, and their consensus number is the same. However, it is unclear how those two objects are related, and further evaluation should be conducted to determine what similarities exist and to what extent those objects can be formalized in an even more general object.

6.3 Model

The communication model used in this chapter is the shared memory model presented in Section 4.1.1.

In the following, it is also assumed there exists dynamic accumulators Acc that supports efficient ZKP of set-membership and set-non-membership. Such ZKP systems have been proposed for proof of set-membership and proof of set-non-membership [120].

6.4 The AllowList and DenyList objects: Definition

Distributed AllowList and DenyList object types are the type of objects that allow a set of managers to control access to a resource. The term "resource" is used here to describe the goal a user wants to achieve and which is protected by an access control policy. A user is granted access to the resource if it succeeds in proving that it is authorized to access it. First, we describe the AllowList object type. Then we consider the DenyList object type.

The AllowList object type is one of the two most common access control mechanisms. To access a resource, a process $p \in \Pi_V$ needs to prove it knows some element v previously authorized by a process $p_M \in \Pi_M$, where $\Pi_M \subseteq \Pi$ is the set of managers, and $\Pi_V \subseteq \Pi$ is the set of processes authorized to conduct proofs. We call verifiers the processes in Π_V. The sets Π_V and Π_M are predefined and static. They are parameters of the object. Depending on the usage of the object, these subset can either be small, or they can contain all the processes in Π.

A process $p \in \Pi_V$ proves that v was previously authorized by invoking a Prove(v) operation. This operation is said to be valid if some manager in Π_M previously invoked an Append(v) operation. Intuitively, we can see the invocation of the Append(v) operation as the action of authorizing some process to access the resource. On the other hand, the Prove(v) operation, performed by a prover process, $p \in \Pi_V$, proves to the other processes in Π_V that they are authorized. However, this proof is not enough in itself. The verifiers of a proof must be able to verify that a valid Prove operation has been invoked. To this end, the AllowList object type is also equipped with a Read() operation. This operation can be invoked by any process in Π and returns all the valid Prove operations invoked, along with the identity of the processes that invoked them. The list returned by the Read operation can be any arbitrary permutation of the list of Prove operations. All processes in Π can invoke the Read operation. [2]

An optional anonymity property can be added to the AllowList object to enable privacy-preserving implementations. This property ensures that other processes cannot learn the value v proven by a Prove(v) operation.

The AllowList object type is formally defined as a sequential object, where each invocation is immediately followed by a response. Hence, the sequence of operations defines a total order, and each operation can be identified by its place in the sequence.

Definition 6.1. The *AllowList* object type supports three operations: Append, Prove, and Read. These operations appear as if executed in a sequence Seq such that:

— *Termination.* A Prove, an Append, or a Read operation invoked by a correct process always returns.

— Append *Validity.* The invocation of Append(x) by a process p is valid if:

2. Usually, AllowList objects are implemented in a message-passing setting. In these cases, the Read operation is implicit. Each process knows a local state of the distributed object, and can inspect it any time. In the shared-memory setting, we need to make this Read operation explicit.

  - $p \in \Pi_M \subseteq \Pi$; and
  - $x \in \mathcal{S}$, where $\mathcal{S}$ is a predefined set.

  Otherwise, the operation is invalid.
- Prove *Validity.* If the invocation of $op = \mathsf{Prove}(x)$ by a process p is valid, then:
  - $p \in \Pi_V \subseteq \Pi$; and
  - A valid $\mathsf{Append}(x)$ operation appears before op in Seq.

  Otherwise, the invocation is invalid.
- *Progress.* If a valid $\mathsf{Append}(x)$ operation is invoked, then there exists a point in Seq such that any $\mathsf{Prove}(x)$ operation invoked after this point by any process $p \in \Pi_V$ will be valid.
- Read *Validity.* The invocation of $op = \mathsf{Read}()$ by a process $p \in \Pi_V$ returns the list of valid invocations of Prove that appears before op in Seq along with the names of the processes that invoked each operation.
- *Optional - Anonymity.* Let us assume the process p invokes a $\mathsf{Prove}(v)$ operation. If the process p' invokes a $\mathsf{Read}()$ operation, then p' cannot learn the value v unless p leaks additional information. [3]

The AllowList object is defined in an append-only manner. This definition makes it possible to use it to build all use cases explored in this chapter. However, some use cases could need an DenyList with an additional Remove operation. This variation is studied in Section 6.8.

The DenyList object type can be informally presented as an access policy where, contrary to the AllowList object type, all users are authorized to access the resource in the first place. The managers are here to revoke this authorization. A manager revokes a user by invoking the $\mathsf{Append}(v)$ operation. A user uses the $\mathsf{Prove}(v)$ operation to prove that it was not revoked. A $\mathsf{Prove}(v)$ invocation is invalid only if a manager previously revoked the value v.

All the processes in Π can verify the validity of a Prove operation by invoking a Read () operation. This operation is similar to the AllowList's Read operation. It returns the list of valid Prove invocations along with the name of the processes that invoked it.

There is one significant difference between the DenyList and the AllowList object types. With an AllowList, if a user cannot access a resource immediately after its authorization, no malicious behavior can harm the system—the system's state is equivalent to its previous state. However, with a DenyList, a revocation not taken into account can let a malicious user access the resource and harm the system. In other words, access to the resource in the DenyList case must take into account the "most up to date" available revocation list.

To this end, the DenyList object type is defined with an additional property. The anti-

3. The Anonymity property only protects the value v. The system considered is eponymous. Hence, the identity of the processes is already known. However, the anonymity of v makes it possible to hide other information. For example, the identity of a client that issues a request to a process of the system. These example are discussed in Section 6.9. Thereby, the anonymity property does not contravene the Read validity property, which only discloses the process identity.

flickering property ensures that if an Append operation is taken into account by one Prove operation, it will be taken into account by every subsequent Prove operation. Along with the progress property, the anti-flickering property ensures that the revocation mechanism is as immediate as possible. The DenyList object is formally defined as a sequential object, where each invocation is immediately followed by a response. Hence, the sequence of operations define a total order, and each operation can be identified by its place in the sequence.

Definition 6.2. The *DenyList* object type supports three operations: Append, Prove, and Read. These operations appear as if executed in a sequence Seq such that:

- *Termination.* A Prove, an Append, or a Read operation invoked by a correct process always returns.
- Append *Validity.* The invocation of Append(x) by a process p is valid if:
  - $p \in \Pi_M \subseteq \Pi$; and
  - $x \in \mathcal{S}$, where $\mathcal{S}$ is a predefined set.

  Otherwise, the operation is invalid.
- Prove *Validity.* If the invocation of a $op =$ Prove(x) by a correct process p is not valid, then:
  - $p \notin \Pi_V \subseteq \Pi$; or
  - A valid Append(x) appears before op_P in Seq.

  Otherwise, the operation is valid.
- Prove *Anti-Flickering.* If the invocation of a operation $op =$ Prove(x) by a correct process $p \in \Pi_V$ is invalid, then any Prove(x) operation that appears after op in Seq is invalid. [4]
- Read *Validity.* The invocation of $op =$ Read() by a process $p \in \Pi_V$ returns the list of valid invocations of Prove that appears before op in Seq along with the names of the processes that invoked each operation.
- *Optional - Anonymity.* Let us assume the process p invokes a Prove(v) operation. If the process p' invokes a Read() operation, then p' cannot learn the value v unless p leaks additional information.

6.5 PROOF-LIST object specification

Section 6.6 and Section 6.7 propose an analysis of the synchronization power of the AllowList and the DenyList object types using the notion of consensus number. Both objects share many similarities. Indeed, the only difference is the type of proof performed by the user and the

4. The only difference between the AllowList and the DenyList object types is this anti-flickering property. As it is shown in Section 6.6 and in Section 6.7, the AllowList object has consensus number 1, and the DenyList object has consensus number $k = |\Pi_V|$. Hence, this difference in term of consensus number is due solely to the anti-flickering property. It is an open question whether a variation of this property could transform any consensus number 1 object into a consensus number k object.

Process	Operation	Initial state	Response	Final state	Conditions
$p_i \in \Pi_M$	$\mathsf{Append}(y)$	$(\textit{listed-values} = \{x \in \mathcal{S}\},$ $\textit{proofs} = (\{(p_j \in \Pi, \widehat{\mathcal{S}} \subseteq \mathcal{S}, \mathsf{P} \in \mathcal{P}_{\mathcal{L}_{\widehat{\mathcal{S}}}})\}))$	`true`	$(\textit{listed-values} \cup \{y\},$ $\textit{proofs})$	$y \in \mathcal{S}$
p_i	$\mathsf{Append}(y)$	$(\textit{listed-values} = \{x \in \mathcal{S}\},$ $\textit{proofs} = (\{(p_j \in \Pi, \widehat{\mathcal{S}} \subseteq \mathcal{S}, \mathsf{P} \in \mathcal{P}_{\mathcal{L}_{\widehat{\mathcal{S}}}})\}))$	`false`	$(\textit{listed-values}, \textit{proofs})$	$p_i \notin \Pi_M \vee y \notin \mathcal{S}$
$p_i \in \Pi_V$	$\mathsf{Prove}(y)$	$(\textit{listed-values} = \{x \in \mathcal{S}\},$ $\textit{proofs} = (\{(p_j \in \Pi, \widehat{\mathcal{S}} \subseteq \mathcal{S}, \mathsf{P} \in \mathcal{P}_{\mathcal{L}_{\widehat{\mathcal{S}}}})\}))$	$(\mathcal{A}, \mathsf{P})$	$(\textit{listed-values},$ $\textit{proofs} \cup \{(p_i, \mathcal{A}, \mathsf{P})\})$	$\forall y \in \mathcal{L}_{\mathcal{A}} \wedge \mathcal{A} \subseteq \textit{listed-values}$ $\wedge \forall \mathsf{P} \in \mathcal{P}_{\mathcal{L}_{\mathcal{A}}} \wedge \mathsf{C}(y, \widehat{\mathcal{S}}) = 1$
p_i	$\mathsf{Prove}(y)$	$(\textit{listed-values} = \{x \in \mathcal{S}\},$ $\textit{proofs} = (\{(p_j \in \Pi, \widehat{\mathcal{S}} \subseteq \mathcal{S}, \mathsf{P} \in \mathcal{P}_{\mathcal{L}_{\widehat{\mathcal{S}}}})\}))$	`false`	$(\textit{listed-values}, \textit{proofs})$	$\forall y \notin \mathcal{L}_{\mathcal{A}} \vee \mathcal{A} \not\subseteq \textit{listed-values}$ $\vee \forall \mathsf{P} \notin \mathcal{P}_{\mathcal{L}_{\mathcal{A}}} \vee \forall p_i \notin \Pi_V$ $\vee \mathsf{C}(y, \widehat{\mathcal{S}}) = 0$
$p_i \in \Pi$	$\mathsf{Read}()$	$(\textit{listed-values} = \{x \in \mathcal{S}\},$ $\textit{proofs} = (\{(p_j \in \Pi, \widehat{\mathcal{S}} \subseteq \mathcal{S}, \mathsf{P} \in \mathcal{P}_{\mathcal{L}_{\widehat{\mathcal{S}}}})\}))$	*proofs*	$(\textit{listed-values}, \textit{proofs})$	

Table 6.1 – Transition function Δ for the PROOF-LIST object.

non-flickering properties. Therefore, this section defines the formal specification of the PROOF-LIST object type, a new generic object that can be instantiated to describe the AllowList or the DenyList object type.

The PROOF-LIST object type is a distributed object type whose state is a pair of arrays (*listed-values*, *proofs*). The first array, *listed-values*, represents the list of authorized/revoked elements. It is an array of objects in a set $\mathcal{S}$, where $\mathcal{S}$ is the universe of potential elements. The second array, *proofs*, is a list of assertions about the *listed-values* array. Given a set of managers $\Pi_M \subseteq \Pi$ and a set of verifiers $\Pi_V \subseteq \Pi$, the PROOF-LIST object supports three operations. First, the $\mathsf{Append}(v)$ operation appends a value $v \in \mathcal{S}$ to the *listed-values* array. Any process in the manager's set can invoke this operation. Second, the $\mathsf{Prove}(v)$ operation appends a valid proof about the element $v \in \mathcal{S}$ relative to the *listed-values* array to the *proofs* array. This operation can be invoked by any process $p \in \Pi_V$. Third, the $\mathsf{Read}()$ operation returns the *proofs* array.

The sets Π_V and Π_M are static, predefined subsets of Π. There is no restriction on their compositions. The choice of these sets only depends on the usage of the AllowList or the DenyList. Depending on the usage, they can either contain a small subset of processes in Π or they can contain the whole set of processes of the system.

To express the proofs produced by a process p, we use an abstract language $\mathcal{L}_{\mathcal{A}}$ of the complexity class $\mathcal{NP}$, which depends on a set $\mathcal{A}$. This language will be specified for the AllowList and the DenyList objects in Section 6.6 and Section 6.7. The idea is that p produces a proof π about a value $v \in \mathcal{S}$. A Prove invocation by a process p is valid only if the proof π added to the *proofs* array is valid. The proof π is valid if $v \in \mathcal{L}_{\mathcal{A}}$—i.e., v is a solution to the instance of the problem expressed by $\mathcal{L}_{\mathcal{A}}$, where $\mathcal{L}_{\mathcal{A}}$ is a language of the complexity class $\mathcal{NP}$ [5] which depends on a subset $\mathcal{A}$ of the *listed-values* array ($\mathcal{A} \subseteq \mathcal{S}$). We note $\mathcal{P}_{\mathcal{L}_{\mathcal{A}}}$ the set of valid proofs relative to the language $\mathcal{L}_{\mathcal{A}}$. $\mathcal{P}_{\mathcal{L}_{\mathcal{A}}}$ can either represent Zero Knowledge Proofs or explicit proofs.

5. In this article, $\mathcal{L}_{\mathcal{A}}$ can be one of the following languages: a value v belongs to $\mathcal{A}$ (AllowList), or a value v does not belongs to $\mathcal{A}$ (DenyList).

If a proof π is valid, then the Prove operation returns $(\mathcal{A}, \mathsf{Acc}.Prove(v, \mathcal{A}))$, where $\mathsf{Acc}.Prove(v, \mathcal{A})$ is the proof generated by the operation, and where $\mathcal{A}$ is a subset of values in *listed-values* on which the proof was applied. Otherwise, the Prove operation returns "`false`". Furthermore, the *proofs* array also stores the name of the processes that invoked Prove operations.

Formally, the PROOF-LIST object type is defined by the tuple (Q, Q_0, O, R, Δ), where:

- The set of valid state is $Q = (\textit{listed-values} = \{x \in \mathcal{S}\}, \textit{proofs} = \{(p \in \Pi, \widehat{\mathcal{S}} \subseteq \mathcal{S}, \mathsf{P} \in \mathcal{P}_{\mathcal{L}_{\widehat{\mathcal{S}}}})\})$, where *listed-values* is a subset of $\mathcal{S}$ and *proofs* is a set of tuples. Each tuple in *proofs* consists of a proof associated with the set it applies to and to the identifier of the process that issued the proof;
- The set of valid initial states is $Q_0 = (\emptyset, \emptyset)$, the state where the *listed-values* and the *proofs* arrays are empty;
- The set of possible operation is $O = \{\mathsf{Append}(x), \mathsf{Prove}(y), \mathsf{Read}()\}$, with $x, y \in \mathcal{S}$;
- The set of possible responses is $R = \left\{\texttt{true}, \texttt{false}, (\widehat{\mathcal{S}} \subseteq \mathcal{S}, \mathsf{P} \in \mathcal{P}_{\mathcal{L}_{\widehat{\mathcal{S}}}}), \{(p \in \Pi, \widehat{\mathcal{S}}' \subseteq \mathcal{S}, \mathsf{P}' \in \mathcal{P}_{\mathcal{L}_{\widehat{\mathcal{S}}}})\}\right\}$, where `true` is the response to a successful Append operation, $(\widehat{\mathcal{S}}, \mathsf{P})$ is the response to a successful Prove operation, $\{(p, \widehat{\mathcal{S}}', \mathsf{P}')\}$ is the response to a Read operation, and `false` is the response to a failed operation; and
- The transition function is Δ. The PROOF-LIST object type supports 5 possible transitions. We define the 5 possible transitions of Δ in Table 6.1.

The first transition of the Δ function models a valid Append invocation, a value $y \in \mathcal{S}$ is added to the *listed-values* array by a process in the managers' set Π_M. The second transition of the Δ function represents a failed Append invocation. Either the process p_i that invokes this function is not authorized to modify the *listed-values* array, i.e., $p_i \notin \Pi_M$, or the value it tries to append is invalid, i.e., $y \notin \mathcal{S}$. The third transition of the Δ function captures a valid Prove operation, where a valid proof is added to the *proofs* array. The function C will be used to express the anti-flickering property of the DenyList implementation. It is a boolean function that outputs either 0 or 1.The fourth transition of the Δ function represents an invalid Prove invocation. Either the proof is invalid, or the set on which the proof is issued is not a subset of the *listed-values* array. Finally, the fifth transition represents a Read operation. It returns the *proofs* array and does not modify the object's state.

The language $\mathcal{L}_{\mathcal{A}}$ does not directly depend on the *listed-values* array. Hence, the validity of a Prove operation will depend on the choice of the set $\mathcal{A}$.

6.6 The consensus number of the AllowList object

This section provides an AllowList object specification based on the PROOF-LIST object. The specification is then used to analyze the consensus number of the object type.

We provide a specification of the AllowList object defined as a PROOF-LIST object, where $\mathsf{C}(y, \widehat{\mathcal{S}}) = 1$ and

$$\forall y \in \mathcal{S}, y \in \mathcal{L}_{\mathcal{A}} \Leftrightarrow (\mathcal{A} \subseteq \mathcal{S} \wedge y \in \mathcal{A}). \tag{6.1}$$

In other words, y belongs to a set $\mathcal{A}$. Using the third transition of the Δ function, we can see that $\mathcal{A}$ should also be a subset of the *listed-values* array. Hence, this specification supports proofs of set-membership in *listed-values*. A PROOF-LIST object defined for such language follows the specification of the AllowList. To support this statement, we provide an implementation of the object.

To implement the AllowList object, Algorithm 1 uses two Atomic Snapshot objects. The first one represents the *listed-values* array, and the second represents the *proofs* array. These objects are arrays of N entries. Furthermore, we use a function "Proof" that on input of a set $\mathcal{S}$ and an element y outputs a proof that $y \in$ *listed-values*. This function is used as a black box, and can either output an explicit proof—an explicit proof can be the tuple $(y, \mathcal{A})$, where $\mathcal{A} \subseteq$ *listed-values*—or a Zero Knowledge Proof.

Shared variables
AS-LV ← N-dimensions Atomic-Snapshot object, initially $\{\emptyset\}^N$;
AS-PROOF ← N-dimensions Atomic-Snapshot object, initially $\{\emptyset\}^N$;

Operation Append(v) **is**
1: **If** $(v \in \mathcal{S}) \wedge (p \in \Pi_M)$ **then**
2: local-values ← AS-LV.Snapshot()[p];
3: AS-LV.Update(local-values ∪ v, p);
4: **Return** `true`;
5: **Else return** `false`;

Operation Read () **is**
6: **Return** AS-PROOF.Snapshot();

Operation Prove(v) **is**
7: **If** $p \notin \Pi_V$ **then**
8: **Return** `false`;
9: $\mathcal{A} \leftarrow$ AS-LV.Snapshot();
10: **If** $v \in \mathcal{A}$ **then**
11: $\pi_{\text{set-memb}} \leftarrow$ Proof($v \in \mathcal{A}$);
12: proofs ← AS-PROOF.Snapshot()[p];
13: AS-PROOF.Update(proofs ∪ $(p, \mathcal{A}, \pi_{\text{set-memb}})$, p);
14: **Return** $(\mathcal{A}, \pi_{\text{set-memb}})$;
15: **Else return** `false`.

Algorithm 1: Implementation of an AllowList object using Atomic-Snapshot objects

Theorem 6.1. Algorithm 1 wait-free implements an AllowList object.

Proof. Let us fix an execution E of the algorithm presented in algorithm 1. Each invocation

is a sequence of a finite number of local operations and Atomic-Snapshot accesses. Because the Atomic Snapshot primitive can be wait-free implemented in the read-write shared memory model, each correct process terminates each invocation in a finite number of its own steps.

Let H be the history of the execution E. We define $\bar{H}$, the completed history of H. Any invocation in H can be completed in $\bar{H}$. We give the completed history $\bar{H}$ of H:

— Any invocation of the Append operation that did not reach line 3 can be completed with the line "**Return** `false`";
— Any invocation of the Prove operation that did not reach line 13 can be completed with the line "**Return** `false`";
— Any invocation of the Append operation that reached line 3 can be completed with line 4; and
— Any invocation of the Prove operation that reached line 13 can be completed with line 14.

The linearization points of the Append, Prove and Read operations are respectively line 3, line 13 and line 6. For convenience, We call any operation in $\bar{H}$ that returns "`false`" an invalid operation. We verify that each operation in $\bar{H}$ respects the specification:

— Any operation in $\bar{H}$ run by a process p that is invalid is an operation that only modifies the internal state of p and that was invoked by a faulty process or that was invoked by a process without the write to invoke the operation. Therefore, these invalid operations do not impact the validity and the progress properties of the AllowList object.
— If an Append operation invoked by a process p in $\bar{H}$ returns "`true`", it implies that p reached line 3. Therefore p appended a value v to the array *listed-values* at the index p. Process p is the only process able to write at this index. Because the Update operation is atomic, and because p is the only process able to write in AS-LV[p], the *listed-values* array append-only property is preserved. Furthermore, the element added to *listed-value* belongs to the set $\mathcal{S}$, and the process that appends the value belongs to the set of managers Π_M. Therefore, any invocation of the Append operation in $\bar{H}$ that returns "`true`" fulfills the Append validity property. Hence, any Append invocation in $\bar{H}$ follows the AllowList specification.
— If an invocation of the Prove operation by a process p in $\bar{H}$ returns $(\mathcal{A}, \pi)$, then $p \in \Pi_V$ reached line 13. Therefore, p appended a proof π to the *proofs* array at the index p, and the proof is a valid proof that $v \in \mathcal{A}$. Process p is the only process allowed to modify the *proofs* array at this index. There is no concurrency on the write operation. Furthermore, the set $\mathcal{A}$, is a subset of the AS-LV array (line 9). Because the only way to add an element to the AS-LV array is via an Append operation, because we consider the linearization point of the Prove operation to be at line 13, the Prove validity property is ensured. The progress property is ensured thanks to the atomicity of the Atomic Snapshot object. If

some process executes line 3 of the Append operation at time t_1, then any correct process that reaches line 8 of the Prove(x) operation at time $t_2 > t_1$ will be valid. Hence, any Prove invocation in $\bar{H}$ follows the AllowList specification.

— A Read operation always returns the values of the AS-PROOF array that were linearized before the execution of line 6, thanks to the atomicity of the Atomic Snapshot object. Furthermore, the returned value is always a set of successful Prove operations (AS-PROOF). This set ois compounded of proofs associated to the name of the process that invoked the operation. Therefore, the Read validity property is ensured. Hence, any Read invocation in $\bar{H}$ follows the AllowList specification.

All operations in $\bar{H}$ follow the AllowList specification. Thus, $\bar{H}$ is a legal history of the AllowList object type, and H is linearizable. To conclude, the algorithm presented in algorithm 1 is a wait-free implementation of the AllowList object type. □

Corollary 6.1. The consensus number of the AllowList object type is 1.

6.7 The consensus number of the DenyList object

In the following, we propose two wait-free implementations establishing the consensus number of the DenyList object type. In this section and in the following, we refer to a DenyList with $|\Pi_V| = k$ as a k-DenyList object. This analysis of this parameter k is the core of the study conducted here. Because it is a statically defined parameter, the knowledge of this parameter can improve efficiency of DenyList implementation by reducing the number of processes that need to synchronize in order to conduct a proof.

6.7.1 Lower bound

Algorithm 2 presents an implementation of a k-consensus object using a k-DenyList object with $\Pi_M = \Pi_V = \Pi$, and $|\Pi| = k$. It uses an Atomic Snapshot object, AS-LIST, to allow processes to propose values. AS-LIST serves as a helping mechanism [121]. In addition, the algorithm uses the progress and the anti-flickering properties of the Prove operation of the k-DenyList to enforce the k-consensus agreement property. The Propose operation operates as follows. First, a process p tries to prove that the element 0 is not revoked by invoking Prove(0). Then, if the previous operation succeeds, p revokes the element 0 by invoking Append(0). Then, p waits for the Append to be effective. This verification is done by invoking multiple Prove operations until one is invalid. This behavior is ensured by the progress property of the k-DenyList object. Once the progress has occurred, p is sure that no other process will be able to invoke a valid Prove(0) operation. Hence, p is sure that the set returned by the Read operation can no longer grow. Indeed, the Read operation returns the set of valid Prove operation that

occurred prior to its invocation. If no valid Prove(0) operation can be invoked, the set returned by the Read operation is fixed (with regard to the element 0). Furthermore, all the processes in Π share the same view of this set.

Finally, p invokes Read() to obtain the set of processes that invoked a valid Prove(0) operation. The response to the Read operation will include all the processes that invoked a valid Prove operation, and this set will be the same for all the processes in Π that invoke the Propose operation. Therefore, up to line 7, the algorithm solved the set-consensus problem. To solve consensus, we use an additional deterministic function $f_i : \Pi^i \to \Pi$, which takes as input any set of size i and outputs a single value from this set.

To simplify the representation of the algorithm, we also use the separator() function, which, on input of a set of proofs $(\{(p \in \Pi, \{\widehat{\mathcal{S}} \subseteq \mathcal{S}, \mathsf{P} \in \mathcal{P}_{\mathcal{L}_\mathcal{S}})\}\})$, outputs *processes*, the set of processes which conducted the proofs, i.e. the first component of each tuple.

Shared variables
  k-dlist $\leftarrow$ k-DenyList object;
  AS-LIST $\leftarrow$ Atomic Snapshot object, initially $\{\emptyset\}^k$
Operation Propose(v) **is**
1: AS-LIST.Update(v, p);
2: k-dlist.Prove(0);
3: k-dlist.Append(0);
4: **Do**
5: ret $\leftarrow$ k-dlist.Prove(0);
6: **Until** (ret $\neq$ `false`);
7: *processes* $\leftarrow$ separator(k-dlist.Read());
8: **Return** AS-LIST.Snapshot()[$f_{|processes|}$(*processes*)].

Algorithm 2: Implementation of a k-consensus object using one k-DenyList object and one Atomic Snapshot

Theorem 6.2. Algorithm 2 wait-free implements a k-consensus object.

Proof. Let us fix an execution E of the algorithm presented in Algorithm 2. The progress property of the k-DenyList object ensures that the while loop in line 4 consists of a finite number of iterations—an Append(0) is invoked prior to the loop, hence, the Prove(0) operation will eventually be invalid. Each invocation of the Propose operation is a sequence of a finite number of local operations, Atomic Snapshot object accesses and k-DenyList object accesses which are assumed atomic. Therefore, each process terminates the Propose operation in a finite number of its own steps. Let H be the history of E. We define $\bar{H}$ the completed history of H, where an invocation of Propose which did not reach line 8 is completed with a line "**return** `false`". Line 8 is the

linearization point of the algorithm. For convenience, any Propose invocation that returns `false` is called an failed invocation. Otherwise, it is called a successful invocation.

We now prove that all operations in $\bar{H}$ follow the k-consensus specification:

— The process p that invoked a failed Propose operation in $\bar{H}$ is faulty—by definition, the process prematurely stopped before line 8. Therefore, the fact that p cannot decide does not impact the termination nor the agreement properties of the k-consensus object.

— A successful Propose operation returns AS-LIST.Snapshot()$[f_{|processes|}(processes)]$. Furthermore, a process proposed this value in line 1. All the processes that invoke Propose conduct an Append(0) operation, and wait for this operation to be effective using the while loop at line 4 to 6. Thanks to the anti-flickering property of the k-DenyList object, when the Append operation is effective for one process—i.e. the Progress happens, in other words,a Prove(0) operation is invalid—, then it is effective for any other process that would invoke the Prove(0) operation. Hence, thanks to the anti-flickering property, when a process obtains an invalid response from the Propose(0) operation at line 5, it knows that no other process can invoke a valid Prove(0) operation. This implies that the Read operation conducted at line 7 will return a fix set of processes, and all the processes that reach this line will see the same set. Furthermore, because each process invokes a Propose(0) before the Append(0) at line 3, at least one valid Propose(0) operation was invoked. Therefore, the *processes* set is not empty. Because each process ends up with the same set *processes*, and thanks to the determinism of the function f_i, all correct processes output the same value v (Agreement property and non-trivial value). The value v comes from the Atomic Snapshot object, composed of values proposed by authorized processes (Validity property). Hence a successful Propose operation follows the k-consensus object specification.

All operations in $\bar{H}$ follow the k-consensus specification. To conclude, the algorithm presented in algorithm 2 is a wait-free implementation of the k-consensus object type. □

Corollary 6.2. The consensus number of the k-DenyList object type is at least k.

6.7.2 Upper bound

This section provides a DenyList object specification based on the PROOF-LIST object. The specification is then used to analyze the upper bound on the consensus number of the object type.

We provide an instantiation of the DenyList object defined as a PROOF-LIST object, where:

$$\forall y \in \mathcal{S}, y \in \mathcal{L}_{\mathcal{A}} \Leftrightarrow (\mathcal{A} \subseteq \mathcal{S} \wedge y \notin \mathcal{A}).$$

And where :

$$\mathsf{C}(y, \widehat{\mathcal{S}}) = \begin{cases} 1, & \text{if } \forall \mathcal{A}' \in \widehat{\mathcal{S}}, y \notin \mathcal{A}' \\ 0, & \text{otherwise.} \end{cases}$$

In other words, the first equation ensures that y does not belong to a set $\mathcal{A}$, while the second equation ensures that the object fulfills the anti-flickering property. Hence, this instantiation supports proofs of set-non-membership in *listed-values*. A PROOF-LIST object defined for such language follows the specification of the DenyList. To support this statement, we provide an implementation of the object.

To build a k-DenyList object which can fulfill the anonymity property, it is required to build an efficient helping mechanism that preserves anonymity. It is impossible to disclose directly the value proven without disclosing the user's identity. Therefore, we assume that a process p that invokes the $\mathsf{Prove}(v)$ operation can deterministically build a cryptographic commitment to the value v. Let C_v be the commitment to the value v. Then, any process $p' \neq p$ that invokes $\mathsf{Prove}(v)$ can infer that C_v was built using the value v. However, a process that does not invoke $\mathsf{Prove}(v)$ cannot discover to which value C_v is linked. If the targeted application does not require the user's anonymity, it is possible to use the plaintext v as the helping value.

Algorithm 3 presents an implementation of a k-DenyList object using k-consensus objects and Atomic Snapshots. The Append and the Read operations are analogous to those of Algorithm 1.

On the other hand, the Prove operation must implement the anti-flickering property. To this end, a set of k-consensus objects and a helping mechanism based on commitments are used.

When a process invokes the $\mathsf{Prove}(v)$ operation, it publishes C_v, the cryptographic commitment to v, using an atomic snapshot object. This commitment is published along with a timestamp [122] defined as follow. A local timestamp (p, c) is constituted of a process identifier p and a local counter value c. The counter c is always incremented before being reused. Therefore, each timestamp is unique. Furthermore, we build the strict total order relation $\mathcal{R}$ such that $(p, c)\mathcal{R}(p', c') \Leftrightarrow (c < c') \vee ((c = c') \wedge (p < p'))$. The timestamp is used in coordination with the helping value C_v to ensure termination. A process p that invokes the $\mathsf{Prove}(v)$ operation must parse all the values proposed by the other processes. If a $\mathsf{Prove}(v')$ operation was invoked by a process p' earlier than the one invoked by p—under the relation $\mathcal{R}$—, then p must affect a set "val" for the Prove operation of p' via the consensus object. The set "val" is obtained by reading the AS-LV object. The AS-LV object is append-only—no operation removes elements from the object. Furthermore, the sets "val" are attributed via the consensus object. Therefore, this mechanism ensures that the sets on which the Prove operations are applied always grow.

Furthermore, processes sequentially parse the CONS-ARR using the counter_p variable. This behavior, in collaboration with the properties of the consensus, ensures that all the process see the same tuples (winner, val) in the same order.

Finally, if a process p observes that a Prove operation conducted by a process $p' \neq p$ is

associated to a commitment C_v equivalent to the one proposed by p, then p produces the proof of set-non-membership relative to v and the set "val" affected to p' in its name. We consider that a valid Prove operation is linearized when this proof of set-non-membership is added to AS-PROOF in line 19. Hence, when p produces its own proof—or if another process produces the proof in its name—it is sure that all the Prove operations that are relative to v and that have a lower index in CONS-ARR compared to its own are already published in the AS-PROOF Atomic Snapshot object. Therefore, the anti-flickering property is ensured. Indeed, because the affected sets "val" are always growing and because of the total order induced by the CONS-ARR array, if p reaches line 25, it previously added a proof to AS-PROOF in the name of each process $p' \neq p$ that invoked a Prove (v) operation and that was attributed a set at a lower index than p in CONS-ARR. Hence, the operation of p' was linearized prior to the operation of p.

A Prove operation can always be identified by its published timestamp. Furthermore, when a proof is added to the AS-PROOF object, it is always added to the index counter_{p_w}. Therefore, if multiple processes execute line 19 for the Prove operation labeled counter_{p_w}, the AS-PROOF object will only register a unique value.

Furthermore, we use a function Proof that on input of a set $\mathcal{S}$ and an element x outputs a proof that $x \notin \mathcal{S}$. This function is used as a black box, and can either output an explicit proof—an explicit proof can be the tuple $(x, \mathcal{S})$—, or a Zero Knowledge Proof.

Theorem 6.3. Algorithm 3 wait-free implements a k-DenyList object.

Proof. Let us fix an execution E of the algorithm presented in Algorithm 3. The strict order relation $\mathcal{R}$ used to prioritize accesses to the CONS-ARR array implies that each process that enters the while loop in line 12 will only iterate a finite number of times. Furthermore, we assume that k-consensus objects and atomic-snapshot objects are atomic. Therefore, each process returns from a Prove, an Append, or a Read operation in a finite number of its own steps.

Let H be the history of E. We define $\bar{H}$, the completed history of H. We associate a specific response with all pending invocations in H. The associated responses are:

— Any invocation of the Append operation that did not reach line 3 can be completed with the line "**Return** `false`".

— Any invocation of the Prove operation that did not reach line 10 can be completed with the line "**Return** `false`".

— Any pending invocation of the Prove operation by the process p that reached line 10 is completed with the line "**Return** (val, π_{SNM});" if $(p,\ \text{value}, \pi_{SNM},\ \text{winner})$ is in the AS-PROOF array, and the value added by process p in line 10 is "winner". Otherwise, the operation is completed with the line "**Return** `false`".

— Any pending invocation of the Append operation that reached line 3 can be completed with line 4.

Shared variables

AS-LV ← N-dimensions Atomic-Snapshot object, initially $\{\emptyset\}^N$;
AS-Queue ← N-dimensions Atomic-Snapshot object, initially $\{\emptyset\}^N$;
CONS-ARR$_p$ ← an array of k-consensus objects of size $l > 0$;
AS-PROOF ← l-dimensions Atomic-Snapshot object, initially $\{\emptyset\}^l$;

Local variables

For each $p \in \Pi_V$:
evaluated$_p$ ← an array of size $l > 0$, initially $\{\emptyset\}^l$;
counter$_p$ ← a positive integer, initially 0;

Operation Append(v) **is**

1: **If** $(v \in \mathcal{S}) \wedge (p \in \Pi_M)$ **then**
2: local-values ← AS-LV.Snapshot()[p];
3: AS-LV.Update(local-values ∪ v, p);
4: **Return** `true`;
5: **Else return** `false`;

Operation Prove(v) **is**

6: **If** $p \notin \Pi_V$ **then**
7: **Return** `false`;
8: C_v ← Commit(v); 9: cnt ← counter$_p$;
10: AS-Queue.Update(((cnt, p), C_v), p);
11: queue ← AS-Queue.Snapshot() \ evaluated$_p$;
12: **While** (cnt, p) ∈ queue **do**
13: oldest ← the smallest clock value in queue under $\mathcal{R}$;
14: prop ← (oldest, AS-LV.Snapshot());
15: (winner, val) ← CONS-ARR[counter$_p$].Propose(prop);
16: ((counter$_{p_w}$, p_w), C^*) ← winner;
17: **If** $C^* = C_v \wedge v \notin$ val **then**
18: π_{SNM} ← Proof($v \notin val$);
19: AS-PROOF.Update((p_w, val, π_{SNM}, winner), counter$_{p_w}$);
20: evaluated$_p$ ← evaluated$_p$ ∪ winner;
21: queue ← queue \ winner;
22: counter$_p$ ← counter$_p$ + 1;
23: **If** $v \notin$ val **then**
24: **Return** (val, π_{SNM});
25: **Else return** `false`;

Operation Read() **is**

26: **Return** AS-PROOF.Snapshot();

Algorithm 3: k-DenyList object type implementation using k-consensus objects and Atomic Snapshot objects.

The linearization point of the Append and Read operations are respectively at line 3 and 26. Let us consider a valid Prove operation invoked by a process p that is attributed a tuple (winner, val) at the index counter_{p_w} of the CONS-ARR array. We say this operation is linearized when the first AS-PROOF.Update labeled with counter_{p_w} in line 19 is executed by any process.

For convenience, we call operations that return `false` invalid operations. The consensus objects in CONS-ARR are accessed at most once by each process. There are only $k = |\Pi_V|$ processes allowed to access these objects. Therefore, the k-consensus objects in the array always return a value different from $\emptyset$. We now prove that all operations in $\bar{H}$ follow the DenyList specification:

— An invalid Append operation in $\bar{H}$ only modifies the internal state of the process. This operation does not modify the state of the shared object. It is either invoked by an unauthorized process which fails in line 1, or by a faulty process. This operation follows the specification;

— An invalid Prove operation in $\bar{H}$ is an operation that returns `false` in line 7 or 25. In the first case, the process was not authorized to propose a proof. In the second case, the value v used by the process is already inside the set "val" the process was attributed by the consensus in line 15. This set is produced from the values added to the AS-LV object. This object begins as an empty set, and values inside this set can only be added using the Append operation. Therefore, the Prove validity property is ensured.

— If an invocation of the Append operation in $\bar{H}$ returns `true`, it implies that process p appended a value v to the *listed-values* array, at the index p at line 3. Because the WRITE operation is atomic, and because p is the only process able to write in AS-ACC[p], the *listed-values* array append-only property is preserved. Hence a successful Append operation follows the specification.

— If an invocation of the Prove operation in $\bar{H}$ returns `true`, it implies that: 1) process p was attributed a k-consensual set "val" on line 15, 2) from line 17, $v \notin$ val, and 3) a proof that $v \notin$ "val" was added to the AS-PROOF object, either by p or by another process performing the helping mechanism. First, to prove the progress property, we assume a history where first, a process p' obtains a positive response from an Append(v) operation. Afterward, a process p invokes a Prove(v) operation. Therefore, the value v will already be included in the AS-LV object at this time because p' received a positive response from its invocation. Any process that executes the line 14 of the Prove operation after the invocation of p will propose a set where v is included. Therefore, the set "val" that will be affected to p by the consensus on line 15 will include v. The Prove(v) operation invoked by p will be invalid. The progress property is ensured.
Second, the anti-flickering property is ensured by the helping mechanism and the k-consensus objects used from line 10 to 22. The processes in Π_V that invoke the Prove(v)

operation will sequentially attribute a set "val" to each proving process, using the set of k-consensus objects. Furthermore, this sequential attribution takes into account the evolution of the AS-LV object. Therefore, the set associated with the object CONS-ARR$[i-1]$ is always included in the set associated with the object CONS-ARR$[i]$.
Furthermore, the CONS-ARR array is browsed sequentially by each process invoking the Prove operation. Therefore, if a process p that invokes a Prove(v) operation with a timestamp t, and this invocation is not valid in the end, p will nonetheless linearize all the Prove(v) operations that have a lower timestamp than t before returning from the operation. Hence, all the valid Prove(v) operations will be linearized before the response of p's invocation, and any invocation of a Prove(v) operation that occurs after the response of p's invocation will fail.
Third, the Prove validity property directly follows from point (2) and the anti-flickering property. Hence a successful Prove operation follows the specification.

— A Read operation always returns, and thanks to the atomicity of the Atomic Snapshot object, it always returns the most up-to-date version of the AS-PROOF array.

All operations in $\bar{H}$ follow the k-DenyList specification. Therefore the algorithm presented in Algorithm 3 is a wait-free implementation of the k-DenyList object type. □

The following corollary follows from Theorem 6.2 and Theorem 6.3.

Corollary 6.3. The k-DenyList object type has consensus number k.

6.8 Variations on the *listed-values* array

In the previous sections, we assumed the *listed-values* array was append-only. Some use cases might need to use a different configuration for this array. In this section, we want to explore the case where the *listed-values* array is no longer append-only.

6.8.1 One-process only

We will first explore a limited scenario where the processes can only remove values they wrote themselves on the *listed-values* array. In this case, there are no conflicts on the append and remove operations. The *listed-values* array can be seen as an array of $|\Pi_V|$ values. A process p_i can write the i-th index of the *listed-values* array. It is the only process that modifies this array. Therefore, there are no conflicts upon writing. We would need to add a Remove operation to the AllowList and DenyList object. Because of this Remove operation, the AllowList could act as a DenyList. Indeed, let us assume the managers adds all elements of the universe of the possible identifiers to the AllowList in the first place. Then, this AllowList can implement a DenyList object, where the Remove operation of the AllowList is equivalent to the Append

operation of the DenyList. Hence, the AllowList object would need an anti-flickering property to prevent concurrent Prove operations from yielding conflicting results. This implies that an AllowList object implemented with a Remove operation is equivalent to a DenyList object and has consensus number k, where k is the number of processes in Π_V.

6.8.2 Multi-process

The generalization of the previous single-write-remove *listed-values* array is a *listed-values* array where k_{AR} (AR for Append/Remove) processes can remove a value appended by process p_i. We assume each process p is authorized to conduct Append and Remove operations on its "own" register. Furthermore, each process p_i has a predefined authorization set $\mathcal{A}_i \subseteq \Pi_M$, defining which processes can Append or Remove on p_i's register. We always have $p_i \in \mathcal{A}_i$. If $p_j \in \mathcal{A}_i$, then p_j is allowed to "overwrite" (remove) anything p_i wrote. In this case, all authorized processes need to synchronize in order to write a value on the *listed-values* array. More precisely, we can highlight two cases.

The first case is the "totally shared array" case, where all processes share the same $\mathcal{A}_i = \Pi_M$. Any modifications on the *listed-values* array by one process p_i can be in competition with any other process $p_j \in \Pi_M$. Therefore, there must be a total synchronization among all the processes of the managers' set to modify the *listed-values* array. When such behaviour is needed, both AllowList and DenyList require solving consensus among at least $|\Pi_M|$ processes to implement the Append and Remove operations.

The second case is the "cluster" case: a subset of processes share a sub-array, which they can write. In this case, each process in a given cluster must synchronize before writing (or removing) a value. The synchronization required is only between this cluster's k_{AR} authorized process. This corresponds to some extent to a sharded network [123].

6.9 Discussion

This section presents several applications where the AllowList and the k-DenyList can be used to determine consensus number of more elaborate objects. More importantly, the analysis of the consensus of these use cases makes it possible to determine if actual implementations achieve optimal efficiency in terms of synchronization. If not, we use the knowledge of the consensus number of the AllowList and DenyList objects to give intuitions on how to build more practical implementations. More precisely, the fact that consensus numbers of AllowList and DenyList objects are (in most cases) smaller than n implies that most implementations can reduce the number of processes that need to synchronize in order to implement such distributed objects. The liveness of many consensus protocols is only ensured when the network reaches a synchronous period. Therefore, reducing the number of processes that need to synchronize can increase the

system's probability of reaching such synchronous periods. Thus, it can increase the effectiveness of such protocols.

6.9.1 Revocation of a verifiable credential

We begin by analyzing Sovrin's Verifiable-Credential revocation method using the DenyList object [52]. Sovrin is a privacy-preserving Distributed Identity Management System (DIMS). In this system, users own credentials issued by entities called issuers. A user can employ one such credential to prove to a verifier they have certain characteristics. An issuer may want to revoke a user's credential prematurely. To do so, the issuer maintains an append-only list of revoked credentials. When a user wants to prove that their credential is valid, they must provide to the verifier a valid ZKP of set-non-membership proving that their credential is not revoked, i.e. not in the DenyList. In this application, the set of managers Π_M consists solely of the credential's issuer. Hence, the proof concerns solely the verifier and the user. The way Sovrin implements this verification interaction is by creating an ad-hoc peer-to-peer consensus instance between the user and the verifier for each interaction. Even if the resulting DenyList has consensus number 2, Sovrin implements the Append operation using an SWMR stored on a blockchain-backed ledger (which requires synchronizing the N processes of the system). Our results suggest instead that Sovrin's revocation mechanism could be implemented without a blockchain. The Append operation could be implemented using FIFO reliable broadcast, and the Prove operation could be implemented using pairwise consensus between users and verifiers. We show in Chapter 10 how to use efficient distributed algorithm to reach this lower bound with a system similar to the one implemented by Sovrin. We also extend this application to the multi-device authorization problem that is studied in Chapter 9.

6.9.2 Distributed e-vote systems

Finally, another direct application of the DenyList object is the blind-signature-based e-vote system with consensus number k, k being the number of voting servers, which we present in Appendix A. Most distributed implementations of such systems also use blockchains, whereas only a subset of the processes involved actually require synchronization.

6.10 Conclusion

This chapter presented the first formal definition of distributed AllowList and DenyList object types. These definitions made it possible to analyze their consensus number. This analysis concludes that no consensus is required to implement an AllowList object. On the other hand, with a DenyList object, all the processes that can propose a set-non-membership proof must synchronize, which makes the implementation of a DenyList more resource intensive.

The definition of AllowList and DenyList as distributed objects made it possible to thoroughly study other distributed objects that can use AllowList and DenyList as building blocks. Importantly, we discussed the use of DenyList to implement revocation lists for PPfDIMSs. We will also use the result presented in this chapter to implement a multi-device feature for PPfDIMS. This implementation is presented in Chapter 9. Overall, AllowList and DenyList are one of the main tools of PPfDIMS. Therefore, the thorough study conducted in this chapter will allow us to determine the optimality of PPfDIMSs in term of synchronization requirements.

Chapter 7

From Zooko's trilemma to the Namespace object: how to allocate scarce names in a distributed system

This chapter studies the problem of attributing names to resources in a distributed system. It analyzes Zooko's trilemma as well as the consensus number of the naming problem. Those analyses are used to find the lowest synchronization requirements when creating DID documents in PPfDIMSs. Those results are used in Chapter 8 to build a new asynchronous naming algorithm and in Chapter 10 to create DIDs and DID documents in the context of PPfDIMS. Many developments presented here arose from discussions with Timothé Albouy.

7.1 Introduction

One of the main components of a PPfDIMS is a DID and DID document-capable ledger. DID documents are used to enable auxiliary features of PPfDIMSs. We discussed some of the distributed problems related to these auxiliary features and, thus, to the usage of DID documents in Chapter 6. However, when an entity creates a DID document, it has to name it. In other words, it must create a unique DID that refers to this DID document. Two different versions of this problem have been studied. The first type of study was conducted in the shared-memory model and is called the renaming problem [124, 125]. The setting of this problem is the following: processes have names at initialization, which are drawn from a large set $\mathcal{N}$. They must rename themselves so that each process has a name from a second set $\mathcal{M}$, and the size of $\mathcal{M}$ is significantly smaller than $\mathcal{N}$. Furthermore, no two processes can share the same name, and each (correct) process must be able to learn the names of the other processes. Importantly, in the renaming problem, processes cannot choose their names. The second type of study looks at the names and the forms they can take in distributed systems. Whereas the first type of study is formally defined and provides proof of what is possible to do or not, it does not directly apply

to most use cases. Specifically, it does not apply to creating DID and DID documents. In this chapter, we are interested in the second type of problem.

In 2001, Wilcox Zooko published a blog article [16] presenting its thoughts on the problem of name allocation in distributed systems. The outcome of this article is the following conjecture: In a distributed system, it is impossible to identify a process with a name that is at the same time secure and human-meaningful. This chapter aims to formally define the different objects and properties at stake in this problem and prove Zooko's statement. To the best of our knowledge, this is the first attempt to provide such formal proof of Zooko's trilemma.

Interestingly, multiple systems achieve these three properties (distributed, human-meaningful, and secure). The most famous one is DNSSec [126], which associates each name with an IP address and adds security through the use of digital signatures. Thus, only the owner of a domain name can modify its IP address. Another well-known system is the Ethereum Name Service (ENS) [115], which allocates domain names using the Ethereum blockchain [127]. Namecoin [116] is the predecessor of ENS and works similarly.

These few examples show a disconnection between state-of-the-art implementations and Zooko's statement. This chapter will show that these systems circumvent the problem rather than solve it. To do so, they associate multiple (often two) identifiers. One provides human meaningfulness and distribution, while the other provides security and distribution.

When we know how to circumvent the problem, it is interesting to understand what is required to build a "Zooko's circumventing" naming system. We will explore the cryptographic minimal requirements of such systems and the distributed-system theory that applies to this problem. To the best of our knowledge, this is the first time such an explicit analysis has been conducted.

In this chapter, we will explain Zooko's triangle problem. Then, we will explain how distributed systems circumvent this problem. Afterward, we will explore the distributed system's notion of naming problem. Finally, we will map the naming problem to Zooko's triangle problem and identify the minimal requirements in term of synchronization necessary to build a naming system.

7.2 Identifiers, resources and namespaces

A naming system aims to associate an identifier with a resource. We define an identifier as a unique and finite string of bits. Identifiers can be augmented with additional properties. These properties are exposed in Section 7.4. Formally:

Definition 7.1. Identifier. An identifier is an arbitrary finite string of bits.

A resource, on the other hand, is any object that an identifier can identify. Intentionally, we keep the definition of a resource vague. Formally:

Definition 7.2. Resource. A resource is an object (either physical or digital) that can be uniquely identified and distinguished from other objects.

The goal of an identifier is to identify a resource. A resource, whether a physical or a digital object, does not have a name by itself. A name is only a cultural construction that has a meaning in a specific environment. We call this environment a namespace:

Definition 7.3. Namespace. A namespace $\mathcal{N}$ is defined for a set of processes Π as a dictionary that associates a resource with an identifier. Each identifier must be unique in a given namespace.

Interestingly, this definition of a namespace does not imply any properties. For example, an identifier can be arbitrarily long. Furthermore, this definition does not imply any security property. For example, with this definition, any process could impersonate another.

7.3 The Namespace object

This section proposes a formal definition of a distributed Namespace object. The specification's goal is to formalize Zooko's statement so that we can formally prove its properties.

We provide a sequential definition of a *Namespace object* (NO) for a unique resource. This specification describes how an identifier is associated with a resource. This definition is based on a bijective function f_{sec} that will be used to define the security property of an identifier.

Definition 7.4. Namespace object. A Namespace object $\mathcal{NO}$ is defined for a set of resources $\mathcal{R}$, a set of identifiers $\mathcal{ID}$, where $id \in \mathcal{ID}$ is a unique arbitrary string of bits, an arbitrary set A and a bijective function $f_{\text{sec}} : A \to \mathcal{ID}$ by a set of three operations $\{\mathcal{NO}.\mathsf{Associate}, \mathcal{NO}.\mathsf{Read}, \mathcal{NO}.\mathsf{ProvePossession}\}$. The operations $\mathcal{NO}.\mathsf{ProvePossession}(a, res)$ and $\mathcal{NO}.\mathsf{Associate}(a, res)$ both have two inputs, $a \in A$, an element in A, and $res \in \mathcal{R}$, a resource. The operation $\mathcal{NO}.\mathsf{Read}(res)$ takes one argument, a resource $res \in \mathcal{R}$. We give a definition of the properties of a Namespace object where each operation appears as in a sequence seq.

- *Termination.* the invocation of the operations $\mathcal{NO}.\mathsf{Associate}$, $\mathcal{NO}.\mathsf{Read}$ and $\mathcal{NO}.\mathsf{ProvePossession}$ terminate.
- $\mathcal{NO}.\mathsf{Associate}$ *validity.* The operation $\mathcal{NO}.\mathsf{Associate}(x, res)$ invoked by the process p_i is valid if $x \in A, res \in \mathcal{R}$ and no valid $\mathcal{NO}.\mathsf{Associate}(\star, res)$ operation appears before in the sequence seq.
- $\mathcal{NO}.\mathsf{Read}$ *validity.* The operation $\mathcal{NO}.\mathsf{Read}(res)$ invoked by the process p_i is always valid. It eventually returns $f_{\text{sec}}(x)$ if a valid $\mathcal{NO}.\mathsf{Associate}(x, res)$ operation appears before in seq. Otherwise, it returns $\perp$.
- $\mathcal{NO}.\mathsf{ProvePossession}$ *validity.* The operation $\mathcal{NO}.\mathsf{ProvePossession}(x, res)$ invoked by pro-

cess p_i is valid if $\mathcal{NO}.\mathsf{Read}(res) = id$ and $f_{\text{sec}}^{-1}(id) = x$, where f_{sec}^{-1} is the inverse function of f_{sec}.

— *Identifier unicity.* Two different invocation of an $\mathcal{NO}.\mathsf{Associate}(x, \star)$ operations cannot appear in seq.

As we can see, the operation $\mathcal{NO}.\mathsf{Associate}(a, res)$ is used to associate an identifier with a ressource. However, the element $a \in A$ used in the operation is not necessarily the actual identifier as it appears to the other processes of the system. Processes that did not invoke the valid $\mathcal{NO}.\mathsf{Associate}(a, res)$ operation can only see the identifier associated with the resource res by invoking the operation $\mathcal{NO}.\mathsf{Read}(res)$. This operation does not return a, it returns $f_{\text{sec}}(a)$. Hence, $f_{\text{sec}}(a)$ is the actual identifier as it appears to the system. This remark will be necessary to define the security of a Namespace object formally.

Notably, the only information processes have access to with a Namespace object is this $f_{\text{sec}}(a)$ value. If the processes have access to any additional value, then the object would not fulfill the Namespace specification, as we are interested in objects where a resource is associated to a *unique identifier*, *i.e.*, a unique string of bits. We study resources associated with multiple identifiers in Section 7.9.

7.4 The Zooko's triangle problem

The Zooko's triangle problem, or Zooko's trilema, is a problem informally introduced by Wilcox Zooko in 2001 [16] in a blog post. In this blog article, Zooko tackles the problem of building a distributed Namespace object that is simultaneously human-meaningful (*e.g.*, which does not consist of a random long string of characters) and cannot be impersonated. Zooko stated that a unique identifier could not fulfill the three requirements (distributed, secure, and human-meaningful) at the same time.

Zooko's problem isolates three desirable identifier properties: decentralization, security, and human-meaningfulness. A name is decentralized if the system consists of more than one process, if there is no trusted third party, and if all the correct processes agree on the associations identifier↔resource. An identifier id associated with a resource res and belonging to a process p_i is secure if no process $p_j \neq p_i$ can modify (or prove the possession of) the resource associated to id even if p_j is Byzantine. Finally, the human-meaningful property is the hardest to describe. It is a property related to a human's ease of remembering and reusing an identifier. Multiple definitions can be proposed to describe this notion. A popular one is the bus test. If an identifier id is written on the side of a bus, and this bus stays less than 3 seconds in the eyesight of an observer, then this observer should be able to recall this name a few minutes later. We prefer to use the "human-choosable" definition. This definition states that a name is human-meaningful if it consists of a string of characters, and each character, along with the length of the string, is

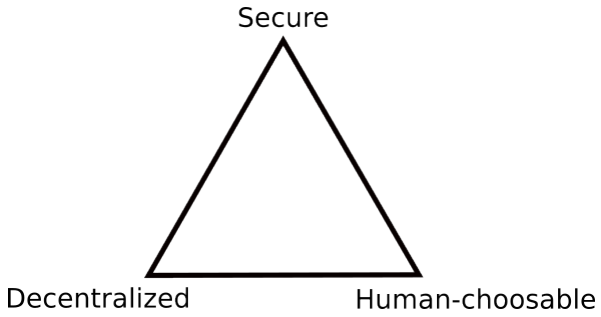


Figure 7.1 – The Zooko's triangle

chosen by the user controlling the process p_i. This second definition also has the advantage of capturing the scarcity of names. Multiple processes might choose the same name, and there are few human-choosable names compared to the space of possible identifiers.

These properties are often represented as a triangle, where each corner represents one property. The sides of the triangles represent names with two properties, and the center of the triangle is the identifier that implements every property. This triangle is represented in Figure 7.1.

It is easy to understand Zooko's statement—which is the title of his blog post—"Names: Decentralized, Secure, Human-meaningful: choose two". In other words, no identifier can fulfill the three properties at the same time. Let us explain this statement.

First, let us place ourselves in a distributed system without trusted third parties. Therefore, the identifiers must fulfill the decentralized property, *i.e.*, every process in the system must agree on which identifier is associated with which process. Let us add human-meaningfulness. Using the human-choosable definition, the identifier must be a string of characters whose size and content are chosen by a human. Now, we add security. No Byzantine process should be able to impersonate another, i.e., no Byzantine process should be able to act using an identifier it does not own. The only known technique to achieve this goal is to associate the identifier with a secret. This secret can be the one of an asymmetric cryptographic systems such as a signature scheme or zero-knowledge proof of knowledge of a secret. However, Zooko's triangle problem is defined for a unique identifier, *i.e.*, for a unique string of bits. [1] Furthermore, a secure asymmetric cryptographic scheme is based on the randomness of the secret used, and thus, the resulting public identifier has to be random too.

The two last points directly indicate that the security and human-meaningful properties cannot be fulfilled simultaneously. Zooko conjectured this impossibility in its article. In the following, we formally prove this impossibility by formalizing the three properties of the trilemma.

The human-choosable property ensures that a process can choose the name it associates with a resource.

1. We will discuss in the following how to circumvent Zooko's triangle problem by mapping multiple identifiers

Definition 7.5. Human-choosable. A Namespace object fulfills the human-choosable property if any process p_i has access to an efficiently computable and deterministic algorithm $x \leftarrow \mathsf{choose}(id)$ such that if it invokes a valid $\mathcal{NO}.\mathsf{Associate}(x, res)$ operation, then $\mathcal{NO}.\mathsf{Read}(res)$ eventually outputs id.

In other words, the human-choosable property states that a process can invert the f_{sec} function. The idea is that the user wants to claim the identifier id. However, processes can only use the $\mathcal{NO}.\mathsf{Associate}$ operation to associate a name to an identifier. Furthermore, the processes that invoke this operation only choose $x \in A$ the pre-image of id relative to f_{sec}. Thus, the choose algorithm is necessary for the process to be able to choose its identifier. The bijectivity of the f_{sec} function implies that the choose function is also bijective. Hence, only one choose function exists. Furthermore, it is assumed that *each* process can associate a resource to *any* identifier. Therefore, all processes have access to the same choose function that can be used to invert the f_{sec} for all the identifiers in $\mathcal{ID}$.

The decentralized property ensures that all the correct processes in the network see the same identifiers associated with a resource. In other words, this property ensures an agreement on the association between a resource and an identifier.

Definition 7.6. Decentralized. Let $|\Pi| = n > 1$. Let $p_i \in \Pi$ be a correct process. Let p_i invoke a valid $\mathcal{NO}.\mathsf{Associate}(x, res)$ operation. A Namsepace Object is decentralized if the invocation of $\mathcal{NO}.\mathsf{Read}(res)$ by a correct process $p_j, \forall j \in \{1, \ldots, n\}$ eventually returns $f_{\text{sec}}(x)$.

Finally, the secure property states that a process must know the secret used to claim the identifier id to invoke a valid $\mathcal{NO}.\mathsf{ProvePossetion}(\star, res)$ operation. Hence, the process that invokes a valid $\mathcal{NO}.\mathsf{ProvePossetion}$ operation proves that it is the rightful owner of the resource res.

Definition 7.7. Secure. A Namespace object is secure if, given $id = \mathcal{NO}.\mathsf{Read}(res)$, the probability for a process p_i to invoke a valid $\mathcal{NO}.\mathsf{Prove}(\star, res)$ operation is lower than ϵ, where ϵ is a security parameter.

This definition also implies that the implementation of the object hides the value of $f_{\text{sec}}^{-1}(id)$. Otherwise, the security property could be trivially broken.

7.5 Formal proof of the Zooko's impossibility

Wilcox Zooko stated in his blog article that no identifier (*i.e.*, Namespace object) can fulfill the human-choosable, secure, and decentralized properties at the same time. This section provides a formal proof of this statement. We recall that this proof considers Namespace objects, *i.e.*, objects where resources are identified by a unique identifier. Hence, the security property must be solely fulfilled by this unique identifier. Any additional information would change the

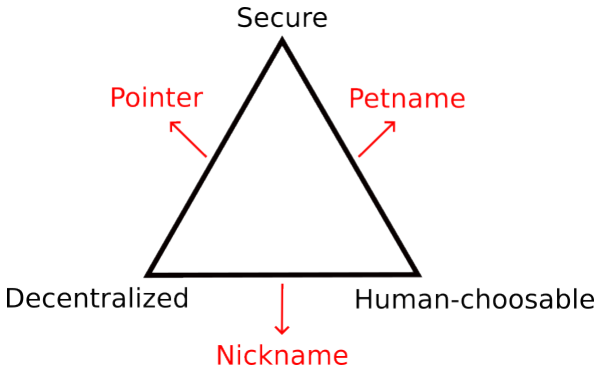


Figure 7.2 – Edges of the Zooko's triangle

object to an Identifier System object. We study Identifier System objects in Section 7.9. However, here we interest ourselves to Namespace objects.

Theorem 7.1. Zooko's triangle problem. A Namespace object cannot fulfill the human-choosable property, the decentralized property, and the secure properties at the same time.

Proof. This proof is a proof by contradiction. We will assume that a Namespace object can fulfill the three properties.

First, let a process p_i invoke a valid $\mathcal{NO}.\mathsf{Associate}(x, res)$ operation. Using the human-choosable property, we know that p_i has access to a function choose such that it can choose an identifier id and use $x = \mathsf{choose}(id)$ in its invocation of $\mathcal{NO}.\mathsf{Associate}(x, res)$. Second, using the decentralized property, we know that there are at least two processes in the system and that they are eventually able to discover id, the identifier associated with res, by invoking $\mathcal{NO}.\mathsf{Read}(res)$. Third, the Human-choosable property states that any process has access to the algorithm choose and that this algorithm is deterministic. In other words, every process in the system knows id and $\mathsf{choose}(id) = f_{\text{sec}}^{-1}(id)$. Hence, each process can compute $x = f_{\text{sec}}^{-1}(Id)$. Therefore, any process in the system will eventually be able to invoke a valid $\mathcal{NO}.\mathsf{Prove}(x, res)$ operation. Hence breaking the security property.

Therefore, a Namespace object cannot simultaneously fulfill the Human-choosable property, the Decentralized property, and the Security property without leading to a contradiction. Hence, Zooko's triangle impossibility is proven. □

7.6 The edges of the triangle

In Section 7.5, we showed that a Namespace object cannot fulfill all three properties of an identifier at the same time. However, an identifier can fulfill two of them. These are the edges of

Zooko's triangle, each with specific properties and usages. Each type of identifier is highlighted in Figure 7.2. They can be defined as:

— **Petname.** A petname is a secure and human-meaningful identifier. A petname is an identifier used on a unique computer or server, *i.e.*, there is no decentralization. There is only one process in the system. Therefore, the process considered is correct. This process can associate an identifier with any resource in its own memory without worrying about security. Therefore, any string of characters chosen by the user will be secure. It directly follows that the human-choosable property is also ensured.

— **Nickname.** A nickname is a decentralized and human-meaningful identifier. A nickname can be easily implemented in a system without Byzantine processes. A user only has to choose an identifier. Then, the process communicates the association between the identifier and the resource the user owns. With this type of identifier, however, a unique Byzantine process could impersonate each identifier.

— **Pointer.** A pointer is a secure and decentralized identifier. A pointer is a public key of some asymmetric cryptographic scheme. This public key must allow the associated secret key holder to prove he or she is (probabilistically) the rightful owner of the identifier and, thus, of the resource. This key can be disseminated among the processes. When a process needs to access a resource, it needs to produce a context-dependent proof that it knows the secret key associated with the identifier.

7.7 The consensus number of the edges

This section analyses the consensus number of a Namespace object and applies this result to the different types of identifiers.

7.7.1 Consensus number of the Namespace object

To prove the consensus number of a distributed object, we use the shared memory model presented in Section 4.1.1. Knowledge of the consensus number of a distributed object can be used to determine whether a consensus algorithm is required to implement the same object in a message-passing system. More precisely, if the consensus number of an object is strictly greater than 1, then its implementation in the message passing model requires the use of a consensus algorithm and, therefore, some degree of synchronization [128].

In this section, we prove that the consensus number of the Namespace object is ∞. The proof is a proof by reduction. Algorithm 4 is an algorithm that implements a consensus object using only a Namespace object and Atomic Snapshot objects. The algorithm lets processes that want to propose a value v add it to the AS-VALUES array. This array is used as a helping mechanism [121]. Then, processes advertise which resource they will name thanks to the AS-RES array.

Afterward, they elect a leader by naming their resource with a predefined identifier, in our case, the identifier 0. Only one process can succeed thanks to the identifier unicity property of the Namespace object. This process is the leader; the value that it added to the AS-VALUES array is chosen as the result of the consensus.

Shared variables
$\mathcal{NO} \leftarrow$ Namespace object;
AS-VALUES $\leftarrow$ Atomic Snapshot object, initially $\{\emptyset\}^n$;
AS-RES $\leftarrow$ Atomic Snapshot object, initially $\{\emptyset\}^n$;
Local variables
$res_i \leftarrow$ a resource, unique for each process in Π;
Operation Propose(v) **is**
1: AS-VALUES.Update(v, i);
2: AS-RES.Update(res_i, i);
3: $\mathcal{NO}$.Associate($0, res_i$);
4: $id \leftarrow \perp$;
5: **While** $id = \perp$ **do**:
6: resources $\leftarrow$ AS-RES.Snapshot();
7: **for** $j \in \{1, \ldots, n\}$ **do**
8: **If** $\mathcal{NO}$.Read(ressources[j]) $\neq \perp$
9: $id \leftarrow \mathcal{NO}$.Read(ressources[$j$]);
10: winner $\leftarrow j$;
11: **Return** AS-LIST.Snapshot()[winner].

Algorithm 4: Implementation of a consensus object using one Namespace object and one Atomic Snapshot, code for process p_i

Lemma 7.1. If p is correct, the loop from line 5 to 10 eventually terminates at process p.

Proof. The $\mathcal{NO}$.Associate validity property ensures that if one or multiple correct processes invoke the $\mathcal{NO}$.Associate$(x, \star)$ operation, then at least one will be valid. The identifier unicity property does not contradict this statement, as this property only avoids a second valid $\mathcal{NO}$.Associate$(x, \star)$ operation when another valid one has already been invoked. Furthermore, the $\mathcal{NO}$.Read validity property states that, if a valid $\mathcal{NO}$.Associate$(\star, res)$ operation is invoked, then eventually, the $\mathcal{NO}$.Read$(\star, res)$ operation outputs a value different from $\perp$. Hence, because p invokes $\mathcal{NO}$.Associate at line 3, and because the $\mathcal{NO}$.Read operation is invoked on all the possible resources of the system, then the $\mathcal{NO}$.Read operation at line 8 eventually outputs a value different from $\perp$ and the condition at line 5 is no longer fulfilled. Thus, the loop from line 5 to 10 eventually terminates at process p. □

Theorem 7.2. Algorithm 4 is a wait-free implementation of a consensus object as defined in Section 4.1.1, for any number of processes.

Proof. Let us fix an execution E of the algorithm presented in Algorithm 4. Using Lemma 7.1, we know that each process terminates the Propose operation in a finite number of its own steps. Let H be the history of E. We define $\bar{H}$ the completed history of H, where an invocation of Propose which did not reach line 3 is completed with a line "return `false`". Line 3 is the linearization point of the algorithm. Any invocation of propose in $\bar{H}$ which reached line 3 and did not terminate is completed with the lines 4 to 11. For convenience, any Propose invocation that returns `false` is called a failed invocation. Otherwise, it is called a successful invocation.

We now prove that all operations in $\bar{H}$ follow the consensus specification:

— The process p that invoked a failed Propose operation in $\bar{H}$ is faulty—by definition, the process prematurely stopped before line 3. Therefore, the fact that p cannot decide does not impact the termination nor the agreement properties of the consensus object.

— Because the loop from line 5 to 10 terminates (c.f. Lemma 7.1), a successful Propose operation returns AS-LIST.Snapshot()[winner]. winner is the identifier of the only process that could associate the identifier 0 to a resource. Thanks to the identifier unicity property, we know that no other process can do so. Furthermore, this process added a value to AS-VALUES beforehand (line 1). Hence, even if this process crashes after the linearization point, all other correct processes will output its proposed value. Therefore, the value this process added to AS-VALUES will be the value output by all the correct processes, thus ensuring the validity, agreement, and non-triviality properties.

All operations in $\bar{H}$ follow the consensus object specification. To conclude, the algorithm presented in algorithm 4 is a wait-free implementation of the consensus object type. □

Corollary 7.1. The consensus number of the Namespace object is ∞.

7.7.2 Consensus requirements in practice

The consensus number of the Namespace object gives information about how a naming system can be implemented. Using the results of Section 7.7.1, we know that implementing a Namespace object in a message-passing system requires a consensus. However, in practice, the probability that two processes associate the same identifier to two different resources with a pointer-based namespace object is negligible. Hence, a pointer-based namespace object can be implemented without consensus. This section aims to give a high-level understanding of this statement. A formal proof of the statement can be found in Section 8.7.1, where we implement a pointer-based Identifier System object (c.f. Section 7.9) without consensus.

To understand this fact, we must cope with the scarcity of names. Indeed, and as pointed out by Kalodner et al. [116], if individuals can freely choose their names, then the "interesting" names

will be scarce. Therefore, the set of nicknames is small compared to the amount of resources that must be identified and cannot be extended. Due to this scarcity, there is a high probability that two processes try to associate the same name with two different resources.

On the other hand, in the case of a pointer, processes cannot choose their identifiers. They usually draw them uniformly at random in a large set. [2] This set can be chosen arbitrarily large so that it is always much greater than the number of resources to identify. More precisely, the set can be designed such that the probability of collision (the probability that two processes draw the same name) is negligible. A low probability of collision is the basis of any asymmetric cryptographic scheme.

Hence, the problem of allocating a name to a resource can be seen as a consensus problem, where different processes can associate a specific resource with an identifier, and the system's processes have to choose which resource will be associated with this identifier. Hence, each existing identifier can be seen as a consensus object. To associate a name with an identifier, processes must propose a resource for this specific identifier. Therefore, the consensus number of any Namespace object should be ∞. However, with a pointer, the probability that two processes choose the same name, *i.e.*, the probability that, given a consensus instance, there exists more than one proposition, is negligible. Hence, and with high probability, a pointer can be implemented without consensus, whereas a nickname has to be implemented using consensus. Finally, the petname is not a distributed problem; it is considered that only one process exists in the system. Hence, we cannot talk about its consensus number, as this notion has no meaning in this case.

Therefore, only implementing a Namespace object based on nicknames requires consensus. We present in Chapter 8 an implementation of a pointer-based Namespace object that does not require consensus. This implementation uses an assumption encapsulating the low probability of collision of two pointers: no two processes can draw and propose the same identifier.

7.8 The difference between the Namespace object specification and the renaming problem

One subject we did not tackle in the previous sections is the relation between the Namespace object and the renaming problem [124, 125]. The renaming problem is well-studied in the shared-memory model. It assumes a set of n processes named with identifiers drawn from a large set (for example, their names are in the set $\{1, \ldots, N\}$, where N is much greater than n). The goal of the processes is to rename themselves, such that the space of new names must be drawn in a smaller set $\{1, \ldots, M\}$, where M is smaller than N, and no two processes share the same name. In this setting, the resources identified are the processes themselves. Furthermore and

2. In fact, they draw the secret associated with the identifier.

importantly, with the renaming problem, processes cannot choose their names.

This setting is different from the one we studied with the Namespace object. With the Namespace object, we do not aim at naming all the resources, *e.g.*, with the DID example, there exists an infinite number of possible DID documents, as the information a DID document can contain is unbounded (at least in theory, in practice, it can be bounded by the system that implements the PPfDIMS, but this bound can be large). Hence, the number of resources is considered unknown. Meanwhile, with the renaming problem, the number of resources is known in advance, and all the resources must be identified.

The consensus number of the renaming problem depends on the number of available names (*i.e.*, on the number M); Attiya et al. [124] proved that, if $M \geq 2n - 1$, then the renaming problem can be solved only using read/write registers. Hence, its consensus number is 1. On the other hand, Castenada et al. [125] proved that the renaming problem with $M < 2n - 1$ has consensus number 2. However, unlike the object presented in Chapter 6, this consensus number does not concern specific processes. To implement a consensus number 2 perfect renaming object [3] from read/write registers and 2-consensus objects, it is required to use a bounded number of "splitters", mutual-exclusion-like objects. The process that is granted the equivalent of the critical section of a given splitter is granted the name associated with it.

A good intuition to understand why the renaming problem is different from the Namespace object is to think about cooperation. The goal of the renaming problem is for all processes to cooperate to assign a predefined number of identifiers to each process. Furthermore, there are always more (or the same number of) identifiers than resources to identify. Meanwhile, with the Namespace object, processes compete to claim the most interesting names. The number of interesting names is statistically less than the number of resources to identify.

7.9 Identifier systems - Circumventing Zooko's impossibility

Section 7.5 proves that a Namespace object cannot be simultaneously human-choosable, secure, and decentralized. We presented in Section 7.6 the three types of identifiers that can be implemented. These three types of identifiers are widely used. However, some use cases may require the achievement of the three properties simultaneously. For example, with a PPfDIMS, an individual may be required to identify and communicate with a service provider using its DID. Hence, this DID needs to be secure to prevent impersonation, distributed because we are in a fully distributed IMS, and human-choosable to allow humans to handle and remember names, thus reducing attacks such as phishing attacks. Therefore, we need a way to circumvent Zooko's impossibility.

3. A perfect renaming is a specific instance of the renaming problem where $M = n$, the number of processes in the system.

The usual—and only—way to circumvent Zooko's triangle impossibility is to use an identifier system. An identifier system associates multiple identifiers to a unique resource. Usually, it is sufficient to associate two identifiers to obtain all three properties.

Two identifier systems are usually used. The petname systems [129], and the nickname systems.

The petname system was proposed to improve PKI-based security. It can be challenging for a human to see the difference between a legitimate website name and a fraudulent one. For example, someone could not spot the difference between "google.com" and "goog1e.com". However, credential-based security is sometimes insufficient to protect users against this attack. The petname system is a countermeasure that proposes a user to locally associate the public key of a website to a name chosen by the user—a petname in this case. The petname is local, human-meaningful, and secure. However, it cannot be shared. This identifier allows the user to spot fraudulent websites and to abort phishing attempts.

On the other hand, the identifier system that will interest us in the following of this thesis is the nickname system. To the best of our knowledge, a nickname system is a widely used identifier system that has never been characterized as such. The idea is that each process is identified by all other processes in the system using a pointer, *i.e.*, the public key of some asymmetric cryptographic scheme. This pointer is then associated with a nickname. Thus, the pointer provides security, while the nickname provides human-choosability. The nickname system is the naming system used by PKI-based certificates (where a name is associated with a public key) [36], by the DNSSec system [126] and by the DID specification of the W3C [8].

7.9.1 The Identifier System Object

We provide a sequential definition of an Identifier System Object (ISO). This specification generalizes the Namespace object definition, where one resource is associated with multiple names, each unique in the namespace. The number of identifiers of an ISO object is denoted k and written as a subscript. Hence, $\mathcal{ISO}_2$ consists of two identifiers.

Definition 7.8. Identifier System Object. An Identifier System Object $\mathcal{ISO}_k$ is defined for a set of resources $\mathcal{R}$, k sets of identifiers $\{\mathcal{ID}_1, \ldots, \mathcal{ID}_k\}$, k different arbitrary sets $\{A_1, \ldots, A_k\}$ and a set of k bijective functions $\{f_{\text{sec}}^{(1)} : A_1 \to \mathcal{ID}_1, \ldots, f_{\text{sec}}^{(k)} : A_k \to \mathcal{ID}_k\}$ by a set of three operations $\{\mathcal{ISO}_k.\mathsf{Associate}, \mathcal{ISO}_k.\mathsf{Read}, \mathcal{ISO}_k.\mathsf{ProvePossession}\}$. We define the properties of the $\mathcal{ISO}_k$ object as if they appear in a sequence seq:

— *Termination.* the invocation of the operations $\mathcal{ISO}_k.\mathsf{Associate}$, $\mathcal{ISO}_k.\mathsf{Read}$ and $\mathcal{ISO}_k.\mathsf{ProvePossession}$ terminate.

— $\mathcal{ISO}_k.\mathsf{Associate}$ *validity.* The operation $\mathcal{ISO}_k.\mathsf{Associate}(\{x_1, \ldots, x_k\}, res)$ invoked by the process p_i is valid if $\forall i \in \{1, \ldots, k\}, x_i \in A^{(i)}$, $res \in \mathcal{R}$ and no valid $\mathcal{ISO}_k.\mathsf{Associate}(\star, res)$ operation appears before in seq.

— ($\text{ISO}_k^{(i)}$.Read *validity.*) The operation $\mathcal{ISO}_k$.Read(res) invoked by the process p_i is always valid. It eventually returns $\{f_{\text{sec}}^{(1)}(x_1), \ldots, f_{\text{sec}}^{(k)}(x_k)\}$ if a valid $\mathcal{ISO}_k$.Associate($\{x_1, \ldots, x_k\}, res$) operation appears before in the sequence seq. Otherwise, it returns $\perp$.

— ($\text{ISO}_k^{(i)}$.ProvePossession *validity.*) The operation $\mathcal{ISO}_k$.ProvePossession($\{x_1, \ldots, x_k\}, res$) invoked by process p_i is valid if $\forall i \in \{1, \ldots, k\}$,$\mathcal{ISO}_k$.Read($res$) = $\{id_1, \ldots, id_k\}$ and $(f_{\text{sec}}^{(i)})^{-1}(id_i) = x_i$, where $(f_{\text{sec}}^{(i)})^{-1}$ is the inverse function of $f_{\text{sec}}^{(i)}$.

— *Identifier unicity.* Let $\mathcal{NO}$.Associate($\{x_1, \ldots, x_k\}, \star$) and $\mathcal{NO}$.Associate($\{y_1, \ldots, y_k\}, \star$) be two valid $\mathcal{NO}$.Associate operations. Then, $x_1 \neq y_1, \ldots, x_k \neq y_k$

7.9.2 Zooko's properties for an identifier system

With the definition of the Identifier System object, it is necessary to redefine the three properties of an identifier. The main difference is the Decentralized property. Whereas the human-choosable and the secure properties stand if only one identifier fulfills the property, the decentralization property only stands if all the identifiers of the Identifier System fulfill the decentralized property. As stated, the decentralized property ensures an agreement on the association between a resource and its identifiers. Thus, this agreement cannot be fulfilled if at least one identifier is not decentralized.

The human-choosable property states that a user can invert one of the $f_{\text{sec}}^{(i)}$ functions. It may be able to invert only one of the $f_{\text{sec}}^{(i)}$ or multiple ones.

Definition 7.9. Human-choosable. An $\mathcal{ISO}_k$ object fulfills the human-choosable property if any correct process p_i has access to at least one efficiently computable and deterministic algorithm $x_i \leftarrow \text{choose}_i(id_i)$ (where $i \in \{1, \ldots, k\}$) such that, if it invokes a valid $\mathcal{ISO}_k$.Associate($\{\star, \ldots, x_i, \ldots, \star\}, res$) operation, then $\mathcal{ISO}_k$.Read(res) outputs $\{\star, \ldots, id_i, \ \ldots, \star\}$.

The secure property of an ISO object is similar to that of a NO object, at the difference that the $\mathcal{ISO}$.ProvePossession operation requires the adversary to find the secrets associated with each identifier.

Definition 7.10. Secure. A Namespace object is secure if, given only $\{id_1, \ldots, id_k\}$ = $\mathcal{ISO}_k$.Read(res), the probability for a process p_i to invoke a valid $\mathcal{ISO}_k$.ProvePossession($\star, res$) operation is lower than ϵ, where ϵ is a negligible value that depends on a security parameter.

The decentralized property ensures that all the correct processes in the network see the same identifiers associated with a given resource. The system processes must agree on all the identifiers associated with this resource.

Definition 7.11. Decentralized. Let $|\Pi| = n > 1$. Let $p_i \in \Pi$ be a correct process. Let p_i invoke a valid $\mathcal{ISO}_k$.Associate($\{x_1, \ldots, x_k\}, res$) operation. An Identifier System object

is decentralized if the invocation of $\mathcal{ISO}_k$.Read(res) by a correct process $p_j, \forall j \in \{1, \ldots, n\}$ eventually returns $\{f_{\text{sec}}^{(1)}(x_1), \ldots, f_{\text{sec}}^{(k)}(x_k)\}$.

Unlike the Namespace object, an Identifier System object can be implemented with these three properties. As stated earlier, the nickname system is the most used identifier system, which associates a pointer with a nickname to achieve the three properties simultaneously. On the other hand, the petname system [129] that we presented earlier does not fulfill the three properties. This is due to the use of a petname, which is not decentralized. Because of the decentralized property of the Identifier System object, we know that if one of the identifiers used is not decentralized, then the whole Identifier System object does not fulfill the decentralized property.

In this thesis, we propose two implementations of a nickname system. The first is presented in Chapter 10 and is a classical nickname system object, which uses a pointer and a nickname to achieve all three properties. The second is presented in Section 8.7.1 and is a special type of nickname system without the human-choosable property. However, it is interesting as it can be implemented without a consensus algorithm in a fully asynchronous system. Its goal is to reduce the entropy of names attributed to processes when contention is low. Rather than being identified by a public key, resources are identified by substrings of these keys. Thus, even if names are not human-choosable, they are easier for humans to handle than public keys.

7.10 Conclusion

In this chapter, we proposed a formalization of the naming problem where names are scarce and resources to identify are potentially infinite. We identified two types of objects. The Namespace object aims to identify a resource with only one identifier, *i.e.*, with only one string of bits. In contrast, the Identifier System object uses multiple identifiers to identify a unique resource. We formally proved that it is impossible to implement a Namespace object that is simultaneously secure, decentralized, and human-choosable. On the other hand, an Identifier System object can be implemented with these three properties.

In this chapter, we also proved that the Namespace object (and, by extension, the Identifier System object) has an infinite consensus number. However, pointers (identifiers that are long random strings of bits) can be probabilistically implemented with an object of consensus number 1 if the probability of collision between two pointers is sufficiently small. We use this result in Section 8.7.1 to implement a new identifier system based on the CAC abstraction. This implementation makes building a specific kind of nickname system possible without consensus.

Identifier systems are one of the main components of Identity Management Systems. They are used to identify resources linked to an individual that may evolve, such as revocation lists or lists of devices authorized to use the verifiable credentials of a given individual.

Chapter 8

A cooperation abstraction when contention is unlikely: the Context Adaptive Cooperation abstraction

This chapter presents CAC, a new cooperation abstraction that reduce synchronization requirements between processes when contention is low. This abstraction is then used to build two algorithms: a consensus algorithm and a naming algorithm. It was written in collaboration with Timothé Albouy, Davide Frey, Michel Raynal and François Taïani. It is currently in a submission process. A version of the paper can be found on arXiv [130].

8.1 Introduction

On distributed abstractions Distributed computing is the science of algorithm-based cooperation. It consists in defining (using precise specifications) and implementing distributed abstractions (distributed objects) that allow a set of computing entities (denoted processes, nodes, peers, actors, *etc.*) to cooperate to reach a common goal. Considering asynchronous n-process message-passing systems, this chapter is on cooperation abstractions that have to cope with Byzantine processes (*i.e.*, processes that may behave arbitrarily, as defined in [131, 132]).

Using Lamport's Paxos terminology [133] when building a cooperation abstraction, all the processes are *acceptors*, and some processes are *proposers*. A proposer contributes to launching a cooperation abstraction instance (*e.g.*, by broadcasting a message), while an acceptor locally (partially or fully) terminates this instance by accepting/deciding/delivering a value.

Due to the very nature of distributed computing, many cooperation abstractions have been proposed, each defining a specific distributed object (see, for example, the textbooks [134, 135, 136] where many of them are described). Two of these cooperation abstractions are particularly important: reliable broadcast [73, 25] and consensus [22, 132] (c.f., Chapter 4).

— In Byzantine Reliable Broadcast (BRB for short), a single process acts as a proposer

while all processes are acceptors. BRB is, therefore, a one-to-all cooperation abstraction in which the non-Byzantine processes must agree on the same value, and if the proposer is not Byzantine, that value is the one proposed by the proposer. It is shown in [73, 25] that BRB can be implemented in an n-process system in the presence of asynchrony and up to $t < n/3$ Byzantine processes. Moreover, this bound on t is tight.

— In the consensus cooperation abstraction, every process is both a proposer and an acceptor. Each process is assumed to propose a value, and the non-Byzantine processes must accept/decide/deliver the very same value (that must satisfy some properties involving the values proposed by the non-Byzantine processes). Despite its practical interest in implementing many fault-tolerant applications (*e.g.*, distributed state machines), consensus is impossible to solve deterministically in the presence of asynchrony if only one process may crash [22]. Due to the practical interest of consensus, several approaches have been proposed to circumvent this impossibility. One such approach enriches the system with randomization (*e.g.*, [137]). Another one extends the system with synchrony assumptions that are as weak as possible (*e.g.*, [138]). A third strategy adds minimal information regarding failures (this is the failure-detector-based approach introduced in [139]).

Crucially, in both BRB and consensus, a non-Byzantine process locally accepts/decides/delivers only one value, which is the same for all the non-Byzantine processes.

Content of this chapter BRB and consensus can be seen as two abstractions that lie at the two ends of a spectrum of cooperation abstractions. This article presents an intermediary cooperation abstraction denoted *Context-Adaptive Cooperation* (CAC), which is formally defined in Section 8.4. Adopting an operational approach, we give below an informal presentation for the reader to get an intuition of it.

— Within a CAC instance, some arbitrary number d (where $1 \leq d \leq n$) of processes propose a value. The number d of actual proposers within a CAC instance is unknown to participating processes, it depends on the run.

— In the course of a CAC execution, each non-Byzantine process accepts pairs $\langle v, i \rangle$ (where v is the value proposed by p_i) so that, eventually, they all accept the same set of ℓ pairs, where $1 \leq \ell \leq d$. A process accepts values one after the other in some arbitrary order, which may vary from one process to another process.

As no process knows d and ℓ, a process can never conclude it has accepted the ℓ pairs composing the final set of accepted pairs (except in some specific cases that will be discussed later). Actually, the values d and ℓ depend on the run (unknown number of proposers, asynchrony and behavior of Byzantine processes). Nevertheless, the reader can see that (1) a CAC instance in which a single process proposes a value boils down to BRB, and (2) a CAC instance in which

Abstraction	Implementation	Byzantine resilience	Asynchronous rounds
BRB	Bracha [73]	$n > 3t$	3
	Imbs-Raynal [140]	$n > 5t$	2
CAC	Section 8.5	$n > 4t$	3
	Section 8.6	$n > 3t$	3
		$n > 5t$	2

Table 8.1 – Good-case latency of sig-free BRB and sig-based CAC w.r.t Byzantine resilience.

all correct processes propose the same value boils down to a particular consensus-like instance. [1]

Interestingly, in addition to eventually providing the same set of ℓ pairs $\langle v, i\rangle$ to non-Byzantine processes, the CAC abstraction provides each process with an imperfect oracle that provides information about the set of pairs that might get accepted in the future. In some particular circumstances, this second set allows processes to know that they have converged to the final set of accepted pairs, and that they will not accept any other pair. CAC thus falls in an intermediate class of cooperation abstractions that dynamically adapts to the number of proposers and their proposed values.

After formally presenting the CAC abstraction, this chapter presents two implementations. One that is easy to implement but sub-optimal (Section 8.5), and a second one that finishes in three asynchronous rounds when $n > 3t$ and finishes in two asynchronous rounds in the best cases if $n > 5t$ (Section 8.6). Hence, this implementation has the same implementation cost as signature-free BRB [73, 140] (see Section 8.1, where "sig" stands for "signature"). This cost is obtained with the help of signatures [2], but without considering other assumptions on synchrony or failures.

The CAC abstraction to address low contention problems The CAC abstraction aims at solving distributed problems with limited contention efficiently. More precisely, we consider objects whose state depends on the order of the operations executed, but only a subset of the operations actually needs to be ordered. The former solution to those problems was using a full-fledged consensus algorithm, thus requiring additional computability power (*e.g.*, partial synchrony or randomization).

When the probability of contention is low, it is more interesting to rely on a lightweight asynchronous cooperation abstraction in good cases, and fall back to consensus only when needed. The CAC abstraction is tailored for such scenarios: thanks to the information provided by the abstraction about the values that might be accepted in the future, processes know if there is contention and which processes are competing. Hence, if there is no contention, they can decide whether or not they can stop prematurely, and if there is contention, they can resolve it by only

1. It is important to notice that this abstraction is less powerful than the consensus, namely, it can be deterministically implemented in failure-prone asynchronous systems [73], while consensus cannot [22].

2. Which is limited in $\mathsf{poly}(\kappa)$, where κ is a security parameter.

communicating with the other processes involved in the conflict, thus improving efficiency. Furthermore, and unlike other optimistically terminating consensus algorithms [141, 142, 143, 144, 145], the use of CAC makes it possible to terminate optimistically even if multiple (different) values are proposed (see Section 8.7).

Interestingly, when there exists a deterministic back-off strategy that can be implemented without consensus, then the CAC abstraction makes it possible to implement fully asynchronous agreement algorithms without synchronization, whereas other solutions would have required the use of consensus. Section 8.7.1 illustrates such an usage of the CAC abstraction. It provides a CAC-based solution to the novel short-naming problem in asynchronous networks, where processes seek to get names with the lowest possible information-theoretic entropy.

Finally, the distributed objects presented in Chapter 6 require synchronization between specific processes. Therefore, the CAC abstraction is particularly interesting to implement them as it makes it possible to determine "on-the-fly" when synchronization is required or not. This behavior of the CAC abstraction is used in Chapter 9 and Chapter 10 to decrease synchronization requirements of PPfDIMS and to increase algorithms efficiency.

Roadmap This chapter consists of 8 sections. Section 8.2 presents the related works, Section 8.3 defines the computing model and Section 8.4 gives a property-based definition of the CAC cooperation abstraction. Then, on the feasibility side, Section 8.5 presents a relatively simple implementation of the CAC abstraction that assumes $t < n/4$. Section 8.6 presents a more efficient implementation that can exhibit optimal Byzantine resilience ($n \geq 3t + 1$) or that can lower the Byzantine resilience for better efficiency. Section 8.7 presents an efficient CAC-based solution of the Cascading Consensus problem. Finally, Section 8.8 concludes the article.

8.2 Related work

The work described in this chapter places itself in the context of *fast/adaptive cooperation distributed algorithms* where an arbitrary, *a priori* unknown subset of processes try to modify a shared object. These algorithms seek to terminate as rapidly as possible in favorable circumstances (*e.g.*, no or few faults, no or little contention) and with as little as possible actions from non-participating processes while maintaining strong safety guarantees in the general case. Such algorithms have been investigated in earlier works.

At the very beginning: mutex algorithms in failure-free systems As far as we know, the notion of *fast concurrent algorithm* was introduced by Lamport [146] in the context of mutual exclusion in failure-free read/write shared memory systems. The idea was to have a complexity (in term of shared memory accesses) depending on the number of processes that actually compete to enter the critical section, rather than a complexity linear in the total number

of processes in the system. More precisely, the previously cited paper presents a mutex algorithm that requires only seven accesses to the shared memory when a call to the critical section occurs in a competition-free context. This notion was then generalized so that the number of memory accesses depends on the number of processes that are concurrently competing to access the critical section. A theoretical foundation of this approach is presented in [147]. Considering shared memory systems, the reader will find more developments of this approach in [148, 149].

Fast/adaptive consensus algorithms in message-passing asynchronous crash-prone/Byzantine systems As stated in [133] (which introduces the fast Paxos algorithm), the notion of fast consensus algorithm in crash-prone message-passing asynchronous system was introduced in [150]. This algorithm was then extended to Byzantine asynchronous systems in [142]. Many efficiency-oriented Byzantine consensus algorithms have since been designed (*e.g.*, [141, 151, 152, 145, 153] to cite a few).

Structuring the space of weak agreement problems The algorithms just discussed are specific to a single problem. In [154], Attiya and Welch go one step further and introduce a new problem termed *Multivalued Connected Consensus*, which captures a range of weak agreement problems such as crusader agreement [155], graded broadcast [156] and adopt-commit agreement [157]. Differently from consensus, these agreement problems can be solved without requiring additional computational power such as synchrony constraints [138], randomization [158], or failure detectors [139].

Interestingly, the decision space of these weak agreement problems can be represented as a spider graph. Such a graph has a central clique (which can be a single vertex) and a set of $|V|$ paths (where V is a finite set of totally ordered values) of length R. Two asynchronous message-passing algorithms that solve Multivalued Connected Consensus are described in [154]. Let n be the number of processes and t the maximal number of processes that can fail. The first algorithm considers crash failures and assumes $t < n/2$, and the second considers Byzantine failures and assumes $t < n/5$. For both of them, the instance with $R = 1$ solves crusader agreement, and the instance $R = 2$ solves graded broadcast and adopt-commit.

8.3 Model

The model used in this chapter is the message passing model presented in Section 4.1.2. We consider one difference compared to this model. In this chapter, we only interest ourselves in the distributed algorithms maintained by processes in Π. We do not interest ourselves in the issuers, the verifiers, or the users. Furthermore, unlike in Chapter 5, in this section, we consider perfect cryptographic primitives that cannot be forged; namely, we assume an unforgeable signature scheme resistant against chosen message attacks.

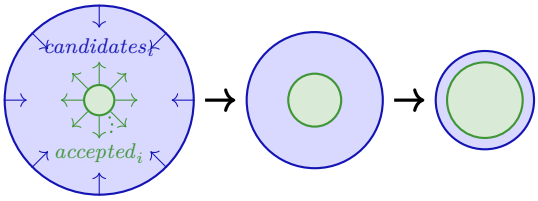


Figure 8.1 – During an execution, the $accepted_i$ and $candidates_i$ sets of a correct process p_i monotonically grows and shrinks, respectively.

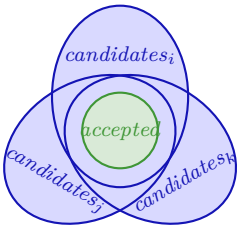


Figure 8.2 – After the execution, the *accepted* set is the same for all correct processes and is contained in the intersection of their *candidates* sets.

8.4 Context-Adaptive Cooperation: Definition

8.4.1 Definition

The Context Adaptive Cooperation (CAC) object provides each process p_i with (1) an operation denoted cac_propose that allows it to propose a value and (2) two sets denoted $accepted_i$ and $candidates_i$. When a process p_i invokes cac_propose(v), we say that "p_i cac-proposes (in short "proposes") the value v" (for clarity sometimes we also say that "p_i cac-proposes the pair $\langle v, i\rangle$"). When a pair $\langle v, j\rangle$ is added to the set $accepted_i$ of a process p_i, we say that "p_i cac-accepts (in short "accepts") $\langle v, j\rangle$". For the sake of simplicity, when a pair $\langle v, j\rangle$ is cac-accepted, a cac_accept(v, j) callback is triggered.

— The set $accepted_i$ is initially empty. It then grows monotonically, progressively adding a pair $\langle v, j\rangle$ for each value v that is cac-accepted by p_i from p_j. Eventually, $accepted_i$ contains all the pairs $\langle v, j\rangle$ accepted by the CAC abstraction (and only them).

— The set $candidates_i$ is initialized to $\top$, where $\top$ is defined as a symbolic value representing the identity element of the $\cap$ operation.[3] Then, $candidates_i$ shrinks monotonically, and contains a dynamic estimation of the pairs $\langle v, j\rangle$ that have been or will be cac-accepted by process p_i. Hence, $accepted_i \subseteq candidates_i$ always holds. More concretely, $candidates_i$ contains all the pairs $\langle v, j\rangle$ that have been already added to the $accepted_i$ set locally by p_i along with some pairs $\langle v, k\rangle$ that may (or may not) be added to the set $accepted_i$ later on. Formally, if τ_1 and τ_2 are two arbitrary time points in the execution of p_i (in no particular order, *i.e.*, with either $\tau_1 \leq \tau_2$ or $\tau_1 \geq \tau_2$) and $x_i^{\tau_k}$ represents the value of variable x_i at time τ_k, then $candidates_i$ verifies $accepted_i^{\tau_2} \subseteq candidates_i^{\tau_1}$. As a result, if a pair $\langle v, k\rangle$ is not in $candidates_i$ at some point, p_i will never cac-accept this pair. Furthermore, if at some point τ, p_i observes $accepted_i^{\tau} = candidates_i^{\tau}$, then p_i knows it has cac-accepted all values for this CAC instance. Let us notice that this case may never happen (see Section 8.4.2). The behavior of both types of sets is summarized in Figures 8.1 and 8.2.

3. That is to say, for any set S, $S \cap \top = \top \cap S = S$, and the statement $S \subseteq \top$ is always true.

CAC specification Given a correct process p_i and its associated $candidates_i$ and $accepted_i$ sets, the following properties define the CAC abstraction. [4]

- CAC-Validity. If p_i and p_j are correct, $candidates_i \neq \top$ and $\langle v, j\rangle \in candidates_i$, then p_j cac-proposed value v.
- CAC-Prediction. For any correct process p_i and for any process identity k, if, at some point of p_i's execution, $\langle v, k\rangle \notin candidates_i$, then p_i *never* cac-accepts $\langle v, k\rangle$ (*i.e.*, $\langle v, k\rangle \notin accepted_i$ holds forever).
- CAC-Non-triviality. For any correct process p_i, $accepted_i \neq \varnothing$ implies $candidates_i \neq \top$.
- CAC-Local-termination. If a correct process p_i invokes cac_propose(v), its set $accepted_i$ eventually contains a pair $\langle v', \star\rangle$ (note that v' is not necessarily v).
- CAC-Global-termination. If p_i is a correct process and $\langle v, j\rangle \in accepted_i$, eventually $\langle v, j\rangle \in accepted_k$ at every correct process p_k.

The CAC-Validity property states that, if a correct process cac-accepts a pair $\langle v, j\rangle$ from a correct process p_j, then p_j cac-proposed value v, *i.e.*, there is no identity theft for correct processes. The CAC-Prediction property states that, if a correct process p_i cac-accepts a pair $\langle v, j\rangle$, then $\langle v, j\rangle$ was present in $candidates_i$ from the start of p_i's execution. In other words, $candidates_i$ provides information about the pairs that might be accepted by p_i in the future. In particular, if, at some point of p_i's execution, some pair $\langle v', k\rangle$ is no longer in $candidates_i$, then p_i will never accept $\langle v', k\rangle$. (However, the converse is generally not true; the prediction provided by $candidates_i$ is, as such, imperfect.) This property is at the core of the cooperation provided by a CAC object. The CAC-Non-triviality property ensures that a trivial implementation that never updates $candidates_i$ is excluded. As soon as some process p_i has accepted some pair $\langle v, k\rangle$, its $candidates_i$ set must contain some explicit information about the pairs that might get accepted in the future. [5]

The CAC-Local-termination property states that if a correct process cac-proposes a value v, its $accepted_i$ set will not remain empty. Notice that this does not mean that the pair $\langle v, i\rangle$ will ever be added to $accepted_i$. Finally, the CAC-Global-termination property states that eventually, the *accepted* sets of all correct processes are equal. Let us notice that, in general, no process p_i can know when no more pairs will be added to its set $accepted_i$.

8.4.2 Termination of the CAC abstraction

It follows from the definition that, for some correct process p_i, if $candidates_i = accepted_i$, then p_i will not cac-accept any new pair $\langle v, j\rangle$. If we use the notations from Section 8.1, we see

4. We present in the following a single-shot version of the CAC abstraction in which a process can propose at most one value. It can easily be extended to a multi-shot version using execution identifiers such as sequence numbers.

5. Ignoring the symbolic value $\top$, $accepted_i$ and $candidates_i$ remain finite sets throughout p_i's execution.

that $d = \ell = |candidates_i| = |accepted_i|$. In this specific case, p_i can detect without ambiguity that the CAC execution has terminated. We say that p_i knows it terminated.

The most obvious example of "known termination" is when only one process cac-proposes (or is perceived to cac-propose) a value. In this case, by CAC-Validity, $|candidates_i|$ eventually equals 1. In this specific setting, the CAC abstraction boils down to the BRB abstraction.

However, in the general case, there might be runs where $|candidates_i| > |accepted_i|$ during the whole execution of the abstraction. In this case, p_i will not be able to know if it has terminated or if new pairs might be added to the $accepted_i$ set. This is an inherent property of the CAC abstraction, but, as we will see in Section 8.7, this does not prevent the abstraction from being appropriate to solve complex coordination problems.

Another side effect of the abstraction is that, it is possible for a correct process p_j to know it terminated because $candidates_i = accepted_i$, while some other correct process p_j might never detect its own termination, because $|candidates_j| > |accepted_j|$ during the whole run.

8.4.3 CAC with proof of acceptance

The properties of the CAC abstraction imply that processes cac-accept pairs asynchronously and in different orders. In some applications, correct processes must prove to others that they have legitimately cac-accepted some pair $\langle v, j \rangle$. To support such use cases, the CAC definition can be enriched with transferable *proofs of acceptance* that a process can use to convince others that the underlying algorithm has been respected.

When using proofs of acceptance, the elements in the $accepted_i$ sets become triplets $\langle v, j, \pi_v \rangle$, where π_v is a cryptographic construct that serves as proof that $\langle v, j \rangle$ was added to $accepted_i$ while following the prescribed algorithm. We say that π_v is valid if there exists a function Verify such that, for any value v and any proof of acceptance π_v pertaining to v, the following property holds:

$$\mathsf{Verify}(v, \pi_v) = \texttt{true} \iff \exists\ p_i \text{ correct such that, eventually, } \langle v, \star, \pi_v \rangle \in accepted_i.$$

When $\mathsf{Verify}(v, \pi_v) = \texttt{true}$, we say that π_v is *valid*, and by extension that the triplet $\langle v, j, \pi_v \rangle$ is valid. When using proofs of acceptance, all properties of the CAC abstraction are modified to use $\langle v, j, \pi_m \rangle$ triplets. In this case, the *accepted* sets contain triplets (the cac_propose operation and the *candidates* sets remain unchanged).

8.5 CAC: a simple, sub-optimal implementation

8.5.1 A simple CAC algorithm

This section presents an implementation of the CAC abstraction for $n > 4t$. Although this implementation is sub-optimal from a resilience perspective, its goal is to convince the reader that the CAC abstraction can be implemented in an easy-to-understand manner. [6]

Algorithm 5 works in two phases (the *witness* phase and the *ready* phase) which each use a specific type of signature (WITSIG and READYSIG). During the witness phase, p_i disseminates WITSIG$(p_i, \langle v, j\rangle)$ to acknowledge that a value v was cac-proposed by process p_j. As it is signed by p_i, it cannot be forged. During the ready phase, processes exchange READYSIG signatures to collect information about potential competing values that have been cac-proposed simultaneously, in order to ensure the CAC-PREDICTION property. A READYSIG(p_i, M_i) signature by p_i embeds a set M_i containing a critical mass of WITSIG signatures. Correct processes need to gather enough READYSIG signatures in order to construct their *candidates* and *accepted* sets.

The algorithm works as follows. When a process p_i invokes cac_propose(v), it first verifies that it has not not already cac-proposed a value, or that it did not already be_broadcast any WITSIG (line 3). If this verification passes, p_i produces a WITSIG for the pair $\langle v, i\rangle$, and be-broadcasts it in a BUNDLE message. This type of message can simultaneously carry WITSIG and READYSIG. As a result, each correct process disseminates its complete current knowledge whenever it be-broadcasts a BUNDLE message. Eventually, this WITSIG will be received by all the correct processes.

Let us consider a correct process p_j that receives the BUNDLE message, which contains the signature WITSIG$(p_i, \langle v, i\rangle)$. Firstly, p_j saves all the valid signatures into the $sigs_j$ variable. Secondly, if p_j did not already sign (and be-broadcast) a WITSIG, it produces a WITSIG for the pair $\langle v, i\rangle$ and be-broadcasts it (lines 10-12). If there are multiple signatures on $\langle v, \star\rangle$ in $sigs_j$, line 11 imposes that the p_j chooses and signs only one of those pairs. Thirdly, p_j checks whether it can sign and send a READYSIG. When it receives WITSIG signatures from more than $n-t$ processes, p_j produces a READYSIG on a set of messages M_j and disseminates it (lines 13-16). M_j contains all the WITSIG received by p_j. This READYSIG is added to the $sigs_j$ set and be-broadcast in a BUNDLE message. Hence, the information about the WITSIGs known by p_j will be received by every correct process along with the READYSIG, thus ensuring the CAC-GLOBAL-TERMINATION property.

Finally, p_i verifies if it can cac-accept a value. To this end, it waits for READYSIG signatures from $n-t$ processes, then it cac-accepts all values that are present in at least $2t+1$ sets M (lines 17-21). The $2t+1$ bound ensures that, if p_i cac-accepts a value later on, then it has already

6. A CAC algorithm that is optimal in terms both of Byzantine resilience and round complexity is presented in Section 8.6. This second algorithm also fulfills the proof of acceptance property.

1 **init:** $sigs_i \leftarrow \varnothing$; $candidates_i \leftarrow \top$; $accepted_i \leftarrow \varnothing$; $M_i \leftarrow \varnothing$.
2 **operation** cac_propose(v) **is**
3 **if** there are no signature by p_i in $sigs_i$ **then**
4 $sigs_i \leftarrow sigs_i \cup \{\text{WITSIG}(p_i, \langle v, i\rangle)\}$; ▷$p_i$ *signs* $\langle v, i\rangle$ *using a* WITSIG *signature*
5 be_broadcast BUNDLE($sigs_i$).
6 **when** BUNDLE($sigs$) **is** received **do**
7 $valid_i \leftarrow \{$all valid signatures in $sigs\}$;
8 **if** $\exists\ p_k, k : \text{WITSIG}(\star, \langle v_k, k\rangle) \in valid_i \wedge \text{WITSIG}(p_k, \langle v_k, k\rangle) \notin valid_i$ **then** return ;
9 $sigs_i \leftarrow sigs_i \cup valid_i$;
10 **if** $\exists\ p_j : \text{WITSIG}(p_j, \langle v, j\rangle) \in sigs_i \wedge \text{WITSIG}(p_i, \langle \star, \star\rangle) \notin sigs_i$ **then**
11 $sigs_i \leftarrow sigs_i \cup \{\text{WITSIG}(p_i, \text{choice}(\{\langle v', k\rangle \mid \text{WITSIG}(p_k, \langle v', k\rangle) \in sigs_i\}))\}$;
 ▷choice *chooses one of the elements in the set given as argument.*
12 be_broadcast BUNDLE($sigs_i$);
13 **if** $|\{j \mid \text{WITSIG}(p_j, \langle \star, \star\rangle) \in sigs_i\}| \geq n - t \wedge \text{READYMSG}(p_i, \star) \notin sigs_i$ **then**
14 $M_i \leftarrow \{\text{WITSIG}(\star, \langle \star, \star\rangle) \in sigs_i\}$;
15 $sigs_i \leftarrow sigs_i \cup \{\text{READYSIG}(p_i, M_i)\}$; ▷$p_i$ *signs* M_i *using a* READYSIG *signature*
16 be_broadcast BUNDLE($sigs_i$);
17 **if** $|\{j \mid \text{READYSIG}(\star, p_j) \in sigs_i\}| \geq n - t$ **then**
18 be_broadcast BUNDLE($sigs_i$);
19 **if** $candidates_i = \top$ **then** ▷*first time a value is accepted*
20 $candidates_i \leftarrow \{\langle v, k\rangle \mid \exists\ M : \text{READYSIG}(\star, M) \in sigs_i \wedge \text{WITSIG}(\star, \langle v, k\rangle) \in M\}$;
21 $accepted_i \leftarrow \left\{ \langle v, k\rangle \in candidates_i \;\middle|\; \begin{array}{l} 2t+1 \textit{ distinct} \text{ processes } p_s \text{ have signed} \\ \text{READYSIG}(\star, M_s) \text{ in } sigs_i \text{ such that} \\ \text{WITSIG}(\star, \langle v, k\rangle) \in M_s \end{array} \right\}$;
22 **for** all pairs $\langle v, k\rangle$ that have just been added to $accepted_i$ **do** cac_accept(v, k).

Algorithm 5: One-shot sig-based CAC implementation assuming $n > 4t$ (code for p_i)

been added to the $candidates_i$ set, thus ensuring the CAC-PREDICTION property. A cac_accept callback is triggered at this point, it is used by algorithms that builds upon CAC to know when new values are added to the *candidates* set.

8.5.2 Proof of the algorithm

The proof that Algorithm 5 is a signature-based implementation of the CAC abstraction under the assumption $n > 4t$ follows from the following lemmas.

In the following, var_x^τ denotes the value of variable *var* at process p_x at time point τ. In particular,

— $sigs_j^\tau$, $candidates_j^\tau$ and $accepted_j^\tau$ are the values of $sigs_j$, $candidates_j$ and $accepted_j$ at time τ,

— $sigs_i^\tau$ is the value of $sigs_i$ at time τ.

Lemma 8.1 (CAC-VALIDITY)**.** If p_i and p_j are correct processes, $candidates_i \neq \top$ and $\langle v, j\rangle \in candidates_i$, then p_j cac-propose value v.

Proof. Let p_i and p_j be two correct processes. If $candidates_i \neq \top$, it implies p_i executed the line 20. Furthermore, if a tuple $\langle v, j\rangle$ is still in the $candidates_i$ set after the execution of line 20 by p_i, it means that there exists a WITSIG$(p_j, \langle v, j\rangle)$ in $sigs_i$ thanks to line 8. We assume p_j is correct. Hence, due to the unforgeability assumption of cryptographic signatures, the only process able to produce such a signature is p_j itself. The only place in the algorithm where a correct process can produce such a signature is during a cac_propose(v) invocation. □

Lemma 8.2. For any two correct processes p_i and p_j, a process p_k (possibly Byzantine), and a value v_k, if $sigs_i$ contains READYSIG$(\star, M_s)$ signatures, $s \in \{1, \ldots, n\}$, from $2t+1$ distinct processes with WITSIG$(\star, \langle v_k, k\rangle) \in M_s$ for all s, then $\langle v_k, k\rangle \in candidates_j$ from the start of p_j's execution.

Proof. Let p_i and p_j be two correct processes, p_k a process (possibly Byzantine), and v_k a value. Assume that at some point τ of p_i's execution $sigs_i$ contains READYSIG$(\star, M_s)$ signatures from $2t+1$ distinct processes such that WITSIG$(\star, \langle v, k\rangle) \in M_s$ for all s, i.e.

$$\left|\{p_s \mid \text{READYSIG}(p_s, M_s) \in sigs_i^\tau : \text{WITSIG}(\star, \langle v_k, k\rangle) \in M_s\}\right| \geq 2t+1. \tag{8.1}$$

Let $\langle v_\ell, \ell\rangle$ be the first value cac-accepted by p_j at line 21. The proof considers two cases according to two time periods: the period before p_j accepts $\langle v_\ell, \ell\rangle$ (Case 1), and the period after (Case 2).

— Case 1. In this case, before p_j executes lines 20 and 21 for $\langle v_\ell, \ell\rangle$, $candidates_j$ retains its initial value, namely $\top$, and by definition of $\top$, $\langle v_k, k\rangle \in candidates_j$, the lemma holds.

— Case 2. Let's now turn to the value of $candidates_j$ after p_j has accepted $\langle v_\ell, \ell\rangle$. Let τ_ℓ be the time when p_j adds $\langle v_\ell, \ell\rangle$ to $accepted_j$. We show that $\langle v_k, k\rangle \in candidates_j^{\tau_\ell}$ when p_j accepts $\langle v_\ell, \ell\rangle$. The proof considers two sets of processes, noted A and B.
  — A is the set of processes whose READYSIG signatures are known to p_j at time point τ_ℓ. Because of the condition at line 17, to add $\langle v_\ell, \ell\rangle$ to $accepted_j^{\tau_\ell}$, A must contain at least $n - t$ processes.
  — B is the set of processes p_s that have signed a READYSIG(p_s, M_s) signature with WITSIG$(\star, \langle v_k, k\rangle) \in M_s$, so that this READYSIG signature is known to p_i at time point τ. From eq. (8.1), B contains at least $2t + 1$ distinct processes.

  We have $|A \cap B| = |A| + |B| - |A \cup B| \geq (n - t) + (2t + 1) - n = t + 1$ processes, which means that there is at least one correct process $p_r \in A \cap B$. Because $p_r \in A$, p_r has signed READYSIG(M_r, r) which was received by p_j by time τ_ℓ, hence READYSIG$(M_r, r) \in sigs_j^{\tau_\ell}$. Furthermore, because p_j is correct, and because $\langle v_\ell, \ell\rangle$ was the first value cac-accepted by p_j, $candidates_j$ was updated at time τ_ℓ at line 20, from which point onward the following holds:

$$\{\langle v, s\rangle \mid \text{WITSIG}(\star, \langle v, s\rangle) \in M_r\} \subseteq candidates_j. \tag{8.2}$$

  Because $p_r \in B$, p_r has signed READYSIG(p_r, M'_r) which was received by p_i by time τ and where WITSIG$(\star, \langle v_k, k\rangle) \in M'_r$. Because p_r is correct, due to the condition at line 13, it only produces at most one READYSIG$(p_r, \star)$ signature, therefore $M_r = M'_r$, and WITSIG$(\star, \langle v_k, k\rangle) \in M_r$. By eq. (8.2), $\langle v_k, k\rangle \in candidates_j^{\tau_\ell}$. Due to the condition at line 19, $candidates_j$ is only updated once, when $\langle v_\ell, \ell\rangle$ is accepted by p_j, as a result $\langle v_k, k\rangle \in candidates_j$ after p_j accepts $\langle v_\ell, \ell\rangle$, which concludes the lemma. □

Lemma 8.3 (Extended Prediction)**.** For any two correct processes p_i and p_j, if $\langle v, k\rangle \in accepted_i$ then $\langle v, k\rangle \in candidates_j$ from the start of p_j's execution.

Proof. Let p_i and p_j be two correct processes. Assume that $\langle v_k, k\rangle \in accepted_i$. Consider the set of processes $S = \{p_s\}$ that have signed a READYSIG$(\star, M_s)$ signature with WITSIG$(\star, \langle v_k, k\rangle) \in M_s$, so that this READYSIG signature is known to p_i when it accepts $\langle v_k, k\rangle$. By construction of $accepted_i$ at line 21, S contains at least $2t + 1$ distinct processes. lemma 8.2 applies, concluding the proof. □

Corollary 8.1 (CAC-PREDICTION)**.** For any correct process p_i and for any process identity k, if, at some point of its execution, $\langle v, k\rangle \notin candidates_i$, then p_i *never* cac-accepts $\langle v, k\rangle$ (*i.e.*, $\langle v, k\rangle \notin accepted_i$ holds forever).

Proof. The corollary follows from the contrapositive of Lemma 8.3 when $p_j = p_i$. □

Lemma 8.4 (CAC-Non-triviality)**.** If process p_i is correct, $accepted_i \neq \emptyset$ implies $candidates_i \neq \top$.

Proof. This is an immediate consequence of lines 20-21 where, if $candidates_i = \top$, it is set to a non-$\top$ value before that $accepted_i$ is updated. □

Lemma 8.5. If a correct process p_i cac-proposes a value v, then each correct process p_j

- broadcasts its own witSig$(p_j, \langle\star, \star\rangle)$ signature in a BUNDLE message at *line 5* or *12*;
- broadcasts its own readySig(p_j, M_j) signature in a BUNDLE message, with $|M_j| \geq n-t$, at *line 16*;
- eventually receives readySig signatures from at least $n - t$ distinct processes.

Proof. Suppose p_i has cac-proposed a value v. In that case, it has necessarily broadcast a BUNDLE$(sigs_i)$ message, where $sigs_i$ contains a signature witSig$(p_i, \langle\star, \star\rangle)$, either during the invocation of cac_propose(v) at line 5 (if it has not done it previously) or during the handling of a received BUNDLE message at line 12. All correct processes will therefore broadcast a BUNDLE message containing their own witSig$(\star, \langle\star, \star\rangle)$ signature, either because they have received p_i's BUNDLE$(sigs_i)$ message, because they have received the BUNDLE message of another process, or because they invoked the cac_propose operation themselves before receiving any valid BUNDLE message. As all $c \geq n - t$ correct processes broadcast their own witSig$(\star, \langle\star, \star\rangle)$ signature in a BUNDLE message, all correct processes eventually receive these messages (thanks to the best effort broadcast properties and since the network is reliable) and pass the condition at line 13. This implies that all correct processes sign and broadcast their own readySig$(\star, M)$ signature in a BUNDLE message (at lines 15 and 16). M contains the whole list of witSig received so far, due to condition at line 13, $|M| \geq n - t$. As with witSig signatures, these messages are eventually received by all correct processes, which eventually receive readySig signatures from at least $n - t$ distinct processes and pass the condition at line 17. □

Lemma 8.6. Let C be a set such that $|\mathsf{C}| \leq c$ (with $c > 0$), where $c \geq 3t + 1$. Let $\mathcal{S} = \{\mathsf{S}_1, \ldots, \mathsf{S}_c\}$ be a set of c subsets of C that each contain at least $c - t$ elements, i.e. $\forall i \in \{1, \ldots, c\}, |\mathsf{S}_i| \geq c - t$. Then, there is at least one element $e \in \mathsf{C}$ that appears in at least $2t + 1$ sets S_i, i.e.

$$\exists\, e \in \mathsf{C} : |\{\mathsf{S}_i \mid e \in \mathsf{S}_i\}| \geq 2t + 1.$$

Proof. We prove Lemma 8.6 by contradiction. Let us assume there are no element $e \in \mathsf{C}$ that appears in at least $2t + 1$ sets S_i. This implies that, in the best case, each element of C appears

at most in $2t$ of the sets in $\mathcal{S} = \{\mathsf{S}_1, \ldots, \mathsf{S}_c\}$, *i.e.*

$$\begin{aligned} &\forall\ e \in \mathsf{C} : |\{\mathsf{S}_i \mid e \in \mathsf{S}_i\}| \leq 2t, \\ &\forall\ e \in \mathsf{C} : \sum_{\mathsf{S}_i \in \mathcal{S}} \mathbb{1}_{\mathsf{S}_i}(e) \leq 2t, \end{aligned} \tag{8.3}$$

where $\mathbb{1}_{\mathsf{S}_i}$ is the indicator function for the set S_i, i.e.

$$\mathbb{1}_{\mathsf{S}_i}(e) = \begin{cases} 1 \text{ if } e \in \mathsf{S}_i, \\ 0 \text{ otherwise.} \end{cases} \tag{8.4}$$

For each $\mathsf{S}_i \in \mathcal{S}$, we further have $\mathsf{S}_i \subseteq C$ and therefore

$$|\mathsf{S}_i| = \sum_{e \in C} \mathbb{1}_{\mathsf{S}_i}(e). \tag{8.5}$$

Combining eq. (8.5) and eq. (8.3) yields

$$\sum_{\mathsf{S}_i \in \mathcal{S}} |\mathsf{S}_i| = \sum_{\mathsf{S}_i \in \mathcal{S}} \sum_{e \in C} \mathbb{1}_{\mathsf{S}_i}(e) = \sum_{e \in C} \sum_{\mathsf{S}_i \in \mathcal{S}} \mathbb{1}_{\mathsf{S}_i}(e) \tag{8.6}$$

(by inverting the sums)

$$\leq \sum_{e \in C} 2t \tag{8.7}$$

(using eq. (8.3))

$$\leq c \times 2t \qquad \text{(as } |\mathsf{C}| \leq c \text{ by assumption.)}$$

However, by lemma assumption $c \geq 3t + 1$ and $\forall \mathsf{S}_i \in \mathcal{S}, |\mathsf{S}_i| \geq c - t$. As a result,

$$\sum_{\mathsf{S}_i \in \mathcal{S}} |\mathsf{S}_i| \geq c(c - t) \geq c \times (2t + 1).$$

As $c > 0$, the two last inequalities contradict each other, proving Lemma 8.6. □

Lemma 8.7 (CAC-Local-termination)**.** If a correct process p_i cac-proposes a value v, then its set $accepted_i$ eventually contains a pair $\langle v', \star \rangle$. (Note that v' can be different from v.)

Proof. Consider a correct process p_i that cac-proposes a value v. lemma 8.5 applies. As each correct process signs and sends a readySig signature using a bundle message, and as bundle messages are disseminated using best effort broadcast, p_i eventually receives the readySig of each correct process, and by extension, it receives each of their M_j sets. Without loss of generality, we assume there are c correct processes, with $n \geq c \geq n - t$, and their identifiers goes from 1 to c, *i.e.*, $p_1, \ldots, p_c$ are correct processes.

By conditions at line 13 and 14, each M_j set sent by a correct process contains at least $n-t$ witSig, and out of those $n-t$ witSig, at least $c-t \geq n-2t$ are witSig signed by correct processes. Let S_j be the set of witSig in M_j signed by correct processes, *i.e.*, $\mathsf{S}_j = \{\text{witSig}(p_k, \langle\star,\star\rangle) \mid \forall p_k \text{ correct }, \text{witSig}(p_k, \langle\star,\star\rangle) \in M_j\}$, therefore, $|\mathsf{S}_j| \geq c-t, \forall j \in \{1,\ldots,c\}$. We note $\mathcal{S} = \{\mathsf{S}_1,\ldots,\mathsf{S}_c\}$ the set of S_j sets sent by correct processes. Finaly, we note $\mathsf{C} = \bigcup_{k=1}^{c} \mathsf{S}_k$ the set of witSig signed by correct processes and sent in the M_j sets by correct processes. As each correct process only produces one witSig signature during an execution, $|\mathsf{C}| = |\bigcup_{k=1}^{c} \mathsf{S}_k| \leq c$. Hence, Lemma 8.6 can be applied.

Therefore, among the c sets S_j that p_i eventually receives, at least one witSig signature is present in $2t+1$ of those sets. Therefore, the pair associated to this witSig will eventually verify condition at line 21 at p_i. Thus, p_i will add this pair to its $accepted_i$ set. □

Lemma 8.8 (CAC-Global-termination)**.** If, for a correct process p_i, $\langle v, j\rangle \in accepted_i$, then eventually $\langle v, j\rangle \in accepted_k$ at each correct process p_k.

This proof is an extended version of the proof of CAC-Global-termination presented in Section 8.5.2

Proof. Consider two correct processes p_i and p_k. Assume p_i adds $\langle v, j\rangle$ to $accepted_i$. By construction of line 21, p_i has saved the readySig signatures of $2t+1$ processes

$$\{\text{readySig}(p_{i_1}, M_1), \text{readySig}(p_{i_2}, M_2), ..., \text{readySig}(p_{i_{2t+1}}, M_{2t+1})\}$$

where $\text{witSig}(p_{\ell_k}, \langle v, \ell_k\rangle) \in M_{i_k}$, for some $\{\ell_1, \ell_2, ...\ell_{2t+1}\} \subseteq [1..n]$. p_i be-broadcasts all these signatures at line 18. Let us note $R_i^{v_j}$ this set of readySig signatures.

Observation 8.1. p_k will eventually receive the $2t+1$ signatures in $R_i^{v_j}$.

Proof. This trivially follows from the best effort broadcast properties and network's reliability. □

Observation 8.2. p_k will eventually receive at least $n-t$ readySig signatures.

Proof. As $R_i^{v_j}$ contains the signatures of $2t+1$ distinct processes (line 21), $t+1$ of these processes must be correct. W.l.o.g, assume p_{i_1} is correct. p_{i_1} has signed $\text{readySig}(p_{i_1}, M_1)$ at line 15 with $\text{witSig}(p_{\ell_1}, \langle v, \ell_1\rangle) \in M_{i_1}$. As a result, at line 14,

$$\text{witSig}(p_{\ell_1}, \langle v, \ell_1\rangle) \in sigs_{i_1}, \tag{8.8}$$

which implies that p_{i_1} be-broadcasts $\text{witSig}(p_{\ell)1}, \langle v, \ell_1\rangle)$ at line 16. As the network is reliable, all correct processes will eventually receive $\text{witSig}(p_{\ell_1}, \langle v, \ell_1\rangle)$, and if they have not done so already will produce a witSig signature at line 11 and be-broadcast it at line 12. As there

Variable	Meaning
$sigs_i$	set of valid signatures known by p_i
$sigcount_i$	sequence number of the signatures generated by p_i

Table 8.2 – CAC algorithm parameters and variables

are at least $n - t$ correct processes, all correct processes will eventually receive $n - t$ witSig signatures, rendering the first part of line 13 true. If they have not done so already, all correct processes will therefore produce a readySig signature at line 15, and be-broadcast it at line 16. As a result p_k will eventually receive at least $n - t$ readySig signatures □

Observation 8.3. There exists some $\ell \in [1...n]$ and some set M_ℓ of witSig signatures such that

— eventually, readySig$(p_\ell, M_\ell) \in sigs_k$;
— and witSig$(p_j, \langle v, j\rangle) \in M_\ell$.

Proof. When p_i accepts $\langle v, j\rangle$, line 21 implies that $\langle v, j\rangle \in candidates_i$, and therefore because of line 20 that $\exists\ p_\ell, M_\ell$: readySig$(p_\ell, M_\ell) \in sigs_i \wedge$ witSig$(p_j, \langle v, j\rangle) \in M_\ell$. Because p_i be-broadcasts $sigs_i$ at line 18, p_k will eventually receive the signatures contained in $sigs_i$, and add then to its own $sigs_k$ at line 9, including readySig(p_ℓ, M_ℓ). □

Observation 8.4. Eventually, $\langle v, j\rangle \in accepted_k$.

Proof. The previous observations have shown the following:

— By observation 8.2, the condition of line 17 eventually becomes true at p_k.
— By observation 8.3, $sigs_k$ eventually contains readySig$(p_\ell, M_\ell) \in$ for some $\ell \in [1...n]$ such that witSig$(p_j, \langle v, j\rangle) \in M_\ell$.
— By observation 8.1, p_k eventually receives the $2t + 1$ signatures in $R_i^{v_j}$.

When the last of these three events occurs, p_k passes through the condition at line 17, then line 20 leads to

$$\langle v, j\rangle \in candidates_k, \tag{8.9}$$

and by Observation 8.1, the selection criteria at line 21 is true for v, which, with Equation (8.9), implies that $\langle v, j\rangle \in accepted_k$, concluding the proof of the Lemma. □

□

```
1 init: accepted_i ← ∅; candidates_i ← ⊤; sigs_i ← ∅; blacklist_i ← ∅; sigcount_i ← 0.
2 function wit_count(⟨v, j⟩) is
3 |  S ← {k : WITSIG(p_k, ⟨v, j⟩, ⋆) ∈ sigs_i};          ▷ p_k has backed ⟨v, j⟩.
4 |  return |S|.
5 operation cac_propose(v) is
6 |  if no WITNESSMSG(⋆) or READYMSG(⋆) already be-broadcast by p_i then
7 |  |  sigs_i ← sigs_i ∪ {WITSIG(p_i, ⟨v, i⟩, sigcount_i)}; sigcount_i++;
8 |  |  be_broadcast WITNESSMSG(sigs_i).
```

Algorithm 6: One-shot signature-based CAC implementation (code for p_i) (Part I)

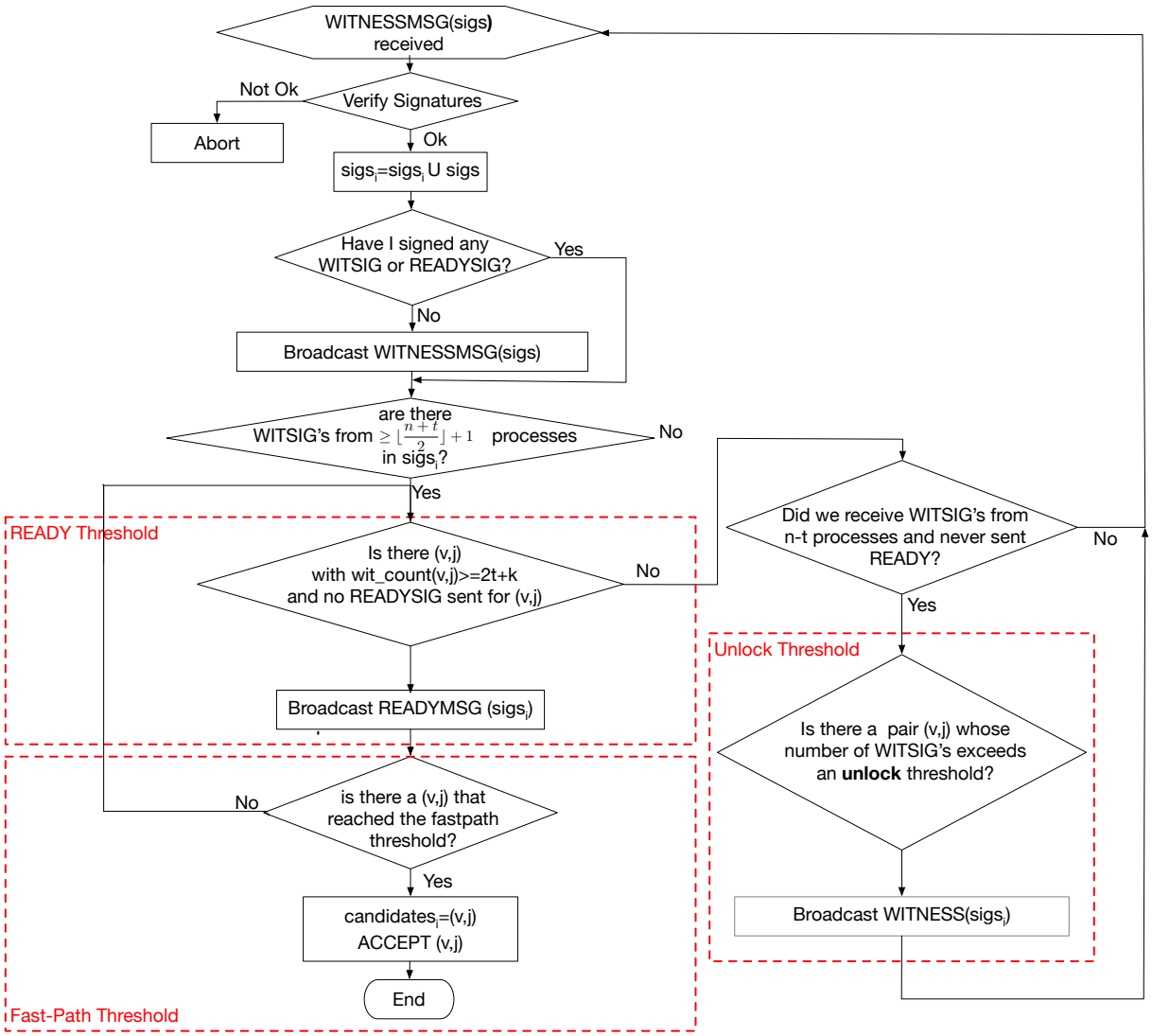


Figure 8.3 – Flowchart of the CAC implementation; workflow to process WITNESSMSG messages for process p_i.

```
 9 when WITNESSMSG(sigs) is received do                    ▷invalid messages are ignored
10   sigs_i ← sigs_i ∪ sigs;
11   if p_i has not signed any WITSIG or READYSIG statement yet then
12     ⟨v, j⟩ ← choice(sigs_i);                            ▷choice chooses one of the elements in sigs_i
13     sigs_i ← sigs_i ∪ {WITSIG(p_i, ⟨v, j⟩, sigcount_i)}; sigcount_i++;
14     be_broadcast WITNESSMSG(sigs_i);
15   if there are WITSIG from at least ⌊(n+t)/2⌋ + 1 processes in sigs_i then
16     for all ⟨v, j⟩ such that wit_count(⟨v, j⟩) ≥ 2t + k do
17       if READYSIG(p_i, ⟨v, j⟩, ⋆) ∉ sigs_i then
18         sigs_i ← sigs_i ∪ {READYSIG(p_i, ⟨v, j⟩, sigcount_i)}; sigcount_i++;
19         be_broadcast READYMSG(sigs_i);
20     if ∃ ⟨v, j⟩ : wit_count(⟨v, j⟩) ≥ n − t and ∀ ⟨v′, j′⟩ ≠ ⟨v, j⟩, wit_count(⟨v′, j′⟩) = 0 and n > 5t
         then
21       if ⟨v, j⟩ has not been accepted yet then
22         candidates_i ← ⟨v, j⟩ ;  accepted_i ← {⟨v, j, sigs_i⟩};
23         cac_accept(v, j) ;                  ▷Fast-path, no other pair ⟨v′, j′⟩ ≠ ⟨v, j⟩ will be accepted.
24   P ← {j | WITSIG(p_j, ⟨⋆, ⋆⟩, ⋆) ∈ sigs_i};
25   if |P| ≥ n − t and READYMSG(⋆) not already broadcast by p_i then
26     if n > 5t and ∃ ⟨v, j⟩ : wit_count(⟨v, j⟩) ≥ |P| − 2t then
27       if WITSIG(p_i, ⟨v, j⟩, ⋆) ∉ sigs_i then
28         sigs_i ← sigs_i ∪ {WITSIG(p_i, ⟨v, j⟩, sigcount_i)};
29         sigcount_i++;  be_broadcast WITNESSMSG(sigs_i);
30     else
31       M ← {⟨v, j⟩ | WITSIG(⋆, ⟨v, j⟩, ⋆) ∈ sigs_i};
32       T ← {⟨v, j⟩ | wit_count(⟨v, j⟩) ≥ max(n − (|M| + 1)t, 1)};
33       for all ⟨v, j⟩ ∈ T such that WITSIG(p_i, ⟨v, j⟩, ⋆) ∉ sigs_i do
34         sigs_i ← sigs_i ∪ {WITSIG(p_i, ⟨v, j⟩, sigcount_i)};
35         sigcount_i++; be_broadcast WITNESSMSG(sigs_i).
36 when READYMSG(sigs) is received do                      ▷invalid messages are ignored
37   if ∃ ⟨v′, k⟩ such that wit_count(⟨v′, k⟩) ≥ 2t + k then
38     sigs_i ← sigs_i ∪ {sigs};
39     for all ⟨v, j⟩ such that wit_count(⟨v, j⟩) ≥ 2t + k do
40       if READYSIG(p_i, ⟨v, j⟩, ⋆) ∉ sigs_i then
41         sigs_i ← sigs_i ∪ {READYSIG(p_i, ⟨v, j⟩, sigcount_i)};  sigcount_i++;
42         be_broadcast READYMSG(sigs_i);
43     candidates_i ← candidates_i ∩ {⟨v, j⟩ : wit_count(⟨v, j⟩) ≥ k};
44     for all ⟨v, j⟩ ∈ candidates_i such that |{j : READYSIG(p_j, ⟨v, j⟩, ⋆) ∈ sigs_i}| ≥ n − t do
45       accepted_i ← accepted_i ∪ {⟨v, j, sigs_i⟩};  cac_accept(v, j).
```

Algorithm 7: One-shot optimal signature-based CAC implementation (code for p_i) (Part II).

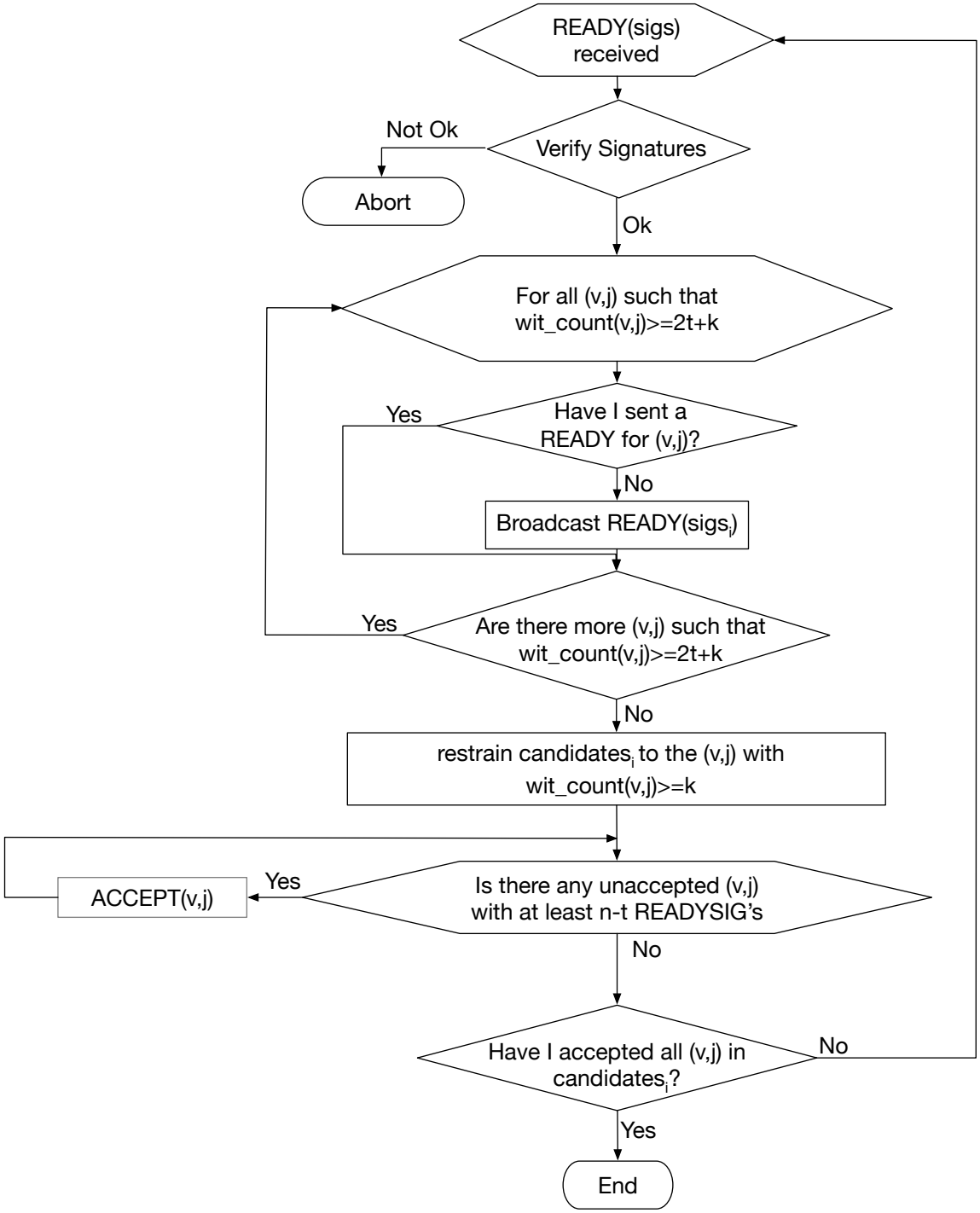


Figure 8.4 – Flowchart of the CAC implementation; workflow to process READYMSG messages for process p_i.

8.6 CAC: An Optimal Implementation

8.6.1 An optimal implementation of the CAC abstraction

Algorithm 6 and Algorithm 7 are the two parts of a signature-based algorithm that implements the CAC abstraction with optimal Byzantine resilience. Furthermore, the implementation has a good case latency of 2 asynchronous rounds when $n > 5t$ and 3 asynchronous rounds when $n > 3t$. Those best-case latencies are optimal as we analyze in Section 8.6.5. The algorithm also respects the proof of acceptance as proven by Lemma 8.15. This optional property comes without additional cost in our implementation. Table 8.2 summarizes the parameters and variables of the implementation and Figure 8.3 and Figure 8.4 is a flow-chart visually describing the algorithm.

In the first part of the description of the algorithm, we omit the "fast-path" mechanism (lines 20 to 22 and lines 26 to 29). Those are totally optional, for example, if $n < 5t$ they are never executed.

The algorithm works in two phases: WITNESS and READY. During each of these phases, correct processes sign and propagate two types of "statements" (WITSIG statements during the WITNESS phase, and READYSIG statements during the READY phase), using two types of messages (WITNESSMSG messages and READYMSG messages). Those statements are signatures of pairs $\langle v, j \rangle$, where v is a value and j is the identifier of the process that initially cac-propose v (if p_j is correct).

The signed statements produced by a node p_i are uniquely identified through a local sequence number $sigcount_i$, which is incremented every time p_i signs a new statement (at lines 7, 13, 18, 34, and 41). When communicating with other processes, a correct process always propagates all the signed statements it has observed or produced so far. (These statements are stored in the variable $sigs_i$.) To limit the power of Byzantine nodes, correct nodes only accept messages that present no "holes" in the sequence of statements they contain, *i.e.*, if a message XXMSG contains a statement signed by p_j with sequence number k, then XXMSG must contain one statement by p_j for all earlier sequence numbers $k' \in \{0, \cdots, k-1\}$ to be considered valid. Furthermore, a valid XXMSG contains a signature of the pair $\langle v, j \rangle$ by p_j, the process that cac-proposed the value. Similarly, a valid message can only contain valid signatures. Invalid messages are silently dropped by correct processes (not shown in the pseudo-code for clarity).

In the first phase, processes exchange WITNESSMSGs to accumulate votes on potential pairs to accept. A vote for a pair takes the form of a cryptographic signature on the message, which we refer to as WITSIG. Each WITNESSMSG can thus contain one or more WITSIGs. In the second phase, processes use READYMSGs to propagate cryptographic proofs that certain pairs have received enough support/votes. We refer to one such proof as READYSIG. Receiving a sufficient number of READYSIGs triggers the cac-acceptance. In Algorithm 7, the notation $\text{WITSIG}(p_i, \langle v, j \rangle, s_i)$

stands for a WITNESS statement signed by the process p_i with sequence number s_i of value v proposed by the process p_j. Similarly, the notation READYSIG$(p_i, \langle v, j \rangle, s_i)$ denotes a READY statement signed by the process p_i with sequence number s_i of value v initiated by the process p_j.

The algorithm relies on a parameter, k, which determines which pairs should enter the $candidates_i$ set. Specifically, a process adds a pair $\langle v, j \rangle$ to $candidates_i$ only if it has received at least k WITSIGs in favor of $\langle v, j \rangle$ from k different processes. The value of k strikes a balance between utility and fault tolerance. In particular, for $k = 1$, any two distinct pairs generated during an execution have a chance of being cac-accepted and thus enter the $candidates_i$ set, thus decreasing the probability of "known" termination for p_i, see Section 8.4.2. But in general, only pairs that k distinct processes have witnessed can enter the $candidates_i$ set. In either case, the algorithm works for $n \geq 3t + k$. Therefore, k must be chosen by the algorithm's implementer to balance Byzantine resilience and known termination probability.

In the following, we begin by describing each of the two phases of the algorithm without the fast-path, while referencing the pseudocode in Algorithm 7. Then, we describe the specificity of the fast path.

8.6.2 WITNESS phase

Let us consider a correct process p_i that cac-proposes value v. If p_i has not yet witnessed any earlier value, it signs $\langle v, i \rangle$ and propagates the resulting WITSIG to all the participants in a WITNESSMSG (lines 6-8). We refer to process p_i as the initiator of value v.

When p_i receives a WITNESSMSG, it accumulates the WITSIGs the message contains into its local signature set $sigs_i$ (line 10). The process then checks whether it has already witnessed an earlier pair (line 11). If it has not, it selects one of the WITSIGs is in its local signature set $sigs_i$, and signs a new WITSIG for the corresponding pair. It then broadcasts a new WITNESSMSG containing all WITSIGs it has observed or produced so far (lines 11 to 14). Because channels are reliable, this behavior ensures that all correct processes eventually witness some pair, which they propagate to the rest.

Once a process has received WITSIGs from a majority of correct processes (line 15), the majority is ensured by the threshold $\lfloor \frac{n+t}{2} \rfloor + 1$), it enters the READY phase of the algorithm. More precisely, it sends—if it has not done so already—a READYMSG for each of the pairs that have collected a quorum of $2t + k$ WITSIGs in their favor (lines 16-19). The READYMSG contains a READYSIG for the considered pair, and each of the WITSIGs received so far. Intuitively, this READY phase ensures that correct processes discover all the pairs that can potentially be *accepted* before accepting their first pair. (We discuss this phase in detail just below.)

However, receiving WITNESSMSGs from a majority of processes does not guarantee the presence of a pair with $2t+k$ WITSIGs. Indeed, up to this point, each correct process was only allowed

to vote once. For example, each correct process can vote for its own value it cac-proposes. Hence, a correct process may even receive $n - t$ witnessMsg without reaching the quorum of $2t + k$ witnessMsg for any pair. When this happens, we say that the algorithm has reached a *locked* state, which can be resolved using an *unlocking mechanism* (lines 25 to 35).

The first unlocking mechanism (from lines 26 to 29) is used when the fast path may have been used and will be described in Section 8.6.4. The second unlocking mechanism ensures that at least one pair reaches the $2t+k$ threshold at line 16 or 39. Once a process enters the unlocking mechanism, it sends a witnessMsg for each pair that received at least $\mathsf{max}(n - (|M| + 1)t, 1)$ witSigs in their favor. This threshold ensures that at least one pair reaches $2t + k$ witSigs. Thanks to this mechanism, all correct processes eventually broadcast at least one readyMsg.

8.6.3 READY phase

The ready phase starts by sending a readyMsg at line 19. When a correct process, p_i, receives a readyMsg, it first checks that it indeed contains at least $2t + k$ valid signatures for a given pair (line 37). If not, the message was sent by a malicious process and is thus ignored. After this verification step, p_i signs and broadcasts a readyMsg for all the pairs with at least $2t + k$ witSigs. This ensures that all correct processes eventually share the same knowledge about potentially acceptable values.

Then, process p_i computes its current $candidates_i$ set by only keeping the values that are backed by at least k witSigs. Then, process p_i cac-accepts all the pairs in the $candidates_i$ set that have received at least $n - t$ witSigs.

8.6.4 Fast-path

We now detail the optimization of the CAC algorithm that introduces an optimal latency path that can be followed by a process p_i when $n \geq 5t + 1$ and all the witSigs that p_i receives are in favor of a unique value. This fast-path can be seen from lines 20 to 22 in Algorithm 7.

The optimization requires an additional condition to the algorithm. If a process uses the fast-path for a pair $\langle v, j\rangle$, its $candidates_i$ set only contains $\langle v, j\rangle$ (line 22). Hence, no pair different from $\langle v, j\rangle$ can be accepted by any correct process to satisfy the CAC-Global-termination and CAC-Prediction properties. Thereby, the unlocking mechanism of the algorithm is also modified. Namely, a condition to send new witnessMsgs is added to the algorithm to ensure that, if a process could have taken the fast-path, then all the correct processes only send witSigs in favor of this pair.

This mechanism (the condition at line 26) is used if a process may have taken the fast-path, whereas the original mechanism at line 33 is used in all other cases. If a process $p_i \neq p_k$ uses the fast-path, then $n \geq 5t + 1$ and it received $n - t$ signatures in favor of one pair, for example, $\langle v, j\rangle$, and no signatures in favor of v'. Therefore, p_k receives a minimum of $n-2t$ messages from

the same processes as p_i, among which t can have been sent by Byzantine processes. Hence, p_k receives at least $n - 3t$ messages in favor of v, and t in favor of v' among the first $n - t$ witnessMsgs it receives. Furthermore, if p_k received t messages from Byzantine processes, it means that it can still receive messages from t correct processes. If p_i did use the fast-path, those new messages will back v. Therefore, if at least $n - 3t \leq |P| - 2t \leq n - 2t$—where $|P|$ is the number of unique processes from which p_i received witSigs—witnessMsg received by p_k are in favor of a unique pair $\langle v, j \rangle$, then another correct process p_i may have taken the fast path. Furthermore, when a correct process does use the fast-path for a pair $\langle v, j \rangle$, it accepts it along with a *candidates* set containing only the pair $\langle v, j \rangle$. In other words, if a process did use the fast-path, then no other pair should be *accepted*. Therefore, if a correct process is in a locked state, and if it detects that a process might have taken the fast-path for a pair $\langle v, j \rangle$, it should only send new witSigs in favor of $\langle v, j \rangle$. If every correct process detects that a process might have taken the fast path, then every process that did not vote in favor of $\langle v, j \rangle$ will do so. Therefore, each correct process will receive at least $2t + k$ witSigs in favor of v and will send a readyMsg in its favor, and no other pair will reach the $2t + k$ threshold.

8.6.5 Proof of the algorithm

We now prove that Algorithm 7 is a valid implementation of the CAC abstraction. The algorithm is proven for $n \geq 3t + k \geq 3t + 1$.

The different lemmas used to prove Algorithm 7 use the following notations: Let $sigs_i^\tau$ be the set $sigs_i$ of the process p_i at time τ. Let $accepted_i^\tau$ be the state of the set $accepted_i$ at time τ. Let $candidates^\tau$ be the state of the set $candidates_i$ at time τ. Let witSig$(sigs_i^\tau, v)$ be the witSigs relative to v in $sigs_i^\tau$, and let readySig$(sigs_i^\tau, v)$ be the readySigs relative to v in $sigs_i^\tau$. Let $\mathsf{max}(sigs_i^\tau, p_j)$ be the readySig or witSig with the greatest *sigcount* from process p_j in $sigs_i^\tau$.

Lemma 8.9 (CAC-Validity)**.** If p_i and p_j are correct, $candidates_i \neq \top$ and $\langle v, j \rangle \in candidates_i$, then p_j cac-proposed value v.

Proof. Let p_i and p_j be two correct processes p_i and let $\langle v_j, j \rangle \in candidates_i^\tau$ for some time τ. Furthermore, let us assume $candidates_i \neq \top$. Hence, $candidates_i$ has been modified by p_i. There are only two lines in Algorithm 7 where p_i can modify $candidates_i$. Either it did it at line 43 and it received k witSigs backing $\langle v_j, j \rangle$, or it modified it at line 22, and $n > 5t$ and p_i received at least $n - t$ witSigs backing $\langle v_j, j \rangle$ and no witSig backing another pair (line 20).

In both cases, p_i considers the pair $\langle v_j, j \rangle$ to be valid only if witSig$(p_j, \langle v_j, j \rangle, \star) \in sigs_i$ [7], *i.e.*, there exist a witSig in $sigs_i$ from the proposer of the value. In both cases, there is a signature of $\langle v_j, j \rangle$ by p_j in $sigs_i$. Hence, at time t, p_i received a witSig from p_j. Furthermore, we assume

7. This condition is an implicit condition stated in the description of the algorithm and assumed by the comment at lines 9 and 36.

that p_j is correct and that cryptographic signatures cannot be impersonated. Therefore, the only process able to sign a value using p_j's secret key is p_j itself. Hence, p_j did cac-propose value v_j in both cases. □

Lemma 8.10. For any two correct processes p_i and p_j, if $\langle v, f, \star\rangle \in accepted_i$ and $\langle v', o, \star\rangle \in accepted_j$, then $\langle v, k\rangle, \langle v', \ell\rangle \in (candidates_i \cap candidates_j)$.

Proof. Let p_i and p_j be two correct processes, and let $\langle v, f, \star\rangle \in accepted_i$ and $\langle v', o, \star\rangle \in accepted_j$. We note τ_v the time $\langle v, f, \star\rangle$ is added to $accepted_i$ and we note $accepted_i^{\tau_v}$, $candidates_i^{\tau_v}$ the state of the sets $accepted_i$ and $candidates_i$ at this time. Using the CAC-Global-termination property, we know that p_i will eventually cac-accept v'.

With this setup, p_i cannot cac-accept one of these tuples using the fast path—if p_i uses the fast path for $\langle v, f, \star\rangle$, there can only be a maximum of $2t$ witSigs in favor of $\langle v', o, \star\rangle$, no correct process will send a readySig in favor of $\langle v', o, \star\rangle$. Hence, both $\langle v, f, \star\rangle$ and $\langle v', o, \star\rangle$ are cac-accepted at line 45.

The following uses a proof by contradiction, we assume $\langle v, f, \star\rangle$ is cac-accepted first and $\langle v', o, \star\rangle \notin candidates_i^{\tau_v}$.

Furthermore, we use the following notations: Let $\mathsf{witness}(sigs_i^t, \langle v, f\rangle)$ be the witSig signatures for the pair $\langle v, f\rangle$ in $sigs_i^\tau$, and let $\mathsf{ready}(sigs_i^\tau, \langle v, f\rangle)$ be the readySigs relative to the pair $\langle v, f\rangle$ in $sigs_i^t$. Let $\mathsf{max}(sigs_i^t, p_j)$ be the readySig or witSig with the greatest *sigcount* from process p_j in $sigs_i^t$.

At τ_v, due to the condition at line 44, $|\mathsf{ready}(sigs_i^{\tau_v}, \langle v, f\rangle)| \geq n - t$. However, $\langle v', o\rangle \notin candidates_i^{\tau_v}$ by assumption. Hence, $|\mathsf{witness}(sigs_i^{t_v}, \langle v', o\rangle)| < k$ (lines 43 and 44). In the following, we use two characteristics of the algorithm:

1. A correct process does not send a witnessMsg if it already sent a readyMsg (lines 11 and 36); and
2. A correct process only accepts complete sequences of messages, *i.e.*, signature received from correct processes can be assumed FIFO. [8]

Among the readySigs in $\mathsf{ready}(sigs_i^{\tau_v}, \langle v, f\rangle)$, at least $n-2t$ are sent by correct processes. Using the second point of the previous remark, we know that if p_l is correct and given $k = \mathsf{max}(sigs_i^\tau, p_l)$, we received all messages from p_l with *sigcount* lesser than k. Furthermore, if there exists a readySig from p_l in $sigs_i^{t_v}$ and if p_l is correct, using the first point of the previous remark, we know that there are no witSigs from p_l in $sigs_i^\tau$ that are not in $sigs_i^{\tau_v}$, for all $\tau \geq \tau_v$.

Hence, the only witSigs in $\mathsf{witness}(sigs_i^\tau, \langle v', o\rangle)$ that are not in $\mathsf{witness}(sigs_i^{\tau_v}, \langle v', o\rangle)$ for $\tau > \tau_v$ are the one sent by correct processes whose readySigs weren't in $\mathsf{ready}(sigs_i^{\tau_v}, \langle v, f\rangle)$—we call them the set of *missed processes*—or the one sent by Byzantine processes. Because

8. This condition is implicitly stated in the description of the algorithm and assumed by the comment at lines 9 and 36.

$\mathsf{ready}(sigs_i^{\tau_v}, \langle v, f\rangle)$ contains the signature from at least $n - t$ processes, we know that the set of missed processes is lesser or equal to t. Hence, a maximum of $2t$ additional witSigs can be received by p_i after τ_v. (Up to t from correct processes whose readySigs weren't in $\mathsf{ready}(sigs_i^{\tau_v}, \langle v, f\rangle)$, and up to t from Byzantine processes that do not respect this constraint.) Therefore, p_i can receive up to $|\mathsf{witness}(sigs_i^{\tau_v}, \langle v', o\rangle)| + 2t$ witSig in favour of $\langle v', o\rangle$ during the whole execution of the algorithm. However, we said that $|\mathsf{witness}(sigs_i^{\tau_v}, \langle v', o\rangle)| < k$. Hence, $|\mathsf{witness}(sigs_i^{\tau_v}, \langle v', o\rangle)| + 2t < 2t + k$

Therefore, $\langle v', o\rangle$ will never reach the $2t + k$ threshold (line 16 or 44) and $\langle v', o, \star\rangle$ cannot be cac-accepted by a correct process, hence contradicting the assumption. Therefore, $\langle v, \star, \star\rangle, \langle v', \star, \star\rangle \in candidates_i \cap candidates_j$. □

Corollary 8.2 (CAC-Prediction)**.** For any correct process p_i and for any process identity k, if, at some point of p_i's execution, $\langle v, k\rangle \notin candidates_i$, then p_i *never* cac-accepts $\langle v, k\rangle$ (*i.e.*, $\langle v, k\rangle \notin accepted_i$ holds forever).

Proof. The corollary follows from the contrapositive of lemma 8.10 when $p_j = p_i$. □

Lemma 8.11 (CAC-Non-triviality)**.** For any correct process p_i, $accepted_i \neq \emptyset \Rightarrow candidates_i \neq \top$.

Proof. This property is directly verified. When a process cac-accepts a value, it first intersects its $candidates_i$ set with a finite set. Hence, at this point in time, $candidates_i \neq \top$. □

Lemma 8.12. If a correct process broadcasts a witnessMsg at *line 8* or *14*, then eventually we have $|\{j : \text{witSig}(p_j, \langle \star, \star\rangle, \star) \in sigs_i\}| \geq n - t$.

Proof. If a correct process broadcasts a witnessMsg at line 8 or 14, it is sure all the correct processes will eventually receive this witnessMsg (thanks to the best effort broadcast properties). Hence, each correct process p_j will answer with a witnessMsg containing a $\text{witSig}(p_j, \langle \star, \star\rangle, \star)$ if they did not already do so (lines 11 to 14). Therefore, if p_i broadcasts a witnessMsg, it is sure that eventually, $|\{j : \text{witSig}(p_j, \langle \star, \star\rangle, \star) \in sigs_i\}| \geq n - t$. □

Lemma 8.13 (CAC-Local-termination)**.** If a correct process p_i invokes cac_propose(v), its set $accepted_i$ eventually contains a pair $\langle v', \star\rangle$ (note that v' is not necessarily v).

Proof. Let a correct process p_i cac-proposes a value v. To prove the CAC-Local-termination property, three different cases must be explored.

— In the first case, p_i signs and be-broadcasts a witSig in favour of $\langle v, i\rangle$. It eventually receives $n - t$ witSigs (Lemma 8.12) among which at least $2t + k$ witSigs are in favour of a unique pair $\langle v', j\rangle$ (either $\langle v', j\rangle = \langle v, i\rangle$ or $\langle v', j\rangle \neq \langle v, i\rangle$), *i.e.*, $\exists \tau$ such that $|\{l :$

WITSIG$(p_l, \langle v', j\rangle, \star) \in sigs_i^\tau\}| \geq 2t + k$. Hence, $\langle v', j\rangle$ satisfies the condition line 16 or 39 and p_i will broadcast a READYMSG along with $sigs_i^\tau$. Thanks to the best effort broadcast properties and because p_i is correct, the $n - t$ correct processes will eventually receive $sigs_i^\tau$.

Let p_κ be a correct process that receives the READYMSG from p_i and $sigs_i^\tau$ at time τ'. It will add all the signatures from $sigs_i^\tau$ to $sigs_\kappa^{\tau'}$ (line 38). Therefore, $|\{l$: WITSIG$(p_l, \langle v', j\rangle, \star) \in sigs_\kappa^{\tau'}\}| \geq 2t + k$ and v' satisfies the condition at line 39. Process p_κ will eventually send a READYMSG at line 42 along with its set $sigs_\kappa$ where READYSIG$(p_\kappa, \langle v', j\rangle, \star) \in sigs_\kappa$ (lines 41 and 42). Each correct process will eventually send such READYMSG. Hence, eventually, p_i will receive READYSIG$(\star, \langle v', j\rangle, \star)$ from the $n - t$ correct processes. Hence, the condition at line 44 will eventually be verified, and p_i will cac-accept $\langle v', j\rangle$.

— In the second case, p_i signs and be-broadcasts a WITNESSMSG in favour of $\langle v, i\rangle$, and among the $n - t$ responses it receives (Lemma 8.12), there are less than $2t + k$ WITSIGs messages in favour of $\langle v, i\rangle$ or any other $\langle v', j\rangle$, *i.e.*, $|\{j \mid$ WITSIG$(p_j, \langle\star, \star\rangle, \star) \in sigs_i^\tau\}| \geq n - t$ and $\nexists \langle v', j\rangle$, such that $|\{$WITSIG$(\star, \langle v', j\rangle, \star) \in sigs_i^\tau\}| \geq 2t + k$. In this case, p_i is stuck. It cannot send a READYMSG or cac-accept a value, but it cannot wait for new WITNESSMSG either, because all the Byzantine processes could act as if they crashed. [9] It must use one of the unlocking mechanisms implemented from lines 25 to 35.

We analyze the two possible unlocking mechanisms:

— The first unlocking mechanism (from line 26 to 29) is used by p_i if a correct process p_j might have used the fast-path. If p_j might have used the fast-path at time τ, $|\{$WITSIG$(\star, \langle v_f, f\rangle, \star) \in sigs_j^{\tau'}\}| \geq n - t$ and $|\{$WITSIG$(\star, \langle v_o, o\rangle, \star) \in sigs_j^{\tau'}\}| = 0, \forall \langle v_f, f\rangle \neq \langle v_o, o\rangle$. Let $|P|$ be the number of processes from which p_i received WITSIGs, *i.e.*, $P = \{j \mid$ WITSIG$(p_j, \langle\star, \star\rangle, \star) \in sigs_i\}$, and $|P| \geq n - t$. We consider the worst case scenario where $T_j^{\tau'} = \{$WITSIG$_t, \cdots,$ WITSIG$_{2t} \cdots,$ WITSIG$_n\}$ is the set of WITSIGs received by p_j at time τ', where $\{$WITSIG$_t, \cdots,$ WITSIG$_{2t}\}$ are messages sent by Byzantine processes and $T_i^\tau = \{$WITSIG$_1, \cdots,$ WITSIG$_{t-1},$ WITSIG$_{t'}, \cdots,$ WITSIG$_{2t'},$ WITSIG$_{2t+1}, \cdots,$ WITSIG$_{|P|}\}$ is the set of WITSIGs received by p_i at time τ where $\{$WITSIG$_{t'}, \cdots,$ WITSIG$_{2t'}\}$ are messages sent by Byzantine processes, WITSIG$_i \neq$ WITSIG$_{i'}, \forall i \in \{t, \cdots, 2t\}$. We have $|T_i^\tau \cap T_j^{\tau'}| \geq |P| - 2t \geq n - 3t$.

Therefore, if $\exists \langle v_f, f\rangle$ such that $|$WITSIG$(\star, \langle v_f, f\rangle, \star) \in sigs_i^\tau| \geq |P| - 2t$, p_j might have used the fast-path (this condition is verified by line 26). In this case, processes send a new WITNESSMSG only in favor of $\langle v_f, f\rangle$ (if they did not already do so). Eventually,

9. Let us recall that, except for the unlocking mechanisms from line 25 to 35, a correct process can only produce one WITSIG during the execution of the algorithm.

p_i will receive all the WITNESSMSG sent by the correct processes. Therefore, if $n - 2t$ correct processes sent a WITNESSMSG message in favor of $\langle v_f, f\rangle$, then the t correct processes that did not vote for this pair in the first place will send a new WITNESSMSG in its favor. Therefore, the correct processes will eventually receive $n - t \geq 2t + k$ WITNESSMSG messages in favor of $\langle v_f, f\rangle$, and they will send a READYMSG in favor of this pair. Hence, each correct process will receive $n - t$ READYSIG in favor of $\langle v_f, f\rangle$, and will cac-accept it (line 45). Otherwise, if there are less than $n-2t$ correct processes that sent WITSIGs in favor of $\langle v_f, f\rangle$ in the first place, p_i will eventually receive the messages from the t correct processes that it missed, the condition at line 26 will no longer be true and p_i will resume to the second unlocking mechanism.

— With the second unlocking mechanism, a correct process sends a new WITNESSMSG only if it received at least $\mathsf{max}(n-(|M|+1)t, 1)$ WITSIGs (line 33) where $M = \{\langle v', \kappa\rangle : \text{WITSIG}(\star, \langle v', \kappa\rangle, \star) \in sigs_i\}$. First, let us prove that either a correct process p_j sends a READYMSG message after receiving the first messages of the $n-t$ correct processes, or a pair $\langle v', \kappa\rangle$ eventually satisfies the following condition at all correct processes: $\{\text{WITSIG}(\star, \langle v', \kappa\rangle, \star) \in sigs_l\} \geq \mathsf{max}(n - (|M| + 1)t, 1), \forall p_l$, a correct process.
Let us assume that the previous assumption is wrong, *i.e.*, no correct process sends a READYMSG message after receiving the first WITNESSMSG from the $n - t$ correct processes, and there is a process p_l such that $\{\text{WITSIG}(\star, \langle v', \kappa\rangle, \star) \in sigs_l\} < \mathsf{max}(n - (|M| + 1)t, 1)$. The first part of the assumption implies that $\forall \langle v', \kappa\rangle$, $\{\text{WITSIG}(\star, \langle v', \kappa\rangle, \star) \in sigs_l\} < 2t + k$, for all p_l correct. We know (thanks to the best effort broadcast properties) that each correct process will eventually receive the first $n - t$ WITSIG sent by correct processes, let $sigs_{tot}$ be this set. Furthermore, the worst case scenario is when each pair in $sigs_{tot}$ is backed by a minimal number of WITSIGs, *i.e.*, the scenario where each pair in $sigs_{tot}$ is backed by $\frac{n-t}{|M|}$ WITSIGs—otherwise, by the pigeonhole argument, we have one pair that is backed by more signatures and which is more likely to reach the $\mathsf{max}(n-(|M|+1)t, 1)$ threshold. Hence, $\forall\langle v', \kappa\rangle$ such that $\lfloor\frac{n-t}{|M|}\rfloor| + 1 \geq |\{\text{WITSIG}(\star, \langle v', \kappa\rangle, \star) \in sigs_{tot}\}| \geq \lfloor\frac{n-t}{|M|}\rfloor$. Furthermore, we see that $\lfloor\frac{n-t}{|M|}\rfloor \geq \mathsf{max}(n - (|M| + 1)t, 1)$. Hence, the hypothesis is contradicted. We know that either a correct process p_j sends a READYMSG while receiving the first value of the $n - t$ correct processes, or a value v' eventually satisfies the following condition at all correct processes: $\{\text{WITSIG}(\star, \langle v', \kappa\rangle, \star) \in sigs_l\} \geq \mathsf{max}(n - (|M| + 1)t, 1), \forall p_l$ correct processes.

In both cases, each correct process will eventually send a READYMSG along with the $2t+k$ WITSIGs they received in favor of a unique pair, hence falling back to the first case.

— The third case occurs when no WITNESSMSG in support of $\langle v, i\rangle$ is sent by p_i to the other processes—another WITNESSMSG was already broadcast (line 6)—*i.e.*,

witSig$(p_i, \langle v, i\rangle, \star) \notin sigs_i^\tau$ for any time τ of the execution. However, even if p_i does not broadcast a witnessMsg in favor of $\langle v, i\rangle$ (due to the condition at line 6), it has already sent a witnessMsg in favor of some pair $\langle v', j\rangle$ (again, because of the condition at line 6), thus falling back to the first or the second case.

Therefore, if a correct process p_i cac-proposes a value, it will always cac-accept at least one pair. □

Lemma 8.14 (CAC-Global-termination)**.** If p_i is a correct process and $\langle v, j\rangle \in accepted_i$, eventually $\langle v, j\rangle \in accepted_k$ at every correct process p_k.

Proof. Let p_i and p_j be two correct processes. Let p_i cac-accept a tuple $\langle v, j, \star\rangle$, but p_j does not. Two cases can be highlighted:

- In the first case, p_i received $n-t$ readySig in favour of $\langle v, j\rangle$ (line 44). The readyMsg that is used to send those signatures contains $2t+k$ witSig in favor of $\langle v, j\rangle$ (thanks to the verification at line 37). Furthermore, it did not satisfy the condition line 20.
- In the second case, $n > 5t$, and p_i received more than $n-t$ witSig in favour of $\langle v, j\rangle$ and no signatures in favour of another pair (line 20).

In both cases, p_i broadcasts a readyMsg in favour of $\langle v, j\rangle$ (line 19 or 42). Each readyMsg in favor of $\langle v, j\rangle$ sent by a correct process contains at least $2t+k$ valid witSig in favor of $\langle v, j\rangle$ (lines 16 and 39). Process p_i is correct, therefore each correct process will eventually receive at least one readyMsg associated with the proof that $2t+k$ witSig in favour of $\langle v, j\rangle$ exists. Hence, $\langle v, j\rangle$ will eventually reach the $2t+k$ threshold at each correct process. When a correct process receives $2t+k$ witnessMsg in favor of a pair, it sends a readyMsg in its favor (line 19 or 42). Therefore, each correct process will send a readyMsg relative to $\langle v, j\rangle$ at line 42. Because there are $n-t$ correct processes, each correct process will receive $n-t$ readyMsg in favor of $\langle v, j\rangle$, and p_j will eventually cac-accept $\langle v, j\rangle$ (line 45). □

Theorem 8.1. If $n \geq 3t+k \geq 3t+1$, then Algorithm 7 implements the *CAC* abstraction.

Proof. Using Lemma 8.9, Corollary 8.2, Lemma 8.11, Lemma 8.13, and Lemma 8.14, Algorithm 7 implements the CAC abstraction. □

Lemma 8.15 (Proof of acceptance)**.** There exists a function Verify such that, for any proof of acceptance π_v, the following property holds

$$\mathsf{Verify}(v, \pi_v) = \texttt{true} \iff \exists\ p_i \text{ correct such that, eventually, } \langle v, \star, \pi_v\rangle \in accepted_i.$$

Proof. Let π_v be a candidate proof of cac-acceptance. Let the function Verify(π_v, v) return true if and only if π_v contains valid readySigs on the value v from at least $n-t$ processes in Π. Hence,

at least $n - 2t \geq t + k$ correct processes signed READYSIGs in favor of v. Furthermore, correct processes only propagate their signatures via a READYMSG, which is a best-effort broadcast. Therefore, all the correct processes eventually receive those $n - 2t$ READYSIGs. Furthermore, a READYMSG from a correct process contains at least $2t + k$ WITSIGs (line 16 or 39). Hence, all the correct processes will receive $2t + k$ WITSIG backing v, and all the correct processes will send a READYMSG backing v, and they will eventually cac-accept v. In other words, $\mathsf{Verify}(\pi_v, v) = \texttt{true} \Rightarrow \exists p_i$ correct such that p_i cac-accepts $\langle v, \star, \pi_v \rangle$.

Let a correct process p cac-accept a tuple $\langle v, \star, \pi_v = sigs_i \rangle$. Then $sigs_i$ contains all the signatures p sent and received before the cac-acceptance. To cac-accept a value v, $sigs_i$ must contain at least $n - t$ READYSIGs in its favor (line 44). Hence $\mathsf{Verify}(sigs_i, v) = \texttt{true}$. Therefore, $\mathsf{Verify}(\pi_v) = \texttt{true} \Leftrightarrow \exists\ p_i$ correct such that p_i cac-accepts $\langle v, \star, \pi_v \rangle$. □

Optimality of the best case latency

This section explores the theoretical best-case latency of the abstraction. More precisely, it proves that the optimized Algorithm 7 reaches the lower bound with respect to Byzantine resilience when fast-path is enabled.

Theorem 8.2. If a *CAC* algorithm allows processes to cac-accept after all the correct processes have only broadcast one message, then $n \geq 5t + 1$.

Proof. Let T_1, T_2, T_3, T_4 be partitions of Π. Let $|T_1| = |T2| = |T_3| = t$. Let $n \leq 5t$. We consider $p_1 \in T_1$ and $p_2 \in T_2$ two processes. Two values v and v' are cac-proposed by two correct processes p_v and $p_{v'}$ respectively. The assumption is that p_1 cac-accepts a value v after all correct processes broadcast one message. In the best case, p_1 received messages from processes that only received the broadcast from p_v.

In the first execution, processes in T_2 are Byzantine, and act as if they crashed. Processes in T_1, T_3 and T_4 back value v. The second execution is an execution where processes in T_3 are Byzantine; they send votes in favor of v to the processes in T_1 and votes in favor of v' to the processes in T_2. Processes in T_2 vote back v', and processes in T_4 vote back v. In the third execution, processes in T_1 are Byzantine and act as if they crashed. Processes in T_2 and in T_3 both vote in favor of v' while processes in T_4 vote in favor of v.

Because of the asynchrony of the network, from p_1's point of view, the first two executions are indistinguishable if it receives messages from T_1, T_3 and T_4 first. In both of them, it has to accept a value. Thus, in both of them, it cac-accepts the value v in one round because it only received messages about v.

Furthermore, and because of the asynchrony of the network, from p_2's point of view, the second and the third executions are indistinguishable if it receives messages from T_2, T_3 and T_4 first. In both of them, it sees $2t$ votes for v and $2t$ votes for v'. Thus, whether v or v' should be

cac-accepted is undetermined, and processes must send new messages to decide. A second round of communication is necessary, and both values could eventually be accepted.

However, the assumption was that p cac-accepts in one round, *i.e.*, it can participate in the second round of communication, but the result should not impact the fact that only v is cac-accepted. However, v' can also be cac-accepted, hence contradicting the CAC-Prediction and CAC-Global-termination property. Therefore, the proportion of Byzantine processes for a best-case latency of one round is at least $n \geq 5t + 1$. □

Corollary 8.3. The fast-path proposed by *Algorithm 7* is optimal with respect to Byzantine resilience.

Proof of the latency of Algorithm 7

This section analyzes the latency properties of our algorithm.

Theorem 8.3. Let x values be cac-proposed by x processes. In the worst case, processes that implement *Algorithm 7* exchange $2 \times x \times n^2$ messages and cac-decide after four rounds of best-effort broadcast.

Proof. Let x values be cac-proposed by x processes. Each process will broadcast an initial witnessMsg—one best effort broadcast round, and $x \times n$ messages. After the reception, the $n - x$ processes that did not broadcast answer those broadcasts with new witnessMsg—a second best effort broadcast round, and $n(n - x)$ messages. Because of the conflict, the processes have to use the unlocking mechanism for each of the values they did not already witness —third best effort broadcast round and $(x - 1)n^2$ messages. Finally, each process sends a readyMsg in favor of each value and cac-accepts—fourth best effort broadcast round and xn^2 messages. Therefore, in the worst case, the values are cac-accepted after four best-effort broadcast rounds, and $xn + n(n - x) + (x - 1)n^2 + xn^2 = 2xn^2$ messages are exchanged. □

Theorem 8.4. The best case latency of the *Algorithm 7* when $n < 5t + 1$ is three asynchronous rounds.

Proof. When $n < 5t+1$, processes cannot use the fast path. The best case for the implementation is when there are no conflicts. In this case, a process p_i broadcasts an initial witnessMsg in favor of value v—first asynchronous round. Then, each process broadcasts its own witnessMsg in favor of v—second asynchronous round. Finally, each process broadcasts a readyMsg message, and cac-decides—third asynchronous round. The correct processes cac-accept a value after three asynchronous rounds. □

Theorem 8.5. The best case latency for a correct process in *Algorithm 7* is two asynchronous rounds when $n \geq 5t + 1$.

Proof. Let a correct process p_i be the unique process to cac-propose value v. Let $n \geq 5t + 1$. First, it broadcasts a WITNESSMSG in favor of v—first asynchronous round. Then, each correct process broadcasts a WITNESSMSG in favor of v—second asynchronous round. When p_i receives the $n - t$ WITNESSMSG of the correct processes, it uses the fast path and accepts v. Thus, the best-case latency of the optimized version of the CAC implementation is two asynchronous rounds. □

8.7 CAC in Action: Solving Low Contention Problems

The CAC object can solve cooperation problems by combining optimistic conflict avoidance with a deterministic back-off strategy when conflicts occur. This section thoroughly explores two of these applications. The first one is a solution to a new naming problem, called *short naming*. The naming problem makes it possible for processes to claim new names, and to associate them with a public key. Short naming is a variant of the naming problem where the new names attributed to processes have low entropy. The CAC abstraction is, to the best of our knowledge, the only abstraction that makes it possible to solve this problem in the presence of faulty processes and asynchrony.

The second application studied is the well known distributed consensus problem. We explore a CAC-based solution to this problem denoted *Cascading Consensus*, a new optimistically terminating consensus algorithm that ensures an early decision in favorable circumstances. Moreover, this algorithm uses information provided by the CAC abstraction to reduce synchronization and communication complexity in case of contention. More precisely, this algorithm uses the CAC abstraction to disseminate values. If contention occurs, *i.e.*,, if termination is not guaranteed, then only the processes that proposed a value participate in the conflict resolution. This behavior is made possible thanks to the information given by the *candidates* set. In that sense, this algorithm goes beyond similar existing solutions [141, 151, 152, 145, 153], and, to the best of our knowledge, the CAC abstraction is the only existing abstraction that makes it possible to implement an algorithm with such a behavior. These examples could be extended to many other distributed applications, *e.g.*, shared account asset-transfer protocols, access control, naming services, *etc.*

8.7.1 The fault-tolerant asynchronous short-naming problem

Problem presentation

The algorithm presented in this section uses the CAC abstraction to implement a consensus-free naming algorithm. The idea is to allow clients to claim a name as theirs. To do so, they associate a name with a public key and prove that they know the associated secret key. However,

if clients can choose the name they want to claim, multiple clients can claim the same name simultaneously, creating a conflict.

Existing solutions To this day, only two methods are known to solve this problem.

— The first method uses a consensus algorithm. Each client can choose its name, and, in case of contention, the consensus algorithm decides which process wins in a first-come, first-served manner. The problem of this method is that consensus should be used as the main communication primitive, whereas the probability of contention is low. Furthermore, consensus requires additional computability power (*e.g.*, partial synchrony or randomization).

— The second method prevents processes from actually choosing their names and relies on the entropy of random numbers to avoid collisions. It does not require the use of consensus algorithms as processes use their public keys as names. If the underlying cryptography is perfectly secure and secret keys are only known by their legitimate user, then the associated public keys are unique, and no conflict can occur because no two clients can claim the same name. The problem with this method is that public keys consist of long chains of random characters, making them hard for humans to remember. Techniques exist to circumvent this issue. Most of them use functions to map a random string to something that humans can remember: petname systems [129], tripphrases [159], or Proquint IDs [160]. However, those techniques do not reduce the entropy of the identifier, and they are mainly used to prevent identity theft (*e.g.*, phishing).

An efficient CAC-based approach Assuming perfect public/private keys, the CAC primitive makes it possible to propose a new consensus-free naming algorithm that reduces the size (and the entropy) of the claimed names, thus making them easier to remember for humans. The idea is to let clients claim sub-strings of their public keys. For example, let a client c have a public key "`abcdefghij`". It will first claim the name "`a`" using one instance of the CAC primitive. If there is no conflict, *i.e.*, if the size of the *candidates* set is 1 after the first acceptance, then the name "`a`" belongs to c and is associated with its public key. On the other hand, if there is a conflict, *i.e.*, another process claimed the name "`a`" and the size of the candidate set is strictly greater than 1 after the first acceptance, then c claims the name "`ab`". This procedure ensures that a client can always obtain a name. Indeed, because we assume perfect cryptographic primitives, only one client knows the secret key associated with its public key. Therefore, if c conflicts with all its claims on the subsets of its public key, it will eventually claim the name "`abcdefghij`". No other process can claim the same name and prove it knows the associated secret key.

Short naming: formal definition

The short naming cooperation abstraction provides each process with one operation $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$ that allows it to claim a name that is a sub-string of the public key pk. Furthermore, π is the proof of knowledge of the secret key associated with pk. Moreover, the object provides each process with an (initially empty) set $Names_i$, which associates names with public keys. A $Names_i$ set is composed of triples $\langle n, \mathsf{pk}, \pi \rangle$ where n is the attributed name, pk is the associated public key, and π is the proof of knowledge of the secret key associated to the public key. A short naming object provides the following properties.

— *Unicity*. Given a correct process p_i, $\forall \langle Names_j, \mathsf{pk}_j, \pi_j \rangle, \langle n_k, \mathsf{pk}_k, \pi_k \rangle \in Names_i$, either $n_j \neq n_k$ or $j = k$.

— *Short-names*.[10] If all processes are correct, and given one correct process p_i, eventually we have $\forall \langle n_j, \mathsf{pk}_j, \star \rangle, \langle n_k, \mathsf{pk}_k, \star \rangle \in Names_i$:
If $|\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_k)| \geq |\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_\ell)|, \forall \langle \star, \mathsf{pk}_\ell, \star \rangle \in Names_i$ then $|\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_k)| + 1 \geq |n_j|$.

— *Agreement*. Let p_i and p_j be two correct processes. If $\langle n, \mathsf{pk}, \pi \rangle \in Names_i$ and if the process that invoked $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$ is correct, then eventually $\langle n, \mathsf{pk}, \pi \rangle \in Names_j$.

— *Termination*. If a correct process p_i invokes $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$, then eventually $\langle \star, \mathsf{pk}, \star \rangle \in Names_i$.

The *Short-names* property captures the fact that the names given to the processes are the smallest possible. If there are no Byzantine processes, each name should be the smallest possible, when comparing it to other attributed names. The property only considers this difference eventually, *i.e.*, while a process might have successfully claimed a name, it may take a long time for the process it was concurring with to get its own name.

A CAC-based short-naming algorithm

Given a character string s, we denote by $s[i]$ the prefix of s of length i, e.g. "`abcdefghijk`"[3] = "`abc`".

Algorithm 8 implements the short naming abstraction presented in Section 8.7.1. This implementation uses two steps: a claiming phase and a commitment phase. The claiming phase verifies (and proves) that no other process tries to claim the same name. The commitment phase is used to actually associate a name with a public key, once this association has been successfully claimed. The claiming phase uses multiple CAC instances. Each instance is associated with a name. If the CAC instance cac-accepts the value cac-proposed by a process p_i and $|candidates_i| = 1$, it means that there is no contention on the attribution of the name. If p_i is the only process claiming this name, then it can commit to this name; otherwise, there is a conflict.

10. The function Max_Common_Prefix() outputs the longest common prefix between two string, *e.g.*, Max_Common_Prefix("`abcdefg`", "`abcfed`") = "`abc`".

In the latter case, the invoking processes will claim a new name by adding one character from its public key to the old name. The CAC instances for the claiming phase are stored in a dynamic dictionary, *Claim_dict*. This dictionary dynamically associates a CAC instance to a name. It is initiated as an empty dictionary, and whenever a process invokes the cac_propose operation on a specific name—*i.e.*, when a process executes *Claim_dict*[*name*].cac_propose(pk)—or when the first CAC value for a specific name is received, the dictionary dynamically allocate a new CAC object.

The commitment phase uses a new set of CAC instances. Similarly to the claiming phase, Algorithm 8 uses one CAC instance per name. However, unlike the claiming phase, there is one CAC instance per name and per process. When a process knows it successfully claimed a name, *i.e.*, no contention was detected, it informs the other processes by disseminating its public key using its CAC instance associated with the claimed name. Processes can verify that the commitment does not conflict with another process, because they accepted the associated name in the associated CAC instance. However, if a Byzantine process p claimed the same name, it could commit to the name even though the system did not accept its claim. In this case, the commitment would be rejected by correct processes, as the Byzantine process cannot provide a valid proof of acceptance of the claim. The only case where multiple processes can commit to the same name is if they are all Byzantine, and all their claims are cac-accepted. In this case, they could all commit to the same name, which does not violate the specification and the *Agreement* property. In other words, Byzantine processes can share the same names if it does not impact correct processes. Similarly to the claiming phase, CAC instances used during the commitment phase are stored in a dynamic dictionary.

We further assume the CAC instances only accept valid pairs, *i.e.*, for a pair $\langle \mathsf{pk}, \pi \rangle$ and the instance *Commit_dict*[*name*] or *Claim_dict*[*name*], *name* is a sub-string of pkand π is a valid proof of knowledge of the secret key associated to pk.

This algorithm uses a VerifySig(pk, π) algorithm. This algorithm returns "`true`" if and only if π is a valid cryptographic signature by the public key pk.

Proof of the algorithm

The proof that Algorithm 8 implements the Short Naming abstraction defined in Section 8.7.1 follows from the subsequent lemmas.

Lemma 8.16 (*Unicity*)**.** Given a correct process p_i, $\forall \langle Names_j, \mathsf{pk}_j, \pi_j \rangle, \langle n_k, \mathsf{pk}_k, \pi_k \rangle \in Names_i$, either $n_j \neq n_k$ or $j = k$.

Proof. Let p_i be a correct process such that $\exists \langle n_j, \mathsf{pk}_j, \pi_j \rangle, \langle n_k, \mathsf{pk}_k, \pi_k \rangle \in Names_i$ and $n_j = n_k$, $j \neq k$.

The only place in the algorithm where p_i updates $Names_i$ is at line 23. To reach this line,

```
1  init: Names_i ← ∅; Claim_dict ← dynamic dictionary of CAC objects;
2  Commit_dict_i ← dynamic dictionary of CAC objects; prop_i ← ∅.

3  operation Short_Naming.Claim(pk, π) is
4      if VerifySig(pk, π) = false then return ;
5      Choose_Name(1, pk, π).                ▷Queries an unused name, starting with pk[1].

6  internal operation Choose_Name(ℓ, pk, π) is
7      curr_name ← pk[ℓ];
8      while ⟨curr_name, ⋆⟩ ∈ Names_i do          ▷Looks for the first unused name.
9          ℓ ← ℓ + 1;
10         if ℓ > |pk| then return ;
11         curr_name ← pk[ℓ];
12     prop_i ← prop_i ∪ ⟨curr_name, pk, π, ℓ⟩;
13     Claim_dict[curr_name].cac_propose(⟨pk, π⟩).          ▷Claims curr_name.

14 when Claim_dict[name].cac_accept(⟨pk, π⟩, j) do
15     if VerifySig(pk, π) = false or name is a sub-string of pk then return ;
16     if ⟨name, pk′, π′, ℓ⟩ ∈ prop_i then                  ▷If name was claimed by p_i.
17         prop_i ← prop_i \ ⟨name, pk′, π′, ℓ⟩;
18         if |Claim_dict[name].candidates_i| = 1 and π′ = π then
             Commit_dict_i[name].cac_propose(⟨pk′, π′⟩);    ▷If no conflict, commit to name.
19         else Choose_Name(ℓ + 1, pk, π′).    ▷p_i claims a name with more digits (back-off
             strategy).

20 when Commit_dict_j[name].cac_accept(⟨pk, π⟩, j) do
21     if VerifySig(pk, π) = false or name is a sub-string of pk then return ;
22     wait(⟨pk, π⟩ ∈ Claim_dict[name].accepted_i);
23     if ⟨name, ⋆, ⋆⟩ ∉ Names_i then Names_i ← Names_i ∪ {⟨name, pk, π⟩}.
         ▷The association between name and pk is committed by p_i.
```

Algorithm 8: Short naming algorithm implementation (code for p_i)

p_i must verify the condition $\langle name, \star, \star\rangle \notin Names_i$ at line 23. However, this condition can only be valid once per name. Hence, p_i will only update $Names_i$ once per name, and two different tuples $\langle n_j, \mathsf{pk}_j, \pi_j\rangle, \langle n_k, \mathsf{pk}_k, \pi_k\rangle$ cannot be present in $Names_i$ if $j \neq k$. Hence, either $n_j \neq n_k$, or $j = k$ □

Lemma 8.17 (*Agreement*)**.** Let p_i and p_j be two correct processes. If $\langle n, \mathsf{pk}, \pi\rangle \in \mathsf{Name}_i$ and if the process that invoked $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$ is correct, then eventually $\langle n, \mathsf{pk}, \pi\rangle \in \mathsf{Name}_j$.

Proof. Let p_i, p_j and p_k be three correct processes. Let p_k invoke $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$. Let $\langle n, \mathsf{pk}, \pi\rangle \in Names_i$, where $\langle n, \mathsf{pk}, \pi\rangle \notin Names_j$ during the whole execution.

If $\langle n, \mathsf{pk}, \pi\rangle \in Names_i$, then it means that p_i updated $Names_i$ at line 23. This implies that $Commit\_dict_k[n].\mathsf{cac\_accept}(\langle \mathsf{pk}, \pi\rangle, \star)$ was triggered at p_i. Thanks to the CAC-Global-termination property of the CAC abstraction, we know that $Commit\_dict_k[n].\mathsf{cac\_accept}(\langle \mathsf{pk}, \pi\rangle, \star)$ will also eventually be triggered at p_j. Because p_i added $\langle n, \mathsf{pk}, \pi\rangle$ to $Names_i$, we know that the conditions at lines 21 and 22 are verified at p_j. However, the condition $\langle name, \star, \star) \notin Names_i$ at line 23 may not be verified at p_j. However, because p_k is correct, when it invokes $Commit\_dict_k[n].\mathsf{cac\_propose}(\langle \mathsf{pk}, \pi\rangle)$ at line 18, it first verified the condition $|Claim\_dict[n].candidates_i| = 1$ at line 18. Hence, only one $\mathsf{cac\_accept}$ occurs for the name n at all correct processes, thanks to the CAC-Prediction and CAC-Global-termination properties of the CAC abstraction. Therefore, only one tuple can pass the wait instruction at line 22: $\langle \mathsf{pk}, \pi\rangle$, at the index n of $Claim\_dict$. Hence, all conditions from line 21 to 23 will eventually be verified at p_j and, eventually, $\langle n, \mathsf{pk}, \pi\rangle \in Names_j$. This contradicts the hypothesis; thus, the *Agreement* property is verified. □

Lemma 8.18 (*Short-names*)**.** If all processes are correct, and given one correct process p_i, eventually we have $\forall \langle n_j, \mathsf{pk}_j, \star\rangle, \langle n_k, \mathsf{pk}_k, \star\rangle \in Names_i$:
If $|\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_k)| \geq |\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_\ell)|, \forall \langle \star, \mathsf{pk}_\ell, \star\rangle \in Names_i$ then $|\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_k)| + 1 \geq |n_j|$.

Proof. We prove Lemma 8.18 by contradiction. Let all the processes be correct, and let p_i be one of them. We assume that $\exists \langle n_j, \mathsf{pk}_j, \star\rangle, \langle n_k, \mathsf{pk}_k, \star\rangle \in Names_i, \forall \langle n_l, \mathsf{pk}_l, \star\rangle \in Names_i$:

$$|\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_k)| \geq |\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_l)|, \text{ and}$$
$$|\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_k)| + 1 < |n_j|.$$

Let us call p_j the correct process that executed $Short\_Naming.\mathsf{Claim}(\mathsf{pk}_j, \star)$. The only place where $Names_i$ is modified is at line 23. To execute this update, p_i verifies with the condition at line 22 that the tuple $\langle \mathsf{pk}_j, \star\rangle$ was cac-accepted at the index n_j of $Claim\_dict$. The validity

property of the CAC abstraction ensures that p_j cac-proposed $\langle \mathsf{pk}_j, \star \rangle$. Here, we assume that, because p_j is the only process that knows the secret key associated with pk_j, it is the only process able to execute $\mathsf{cac\_propose}(\mathsf{pk}_j, \star)$. The only place where p_j can cac-propose at index n_j of *Claim_dict* is at line 13. Furthermore, correct processes try all the sub-strings of their public keys sequentially, beginning with the first digit of the key. Hence, to cac-propose at index n_j of *Claim_dict*, it implies that, either p_j already added the name $n_j[|n_j|-1]$ to $Names_j$ associated to a public key pk_κ, where $\kappa \neq j$, or that a process p cac-proposed at index $n_j[|n_j|-1]$ of *Claim_dict*, and the $candidates_j$ set of this CAC instance contained $\langle \mathsf{pk}_\kappa, \star \rangle$, where $\kappa \neq j$. In the first case, $|\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_\kappa, \mathsf{pk}_j)| \geq |n_j| - 1$. By the *Termination* property of short naming (Lemma 8.20), we know that, eventually, $\langle n_j[|n_j|-1], \mathsf{pk}_\kappa, \star \rangle \in Names_i$, which violates the assumption. Because p_κ is correct, in the second case, it will eventually add an name whose size is greater or equal to $|n_j|$ with pk_κ as a public key to $Names_\kappa$ (*Termination*). By the *Agreement* property of short naming (Lemma 8.17), this name will be added to $Names_i$. Hence, eventually, $|\mathsf{Max\_Common\_Prefix}(\mathsf{pk}_j, \mathsf{pk}_\kappa)|+1 \geq |n_j|$, and $\langle \star, \mathsf{pk}_i, \star \rangle, \langle \star, \mathsf{pk}_\kappa, \star \rangle \in Names_i$, thus violating the assumption and concluding the proof. □

Lemma 8.19. If a correct process p_i executes $\mathsf{Choose\_Name}(j, \mathsf{pk}, \pi)$, $\forall\ j \in \{1, \cdots, |\mathsf{pk}|\}$, then either it eventually invokes $Claim\_dict[\star].\mathsf{cac\_propose}(\langle \mathsf{pk}, \pi \rangle)$, or $\langle \star, \mathsf{pk}, \star \rangle \in Names_i$.

Proof. Assuming p_i is correct and $\langle \star, \mathsf{pk}, \star \rangle \notin Names_i$, let p_i execute $\mathsf{Choose\_Name}(j, \mathsf{pk}, \pi)$, $\forall\ j \in \{1, \cdots, |\mathsf{pk}|\}$ and p_i does not invoke $Claim\_dict[\star].\mathsf{cac\_propose}(\langle \mathsf{pk}, \pi \rangle)$. Then, the process must have returned at line 10, and the condition at line 10 must be verified, *i.e.*, $i > |\mathsf{pk}|$. Hence, all the names from $\mathsf{pk}[i]$ to $\mathsf{pk}[|\mathsf{pk}|]$ were already attributed to public keys different from pk. Hence, there exists a tuple $\langle \mathsf{pk}, \mathsf{pk}', \pi \rangle \in Names_i$ where $\mathsf{pk} \neq \mathsf{pk}'$. If $Names_i$ is updated, then line 23 has necessarily been executed, and the condition at line 21 was verified. Therefore, using the perfect cryptography assumption, we have $\mathsf{pk} = \mathsf{pk}'$. □

Lemma 8.20 (*Termination*)**.** If a correct process p_i invokes $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$, then eventually $\langle \star, \mathsf{pk}, \star \rangle \in Names_i$.

Proof. Let p_i be a correct process that invokes $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$. Then, it will execute $\mathsf{Choose\_Name}(1, \mathsf{pk}, \pi)$. Using Lemma 8.19, we know that either $\langle \star, \mathsf{pk}, \star \rangle \in Names_i$, or p_i invoked $Claim\_dict[name].\mathsf{cac\_propose}(\langle \mathsf{pk}, \pi \rangle)$. In the first case, Lemma 8.20 is trivially verified. In the second case, the CAC-Local-termination property of the CAC primitive ensures that $Claim\_dict[name].\mathsf{cac\_accept}(\langle \mathsf{pk}', \star \rangle, \star)$ will be triggered at line 14. Again, two cases can arise. In the first case, p_i invokes $Claim\_dict[name].\mathsf{cac\_propose}(\langle \mathsf{pk}', \pi' \rangle)$ at line 18. In the second case, $|Claim\_dict[name].candidates_i| > 1$ and $\mathsf{Choose\_Name}(i+1, \mathsf{pk})$ is executed. Let us study the second case first. Multiple recursions might occur between the $\mathsf{Choose\_Name}$ function and the $Claim\_dict[name].\mathsf{cac\_accept}(\langle \mathsf{pk}', \star \rangle, \star)$ callback. However,

either we will end up in the first case and a cac-propose will be invoked by p_i at line 18, or Choose_Name will be eventually executed with $i = |\mathsf{pk}|$. Using the same reasoning as in Lemma 8.19, we know that p_i only receives $Claim\_dict[name].\mathsf{cac\_accept}(\langle \mathsf{pk}', \star\rangle, \star)$ if *name* is a sub-string of pk'. Hence, when $Claim\_dict[\mathsf{pk}].\mathsf{cac\_accept}(\langle \mathsf{pk}', \star\rangle, \star)$ is triggered, $\mathsf{pk} = \mathsf{pk}'$. Therefore, p_i cac-proposes $Commit\_dict_i[name].\mathsf{cac\_propose}(\langle \mathsf{pk}, \pi\rangle)$ at line 18. More precisely, we know that, if p_i invokes $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$, then p_i will eventually invoke $Commit\_dict_i[name].\mathsf{cac\_propose}(\langle \mathsf{pk}, \pi\rangle)$ or $\langle \star, \mathsf{pk}, \star\rangle \in Names_i$.

When $Commit\_dict_i[name].\mathsf{cac\_propose}(\langle \mathsf{pk}, \pi\rangle)$ is invoked by p_i, and because p_i is correct, we know that $Commit\_dict_i[name].\mathsf{cac\_accept}(\langle \mathsf{pk}, \pi\rangle, \star)$ will be triggered. Because p_i is correct, *name* is a sub-string of pk. Furthermore, the only place where p_i can cac-propose such value is at line 18. Hence, before this proposition, p_i accepted $Claim\_dict[name].\mathsf{cac\_accept}(\langle \mathsf{pk}, \pi\rangle, \star)$. Thus, at this time, condition line 22 is always verified. Hence, $\langle name, \mathsf{pk}, \pi\rangle$ is added to $Names_i$.

Therefore, when a correct process executes $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$, eventually $\langle \star, \mathsf{pk}, \star\rangle$ is added to $Names_i$. □

8.7.2 A "synchronize only when needed" CAC-based consensus algorithm: Cascading Consensus

This section present the Cascading Consensus algorithm, a CAC-based consensus algorithm (c.f. Section 4.2.1) with reduces synchronization requirements.

A CAC-based cascading consensus algorithm

Cascading Consensus (CC) builds upon the CAC abstraction to provide an *optimistic context-adaptive* algorithm in the sense that, in favorable circumstances, processes that do not propose a value do not need to synchronize with other participants. [11] Unlike other optimistally terminating consensus algorithms [141, 151, 152, 145, 153], CC achieves low latency by using information about contention (namely the *candidates* set of CAC), even when the most optimistic conditions are not met. This algorithm can be seen as an evolution of earlier optimistically terminating consensus algorithms [141, 142, 143, 144, 145]. Compared to existing solutions, using the CAC abstraction makes it possible to weaken synchronization assumptions and deal with contention more efficiently. Indeed, state-of-the-art fast consensus algorithms use a "slow-path" as soon as several different values are proposed. By contrast, the fast consensus algorithm proposed below leverages the optimal CAC implementation from Section 8.6, which provides a parameter k that balances between the size of the *candidates* set (and thus the contention) and Byzantine resilience. [12] When optimistic conditions are not met and under low contention, only

11. Differently from other consensus definitions, it is not assumed that all correct processes propose a value.
12. More precisely, the algorithm presented in Section 8.6 requires $n \geq 3t + k$, and, all things being equal, a higher value of k reduces the number of pairs in the *candidates* sets.

a few processes need to interact to resolve possible conflicts. Restricting the algorithm to small subsets of processes has two advantages. (i) These few processes are more likely to experience synchronous network phases (which are required to guarantee that a deterministic consensus algorithm terminates [161, 162, 138]), and (ii) these "restricted" synchronous phases tend to exhibit smaller network delays, leading to overall heightened efficiency.

In a nutshell, CC disseminate messages over the entire system using the CAC implementation of Section 8.6 extended with proofs of acceptance (cf. Section 8.4.3) and exploits a *Restrained Consensus algorithm* (see Section 8.7.2). Both the CAC and Restrained Consensus algorithms are fully asynchronous, *i.e.*, they do not require any (partial) synchrony assumptions. CC combines Restrained Consensus with timers to resolve conflicts rapidly among a small subset of processes during favorable synchronous phases. When circumstances are unfavorable (e.g. when network delays exceeds timeouts, or under Byzantine failures), CC falls back to a slow-path mode, which guarantees safety properties in all cases, and terminates (albeit more slowly) provided the overall system remains partially synchronous [163, 164, 162].

CC uses four sub-algorithms (two CAC instances, an instance of Restrained Consensus, and an instance of a standard consensus, which is used as fallback). It works in four steps, each step being associated with a termination condition that is more likely to be met than the previous one.

Once a process p_i has proposed a value using the first CAC instance, if there is no contention (context-adaptiveness) *i.e.*, the $candidates_i$ set of the first CAC instance has size 1, p_i can terminate: the value proposed by p_i becomes the decided value. Otherwise, if the size of the $candidates_i$ set is greater than 1 after cac-accepting the value proposed by p_i, p_i must resolve the conflict with the other processes that proposed a value, whose pairs are in $candidates_i$ (context-adaptiveness). Conflicting processes do so by using an instance of the *Restrained Consensus* (RC) algorithm presented in Section 8.7.2. If the conflicting processes are correct and benefit from stable network delays, the RC algorithm is guaranteed to succeed. In this case, the concerned processes disseminate the result of this step to the whole system using the second CAC instance. If, on the other hand, some of the processes participating in the RC algorithm are Byzantine, or if messages from correct processes are delayed too much, the RC algorithm fails. This failure is detected by the second CAC instance, which returns *candidates* sets with more than 1 pair (context-adaptiveness). In this case, the Cascading Consensus algorithm hands the final decision to its final building block, made up of a *Global Consensus* (GC), *i.e.*, any consensus algorithm based on additional assumptions such as partial synchrony [138, 161, 162], randomization [158, 165], or information on failures [139]. The implementation of GC can be chosen without any constraint. However, if an asynchronous consensus algorithm is chosen to instantiate GC, then CC implements consensus under fully asynchronous assumptions.

Table 8.4 summarizes the termination conditions of the CC algorithm and their associated

Abstraction	Operations	Communication model	Nb of participants
Context-Adaptive Cooperation (CAC)	cac_propose(v) cac_accept(v, i)	Asynchronous	n
Cascading Consensus (CC)	ccons_propose(v) ccons_decide(v)	Async. for whole system Sync. for RC	n
Restrained Consensus (RC)	rcons_propose(v) rcons_decide(E, S_e, S_r) rcons_no_decision()	Synchronous	ℓ (where $\ell \ll n$)
Global Consensus (GC)	gcons_propose(v) gcons_decide(v)	Any	n

Table 8.3 – Notations for the different abstractions used in this section.

Condition	Assumption needed	Execution path	Nb of system-wide rounds	Nb of RC rounds
No conflict	$n > 5t$	CAC_1 (fast path)	1	N/A
No conflict (slow path)	$n > 3t$	CAC_1 (slow path)	2	N/A
All procs of RC correct and sync.	$n > 5t$	CAC_1; RC; CAC_2 (fast path)	3	1
All procs of RC correct and sync.	$n > 3t$	CAC_1; RC; CAC_2 (slow path)	4	1
≥ 1 Byzantine proc. in RC or async. period	$n > 3t$	CAC_1; RC; CAC_2; GC	4 + GC round	1

Table 8.4 – Summary of the "progressively degrading" conditions of Cascading Consensus (instantiated with the optimal CAC algorithm of Section 8.6), and their associated round complexity.

round complexity. The table considers two types of rounds: the fourth column counts *system-wide rounds—i.e.*, for one asynchronous round, each process has to send n messages. The final column counts the asynchronous rounds executed by RC—*i.e.*, for RC round, the ℓ processes that execute RC have to send a message to all other $\ell-1$ involved processes. With fewer processes involved, the asynchronous rounds of RC will typically be faster (measured in wall-clock time) than those of the whole network. The "execution path" column details where in the algorithm a process terminates by listing the sub-algorithm instances (noted CAC_1, RC, CAC_2, and GC) that intervene in a process execution. For instance, the first row describes the most favorable scenario in which a correct process terminates after the first round of the first CAC instance.

In the following, we first detail the workings of the Restrained Consensus algorithm (RC) (Section 8.7.2), before diving into the operations of Cascading Consensus (Line 16).

Restrained consensus (RC)

Restrained Consensus (RC) is an abstraction used uniquely as a building block of CC. It consists of a weakened consensus algorithm in which only a subset Π' of the processes in Π interact. It aims to resolve a conflict between the processes that proposed a value in the first CAC instance of a the CC algorithm. When all participating processes are correct and enjoy favorable

network conditions, Restrained Consensus terminates with a decision. However, in the presence of asynchrony or Byzantine faults, it may fail to agree on a proposed value. This behavior allows Cascading Consensus to resolve a conflict efficiently in good cases while falling back to full-fledged consensus when the restrained-consensus algorithm fails. Formally, Restrained Consensus has one operation rcons_propose(C, π), where C is a set of tuples $\langle v, i \rangle$—with v a value and i the identifier of a process in Π—and π is a proof of acceptance of one of the pairs in C obtained after the first CAC instance (which uses proofs of acceptance). It proves that the process that invokes rcons_propose() is legitimate to do so, as processes can only participate in the Restrained Consensus algorithm if they cac-proposed a value in the first CAC instance, and if this value was cac-accepted.

RC is parameterized by a timeout parameter δ_{RC} and is defined for two sets Π and Π' of processes, such that $\Pi' \subseteq \Pi$. A correct process $p_i \in \Pi$ is in Π' if it invokes the rcons_propose operation with a valid proof of acceptance π, or if it is in the *candidates* set of a process that invoked the rcons_propose operation.

RC has two callbacks: rcons_no_decision() and rcons_decide(E, *endorse_sigs*, *retract_sigs*), where E is a set of tuples $\langle v, i \rangle$ with v a value and i a process identifier; *endorse_sigs* is a set of signatures on E by all the processes p_i in E; and *retract_sigs* is a set of signatures of the string `"RETRACT"`. Restrained Consensus is defined by the following properties. [13]

- *Weak validity.* If all the processes in Π' are correct, the network delays between the processes of Π' are less than or equal to δ_{RC}, and a process invokes rcons_propose, then at least one process p_i executes the callback rcons_decide(E, *endorse_sigs*, *retract_sigs*) and *endorse_sigs* $\cup$ *retract_sigs* contains a signature from each process in Π'. Furthermore, E is a subset of the values proposed by p_i.
- *Weak agreement.* If all the processes in Π' are correct and if a process p_i executes the callback rcons_decide(E, *endorse_sigs*, *retract_sigs*) and a process p_j executes the callback rcons_decide(E', *endorse_sigs*$'$, *retract_sigs*$'$), then $E = E'$ and *endorse_sigs* = *endorse_sigs*$'$.
- *Integrity.* A correct process p_i invokes at most once either rcons_decide($\star, \star, \star$) or rcons_no_decision() (but not both in the same execution).
- *Termination.* Any correct process in Π' eventually invokes rcons_no_decision() or rcons_decide().

Restrained consensus: algorithm

Algorithm 9 implements the Restrained Consensus abstraction. It is designed to be used as a component of a Cascading Consensus algorithm. In particular, it relies on the presence of a correct proof of acceptance π to identify the processes in Π' that invoke the rcons_propose

13. An implementation of Restrained Consensus is presented in Section 8.7.2, along with its proof.

operation. The implementation is based on signatures. The goal of a process that participates in the algorithm is to gather one signature from each process in Π'. The algorithm uses two types of messages, RCONS-SIG and RCONS-RETRACT. The RCONS-SIG message is used as the primary mechanism to propagate and gather signatures. For a correct process to send a RCONS-SIG, it must possess the proof of acceptance of one of its own messages. When a correct process receives a RCONS-SIG message, it has to immediately respond with another RCONS-SIG message if it possesses a proof of acceptance for one of its values. Otherwise, it has to answer with a RCONS-RETRACT message. The RCONS-RETRACT message type proves that the process that sends it does not participate in the Restrained Consensus.

The algorithm relies on a timer T_{RC} that must be chosen to allow correct processes in Π' to terminate their operations if they are in a synchronous period. If this condition holds, the algorithm can terminate in two synchronous periods. The first is the time it takes for the initial broadcast to reach all participants, and the second is the time to respond to this initial broadcast. Therefore, the duration of T_{RC} can be chosen as two times the expected latency of the network δ_{RC} composed of the processes in Π', *i.e.*, $T_{RC} = 2 \times \delta_{RC}$.

Restrained consensus: proof

The proof that Algorithm 9 implements the Restrained Consensus abstraction defined in Section 8.7.2 follows from the following lemmas.

Lemma 8.21 (*Weak validity*)**.** If all the processes in Π' are correct, the network delays between the processes of Π' are lesser or equal to δ_{RC}, and a process invokes rcons_propose, then at least one process p_i executes the callback rcons_decide(E, *endorse_sigs*, *retract_sigs*) and *endorse_sigs* $\cup$ *retract_sigs* contains a signature from each process in Π'.

Proof. If all the processes in Π' are correct the network delays between the processes of Π' are lesser or equal to δ_{RC}, and a process invokes rcons_propose, then at least one process p_i verifies the condition at line 3. This process broadcasts a RCONS-SIG message to all the processes in Π'. All the processes in Π' are correct and synchronous. Therefore, they will answer this message either with a RCONS-RETRACT message or with another RCONS-SIG message before the timer T_{RC} of p_i runs out. Hence, p_i receives a signature from all the processes in Π' before T_{RC} runs out. Either the received signatures are in $endorse\_sigs_i$ (*i.e.*, the process answered with a RCONS-SIG message) or in $retract\_sigs_i$ (*i.e.*, the process answers with a RCONS-RETRACT message). In any cases, the condition at line 11 is verified at p_i, and p_i executes rcons_decide(E, $endorse\_sigs_i$, $retract\_sigs_i$), where $endorse\_sigs_i \cup retract\_sigs_i$ contains the signatures of all the processes in Π'. Furthermore, the values in E are intersections with the original value proposed by p_i (lines 4 and 20). Hence, E is a subset of the values proposed by p_i. □

```
1  init: retract_i ← false; power_C_i ← ∅; endorse_sigs_i ← ∅; retract_sigs_i ← ∅; π_i ← ∅.
2  operation rcons_propose(C, π) is
3      if retract_i = false and p_i has not already rcons-proposed and π is valid then
4          power_C_i ← P(C);                          ▷P denotes the powerset function
5          endorse_sigs_i ← endorse_sigs_i ∪ {signature by p_i for each element in power_C_i};
6          π_i ← {π};
7          be_broadcast RCONS-SIG(endorse_sigs_i, power_C_i, π_i);
8          T_RC.start().
9  internal operation check_decision() is
10     if endorse_sigs_i contains the signatures of all the processes of the largest element E
        in power_C_i and rcons_decide or rcons_no_decision have not already been invoked
        then
11         rcons_decide(E, endorse_sigs_i, retract_sigs_i).
12 when RCONS-SIG(endorse_sigs, power_C, π) is received do
13     if π is invalid or a signature in endorse_sigs has no proof or a value with a proof in π
        is not in C or a signature in endorse_sigs is invalid then rcons_no_decision();
14     π_i ← π_i ∪ π; endorse_sigs_i ← endorse_sigs_i ∪ endorse_sigs;
15     if p_i has not rcons-proposed before and retract_i = false then
16         retract_i ← true;
17         power_C_i ← power_C;
18         be_broadcast RCONS-RETRACT({sig. of "RETRACT" by p_i});
19         if T_RC has not been started then T_RC.start().
20     power_C_i ← power_C_i ∩ power_C;
21     check_decision();
22 when RCONS-RETRACT(retract_sigs) is received do
23     power_C_i ←
        power_C_i \ {value in power_C_i associated to the process that signed retract_sigs};
24     retract_sigs_i ← retract_sigs_i ∪ {retract_sigs};
25     check_decision().
26 when T_RC.end() do rcons_no_decision().
```

Algorithm 9: Restrained consensus implementation (code for p_i).

Lemma 8.22 (*Weak agreement*)**.** If all the processes in Π' are correct and if a process p_i executes the callback rcons_decide(E, *endorse_sigs*, *retract_sigs*) and a process p_j executes the callback rcons_decide(E', *endorse_sigs*$'$, *retract_sigs*$'$), then $E = E'$ and *endorse_sigs* = *endorse_sigs*$'$.

Proof. Let all the processes in Π' be correct and synchronous. Let $R = \{p_1, \cdots, p_r\}$ be the set of processes that executed the callback rcons_decide. Let $\{\langle E_1, endorse\_sigs_1, retract\_sigs_1\rangle, \cdots, \langle E_r, endorse\_sigs_r, retract\_sigs_r\rangle\}$ be the tuples returned to $\{p_1, \cdots, p_r\}$ respectively. All the processes in R invoked the operation rcons_propose, and they verified the condition at line 3. Therefore, they all broadcast their signatures on all the possible sub-elements of C. Furthermore, the only processes able to broadcast a RCONS-SIG message are the processes with a proof that their value is valid. By assumption, these processes are included in all the *candidates* set of all the participating processes. Hence, all the processes in R will receive the signatures of all the other processes in R. Furthermore, they will all receive the same set of *power_C* sets. Therefore, $\forall\ i, j \in \{1, \cdots, r\} : endorse\_sigs_i = endorse\_sigs_j$ and $E_i = E_j$. □

Lemma 8.23 (*Integrity*)**.** A correct process p_i can invoke at most once either rcons_decide($\star, \star, \star$) or rcons_no_decision (but not both in the same execution).

Proof. This lemma is trivially verified by the condition line 10. This condition ensures that rcons_decide callback can only be triggered once, and cannot be triggered if rcons_no_decision has already been triggered. Furthermore, the timer T_{RC} is only started once (lines 3 and 19), hence, the callback rcons_no_decision can only be triggered once. □

Lemma 8.24 (*Termination*)**.** Any correct process in Π' eventually executes rcons_no_decision or rcons_decide.

Proof. A process in Π' is a process that executed the rcons_propose operation without receiving any prior message, or that received a RCONS-SIG message before executing the rcons_propose operation. In the first case, the process will start T_{RC} at line 8. In the second case, the process does not meet the condition at line 15. Therefore, it start T_{RC} at line 19. These two cases are mutually exclusive. Once a process starts T_{RC}, it cannot start it again (lines 3 and 19). Finally, when the timer expires, it executes the callback rcons_no_decision. Therefore, any correct process in Π' will terminate. □

```
1  init: π_i ← ∅.
2  operation ccons_propose(v) is CAC_1.cac_propose(v).
3  when CAC_1.cac_accept(v, j, π) do
4  |  π_i ← π_i ∪ {π};
5  |  if (|CAC_1.candidates_i| = 1 or all values in CAC_1.candidates_i are the same) and
   |   ccons_decide has not already been triggered then ccons_decide(v);
6  |  else if j = i then rcons_propose(cac_1.candidates_i, π) ;
7  |  else T_CC.start().                    ▷start timer with a duration of 2 × δ_RC + δ_CC
8  when RC.rcons_decide(E, endorse_sigs, retract_sigs) do
9  |  CAC_2.cac_propose(⟨E, endorse_sigs, retract_sigs, π_i⟩).
10 when RC.rcons_no_decision() is invoked or T_CC.end() do
11 |  CAC_2.cac_propose(⟨CAC_1.accepted_i, ∅, ∅, π_i⟩).
12 when CAC_2.cac_accept(⟨E, ⋆, ⋆, ⋆⟩, j, π) do
13 |  if (|CAC_2.candidates_i| = 1 or all values in CAC_1.candidates_i are the same) and
   |   ccons_decide has not already been triggered then ccons_decide(choice(E));
14 |  else if p_i has not already ccons-proposed a value then
   |   GC.gcons_propose(⟨CAC_2.candidates_i, π⟩).
15 when GC.gcons_decide(⟨E, ⋆⟩) do
16 |  if ccons_decide has not already been triggered then ccons_decide(choice(E)).
```

Algorithm 10: Cascading Consensus implementation (code for p_i).

Context-adaptive Cascading Consensus: implementation

Algorithm 10 presents the Cascading Consensus algorithm. It relies on two instances of the CAC abstraction, CAC_1 and CAC_2, one instance of Restrained Consensus (RC) and one instance of Global Consensus (GC) (as all the processes in Π participate in it). The list of all different abstractions is summarized in Table 8.3 (where *endorse_sigs* and *retract_sigs* are respectively replaced by S_e and S_r).

When a process p_i cac-accepts a tuple from one of the CAC instances, it can fall into either of the following two cases.

1. $|candidates_i| = 1$: p_i detects there is no conflict, so it knows that other correct processes cannot cac-accept any other value, and it can immediately decide the value it received. [14]
2. $|candidates_i| > 1$: p_i detects multiple candidate values, so it must continue the algorithm to resolve the conflict.

A conflict in CAC_1 leads to the execution of Restrained Consensus (RC) among the par-

14. Multiple processes can propose the same value, while being detected by the CAC abstraction as conflicting propositions. Therefore, if $|candidates_i| > 1$, but all the proposed values are the same, *e.g.*, v, then v is immediately decided.

ticipants involved in the conflict (line 6). A conflict in CAC_2 leads to the execution of Global Consensus (GC) among all the system participants (line 14).

In CAC_2, the set of values cac-accepted in the prior steps are proposed. To simplify the presentation of the algorithm, the pseudo-code omits some implementation details. In particular, CAC_2 verifies the proofs associated with the proposed values. A correct process p_i considers a set of values E cac-proposed by a process p_j in CAC_2 only if either one of the following conditions holds:

— p_j did not propose one of the values in E during CAC_1—*i.e.*, it did not participate in RC—and each value in E is associated with a valid proof of acceptance.
— p_j proposed one of the values in E during CAC_1—*i.e.*, it participates in RC—and E is signed by all the processes that proposed values in E. Furthermore, each process whose value proposed in CAC_1 is eventually accepted signed the string "`RETRACT`".

Similarly, the values proposed in the Global Consensus are also associated with a proof of acceptance from the second instance of the CAC algorithm. We assume that the Global Consensus implementation cannot decide a value not associated with a valid proof of acceptance.

Note that, if a correct process p_i cac-accepts a value with $|candidates_i| = 1$, it does not necessarily imply that other correct processes will have the same *candidates* set. The processes that detect a conflict execute one of the consensuses, restrained or global. However, the algorithm ensures, using acceptance proofs, that only a value that has been cac-accepted in a previous step can be proposed for the next step. Hence, if a correct process p_i cac-accepts a value v with $candidates_i = \{\langle v, \star \rangle\}$, the other processes will not be able to propose $v' \neq v$ in the following steps of the algorithm—by the prediction property of the CAC abstraction. In other words, some correct processes may terminate faster than others, but this early termination does not impact the agreement of the protocol.

Like the RC algorithm described in Section 8.7.2, the Cascading Consensus algorithm uses a timer, T_{CC}. This timer provides the operation T_{CC}.start() to start the timer, and the callback T_{CC}.end(), which is invoked once the time has elapsed. The duration of T_{CC} should be long enough to allow the processes participating in Restrained Consensus to terminate if they are in a synchronous period. Subject to this condition, the algorithm can terminate in 2 synchronous periods for Restrained Consensus plus 1 synchronous period to initiate the second instance of the CAC abstraction. Therefore, the duration of T_{CC} should ideally equal $2 \times \delta_{RC} + \delta_{CC}$, where δ_{RC} is the likely latency of the sub-network of all participants of Restrained Consensus and δ_{CC} is the likely latency of the network composed of all the processes in Π. However, if T_{CC} is chosen too small, the safety and liveness properties of CC are still ensured.

8.7.3 Cascading Consensus: proof

The proof of correctness that the Cascading Consensus algorithm presented in Algorithm 10 implements consensus follows from the subsequent lemmas.

Lemma 8.25 (*Validity*)**.** If all processes are correct and a process decides a value v, then v was proposed by some process.

Proof. By exhaustion, we explore the three following cases.

— If a value is decided at line 5, then it is the result of the first CAC instance. Thanks to the CAC-Validity property, we know that a process in Π proposed this value.

— If a value is decided at line 13, then it is the result of CAC_2. The only values cac-proposed using CAC_2 are a set of values cac-accepted from CAC_1, either they are cac-proposed by a process that participated in rcons or not. Thanks to the CAC-Validity property, we know that a process in Π proposed this value.

— If a value is decided at line 16 then the value was decided by the GC instance. However, the values proposed to GC are values accepted by CAC_2. Thanks to the *Validity* of GC and CAC-Validity of CAC_1 and CAC_2, we know that a process in Π proposed this value. □

Lemma 8.26 (*Agreement*)**.** No two correct processes decide different values.

Proof. A correct process that participates in the Cascading Consensus can decide at different points of the execution of the algorithm: lines 5, 13 or 16. However, if a correct process decides at line 5 or 13, not all correct processes will necessarily do so.

Nonetheless, the CAC-Global-termination property ensure that if a correct process decides before the others, all the correct processes will decide the same value.

Let us assume that a correct process p_i decides a value v at line 5. This implies that CAC_1 outputs a $CAC_1.candidates_i$ set of size 1 or all the values in $CAC_1.candidates_i$ are the same for p_i after the first cac-acceptance. Using the CAC-Prediction and CAC-Global-termination, we know that p_i will not cac-accept any value different from v with the CAC_1 instance. Furthermore, using CAC-Global-termination, we know that no other correct process can cac-accept a value $v' \neq v$. Otherwise, p_i would also cac-accept it, contradicting the CAC-Prediction property. Therefore, if p_i decides v at line 5, all correct processes that do not ccons-decide at this point will only cac-accept v with CAC_1. Furthermore, the values cac-proposed in CAC_2 are those that were cac-accepted by CAC_1. Therefore, v is the only value that is cac-proposed in CAC_2. Using the CAC-Global-termination and CAC-Prediction property, we know that all the correct processes will ccons-decide v at line 13.

Similar reasoning can be applied if a correct process decides at line 13 whereas others do not. The only values that can be gcons-proposed are those cac-accepted in CAC_2. Therefore,

using the CAC-Global-termination and CAC-Prediction properties of CAC, we know that if a correct process ccons-decided a value v at line 13, then all correct processes that did not ccons-decide at this point will ccons-decide v at line 16.

Finally, if no process decides at line 5 or 13, then the *Agreement* property of consensus ensures that all the processes ccons-decide the same value at line 16. □

Lemma 8.27 (*Integrity*)**.** A correct process decides at most one value.

Proof. This lemma is trivially verified. All the lines where a process can decide (lines 5, 13 and 16) are preceded by a condition that can only be verified if the process did not already triggered ccons_decide. Hence the *Integrity* property is verified. □

Lemma 8.28 (*Termination*)**.** If a correct process proposes value v, then all correct processes eventually decide some value (not necessarily v).

Proof. All the sub-algorithms used in Cascading Consensus (CAC, RC, and GC) terminate. Furthermore, each algorithm is executed sequentially if the previous one did not decide a value. The only algorithm that may not be triggered is RC if CAC_1 terminates with a $candidates_i$ set whose size is greater than 1 at p_i, and if p_i did not cac-proposed one of the values in $candidates_i$. However, we observe that processes not participating in the RC algorithm set a timer T_{CC} when CAC_1 returns. Once this timer expires, these processes cac-propose a value using CAC_2. Therefore, any correct process that participates in the Cascading Consensus terminates. □

8.8 Conclusion

This chapter introduced a new cooperation abstraction denoted "Context-Adaptive Cooperation" (CAC). This abstraction allows multiple processes to propose values while multiple value acceptances are triggered. Furthermore, each acceptance comes with information about other acceptances that can possibly occur. Two implementations of CAC have been presented. The first one is a simple algorithm that works in asynchronous networks when $n > 4t$. The second is a latency and Byzantine resilient optimal implementation, which uses fine-tuned thresholds to improve efficiency and reduce the probability of contention. This second implementation works in three asynchronous rounds if $n > 3t$ and in two asynchronous rounds in favorable cases when $n > 5t$.

This new cooperation abstraction can be used in low-contention distributed applications to improve efficiency or remove the need for synchronization. This chapter proposed two such examples, where the CAC abstraction can be used to build distributed algorithms. The first is an optimistically terminating consensus algorithm denoted Cascading Consensus. This algorithm (as some other consensus algorithms, *e.g.*, [150, 144]), can optimistically terminate when there is

no contention or the inputs satisfy specific patterns. However, differently from other algorithms that do not use the CAC abstraction, Cascading Consensus is the first to use information about contention to restrain synchronization to the processes that actually proposed a value. Furthermore, and unlike other optimistically terminating consensus algorithms, cascading consensus terminates optimistically even if multiple processes propose different values. The second example is a short-naming algorithm, which works deterministically in fully asynchronous networks. It allows processes to claim names based on their public keys. However, contrary to other asynchronous naming algorithms, the claimed name is a sub-string of the public key, thus reducing the size of the name space, making it easier for humans to handle.

Both example can be used to improve PPfDIMS efficiency. In the following (Chapters 9 and 10), we use the cascading consensus abstraction as our main communication abstraction. Furthermore, we use the short-naming algorithm to create DIDs when the Human-Choosable property is not required (c.f. Chapter 7), *i.e.*, when the actor that creates the DID is an individual that will not act as an identity provider or a service provider.

Chapter 9

An efficient solution to the multi-device authorization problem: the Anonymous Agreement Proof

This chapter uses the results from Chapter 6 to build a multi-device authorization framework for PPfDIMS. This multi-device authorization framework is a new solution to an important problem in the PPfDIMS ecosystem. It uses a new abstraction, the Anonymous Agreement Proof (AAP) abstraction. The AAP abstraction is formally defined, and an implementation based on Threshold Anonymous Credentials is proposed. Then, the AAP abstraction is used to build a multi-device authorization framework for PPfDIMS.

9.1 Introduction

In the DIMS world, there is a consensus building [13, 12, 14] on the fact that distributed ledgers should only be used to enable auxiliary features. More precisely, the authorization mechanism should be a bipartite protocol between a holder and a verifier. It should not require the intervention of any third party, be it an issuer, a trusted third party, or a distributed ledger. It should be conducted through the creation and the exchange of a Verifiable Presentation (VP) from a Verifiable Credential (VC) [5].

However, a credential's presentation cannot, in itself, provide authentication or other auxiliary features. Among the features that require the intervention of a third party, we can cite the identification of schemas used in the VP, the verification of the revocation of the credentials used to build the VP, and the non-transferability of a credential [166] while enabling multi-device authorization capabilities [19]. In this section, we focus on the latter.

Non-transferability is the ability to link a credential to a specific individual. We only want the legitimate holder of a credential to be able to build VPs from it. Non-transferability is often required by IMSs as presenting a stolen (or shared) credential that do not implement the non-

transferability property can authorize the wrong person to access a potentially sensitive service or information. For example, article 441-1 of the French penal code states that the forgery of a document (which includes the usage of a stolen document) is condemned by 3 years of imprisonment and a 45000€ penalty. However, non-transferability can conflict with usability. If non-transferability is enforced by linking a credential to a device, [1] then the usage of this credential is limited. For example, if the credential is linked to a phone, then if the phone is lost or damaged, the credential holder must request a new issuance for the same credential. To avoid those problems, PPfDIMSs must provide multi-device authorization capability. The multi-device authorization capability authorizes multiple devices to use a credential while maintaining strong authentication capabilities, *i.e.*, credentials cannot be transferred to non-authorized devices.

Non-transferability and multi-device authorization capability can be seen as two contradictory properties. The first one tries to limit the possibilities of presenting a credential, whereas the second one tries to make it possible to share a credential between a limited number of predefined devices. Another challenge is to provide such properties while preserving user's privacy. Privacy requirements directly discard approaches based on unique identifiers revealed to the verifiers, *e.g.*, a method consisting of linking a credential to a DID and explicitly presenting this DID to a verifier to prove that a device is authorized for this DID is therefore discarded.

In the early days of anonymous credentials, only two approaches existed to enable non-transferability. Both worked in the same way. Verifiable Credentials are represented as a signature on a vector of messages. One of these messages is a secret identifier. When presenting the credential, the holder produces a ZKP of knowledge of this identifier without revealing its exact value. Then, the first approach is the biometrically enforced non-transferability [167, 168]. This method consists of building the secret identifier using biometric hardware. Therefore, the credential can only be used by a specific biometric feature holder. We will not explore this approach in the rest of this thesis, as it requires specific hardware and additional security assumptions. The second approach is the PKI-assured-non-transferability. With this approach, the identifier is a secret whose associated public key is registered at a trusted third party (the PKI). Proving knowledge of the secret at the PKI makes it possible to access valuable assets like sensitive information or a large amount of money. An evolution of this scheme is the all-or-nothing non transferability [34], where knowledge of one secret implies knowledge of all the potential secrets of an individual.

In 2019, the Hyperledger Aries group proposed a new way of enabling non-transferability and multi-device authorization capabilities [19]. This method relies on the existence of a DID-capable ledger. The idea is to associate each device owned by an individual with a specific public/secret key pair. The public keys are registered in the user's DID document. Those public keys identify the devices allowed to use VCs associated with this DID. Each user's credential embeds the

1. This is the method used by AnonCred [33]. This link is created through an element called a *linked secret*.

DID identifier. When the user presents a credential, he also provides a (zero knowledge) proof that the secret key they know is associated with the DID embedded in the credential. The issue with the solution proposed by Hyperledger is that it relies on heavy cryptographic primitives: multiple commitment schemes, multiple Zero Knowledge Proof of Knowledge, and multiple Zero Knowledge Proof of Set Membership. Furthermore, only a draft of the protocol exists. Therefore, analyzing its security, performance, and privacy properties is challenging. Finally, their have been no update since 2019 about their multi-device authorization framework. Thus it is probable that they no longer consider it a viable solution.

Interestingly, the DIF Wallet Security working group [169], created in 2023, is currently working on a new Distributed Key Management System (DKMS) standard. This standard uses the Hyperledger Aries DKMS [19] as one of its building blocks. Furthermore, the problem of multi-device authorization has been assessed in a recent paper [170] about challenges for Self Sovereign Identity. Therefore, we may see interesting new developments in the near future concerning non-transferability and multi-device authorization capabilities for PPDIMSs.

Because of the lack of usable propositions to solve the multi-device authorization problem, we propose in this chapter a new way to provide non-transferability and multi-device authorization capabilities for PPfDIMSs. This proposal is based on embedded valuable secrets and uses DIDs like the Hyperledger Aries proposal. Unlike the Hyperledger Aries proposal, we provide a new way to efficiently authenticate a device and prove it is authorized. This method uses a threshold anonymous credential scheme (whose only representant to this day is the Coconut threshold anonymous credential scheme [20]), along with the access control management policy formerly used by Hyperledger Aries Distributed Key Management System [19]. The threshold AC scheme makes it possible to have as much flexibility as Hyperledger Aries' DKMS, without the heavy cryptographic computations and storage overhead. This new way of anonymously proving information stored in a distributed ledger is formalized as a new distributed abstraction called the Anonymous Agreement Proof (AAP) abstraction.

The contributions of this chapter are the following:

— We formalize the need for a multi-device authorization scheme with non-transferability;
— We formalize the new AAP abstraction;
— We propose an implementation of the AAP abstraction in message-passing, and we prove it implements the AAP abstraction; and
— We use the AAP abstraction to build a PPfDIMS with the multi-device authorization capability and the non-transferability property.

9.2 System model

We consider a message-passing model as defined in Section 4.1.2.

Furthermore, we assume we have access to a theoretical Verifiable Credential (VC) scheme which consists of three algorithms $\mathsf{Issue}, \mathsf{Present}$ and Verify. The algorithm $vc \leftarrow \mathsf{Issue}(att, \mathsf{sk}_I, \mathsf{pk}_d, \mathsf{aux})$ is invoked by an issuer $I \in \mathcal{I}$. It takes as input a set of attributes $att \subseteq \mathcal{ATT}$, where $\mathcal{ATT}$ is the set of all potential attributes that can exist, the public key pk_d of a device $d \in \mathcal{D}$, the secret key $\mathsf{sk}_I \in \mathcal{SK}$ of the issuer I and some auxiliary inputs aux. It outputs $vc \in \mathcal{VC}$ a Verifiable Credential that certifies that I has verified that the device holder d is characterized by the identity elements att.

The algorithm $(vp, \pi_I) \leftarrow \mathsf{Present}(vc, \mathsf{sk}_d, T_v, \mathsf{aux})$ takes as input a verifiable credential $vc \in \mathcal{VC}$, a device's secret key sk_d, a set of issuers T_v and some auxilary information aux. It outputs a verifiable presentation $vp \in \mathcal{VP}$, where $\mathcal{VP}$ is the set of all potential verifiable presentations. vp is a curated version of vc, where some identity elements in att may be hidden, and others may be used to perform zero-knowledge proofs. Furthermore, as discussed in the rest of this chapter, vp must contain proof that the device that produced it is authorized to do so. The operation also returns π_I, a proof that vc was issued by one of the issuers in T_V.

Finally, the algorithm $\{0, 1\} \leftarrow \mathsf{Verify}(vp, T_v, \pi_I, \mathsf{aux})$ takes as input a verifiable presentation $vp \in \mathcal{VP}$, a set of trusted issuers T_v, a proof π_I that proves that the issuer I that issued the verifiable credential vc that was used to construct vp is in T_v and some auxiliary information aux. It outputs 1 if the verifiable presentation vp is valid, I is the issuer of vc, and vc was used to build vp. It outputs 0 otherwise.

A Verifiable Credential scheme fulfills the correctness, trusted issuer, and Existential Unforgeability Against Chosen Message Attack properties defined in Chapter 5. Additionally, it can fulfill the issuer-indistinguishability property.

9.3 Problem statement

A PPfDIMS must fulfill multiple requirements to enable the multi-device authorization capability while maintaining strong authentication properties. These requirements can be separated into a usability requirement, two security requirements, and a privacy requirement. This section defines those requirements.

First, the usability requirement states that the PPfDIMS must enable the multi-device authorization capability. A user who owns a VC must be able to share it with its different devices. The user must be able to authorize those devices, and the user must be able to build verifiable presentations from each authorized device. The authorization process is done using the $\mathsf{Authorize}(vc, \mathsf{pk}, \mathsf{sk}, \mathsf{aux})$ operation. This operation takes as input a verifiable credential $vc \in \mathcal{VC}$, the public key pk of the device d_a that is being authorized, the secret key sk of the device that authorizes d_a and some auxiliary information aux. Furthermore, the user needs to be able to revoke devices. The revocation is conducted through the $\mathsf{Revoke}(vc, \mathsf{pk}, \mathsf{sk}, \mathsf{aux})$ operation, where

vc is a verifiable credential, pk is the public key of the revoked device d_r, sk is the secret key of the device that revokes d_r and aux is a variable containing auxiliary information. Formally, the usability property can be stated as follows:

Definition 9.1. Multi-device authorization capability. Let $d_1 \in \mathcal{D}$, $d_2 \in \mathcal{D}$, $d_a \in \mathcal{D}$ and $d_r \in \mathcal{D}$ be four devices, let sk_{d_1}, sk_{d_2}, sk_{d_a} and sk_{d_r} be their secret keys respectively and let pk_{d_1}, pk_{d_2}, pk_{d_a} and pk_{d_r} be their public keys respectively. Let $I \in \mathcal{I}$ be an issuer, and let sk_I be its secret key. Let att be a set of attributes. The issuer computes the verifiable credential $vc = \mathsf{Issue}(att, \mathsf{sk}_I, \mathsf{pk}_{d_1}, \star)$ and sends it to d_1. We assume d_2 and d_r are able to build a verifiable presentation $(vp_i, \pi_i) = \mathsf{Present}(vc, \mathsf{sk}_{d_i}, T_v, \star), \forall i \in \{2, 4\}$ such that $\mathsf{Verify}(vp_i, T_v, \pi_i, \star) = 1$. A PPfDIMS fulfills the multi-device authorization capability property if:

— There exists an operation $\mathsf{Authorize}$ such that, if d_2 invokes $\mathsf{Authorize}(vc, \mathsf{pk}_{d_a}, \mathsf{sk}_{d_2}, \mathsf{aux})$, then d_a is authorized to build verifiable presentations from vc, *i.e.*, if $(vp', \pi'_I) = \mathsf{Present}(vc, \mathsf{sk}_{d_3}, T_v, \star)$ then $\mathsf{Verify}(vp', T_v, \pi'_I, \star) = 1$.

— There exists a Revoke operation, such that if d_2 invokes $\mathsf{Revoke}(vc, \mathsf{pk}_{d_r}, \mathsf{sk}_{d_2}, \mathsf{aux})$, then d_r is no longer able to build verifiable presentations from vc, *i.e.*, $(vp'', \pi''_I) = \mathsf{Present}(vc, \mathsf{sk}_{d_r}, T_v, \star)$ then $\mathsf{Verify}(vp'', T_v, \pi''_I, \star) = 0$.

This definition does not describe how d_2 itself is authorized to invoke the $\mathsf{Authorize}$ operation nor the Revoke operation. This is intentional; we define how transitivity makes it possible to authorize devices later. We do not want the device that originally receives the credential to be the only device able to invoke the $\mathsf{Authorize}$ or the Revoke operation. Otherwise, if this device is lost, the credentials that were issued to this device will become unusable after some time. Therefore, we consider that any device authorized to present a credential can also authorize other devices to present it. [2]

In order to present the security requirements, we need to analyze an execution of a PPfDIMS where multiple $\mathsf{Authorize}$ and Revoke operations are invoked in a sequence. In this case, two consecutive $\mathsf{Authorize}(vc, \mathsf{pk}, \mathsf{sk}, \mathsf{aux})$ followed by a $\mathsf{Revoke}(vc, \mathsf{pk}, \mathsf{sk}, \mathsf{aux})$ are different from an $\mathsf{Authorize}(vc, \mathsf{pk}, \mathsf{sk}, \mathsf{aux})$ operation, followed by a $\mathsf{Revoke}(vc, \mathsf{pk}, \mathsf{sk}, \mathsf{aux})$ operation, itself followed by a final $\mathsf{Authorize}(vc, \mathsf{pk}, \mathsf{sk}, \mathsf{aux})$ operation. In the first case, the device d associated with pk is authorized by the first $\mathsf{Authorize}$ operation, the second $\mathsf{Authorize}$ operation has no effect, and the Revoke operation revokes d. Ultimately, d is not authorized to build verifiable presentations for vc. In the second case, however, d is first authorized, then revoked, and finally re-authorized. In the end, d is authorized. Hence, $\mathsf{Authorize}$ and Revoke operations are not commutative for a given verifiable credential. Thus, a PPfDIMS with multi-device authorization and authentication capabilities must implement a total order on the $\mathsf{Authorize}$ and Revoke operations of a given VC:

2. In Section 9.7, we restrict this statement. We differentiate the list of devices allowed to present credentials from those allowed to authorize and revoke other devices. This separation was originally proposed by the Hyperledger Aries project [19] and makes it possible to have better control over authorization policies. However, the formal definition presented here mixes the access control role with the presentation role for simplicity.

Definition 9.2. Authorize **and** Revoke **order.** For a given VC vc, the Authorize and Revoke operations appear to all the correct processes in Π as in a sequence seq_{vc}. The order of the operations is seq_{vc} are the same for each correct process in vc.

The first security requirements restrict the use of a VC to authorized devices. A device $d \in \mathcal{D}$ is authorized to present a VC vc if the credential was directly issued to d (*i.e.*, a VC vc is issued to d if the issuer I whose secret key is sk_I invoked the operation $vc = \mathsf{Issue}(\star, \mathsf{sk}_I, \mathsf{pk}_d, \star)$), or if an authorized device invoked the Authorize operation in favor of d, and no consequent Revoke operation was invoked against d. Therefore, this security property defines a transitive relation of authorized devices. We call this property the strong authorization property. Before formally defining the strong authorization property, we define the concept of an authorized device. Informally, an authorized device is a device that can build a valid, verifiable presentation from a verifiable credential.

Definition 9.3. Authorized device. Let I be an issuer, and sk_I be its secret key. A device $d \in \mathcal{D}$, whose secret key is sk_d and whose public key is pk_d is said to be an authorized device at time τ of $\mathsf{seq}_{\mathsf{vc}}$ for the verifiable credential $vc = \mathsf{Issue}(\star, \mathsf{sk}_I, \star, \star)$ if, given any verifier $v \in \mathcal{V}$ and its set of trusted issuers T_v where $I \in T_v$, the invocation of $\mathsf{Verify}(vp, T_v, \pi_I, \star)$ outputs 1 at time τ in $\mathsf{seq}_{\mathsf{vc}}$, where $(vp, \pi_I) = \mathsf{Present}(vc, \mathsf{sk}_d, T_v, \star)$.

Using the authorized device definition, we can formally define the strong authorization property:

Definition 9.4. Strong authorization for PPfDIMS. Let $d \in \mathcal{D}$ be a device, let sk_d be its secret key, and let pk_d be its public key. Let $I \in \mathcal{I}$ be an issuer, and let sk_I be its secret key. Let vc be a verifiable credential such that $vc = \mathsf{Issue}(\star, \mathsf{sk}_I, \mathsf{pk}_{d'}, \star)$, where $d' \in \mathcal{D}$. The device d is an authorized device at time τ of $\mathsf{seq}_{\mathsf{vc}}$ for the credential vc if $d = d'$ or if another device d'' whose secret key is $\mathsf{sk}_{d''}$ is an authorized device for vc at time $\tau' \leq \tau$ of $\mathsf{seq}_{\mathsf{vc}}$ and d'' invoked the $\mathsf{Authorize}(vc, \mathsf{pk}_d, \mathsf{sk}_{d''}, \star)$ operation at time τ'. Furthermore, no $\mathsf{Revoke}(vc, \mathsf{pk}_d, \star, \star)$ operation appears in seq_{vc} before time τ.

The definition of the strong authorization property implicitly implies that a process that is not authorized by the issuer (through the Issue operation) or by another authorized device (through the Authorize operation) cannot create a verifiable presentation from vc. However, the authorized device definition does not restrict some non-authorized devices from having a non-negligible probability of succeeding in a the creation of a verifiable presentation from vc. As this would be equivalent to identity theft, we need a second security property to restrict this behavior. This property is called theft resistance and can be seen as an authentication property. The theft resistance property is defined as follows:

Definition 9.5. Theft resistance. Let $d \in \mathcal{D}$ be a device, let sk_d be its secret key, and let pk_d be its public key. Let $I \in \mathcal{I}$ be an issuer, and let sk_I be its secret key. Let att be a set

of attributes. Let vc be a verifiable credential such that $vc = \mathsf{Issue}(att, \mathsf{sk}_I, \mathsf{pk}_d, \star)$. At time τ in $\mathsf{seq}_{\mathsf{vc}}$, vc is the only VC that any issuer in I has issued. Let $auth$ be a set of authorized devices for the verifiable credential vc at time τ. Let $\mathcal{A}$ be an adversary with access to vc, to all the issuers' public keys, and to all the public and secret keys in the system, except the secret keys of the devices in $auth$ and the issuer's secret keys. Furthermore, $\mathcal{A}$ controls the Byzantine processes in Π. A PPfDIMS fulfills the theft resistance property if the probability for $\mathcal{A}$ to output a tuple (vp^*, π_I^*) such that $\mathsf{Verify}(vp^*, \star, \pi_I^*, \star) = 1$ at time τ is negligible.

The last property required to build a multi-device authorization mechanism for a PPfDIMS is one that preserves users' privacy during the presentation of a VC. The privacy requirement comes from the minimalization property stated by Allen [1]. This implies that the multi-device authorization capability must not impact the user's privacy. Therefore, the multi-device authorization method cannot rely on explicitly revealing uniquely identifying identifiers or revealing extra information about the user rather than the information required to be revealed to be authorized by the verifier. This privacy requirement assumes that all the issuers, all the processes implementing the distributed ledger, and all the verifiers can collude and act in an honest but curious manner. Furthermore, Byzantine processes that implement the distributed ledger can exhibit malicious behavior. The definition of this property is expressed with an adversary and a challenger. Each actor's role is similar to the role of the challenger and the adversary presented in Chapter 5. The adversary tries to identify a user, while the challenger is a ubiquitous actor that provides elements to the adversary. The privacy property defines the indistinguishability between two verifiable presentations after the multi-device authorization capability has been activated. It is formalized as follows:

Definition 9.6. Privacy preserving multi-device authorization capability. Let $I \in \mathcal{I}$ be an issuer whose secret key is sk_I and whose public key is pk_I. Let $d_1, d_2, d_1', d_2' \in \mathcal{D}$ be four devices whose secret keys are $\mathsf{sk}_{d_1}, \mathsf{sk}_{d_2}, \mathsf{sk}_{d_1'}$ and $\mathsf{sk}_{d_2'}$ respectively and whose public keys are $\mathsf{pk}_{d_1}, \mathsf{pk}_{d_2}, \mathsf{pk}_{d_1'}$ and $\mathsf{pk}_{d_2'}$ respectively. Let $v \in \mathcal{V}$ be a verifier whose set of trusted issuers is T_v. Let $I \in T_v$. We assume an adversary $\mathcal{A}$ that controls all the issuers in $\mathcal{I}$, all the Byzantine processes in Π, and the verifier v. $\mathcal{A}$ knows all the secret keys existing in the system except $\mathsf{sk}_{d_1}, \mathsf{sk}_{d_2}, \mathsf{sk}_{d_1'}$ and $\mathsf{sk}_{d_2'}$. All processes can act maliciously except the correct processes in Π that act in an honest but curious manner. The challenger controls d_1, d_2, d_1' and d_2'. Let att be a set of attributes. I issues two credentials vc and vc' such that $vc = \mathsf{Issue}(att, \mathsf{sk}_I, \mathsf{pk}_{d_1}, \star)$ and $vc' = \mathsf{Issue}(att, \mathsf{sk}_I, \mathsf{pk}_{d_1'}, \star)$. The challenger invokes $\mathsf{Authorize}(vc, \mathsf{pk}_{d_2}, \mathsf{sk}_{d_1}, \star)$ and $\mathsf{Authorize}(vc', \mathsf{pk}_{d_2'}, \mathsf{sk}_{d_1'}, \star)$. No other $\mathsf{Authorize}$ nor Revoke operation are invoked during the execution. The challenger tosses a uniformly random bit $b \leftarrow\!\!\$\, \{0, 1\}$. If $b = 0$, then the Challenger computes $(vp, \pi_I) = \mathsf{Present}(vc, \mathsf{sk}_{d_2}, T_v, \star)$. Otherwise it computes $(vp, \pi_I) = \mathsf{Present}(vc', \mathsf{sk}_{d_2'}, T_v, \star)$. The challenger transfers vp and π_I to $\mathcal{A}$. The adversary outputs b', a bit in $\{0, 1\}$. We say that a PPfDIMS enables privacy-preserving multi-device au-

thorization capability if the probability for the adversary to output $b = b'$ is lesser or equal to $\frac{1}{2} + \epsilon$, where ϵ is a negligible value.

9.4 Data model and authorization mechanism

The solution proposed for the multi-device authorization problem in this chapter uses the same data model as the one proposed by Hyperledger Aries' Distributed Key Management System (DKMS) [171]. This model uses DID and DID documents for their authorization mechanism. In this model, VCs are linked to DIDs. To prove that a device is authorized to present a VC vc, a device must prove that it knows a secret key associated with one of the public keys listed in the DID document associated to the DID.

When a user enters the system, he creates a DID and a DID document. He creates it using a public key and a secret key. The public key is stored by the ledger in the DID document, and the secret key is stored on the user's device. This device becomes the only authorized device for this DID. The data model exhibits two different roles associated with two different types of authorizations. The first role is the prover role. A device (the public/secret key pair associated with a device) with this role is allowed to build verifiable presentations from any credential that is associated with the DID. The second role is the manager role. A device with the manager role can grant or revoke authorizations to other devices. This separation of roles improves security, as a device with a high probability of getting corrupted can still be used to present credentials. However, if it gets stolen, the thief cannot revoke access to other devices.

As stated earlier, this data model assumes that credentials are not linked to devices but to DIDs (in a privacy-preserving manner that we will describe later). Hence, and unlike the formal definition we gave in Section 9.3, one cannot authorize a device for a given credential but for all the credentials associated with a DID. DID-based authorization simplifies the access control policies, thus improving usability.

The authorization mechanism works as follows. Any verifiable credential can be associated with a DID (in a way we will describe later). When a device presents this credential, it builds a proof that the device's public key is associated with the credential's DID (with the prover role). Furthermore, the device proves it knows the secret key associated with the public key. Thus, the verifier knows the device is authorized to use the credential. This behavior is equivalent to an AllowList that implements the Append and the Remove operations.

9.5 Anonymous Agreement Proof: an abstraction to efficiently prove ledger-agreed data

This section defines the Anonymous Agreement Proof (AAP) abstraction. It is a new abstraction that lets a process p from a set of processes $\mathcal{P}$ prove to the other processes in $\mathcal{P}$ that another set of processes Π agreed on the state s of an object in a privacy-preserving manner. Privacy is the main constraint in this setting. It is the reason why numerous previous works failed to propose a usable solution to the multi-device authorization problem. The only previous solution proposed (the Hyperledger Aries draft [19]) did so by letting the ledger processes build a table of each relations between a device's public key and a DID. With this solution, the ledger processes must maintain multiple accumulators representing those relations. Furthermore, the prover and the verifier have to agree on the state of those accumulators. Finally, the prover has to build multiple zero-knowledge proofs of set membership to prove that they know the secret key associated with one of the public keys authorized for a given DID in the list of relations. This implies heavy computation for the verifier and the prover and heavy computation and storage cost for the ledger.

The solution proposed in this section circumvents the problem. Rather than proving that there exist a link between a public key and a DID registered by the ledger among all the links registered by the DID, our solution is to prove the state of a DID document off-ledger. We do this by letting processes in Π build a proof of the state of a given DID document. This proof is non-interactive, *i.e.*, its verification does not requires the participation of the processes of the ledger. The ledger computes this proof in a privacy preserving manner. The proof is then stored by each DID owner. To verify the state of a DID document, processes only have to verify that the proof of the state of a DID document is valid, *i.e.*, it has been produced by processes in Π. However, this proof of the state of a DID document only informs the verifier that this is a state that existed, not that it is the current state of the DID document. Hence, the prover and the verifier have to synchronize their views on the list of up-to-date DID documents of the ledger, and the prover must prove that the state whose validity was proven is in the set of up-to-date DID documents. They could only synchronize their view on the specific DID document used by the prover, but this would make the identification of the user trivial; thus, they have to synchronize their view on the state of all DID documents. The processes in Π only stores the updated states and proofs it issued.

Therefore, when a device wants to present a credential to a verifier, it only has to prove that it is authorized to do so using the non-interactive proof issued by the ledger and by proving that this proof is associated with one of the up-to-date states as seen by processes in Π. This last proof is a proof of set-membership. As stated in Chapter 6, techniques exist to build such proofs in Zero Knowledge. Thus, if the non-interactive proof provided by the ledger preserves

the user's privacy, the scheme we propose trivially preserves the user's privacy.

This section formalizes the Anonymous Agreement Proof (AAP) abstraction. This distributed abstraction makes it possible to prove to a set of processes $\mathcal{P}$ that some other processes Π agreed on the state s of an object and that this state is not outdated. The AAP abstraction is the privacy-preserving version of the Agreement Proof (AP) abstraction, which has been formalized by Albouy et al. [172]. On the other hand, even though it has not been formalized as such, ZEF [173], an anonymous asset transfer protocol, also uses a weaker form of Anonymous Agreement Proof.

We define the AAP abstraction for objects identified by identifiers. Identifiers are DIDs in the context of this thesis. However, they can be generalized to other identifiers, *e.g.*, a coin in an anonymous asset transfer protocol (c.f., ZEF anonymous asset transfer protocol [173]). The AAP defines an access control policy where the last updated document (in the case of a DID document) is the only valid document. Thus, the abstraction can be seen as a specific type of Append/Remove AllowList, where the manager set is Π, and the prover set is $\mathcal{P}$.

Definition 9.7. The AAP abstraction has two operations, Update and Prove, and three callbacks Updated, Authorized and NotAuthorized. It is defined for a set of objects identified by identifiers in the set $\mathcal{ID}$. Furthermore, we define the set $\mathcal{S}$ of all valid states of the system's objects. We also define the set Π of n_Π processes $p_1, \ldots, p_{n_\Pi}$ that participate in any Update operation and the set $\mathcal{P}$ of $n_\mathcal{P}$ processes $\rho_1, \ldots, \rho_{n_\mathcal{P}}$ that participate in any Prove operation. Both sets do not have to be disjoint. We define a threshold $t_\Pi < n_\pi$ on the maximum number of Byzantine processes in Π, and a threshold $t_\mathcal{P} < n_\mathcal{P}$ on the maximum number of Byzantine processes in $\mathcal{P}$. For each object, we define the set $\mathcal{M}_{Id}^{(s)} \subseteq \Pi$ of managers for a given state s of the object with identifier Id and a set $\Psi_{Id}^{(s)} \subseteq \mathcal{P}$ of provers for a given state s of the object with identifier Id.

The Update$(Id, \pi_{\text{UpdateAuth}}, s, \mathsf{aux})$ operation is invoked by a correct process $p \in \Pi$. It takes as input Id the identifier of the object modified, $\pi_{\text{UpdateAuth}}$, a proof that p is authorized to modify the object identified with Id (a proof that p is in the set $\mathcal{M}_{Id}^{(s')}$ of managers, where s' is the previous state of the object identified by Id), $s \in \mathcal{S}$ the new state of the object, and some auxiliary parameters aux. The proof $\pi_{\text{UpdateAuth}}$ must be context-aware, *i.e.*, it must be resistant against replay attacks. The Updated$(Id, \pi_{\text{UpdateAuth}}, \sigma, s)$ is eventually trigerred at all correct processes in Π if an Update$(id, \pi_{\text{UpdateAuth}}, s, \mathsf{aux})$ operation was invoked, and if $\pi_{UpdateAuth}$, s and aux are valid. The σ value returned is a proof of agreement of the processes in Π on the updated state s of the object Id.

The Prove$(s, \sigma, nonce, \pi_{\text{auth}}, \mathsf{aux})$ operation is invoked by an authorized device $\rho \in \mathcal{P}$. It takes as input s the state of an object, σ an agreement proof that s is an up-to-date state of an object, π_{auth} a context-aware proof that ρ is authorized to invoke the Prove operation (a proof that ρ is in the set $\Psi_{Id}^{(s')}$ of provers, where s' is the previous state of the object identified by Id), *nonce*

is an identifier that (anonymously) links the Prove operation to subsequent operations invoked by p [3] and some auxiliary parameters aux.

The Authorized$(C_s, nonce)$ callback is triggered at all processes in the set of verifiers $\mathcal{P}$ after a valid Prove operation is invoked. It returns *nonce*, an identifier that (anonymously) identifies the process ρ_i that invoked the Prove operation, and C_s, a commitment to the state of an object. The NotAuthorized$(s, nonce)$ callback, on the other hand, is triggered at all the correct processes if the state s is not the most up-to-date state (a new Updated$(Id, \star, s', \star)$ callback has been triggered at a correct process).

The operations and callbacks must fulfill the following properties:

— Update *termination.* If a correct process $p \in \Pi$ invokes Update$(Id, \pi_{\text{UpdateAuth}}, s, \star)$ with $Id \in \mathcal{ID}$, $s \in \mathcal{S}$ and $\pi_{\text{UpdateAuth}}$ is a correct context-aware proof that $p \in \mathcal{M}_{Id}^{(s')}$ where s' is the previous state of the object identified by Id, then a callback Updated$(Id, \star, \star, \star)$ is eventually trigerred at p.

— Update *validity.* If the callback Updated$(Id, \pi_{\text{UpdateAuth}}, \star, s)$ is triggered at a correct process $p \in \Pi$, then $\pi_{\text{UpdateAuth}}$ is a valid context-aware proof of authorization that the process that built the proof is in $\mathcal{M}_{Id}^{(s')}$, where s' is the previous state of the object identified by Id.

— Update *agreement.* If the callback Updated$(Id, \pi_{\text{UpdateAuth}}, \sigma, s)$ is trigerred at a correct process $p \in \Pi$, then it is eventually trigerred at each the correct processes $p' \in \Pi$.

— *State agreement.* For an object identified by the identifier Id, Updated callbacks are triggered in the same order at all correct processes $p \in \Pi$.

— Prove *validity.* If the callback Authorized$(C_s, \star)$ is trigerred at a correct process $\rho \in \mathcal{P}$, where C_s is a commitment to a state $s \in \mathcal{S}$, then the callback Updated$(\star, \star, \star, s)$ was trigerred at at least one correct process $p' \in \Pi$.

— Prove *agreement.* If the callback Authorized$(C_s, nonce)$ is trigerred at a correct process $\rho \in \mathcal{P}$, then it is eventually trigerred at all the correct processes $\rho' \in \mathcal{P}$.

— *Local* Append/Remove *antiflickering.* If an Authorized$(C_s, nonce)$ callback is triggered at a correct process $\rho \in \mathcal{P}$ at time τ_1 of its local clock and then, the invocation of Prove$(s, \star, nonce, \star, \star)$ by ρ triggers the NotAuthorized$(nonce)$ callback at time $\tau_2 > \tau_1$ of the local clock of ρ, then no Authorized$(C'_s, \star)$ can be trigerred at ρ after τ_2, where C_s and C'_s are commitments to the state s.

— *Progress.* Let an Updated$(Id, \star, \star, s)$ callback be triggered at a correct process $p \in \Pi$ at time τ_1 of its local clock, and an Updated$(Id, \star, \star, s')$ callback be triggered at p at time $\tau_2 > \tau_1$ of its local clock. No other Updated$(Id, \star, \star, \star)$ operation is triggered at p during the algorithm's execution. Let $\rho_1 \in \mathcal{P}$. Then, eventually no Authorized$(C_s, \star)$ callback can

3. *nonce* can be the public key of a secret/public key pair or the hash of a statement that will be proven later if ρ is authorized.

be triggerred at ρ_1 where C_s is a commitment to s.
Furthermore let $\rho_2 \in \mathcal{P}$ be a correct process such that $\rho_2 \notin \Psi^{s}_{Id}$ and $\rho_2 \in \Psi^{s'}_{Id}$. If other processes in $\mathcal{P}$ are correct and do not invoke Prove operations, then, eventually, a Prove$(s', \sigma, nonce, \pi_{\text{Auth}}, \star)$ operation invoked by ρ_2 triggers an Authorized$(C_{s'}, nonce)$ callback where $C_{s'}$ is a commitment to s' if π_{Auth} is a correct context-aware proof that $\rho_2 \in \Psi^{(s')}_{Id}$ and σ is a valid proof that the Updated$(\star, \star, \star, s')$ callback was triggered at a correct process in Π.

— *Anonymity.* Let p be a process controlled by a PPT adversary $\mathcal{A}$. $\mathcal{A}$ also controls Byzantine processes in $\mathcal{P}$ and Π. When the callback Authorized$(C_s, nonce)$ is triggered at p, where C_s is the state of an object identified by Id, then the probability for p to output Id or s is negligible.

The AAP abstraction can be seen as a specific case of privacy preserving AllowList, which supports the Remove operation. In this case, the Update operation is an atomic Remove and Append operation, where the old document's state is revoked, and the new one is authorized atomically.

The Update termination property ensures that if a correct process p invokes the Update$(Id, \star, s, \star)$ operation with valid arguments, then the Updated callback will eventually be triggered. However, this property does not ensure that the new state of the Id updated by p will be s. A concurrent Update operation may occur and modify Id differently. On the other hand, the Update validity property ensures that if the Updated callback is triggered at a correct process, then valid arguments are used to trigger this callback. In other words, if the invoking process of the Update operation is correct, it uses valid arguments. If the process is Byzantine, it acts similarly to a correct process.

The Update agreement and the state agreement properties are used to ensure that correct processes in Π agree on the evolution of the states of an object Id. They have to see the same state and in the same order for a given object. Those properties thus ensure that the result of the Prove operations are consistent among different processes.

The Prove validity property ensures that an Authorized callback can only be triggered at a correct process if the process that invoked the Prove operation dit it with valid arguments, or the invoking process is Byzantine, however, it acted similarly to a correct process.

The Prove agreement property ensures consistency on the Authorized callbacks. All the processes in $\mathcal{P}$ must receive the same authorized callbacks. This property is the translation in message passing of the READ validity property defined in Chapter 6.

The Append/Remove antiflickering property is the translation in message passing of the antiflickering property defined in Chapter 6. It has been modified to cope with Append and Remove operations used in the AAP abstraction. It ensures that an Authorized callback cannot be triggered for a commitment to a state s that has already been seen as outdated by the

processes in $\mathcal{P}$. In other words, no process can force the processes in $\mathcal{P}$ to accept an outdated state.

The progress property is also similar to the progress property defined in Chapter 6. The property defined in this chapter considers the progress property of an AllowList and the progress property of a DenyList. This behavior is necessary as the AAP object behaves as a combination of an AllowList and DenyList (an AllowList with Append and Remove operations). The negative part of the property (the part corresponding to the DenyList's progress property) is equivalent to the one defined in Chapter 6, and eventually, an Update operation will be taken into account by the processes in $\mathcal{P}$ for the Prove operations. On the other hand, the positive part of the property (the part corresponding to the AllowList's progress property) is less restrictive than the one defined in Chapter 6. Due to concurrency, it is not possible to guarantee that a $\mathsf{Prove}(s, \star, \star, \star)$ operation will result in a $\mathsf{Authorized}(C_s, \star)$ callback (where C_s is a commitment to s). Therefore, the property only copes for progress if there is no concurrency. This trade-off have been made due to the difference between the shared memory model and the message passing model. However, this behavior does not impact security as the most important property for security is the negative part of the property.

Finally, the anonymity property ensures that a $\mathsf{Prove}(s, \star, star, \star)$ operation does not reveal information about the state s nor the identifier Id of the object defined by this state.

The AAP abstraction is designed to work in a client/node model. This model copes with the reality of distributed algorithms. Because consensus algorithms require heavy assumptions for processes, *i.e.*, processes cannot disconnect and they have a high computational burden, it is likely that many individuals will not possess a process that can participate in the system. Therefore, the client/node model assumes that there are few processes in Π, *i.e.*, processes that maintain the system. On the other hand, the system users can query the processes in Π and benefit from the distributed algorithm they implement. The owners of the objects are, therefore, those clients, whereas the nodes, *i.e.*, the processes in Π, are here only to maintain the system. Therefore, we use proofs of authorization (the proofs $\pi_{\text{UpdateAuth}}$) rather than authorizing specific processes in Π to modify a specific object to cope with this specific setting.

On the other hand, the set $\mathcal{P}$ can be composed of the clients directly. Indeed, in our use case, this set will generally only consist in two processes, the prover and the verifier. Thus, ad hoc $\mathcal{P}$ sets can be built for each new relationship between a prover and a verifier. More formally, this case that will be explored in Section 9.7 can be seen as a set of ℓ AAP objects, where ℓ is the number of different potential prover/verifier pairs. The set Π and the state of the objects (*i.e.*, the Updated callbacks) are shared for each AAP object. However, the ℓ sets $\mathcal{P}$ and the associated Authorized callbacks are unique for each AAP object.

9.6 Implementation of the AAP abstraction using threshold anonymous credential scheme

This section presents an algorithm that implements the Anonymous Agreement Proof abstraction. This implementation is based on two main components: a distributed ledger maintained by the processes in Π and a proving mechanism of the state of the objects as seen by the processes of this distributed ledger. The distributed ledger is used to implement the Update operations. Processes in Π verify each update of the state of an object by verifying that the process that invokes the operation is authorized to do so using the $\pi_{\text{UpdateAuth}}$ proof. On the other hand, the Prove operation is handled by other processes that may be different from the processes in Π. Similarly to the Agreement Proof implementation proposed by Albouy et al. [172], our implementation is based on an "off-line" proof that correct processes agreed on the state s of an object identified with Id. Furthermore, we need a way to prove that this state s is up to date, *i.e.*, no $\mathsf{Update}(Id, \star, \star, \star)$ was invoked in the meantime.

9.6.1 Cryptographic tools

The Update operation must let the processes in Π issue to the process that requested its invocation an "off-line" proof of agreement. This proof of agreement must prove that processes in Π agreed on the state s of an object Id and that this proof is up to date to be usable in a Prove operation. The first part of this problem can be solved by proving that at least one correct process did agree on the state s of the object. Indeed, the Update agreement property ensures that if one correct process agrees on a state, all correct processes will eventually agree. A tool to provide such an agreement proof is to use cryptographic signatures. However, a cryptographic signature does not fulfill the anonymity property of the abstraction by default. Indeed, several problems arise. First, a signature that is not randomizable acts as a unique identifier. An AAP scheme may require multiple presentations of proof of agreement. Thus, the signature must be randomizable. Furthermore, if the adversary can be one of the signers, which is the case in our threat model, the verifier's knowledge of the signed message may identify the user. Therefore, ZKP of signatures are required. Finally, Byzantine processes may try to identify the Id of an object during a verification by acting maliciously. An example of such behavior is a Byzantine process that only signs one agreement proof during the whole execution of the protocol. If it sees its signature in a verification afterward (if it has to use its own public key to verify it), it knows which invocation of the Update operation led to the creation of this signature. Therefore, we need a scheme that solves those three problems at the same time. The two first problems can be solved using anonymous credentials (c.f., Chapter 5). However, a classical anonymous credential scheme does not solve the third challenge. A modified version of a hidden issuer anonymous credential scheme could be a solution. However, our first explorations in this direction led to impractical,

computational-heavy, and storage-heavy solutions. A second solution that only tackles the third challenge is using threshold signatures [87]. In a system of n processes, a threshold signature makes it possible, given signatures from t processes, to build a unique signature that proves that t processes signed a message without revealing their identity. However, a threshold signature scheme in itself does not solve the two first requirements. Interestingly, a paper by Danezis et al. proposed an implementation of a threshold anonymous credential [20]. This scheme solves the abovementioned three points by modifying the Pointcheval Sanders [84] signature scheme to add threshold signature capabilities. In the rest of this section, we assume the existence of such threshold anonymous credential scheme noted TAC. As stated in Chapter 4, this AC scheme has two additional algorithms in comparison with a classical AC scheme, $TAC.\mathsf{AggKey}$ that is used to create the public key of the system used to verify an aggregated signature and $TAC.\mathsf{AggCred}$ that is used to aggregate signatures. We write pk_{system}, the public key used to verify an aggregated signature from the processes in Π.

Our AAP implementation requires two additional cryptographic schemes. Those schemes can be implemented in numerous ways. Therefore, we only give a high-level description of their inputs and outputs. We also give ways of implementing those schemes without choosing a specific one. First, we need an authentication protocol. The goal of this protocol is for a process p to prove that it is authorized to perform an action. This proving mechanism is done using any signature scheme or ZKP scheme. In both cases, p signs a challenge built by the processes in Π (or a challenge built from the context of the Update operation) to authenticate itself and then sends this signature to the processes that need to authenticate p. This protocol is secure assuming that no process shares its secret keys (except Byzantine processes, but the abstraction properties are not guaranteed for them). The public and secret keys are created with the algorithm $\mathsf{sk}, \mathsf{pk} \leftarrow Auth.\mathsf{KeyGen}()$. The proof of knowledge of the secret key is created with an algorithm $\pi_{auth} \leftarrow Auth.\mathsf{Prove}(\mathsf{sk}, \mathsf{pk})$. This proof is verified with the algorithm $\{0,1\} \leftarrow Auth.\mathsf{Verify}(\mathsf{pk}, \pi_{auth})$ that outputs 1 only if π_{auth} is a valid proof of knowledge of the secret key sk associated to the public key pk. Alternatively, pk can be a randomizable Pedersen commitment. Thus, the verifier does not learn the actual value of pk when it invokes $Auth.\mathsf{Verify}$. This additional feature will be necessary to fulfill the anonymity property for the Prove operation.

The second cryptographic tool required is a Zero Knowledge Proof system for Set Membership (ZKPoSM). This is used to ensure that the state of an object is up-to-date without revealing the identifier or the precise state of the object. As stated in Chapter 6, such schemes exist and can be efficiently computed [120]. A ZKPoSM has two operations $\pi \leftarrow ZKPoSM.\mathsf{Prove}(value, set)$ which proves that the value $value$ is in the set set and $\{0,1\} \leftarrow ZKPoSM.\mathsf{Verify}(\pi, set)$ which outputs 1 if π is a valid proof of set membership of a value in the set set. The proof π does not reveal any information about the value $value$ used to compute it.

9.6.2 Communication primitives

Our AAP implementation uses two communication primitives. The main primitive is a unicast/best-effort broadcast primitive. This type of primitive is used when no ordering of messages is required. The second type of communication primitive used is a consensus algorithm. This type of communication primitive is used when ordering is necessary, *i.e.*, to fulfill the local Append/Remove antiflickering and the state agreement properties of the AAP specification. We use the Cascading Consensus algorithm presented in Chapter 8 for efficiency reasons. This algorithm makes it possible to be optimal in terms of synchronization requirements. However, we must carefully partition the Cascading Consensus instances to avoid unnecessary contention. To do so, each object modified by the Update operation is associated with its own Cascading Consensus instance. Each Prove operation is also associated with its own Cascading Consensus. Furthermore, the abstraction authorizes multiple Update and Prove operations to be invoked sequentially. Therefore, we need multi-shot consensus algorithms. To achieve this goal, we slightly modify the parameters of the Cascading Consensus primitives. The Cascading Consensus used during an Update operation is augmented with the identifier of the modified object and the sequence number of this modification. This sequence number starts at 0 and monotonically grows. Each sequence number is unique for each identifier. The updated parameters of the Update's Cascading Consensus are $\mathsf{ccons\_propose}_{\mathsf{Update}}(Id, sn, value)$ where Id is the object's identifier, sn is the sequence number, and $value$ is the value of the message. Each unique (Id, sn) pair constitute a unique Cascading Consensus instance. This consensus is conducted among the processes in Π. For a Prove operation, we only use a sequence number written Seq_Prove. This sequence number starts at 0 and monotonically grows. The Seq_Prove counter needs to be shared for the different object's proofs as the anonymity property prevents the use of an object's identifier, *i.e.*, Seq_Prove grows with each new Prove operations, even if those operations are not related to the same object. The parameters of the Cascading Consensus used during a Prove operation are $\mathsf{ccons\_propose}_{\mathsf{Prove}}(Seq\_Prove, value)$. Each unique sequence number Seq_Prove constitute a unique Cascading Consensus instance.

The AAP implementation uses an additional tool, the RequestObjValue(Π) operation. This operation, invoked by a process in $\mathcal{P}$, requests to a process, or to processes in Π, their current view on the state of the objects in the system (their Obj_i variable, c.f. Section 9.6.3) along with the TAC signatures of those states. This operation is necessary as processes in $\mathcal{P}$ need to know the result of the Update operations whereas, if they are not in Π, they do not receive the Updated callbacks. This operation can be implemented in three different ways. The first method can be applied in the special case where the invoking process $p \in \mathcal{P}$ is also in Π; it only needs to use its local value. The second method can be applied if the invoking process $p \in \mathcal{P}$ trusts at least one process in Π, then it only needs to request this process that is assumed correct and uses its response. Finally, if the invoking process $p \in \mathcal{P}$ does not trust any process in Π, it can request at

least $t_\Pi + 1$ processes in Π and uses the most up-to-date version of each object's state. p verifies the signatures of each response, and then, for each object Id, it uses the most up-to-date state. Thus, it knows it has the most up-to-date view of those $t_\Pi + 1$ processes. Furthermore, verifying the TAC signatures ensures that the chosen states were signed using at least one correct process. Moreover, because there can only be t_Π Byzantine processes in Π, p knows one of the responses came from a correct process. This implies that Byzantine processes cannot break the progress property by answering the RequestObjValue(Π) operation with outdated versions of the objects on purpose.

9.6.3 Implementation

The working of our Anonymous Agreement Proof algorithm is the following. Each object identified by the identifier $Id \in \mathcal{ID}$ is associated with a set of managers' public keys $\mathcal{M}_{Id}^{(s)}$ and a set of provers' public keys $\Psi_{Id}^{(s)}$. Those sets are encapsulated in the object itself for simplicity. A process p whose secret key is sk_p and public key $\mathsf{pk}_p \in \mathcal{M}_{Id}^{(s)}$ has control over an object identified by the identifier Id, *i.e.*, it can invoke the Update operations on this object. For simplicity's sake, we assume each object exists at initialization. This assumption can be relaxed using an Identifier System object (c.f., Chapters 7 and 10). Each object consists of four fields:

— the set of managers' public keys $\mathcal{M}_{Id}$, initiated with one public key, it defines the processes authorized to invoke Update operations;
— the set of provers' public keys Ψ_{Id}, initialized at $\emptyset$, it defines the processes authorized to invoke Prove operations;
— a sequence number sn_{Id} which identifies the version of the object, it is initialized at 0 and monotonically grows;
— a random identifer r_{Id} initialized at 0. It changes with each Update and consists of the public key of a public/secret key pair and is used to conduct the ZKPoSM; and
— the object description, that is initialized at $\emptyset$ and that depends on the implemented object itself.

To update an object Id (to invoke a valid Update operation), an authorized process first needs to draw a new secret/public key pair. The public key r'_{Id} is used as the new random identifier for the object, whereas the secret key is used to avoid impersonation attacks. Then, p shares with processes in Π the new state s' of the object, the new random identifier r'_{Id}, and $\pi_{r'_{Id}}$ the proof of knowledge of the secret key associated to r'_{Id}. This dissemination is operated with a Cascading Consensus instance $ccons_{\mathsf{Update}}$. Using a consensus algorithm makes it possible to ensure the state agreement property. Once the dissemination ends, each process signs the new state s' of the object and its sequence number r'_{Id_p} using the threshold anonymous credential signature scheme and broadcasts its share to all the processes in Π. Processes in Π store the new state s' of the object Id. Processes aggregate $t_\Pi + 1$ shares of TAC signatures, where t_Π is the

maximum number of Byzantine processes in Π. This threshold ensures that at least one correct process agreeed on the new state of the object. Thus, all correct processes will eventually agree on this state. The aggregated signature is stored by processes in Π, along with the new state of the object. The aux variable is used during an Update operation to store $\pi_{r'_{Id}}$ the proof of knowledge of the secret key associated with r'_{Id}.

Importantly, we assume a value v can be $\mathsf{ccons\_decide}_{\mathsf{Update}}$ at line 9 of Algorithm 11 only if it passes conditions lines 3 and 7 pass. This assumption can be fulfilled if the $ccons_{\mathsf{Update}}$ algorithm is modified to only accept values that passes those conditions.

The Prove operation consists of proving that an aggregated signature σ is a valid proof of agreement of correct processes in the system, that the associated state s of the object Id is not outdated, and that the process that invokes the operation is authorized to invoke it, *i.e.*, that it knows a secret key associated to one of the public keys in Ψ_{Id}. Additionally, this operation cannot reveal s nor Id. The algorithm's main part consists of synchronizing the processes' views in $\mathcal{P}$ concerning the state of each object $\forall Id \in \mathcal{ID}$. To do so, the invoking process $p \in \mathcal{P}$ sends a ProveReq message. This message is used as a request to the other processes to share their view on the states of the objects. The ProveReq message is associated with a sequence number Seq_Prove. This sequence number is used to enforce the antiflickering property. Only one Prove operation can succeed for each Seq_Prove. After receiving a ProveReq message, processes will update their view of the objects by using the RequestObjValue operation (c.f. Section 9.6.2). They compare the operation result with their local view of the states of the objects, and, for each Id, they only keep the state with the greatest sequence number sn (the most up-to-date state). Finally, they send the resulting set to p with a ProveAns message and a signature of the set.[4] p gathers at least $n_{\mathcal{P}} - t_{\mathcal{P}}$ of those signatures and for each Id, it selects the state with the greatest sequence number sn (the most up-to-date state). The resulting set is called $Prove\_Obj$. Then, if $s \in Prove\_Obj$, p builds a ZKP of set membership for s. It sends this set membership proof to the other processes in V using the $\mathsf{ccons\_propose}_{\mathsf{Prove}}$ operation. The consensus algorithm enables the antiflickering property (see Chapter 6). Additionally, p needs to prove that $Prove\_Obj$ is a set of states that were the result of the Updated callback and that they are the most up-to-date sets as seen by the $n_{\mathcal{P}} - t_{\mathcal{P}}$ processes in $\mathcal{P}$ that sent a ProveAns message. This is done by sharing the signatures of those $n_{\mathcal{P}} - t_{\mathcal{P}}$ processes and their local view of the states of the objects. When processes receive the $\mathsf{ccons\_decide}(SeqProve, \star)$ callback, they verify each signature and they verify that the $Prove\_Obj$ set received is the most up-to-date version of each state for each Id as seen by the $n_{\mathcal{P}} - t_{\mathcal{P}}$ that signed messages with a sequence number $SeqProve$. Once this verification is complete, processes only have to verify the different ZKPs, and they can trigger the Accepted callback.

4. Processes sign the set along with the sequence number Seq_Prove. This additional element is used to mitigate replay attacks.

However, s may no longer belong to $Prove\_Obj$ when p receives the $n_{\mathcal{P}} - t_{\mathcal{P}}$ ProveAns messages. In this case, p needs to inform the other processes in the system to ensure the antiflickering property. Therefore, it only sends the curated set $Prove\_Obj$ along with the signatures of the processes that sent ProveAns messages. Therefore, processes in $\mathcal{P}$ can update their local view of the states of the objects and trigger the NotAuthorized callback, thus enforcing the antiflickering property.

The protocol is presented in Algorithms 11, 12 and 13.

```
1  operation Update(Id, π_UpdateAuth, s, aux_Update) is
2    if Obj_i[Id].sn ≥ s.sn then return ;                    ▷ Update already registered.
3    if Obj_i[Id].sn ≠ 0 then
4      Wait until Updated(Id, ⋆, ⋆,old_s) is triggerred and old_s.sn = s.sn − 1
5    old_M, old_sn, old_r ← Obj_i[Id].M, Obj_i[Id].sn, Obj_i[Id].r;
6    pk, π_r ← aux_Update;
7    if not (pk ∈old_M and Auth.Verify(pk, π_UpdateAuth) and Auth.Verify(π_r, s.r)) then return ;
8    Propose ccons_propose_Update(Id, s.sn, (pk, π_UpdateAuth, s, π_r)) to processes in Π .

9  when ccons_decide_Update(Id, sn, (pk, π_UpdateAuth, s, π_r)) is received do
10   if Obj_i[Id].sn ≥ s.sn or sn ≠ s.sn then return ;      ▷ Update already registered.
11   if Obj_i[Id].sn ≠ 0 then
12     Wait until Updated(Id, ⋆, ⋆,old_s) is triggerred and old_s.sn = sn − 1
13   old_M, old_sn, old_r ← Obj_i[Id].M, Obj_i[Id].sn, Obj_i[Id].r;
14   if not (pk ∈old_M and Auth.Verify(pk, π_UpdateAuth) and Auth.Verify(π_r, s.r)) then
15     Obj_i[Id].sn ← Obj_i[Id].sn + 1;                    ▷ sn increment to prevent deadlock of object Id.
16     return
17   Pending_Obj_i[Id][sn] ← s;
18   σ_frag ← TAC.Sign(s, sk_{p_i});
19   be_broadcast (UpdateAns(Id, sn, (π_UpdateAuth, σ_frag))) to processes in Π .

20 when UpdateAns(Id, sn, (π_UpdateAuth, σ_frag)) is received from p_j do
21   if Obj_i[Id].sn ≠ 0 then
22     Wait until Updated(Id, ⋆, ⋆,old_s) is triggerred and old_s.sn = s.sn − 1
23   if An UpdateAns for Id and sn has already been received from p_j then return ;
24   if not TAC.Verify(σ_frag, pk_{p_j}, Pending_Obj_i[Id][sn]) then return ; ▷ The signature received is
       invalid
25   New_State_i[Id][sn] ← New_State_i[Id][sn] ∪ σ_frag;
26   if |New_State_i[Id][sn]| > t_Π and Updated has not been triggerred for Id and sn then
27     Obj_i[Id] ← Pending_Obj_i[Id][sn];
28     σ = TAC.Aggregate(NewState[Id][sn]);
29     σ_List_i[Id] ← σ;
30     Trigger Updated(Id, π_UpdateAuth, σ, Obj_i[Id]).
```

Algorithm 11: Update operation of the Anonymous Agreement Proof algorithm (code for p_i).

Variable	Purpose	Initial Value
TAC	A threshold anonymous credential scheme	
pk_{system}	The system public key of the threshold anonymous credential scheme where the threshold is setted up at $t_\Pi + 1$ out of n_Π processes	$\mathsf{pk}_{system} \leftarrow \mathsf{AggKey}(\{\ldots, \mathsf{pk}_{p_1} pk_{p_{n_\Pi}}\})$
Obj_i	A dictionnary of objects, stores the states of the objects	$Obj_i[Id].\mathcal{M}, \forall\ Id \in \mathcal{ID}$ is initiated with the initially authorized process' public key, all other elements are initialized to $\emptyset$
σ_List_i	A dictionnary of signatures, stores the last up-to-date aggregated signature of each object	$\sigma\_List_i[Id] \leftarrow \emptyset$, $\forall\ Id \in \mathcal{ID}$
$Pending\_Obj_i$	A dictionnary, stores the object that won the consensus during an Update operation for a given Id and sn	$Pending\_Obj_i[Id][sn] \leftarrow \emptyset$, $\forall\ Id \in \mathcal{ID}, \forall\ sn \in \mathbb{Z}$
New_State_i	A dictionnary, stores signature fragments of the TAC scheme before they reach the $t_\Pi + 1$ threshold	$New\_State_i[Id][sn] \leftarrow \emptyset$, $\forall\ Id \in \mathcal{ID}, \forall\ sn \in \mathbb{Z}$
Seq_Prove_i	A counter, stores the sequence number of the last $\mathsf{ccons\_decide}_{\mathsf{Prove}}$ received.	$Seq\_Prove_i \leftarrow 0$
$Values_i$	A dictionnary, stores values of a Prove operation during the synchronization of the AllowList	$Values_i[nonce] \leftarrow \emptyset$, $\forall\ nonce \in \mathbb{Z}$
$Process\_Obj_i$	A dictionnary, stores the most up-to-date version of the object Id as seen by each process in $\mathcal{P}$	$Obj\_List_i[j][Id] \leftarrow \emptyset$, $\forall\ Id \in \mathcal{ID}, \forall\ j \in \{1, \ldots, n_\Pi\}$
$\sigma\_Process\_Obj_i$	A dictionnary, stores the signature of the most up-to-date version of the object Id as seen by each process in $\mathcal{P}$	$Obj\_List_i[j][Id] \leftarrow \emptyset$, $\forall\ Id \in \mathcal{ID}, \forall\ j \in \{1, \ldots, n_\Pi\}$
$Prove\_Obj_i$	A dictionnary, stores the most up-to-date version of the object Id as seen by the local process p_i	$Prove\_Obj\_List_i[Id] \leftarrow \emptyset$ $\forall\ Id \in \mathcal{ID}$
$\sigma\_Prove\_Obj_i$	A dictionnary, stores the signature of the most up-to-date version of an object Id as seen by the local process p_i	$\sigma\_Prove\_Obj\_List_i[Id] \leftarrow \emptyset$ $\forall\ Id \in \mathcal{ID}$
$\pi_{\text{UpdateAuth}}$	A proof of authentication given as an input of the $\mathsf{Update}(Id, \star, \star, \star)$ operation computed as $Auth.\mathsf{Prove}(\mathsf{pk}, \mathsf{sk})$ where sk is the secret key associated to $\mathsf{pk} \in \mathcal{M}_{Id}^{(\star)}$	
π_{SetMem}	A ZKP of set-membership computed as $ZKPoSM.\mathsf{Prove}(s.r, \{Prove\_Obj_i[Id].r\}_{\forall Id \in \mathcal{ID}})$ where s is a state in $Prove\_Obj_i$ after view synchronization of processes in $\mathcal{P}$.	
π_r	A proof of knowledge of the secret sk_r associated to r where $\pi_r \leftarrow Auth.\mathsf{Prove}(r, \mathsf{sk}_r)$	
$\mathsf{aux}_{\mathsf{Update}}$	Auxiliary values given as parameters of update operations, contains pk the public key of the device that request the operation and π_r a proof of knowledge of the secret key associated to r	
$\mathsf{aux}_{\mathsf{Prove}}$	Auxiliary values given as parameters of update operations, not used in our algorithm	

Table 9.1 – Variables used by Algorithm 11, Algorithm 12 and Algorithm 13 to implement the Anonymous Agreement Proof protocol.

```
31 operation Prove(s, σ, nonce, π_Auth, aux_Prove) is
32   if not (TAC.Verify(σ, Commit(s), pk_system)) then return ;
33   Values_i[nonce] ← (s, σ, π_Auth);
34   be_broadcast(ProveReq(Seq_Prove_i, nonce))) to processes in P;
35 when ProveReq(Seq_Prove, σ_List, nonce) is received from p_j do
36   Wait until ccons_decide_Prove(Seq_Prove − 1, ⋆) is received;
37   Obj, σ_List ← RequestObjValue(Π) ;
38   Tmp_Obj ← Obj ;
39   Tmp_σ_List ← σ_List;
40   for Id ∈ ID do
41     if Prove_Obj_i[Id].sn > Obj[Id].sn then
42       Tmp_Obj[Id] ← Prove_Obj_i[Id];
43       Tmp_σ_List[Id] ← σ_Prove_Obj_i[Id]
44   σ ← Sign((Tmp_Obj, Seq_Prove);
45   if no ProveAns message was sent by p_i to p_j for Seq_Prove then
46     Send ProveAns(Seq_Prove, Tmp_Obj, σ, sk_{p_i}), σ_List, nonce) to p_j.
47 when ProveAns(Seq_Prove, Obj, σ_Obj, σ_List, nonce) is received from p_j do
48   if A ProveReq message from p_j was already received for SeqProve then return ;
49   if Values_i[nonce] ≠ ∅ then                          ▷p_i initiated the operation.
50     if not (∀Id ∈ ID, TAC.Verify(σ_List[Id], Obj[Id], pk_system) and
         VerifySig(σ_Obj, (Obj, SeqProve), p_j)) then return ;
51     σ_Process_Obj_i[Seq_Prove][j] ← σ_Obj;
52     Process_Obj_i[Seq_Prove][j] ← Obj;
53     for Id ∈ ID do
54       if Obj[Id].sn > Prove_Obj_i[ID].sn then
55         σ_Prove_Obj_i[Id] ← σ_List[Id];
56         Prove_Obj_i[Id] ← Obj[Id];
57     if ProveReq(Seq_Prove, ⋆, ⋆, ⋆, nonce) messages were received from at least than n_P − t_P
         processes then
58       s, σ, π_Auth ← Values_i[nonce];
59       if s.r ∉ {Prove_Obj_i[Id].r}_∀Id∈ID then          ▷s is outdated.
60         ccons_propose_Prove(Seq_Prove, (Prove_Obj_i, σ_Prove_Obj_i,
             Process_Obj_i[Seq_Prove], σ_Process_Obj_i[Seq_Prove], ∅, ∅, ∅, ∅))
61       else
62         σ′ ← TAC.Randomize(σ);
63         π_SetMem ← ZKPoSM.Prove(s.r, {Prove_Obj_i[Id].r}_∀Id∈ID);
64         C_s ← Commit(s);
65         ccons_propose_Prove(Seq_Prove, (Prove_Obj_i, σ_Prove_Obj_i,
             Process_Obj_i[Seq_Prove], σ_Process_Obj_i[Seq_Prove], C_s, σ′, π_Auth, π_SetMem)).
```

Algorithm 12: Prove operation of the Anonymous Agreement Proof algorithm (code for ρ_i) (Part I).

when ccons_decide$_{\mathsf{Prove}}$(Seq_Prove, ($nonce$, $Prove\_Obj$, σ_Prove_Obj, $Process\_Obj$, $\sigma\_Process\_Obj$, C_s, σ, π_{Auth}, π_{SetMem})) **is** received **do**
  if $Seq\_Prove \neq 0$ **then** Wait until ccons_decide($Seq\_Prove - 1, \star$) is received;
  $Seq\_Prove_i \leftarrow Seq\_Prove + 1$;
  if $|\sigma\_Process\_Obj| < n_{\mathcal{P}} - t_{\mathcal{P}}$ **then** return ;
  if not TAC.Verify($\sigma\_Prove\_Obj[Id], Prove\_Obj[Id], \mathsf{pk}_{system}$), $\forall\ Id \in \mathcal{ID}$ **then**
    return
  if $\forall\ k \in \mathcal{P}$, not VerifySig($\sigma\_Process\_Obj[k], (Process\_Obj[k], Seq\_Prove), \mathsf{pk}_{p_k}$), **then** return ;
  if not ($\forall Id\ \in \mathcal{ID}, Prove\_Obj[Id] \in \{Process\_Obj[k][Id]\}_{\forall k \in \mathcal{P}}$) **then** return ;
  if $\forall\ k \in V, \forall\ Id \in \mathcal{ID}$ at least one sequence number $Process\_Obj[k][Id].sn$ is greater than $Prove\_Obj[Id].sn$ **then** return ; ▷ *The invoking process is Byzantine and tries to break progress*
  if $C_s = \emptyset$ or $\sigma = \emptyset$ or $\pi_{\text{Auth}} = \emptyset$ or $\pi_{\text{SetMem}} = \emptyset$ **then**
    $Prove\_Obj_i \leftarrow Prove\_Obj$; ▷ *Updates the list of up-to-date states.*
    $\sigma\_Prove\_Obj_i \leftarrow \sigma\_Prove\_Obj$; ▷ *Updates the list of signatures of up-to-date states.*
    Trigger NotAuthorized($nonce$).
  else
    if not (TAC.Verify($\sigma, C_s, \mathsf{pk}_{system}$) **then** return ;
    if not ($ZKPoSM$.Verify($\pi_{\text{SetMem}}, C_s, \{Prove\_Obj[Id].r\}_{\forall\ Id \in \mathcal{ID}}$) and $Auth$.Verify(π_{auth}, C_s)) **then** return ; ▷ *Verifies that C_s is a commitment to one of the s values $Prove\_Obj$*
    $Prove\_Obj_i \leftarrow Prove\_Obj$; ▷ *Updates the list of up-to-date states.*
    $\sigma\_Prove\_Obj_i \leftarrow \sigma\_Prove\_Obj$; ▷ *Updates the list of of up-to-date signatures.*
    Trigger Authorized($C_s, nonce$).

Algorithm 13: Prove operation of the Anonymous Agreement Proof algorithm (code for ρ_i) (Part II).

9.6.4 Proof of the AAP algorithm

This section proves that Algorithms 11, 12, and 13 implement the AAP abstraction. The proof uses the assumptions that $n_\pi \geq 3t_\pi + 1$ and $n_\mathcal{P} \geq 3t_\mathcal{P} + 1$. Those assumptions are mandatory for the Cascading Consensus algorithms to work. In the following, we write $var^{(\tau,p)}$ the state of a variable var at time τ as seen by the process p.

We begin by proving the Update termination property in a more restrictive form. We guarantee the property if the operation is called with the right parameters, *i.e.*, the invoking process provides a valid proof of knowledge of the secret associated with $s.r$ and the sequence number $s.sn$ of the new state is a successor of the sequence number of the actual state of the object. The difference between the abstraction and the implementation is due to implementation details, where extra verifications are conducted to fulfill the other properties, whereas that verification cannot be added to the abstraction as they consist of implementation-specific details.

Lemma 9.1. Algorithms 11, 12 and 13 fulfill the Update termination property, *i.e.*, if a correct process $p \in \Pi$ invokes $\mathsf{Update}(Id, \pi_{\text{UpdateAuth}}, s, \mathsf{aux})$ with $Id \in \mathcal{ID}$, $s \in \mathcal{S}$, $\pi_{\text{UpdateAuth}}$ is a correct context-aware proof that p is one of the processes in $\mathcal{M}_{Id}^{(s')}$ where s' is the previous state of the object identified by Id, $\mathsf{aux} = (\mathsf{pk}, \pi_r)$, where π_r is a correct proof of knowledge of the secret associated to $s.r$, and an $\mathsf{Updated}(Id, \star, \star, old\_s)$ with $old\_s.sn = s.sn - 1$ is eventually triggerred at p, then a callback $\mathsf{Updated}(Id, \star, \star, s')$ is eventually trigerred at p, where $s'.sn = s.sn$.

Proof. We prove Lemma 9.1 by contradiction. Let us assume a correct process $p \in \Pi$ invokes $\mathsf{Update}(Id, \pi_{\text{UpdateAuth}}, s, \mathsf{aux})$ such that $Id \in \mathcal{ID}$, $s \in \mathcal{S}$, $\pi_{\text{UpdateAuth}}$ is a correct context-aware proof that p is one of the processes in $\mathcal{M}_{Id}^{(s)}$ where s' is the previous state of the object identified by Id, $\mathsf{aux} = (\mathsf{pk}, \pi_r)$, where π_r is a correct proof of knowledge of the secret associated to $s.r$ and an $\mathsf{Updated}(Id, \star, \star, old\_s)$ with $old\_s.sn = s.sn - 1$ is eventually triggerred at p. However, no $\mathsf{Updated}(Id, \star, \star, s')$ is trigerred at p during the execution of the algorithm with $s'.sn = s.sn$.

By assumption, verifications lines 3 and 7 pass. The only verification that may not pass during the invocation is conducted at line 2. However, if this verification does not pass, and because Obj_i is only updated at line 27, it implies that the Updated callback has been triggered at p at line 30. Therefore, if the condition is not fulfilled, the $\mathsf{Updated}(Id, \star, \star, s')$ with $s'.sn = s.sn$ has already been triggered. On the other hand, if the condition line 2 passes, then p will eventually invoke the ccons_propose operation at line 8.

Thanks to the termination and agreement properties of the Cascading Consensus algorithm, we know that all the correct processes in Π will eventually receive the $\mathsf{ccons\_decide}(Id, sn, \star)$ callback. Because of the assumption that the $\mathsf{ccons\_decide}_{\mathsf{Update}}$ callback can only be triggered if conditions at lines 11 and 14 are verified, we know that those verifications will pass. Furthermore, as in the previous paragraph, if the condition line 10 does not pass, the termination property is

verified. Therefore, all the correct processes will eventually broadcast an UpdateAns$(Id, sn, \star)$ message at line 19. Therefore, p will eventually receive an UpdateAns$(Id, sn, \star)$ from each correct process. Using the assumption $n_\pi \geq 3t_\pi + 1$, eventually p receives σ_{frag} from strictly more than t_Π processes. Therefore, the condition at line 26 eventually passes. Hence, an $\mathsf{Updated}(Id, \pi_{\text{UpdateAuth}}, \sigma, s')$ callback is eventually triggerred and $s'.sn = s.sn$. Thus contradicting the hypothesis and proving Lemma 9.1. □

Lemma 9.2. Algorithms 11, 12 and 13 fulfill the Update validity property, *i.e.*, if the callback $\mathsf{Updated}(Id, \pi_{\text{UpdateAuth}}, \star, s)$ is triggered at a correct process $p \in \Pi$, then $\pi_{\text{UpdateAuth}}$ is a valid context-aware proof of authorization that the process that built the proof is in $\mathcal{M}_{Id}^{(s')}$, where s' is the previous state of the object identified by Id.

Proof. Let $p \in \Pi$ be a correct process. We prove Lemma 9.2 by contraction, *i.e.*, assuming that the callback $\mathsf{Updated}(Id, \pi_{\text{UpdateAuth}}, \star, s)$ is triggered at a correct process $p \in \Pi$ and that $\pi_{\text{UpdateAuth}}$ is not a valid context-aware proof of authorization that the process that built the proof is in $\mathcal{M}_{Id}^{(s')}$, where s' is the previous state of the object identified by Id.

The $\mathsf{Updated}(Id, \pi_{\text{UpdateAuth}}, \star, s)$ can only be trigerred at line 30 of Algorithm 12. If this line is triggered, it means that p received UpdateAns messages from $t_\Pi + 1$ different processes (lines 23 and 26). The previous implication implies that at least one correct process p' did broadcast an UpdateAns message. p' can only broadcast this message at line 19. To reach this line, the condition at line 14 must be fulfilled at p'. Thus the process that invoked the operation provided a proof $\pi_{\text{UpdateAuth}}$ such that $Auth.\mathsf{Verify}(\pi_{UpdateAuth}, \text{old\_}\mathcal{M})$ outputs 1. By construction of the authentication protocol, this means that the invoking process proved it knows a secret key associated with one of the public keys listed in $\mathcal{M}_{Id}^{(s')}$, where s' is the previous state of the object identified by Id. Thus contradicting the hypothesis and proving the Update validity property. □

Lemma 9.3. Algorithms 11, 12 and 13 fulfill the Update agreement property, *i.e.*, if the callback $\mathsf{Updated}(Id, \pi_{\text{UpdateAuth}}, \sigma, s)$ is trigerred at a correct process $p \in \Pi$, then it is eventually trigerred at each the correct processes $p' \in \Pi$.

Proof. We prove Lemma 9.3 by contradiction. Let us assume the callback $\mathsf{Updated}(Id, \pi_{\text{UpdateAuth}}, \sigma, s)$ is trigerred at a correct process $p \in \Pi$. However, the callback is never triggered at a second correct process $p' \in \Pi$.

If the callback $\mathsf{Updated}(Id, \pi_{\text{UpdateAuth}}, \sigma, s)$ is triggered at a correct process $p \in \Pi$, it implies that p received UpdateAns messages from $t_\Pi + 1$ different processes (lines 23 and 26). The previous implication implies that at least one correct process p'' did broadcast an UpdateAns message. p'' can only broadcast this message at line 19. To reach this line, p'' must have received a $\mathsf{ccons\_decide}_{\mathsf{Update}}(Id, sn, (\pi_{\text{UpdateAuth}}, s, \star))$, where $s.sn = sn$. Thanks to the termination and agreement properties of the Cascading Consensus,

we know that $\mathsf{ccons\_decide}_{\mathsf{Update}}(Id, sn, (\pi_{\text{UpdateAuth}}, s, \star))$ will be triggered at all the correct processes. Because conditions at lines 10, 11 and 14 passed at p'', they will pass at all the other correct processes. Hence, each correct process will best-effort broadcast an UpdateAns$(Id, sn, (\pi_{\text{UpdateAuth}}, \sigma_{frag}))$ message. Those messages are sent by correct processes. Therefore, they are eventually received by p'. Furthermore, because the senders are correct, conditions lines 23 and 24 will pass at p'. Additionally, there are $n_\Pi - t_\Pi \geq 2t_\pi + 1 \geq t_\pi + 1$ correct processes, hence, condition at line 26 eventually passes at p'. Hence, p' eventually reaches line 30. Therefore, $\mathsf{Updated}(Id, \pi_{\text{UpdateAuth}}, \sigma, s)$ is eventually trigerred at p', thus contradicting the hypothesis and proving Lemma 9.3. □

Lemma 9.4. Algorithms 11, 12 and 13 fulfill the state agreement property, *i.e.*, for an object Id, Updated callbacks are triggered in the same order at all correct processes $p \in \Pi$.

Proof. Lemma 9.4 is trivially verified by construction. The conditions at line 11 and 21 ensure that the callback $\mathsf{Updated}(Id, \star, \star, s)$ cannot be triggerred at a correct process $p \in \Pi$ if the callback $\mathsf{Updated}(Id, \star, \star, old\_s)$ has not been triggerred, with $s.sn - 1 = old\_s.sn$. Furthermore, thanks to the agreement property of the Cascading Consensus, processes only receive one state for each pair of object's identifier Id and object's sequence number sn. □

Lemma 9.5. Algorithms 11, 12 and 13 fulfill the Prove validity property, *i.e.*, if the callback $\mathsf{Authorized}(C_s, \star)$ is trigerred at a correct process $p \in \mathcal{P}$, where C_s is a commitment to a state $s \in \mathcal{S}$, then the callback $\mathsf{Updated}(\star, \star, \star, s)$ was trigerred at at least one correct process $p \in \Pi$.

Proof. We prove Lemma 9.5 by contradiction. We assume the callback $\mathsf{Authorized}(C_s, \star)$ is triggered at a correct process $p \in \mathcal{P}$, where C_s is a commitment to a state $s \in \mathcal{S}$, and no callback $\mathsf{Updated}(\star, \star, \star, s)$ is triggered at any correct process $p' \in \Pi$ during the whole execution.

If the callback $\mathsf{Authorized}(C_s, \star)$ is triggerred at p, it is at line 84. Therefore, the condition at line 80 passed at p. This condition verifies that $TAC.\mathsf{Verify}(\sigma, C_s, \mathsf{pk}_{system}) = 1$. For this statement to be true, and by definition of the TAC scheme, it implies that $t_\Pi + 1$ different processes in Π signed the state s, where C_s is a commitment to s. Because there are at most t_Π Byzantine processes in Π, it implies that at least one correct process $p_c \in \Pi$ signed the state s. The only line where p_c can sign this state is at line 18. This line can be reach by p_c only if it received a $\mathsf{ccons\_decide}_{\mathsf{Update}}(\star, \star, \star, s, \star))$ message. Using the termination and agreement properties of the Cascading Consensus, we know that $\mathsf{ccons\_decide}_{\mathsf{Update}}(Id, sn, \star, s, \star))$ will be triggered at all the correct processes in Π, for Id some object identifier and sn a sequence number. Because conditions at lines 10, 11 and 14 passed at p_c, they will pass at all the other correct processes. Hence, each correct process best-effort broadcasts an UpdateAns$(Id,\ sn, \star, \sigma_{frag}))$ message where σ_{frag} is a TAC signature of the state s. Hence, p_c eventually receives $n - t_\pi \geq 2t_\Pi + 1 \geq t_\Pi + 1$ UpdateAns$(Id,\ sn, \star, \star)$ messages from correct processes. Therefore, conditions

lines 23, 24 and 26 eventually pass at p_c. Therefore, the $\mathsf{Updated}(\star, \star, \star, s)$ callback is eventually triggerred at p_c at line 30. Thus contradicting the hypothesis and proving Lemma 9.5 □

Lemma 9.6. Algorithms 11, 12 and 13 fulfill the Prove agreement property, *i.e.*, if the callback $\mathsf{Authorized}(C_s, nonce)$ is triggered at a correct process $p \in \mathcal{P}$, then it is eventually triggered at all the correct processes $p' \in \mathcal{P}$.

Proof. Let us prove Lemma 9.6 by contradiction. We assume there exists a process $p' \in \mathcal{P}$ such that the callback $\mathsf{Authorized}(C_s, nonce)$ is triggered at a correct process $p \in \mathcal{P}$ and not at p'.

If the callback $\mathsf{Authorized}(C_s, \star)$ is triggerred at p, it is at line 84. It implies that p received a $\mathsf{ccons\_decide_{Prove}}(Seq\_Prove, (Prove\_Obj, \sigma\_Prove\_Obj, Obj\_List,\ \sigma\_Obj\_List, C_s, \sigma, \pi_{\text{Auth}}, \pi_{\text{SetMem}}))$ message. Using the termination and agreement properties of the Cascading Consensus, we know that p' eventually receives the same $\mathsf{ccons\_decide_{Prove}}$. The condition at line 67 is trivially verified if $Seq\_Prove = 0$. In any other cases, because the condition passed at p, it means p received a $\mathsf{ccons\_decide_{Prove}}(Seq\_Prove, \star)$ message. Using the termination and agreement properties of the Cascading Consensus, we know that p' eventually receives this message and that the condition at line 67 eventually passes at p'. Conditions from line 69 to 74 ans conditions lines 80 and 81 also passed at p. Therefore, they will also pass at p'. Furthermore, p entered the Else condition at line 79; thus, p' will also enter this condition. Hence, p' eventually reaches line 84. Thus, the Authorized callback is eventually triggered at p', contradicting the hypothesis and proving Lemma 9.6. □

Lemma 9.7. Let $\mathsf{Updated}(Id, \star, \star, s)$ and $\mathsf{Updated}(Id, \star, \star, s')$ be two callbacks triggerred at a correct process $p' \in \Pi$. Then $s \neq s'$.

Proof. This lemma is verified by the construction of Algorithm 11. Indeed, thanks to the use of the Cascading Consensus (and thanks to the agreement property of this abstraction), only one state can be accepted for a specific identifier Id and sequence number sn at line 30. Furthermore, the sequence number of a state is embedded in this state. Thus, two different states of a given object Id are necessarily different. □

Lemma 9.8. Let two consecutive Authorized callbacks (or an Authorized and a NotAuthorized callback) be triggered at a correct process $p_1 \in \mathcal{P}$. The first callback is triggered at time τ of the local clock of p_1, and the second is triggered at time τ' of the local clock of p_1 with $\tau < \tau'$. Let $Prove\_Obj_1^{(\tau, p_1)}$ be the state of the $Prove\_Obj_1$ variable at line 84 at time τ, and let $\sigma\_Prove\_Obj_1^{(\tau, p_1)}$ be the state of $\sigma\_Prove\_Obj_1$ at line 84 at time τ. Then, $\forall Id \in \mathcal{ID}, Prove\_Obj_1^{(\tau', p_1)}[Id].sn \geq Prove\_Obj_1^{(\tau, p_1)}[Id].sn$.

Proof. The condition at line 73 ensures that the states in $Prove\_Obj_1^{(\tau, p_1)}$ are chosen in $Process\_Obj^{(\tau, p_1)}$, and that the states in $Prove\_Obj_1^{(\tau', p_1)}$ are chosen in $Process\_Obj^{(\tau', p_1)}$.

Because of condition line 69, we know that $Process\_Obj^{(\tau',p_1)}$ consists of at least $n_{\mathcal{P}} - t_{\mathcal{P}}$ different sets sent by $n_{\mathcal{P}} - t_{\mathcal{P}}$ different processes. Thus, at least $n_{\mathcal{P}} - 2t_{\mathcal{P}} \geq t_{\mathcal{P}} + 1$ of those sets have been sent by correct processes. Let p_c be one of those correct processes that participated in the construction of $Process\_Obj^{(\tau',p)}$. If it did so, it shared its local view of the Tmp_Obj variable at line 46. The Tmp_Obj variable is built from lines 38 to 43. Those lines build Tmp_Obj as the most up-to-date version of each object between the Obj and $Prove\_Obj_c$ variables. The $Prove\_Obj_c$ can only be modified at line 76, 82 or 56. Thanks to the agreement property of the Cascading Consensus abstraction, we know that $Prove\_Obj_c^{(\tau,p_c)} = Prove\_Obj_1^{(\tau,p_1)}$.[5] Hence, the $Process\_Obj(\tau', p_1)$ variable contains p_c's Tmp_Obj variable. p_c's Tmp_Obj either contains the same states as the one used at time τ or states with bigger sequence numbers. As stated earlier, $Process\_Obj(\tau', p_1)$ consists in at least $t_{\mathcal{P}} + 1$ of those states. The verifications at line 74 ensures that $Prove\_Obj_1^{(\tau',p_1)}$ consists in the most up to date states in $Process\_Obj(\tau', p_1)$. Therefore, and using the previous argument, sequence numbers of the objects in $Prove\_Obj_1^{(\tau',p_1)}$ are greater or equal to the sequence number of the objects in $Prove\_Obj_1^{(\tau,p_1)}$. □

Lemma 9.9. Algorithms 11, 12 and 13 fulfill the local Append/Remove antiflickering property, *i.e.*, if an Authorized$(C_s, nonce)$ callback is triggered at a correct process $p \in \mathcal{P}$ at time τ_1 of its local clock and then, the invocation of Prove$(s, \star, nonce, \star, \star)$ by p triggers the NotAuthorized$(nonce)$ callback at time $\tau_2 > \tau_1$ of the local clock of p, then no Authorized$(C'_s, \star)$ can be trigerred at p after τ_2, where C_s and C'_s are commitments to the state s.

Proof. We prove Lemma 9.9 by contradiction. We assume that an Authorized$(C_s, nonce)$ callback is triggered at a correct process $p \in \mathcal{P}$ at time τ_1 of its local clock and then, the invocation of Prove$(s, \star, nonce, \star, \star)$ by p triggers the NotAuthorized$(nonce)$ callback at time $\tau_2 > \tau_1$ of the local clock of p. However, an Authorized$(C'_s, \star)$ callback is triggerred at p at time $\tau_3 > \tau_2$, where C_s and C'_s are commitments to the state s.

Let an Authorized$(C_s, \star)$ callback be triggerred at p at time τ_1. Thanks to the verification at line 81, we know that, if C_s is a commitment to s, then $s \in Prove\_Obj^{(\tau_1,p)}$. Second, let the NotAuthorized$(nonce)$ callback be triggerred at time τ_2 at p after it invokes the Prove$(s, \star, nonce, \star, \star)$ operation. The only way this callback can be triggered at p is if it sets $C_s, \sigma', \pi_{\text{Auth}}$ and π_{SetMem} at $\emptyset$ at line 60. This line is reached if $s.r \notin \{Prove\_Obj_i[Id].r\}_{\forall Id \in \mathcal{ID}}$ (line 59). In other words, $s \notin Prove\_Obj^{(\tau_2,p)}$. Therefore, using Lemma 9.8 and Lemma 9.7, we know that $\forall \tau_3 > \tau_2, s \notin Prove\_Obj^{(\tau_3,p)}$. Therefore, after time τ_2, the condition line 81 cannot be verified at p for a commitment C'_s to the state s. Therefore, no Authorized$(C'_s, \star)$ callback is triggerred at p at time $\tau_3 > \tau_2$. Thus contradicting the hypothesis and proving Lemma 9.9. □

5. This is a simplification as there is no common time reference between p_1 and p_c. We should have written that, when reaching line 76 or 82 after receiving the same ccons_decide$(Seq_Prove, \star)$ message, the variable $Prove\_Obj_c$ as seen by p_c is the same as the variable $Prove\_Obj_1$ as seen by p_1.

Lemma 9.10. Algorithms 11, 12 and 13 fulfill the local progress property, *i.e.*, let an $\mathsf{Updated}(Id, \star, \star, s)$ callback be triggered at a correct process $p \in \Pi$ at time τ_1 of its local clock, and an $\mathsf{Updated}(Id, \star, \star, s')$ callback be triggered at p at time $\tau_2 > \tau_1$ of its local clock. No other $\mathsf{Updated}(Id, \star, \star, \star)$ operation is triggered at p during the algorithm's execution. Let $p_1 \in \mathcal{P}$. Then, eventually no $\mathsf{Authorized}(C_s, \star)$ callback can be triggerred at p_1 where C_s is a commitment to s.

Furthermore let $p_2 \in \mathcal{P}$ be a correct process such that $p_2 \notin \Psi_{Id}^{s}$ and $p_2 \in \Psi_{Id}^{s'}$. If other processes in $\mathcal{P}$ are correct and do not invoke Prove operations, then, eventually, a $\mathsf{Prove}(s', \sigma, nonce, \pi_{\text{Auth}}, \star)$ operation invoked by p_2 triggers an $\mathsf{Authorized}(C_{s'}, nonce)$ callback where $C_{s'}$ is a commitment to s' if π_{Auth} is a correct context-aware proof that $p_2 \in \Psi_{Id}^{(s')}$ and σ is a valid proof that the $\mathsf{Updated}(\star, \star, \star, s')$ callback was triggered at a correct process in Π.

Proof. We prove the two parts of Lemma 9.10 by contradiction. First, we focus on the first part of the lemma. We prove that the following statement leads to a contradiction. Let an $\mathsf{Updated}(Id, \star, \star, s)$ callback be triggered at a correct process $p \in \Pi$ at time τ_1 of its local clock, and an $\mathsf{Updated}(Id, \star, \star, s')$ callback be triggered at p at time $\tau_2 > \tau_1$ of its local clock. No other $\mathsf{Updated}(Id, \star, \star, \star)$ operation is triggered at p during the algorithm's execution. Let $p_1 \in \mathcal{P}$ be a process such that $p_1 \in \Psi_{Id}^{s}$ and $p_1 \notin \Psi_{Id}^{s'}$. Let τ_3 be a time at the local clock of p_1, τ_3 can be chosen as large as we want. Then, the $\mathsf{Authorized}(C_s, \star)$ callback is triggerred at p_1 at time τ_2.

The $\mathsf{Authorized}(C_s, \star)$ callback is triggerred at p_1 at time τ_3. This callback can only be triggered at line 84. At line 81, it is verified that C_s is a commitment to a state s, such that $s \in Prove\_Obj_1^{(\tau_3, p_1)}$. Using the condition at line 73, we know that $Prove\_Obj_1^{(\tau_3, p_1)}$ is built from elements in the sets $Process\_Obj[k]^{(\tau_3, p_1)}, \forall k \in \mathcal{P}$. Furthermore, the condition at line 74 verifies that each object identified by $Id \in \mathcal{ID}$ in $Prove\_Obj^{(\tau_3, p_1)}$ is chosen as the most up-to-date (in term of sequence number) element in $\{Process\_Obj[k][Id]^{(\tau_3, p_1)}\}_{\forall k \in \mathcal{P}}$. Moreover, using condition line 69, we know that $Process\_Obj^{(\tau 3, p_1)}$ consists of at least $n_{\mathcal{P}} - t_{\mathcal{P}} \geq 2t_{\mathcal{P}} + 1$ sets. For all $k \in \mathcal{P}$, the set $Process\_Obj[k]$ along with the Seq_Prove variable is signed by the process k (line 72). Out of the $2t_{\mathcal{P}} + 1$ processes that signed sets in $Process\_Obj[k]^{(\tau 3, p_1)}$, at least $t_{\mathcal{P}} + 1$ are correct.

Let p_c be a correct process that signed a set in $Process\_Obj[k]^{(\tau 3, p_1)}$. This process signed this set at line 46. It signed at its local time τ_c the set $Tmp\_Obj^{(\tau_c, p_c)}$. Using lines 38 to 43, we see that for each $Id \in \mathcal{ID}$, $Tmp\_Obj[Id]^{(\tau_c, p_c)}$ is the most recent state of the object Id between the variable $Prove\_Obj_i[Id]^{(\tau_c, p_c)}$ and the variable $Obj^{(\tau_c, p_c)}$. The variable $Obj^{(\tau_c, p_c)}$ gets its value from the $\mathsf{RequestObjValue}(\Pi)$ operation. By construction, this operation returns the most up-to-date version of the state of each object $Id, \forall Id \in \mathcal{ID}$ (the Obj_i variable) of some correct process $p_\Pi \in \Pi$. Using the Update agreement property, we know that the $\mathsf{Updated}(Id, \star, \star, s')$ callback is eventually triggerred at p_Π. Let us note τ_Π the time this callback is triggered at p_Π at its local clock.

τ_c is causaly before τ_3. Furthermore, τ_3 can be chosen as large as we want. Therefore, τ_c can also be chosen as large as we want. Hence, we choose τ_c to be causally after τ_Π. Therefore, and because no other $\mathsf{Updated}(Id, \star, \star, \star)$ callback is triggerred at p_Π, $Tmp\_Obj[Id]^{(\tau_c, p_c)} = s'$. Hence, $s \notin Tmp\_Obj[Id]^{(\tau_c, p_c)}$, $s \notin Prove\_Obj_1^{(\tau_3, p_1)}$. Therefore, the condition at line 81 cannot pass for C_s a commitment to s. Therefore, the $\mathsf{Authorized}(C_s, \star)$ callback cannot be triggerred at p_1 at time τ_3. Thus contradiction the hypothesis and proving the first part of Lemma 9.10.

To prove the second part of Lemma 9.10, we prove that the following statement leads to a contradiction. Let an $\mathsf{Updated}(Id, \star, \star, s)$ callback be triggered at a correct process $p \in \Pi$ at time τ_1 of its local clock, and an $\mathsf{Updated}(Id, \star, \star, s')$ callback be triggered at p at time $\tau_2 > \tau_1$ of its local clock. No other $\mathsf{Updated}(Id, \star, \star, \star)$ operation is triggered at p during the algorithm's execution. Let $p_2 \in \mathcal{P}$ be a correct process such that $p_2 \notin \Psi_{Id}^{s}$ and $p_2 \in \Psi_{Id}^{s'}$. Other processes in $\mathcal{P}$ are correct and do not invoke Prove operations. However, no $\mathsf{Prove}(s', \star, \star, \star, \star)$ operation invoked by p_2 triggers an $\mathsf{Authorized}(C_{s'}, nonce)$ callback.

Let p_2 invoke the $\mathsf{Prove}(s', \sigma, nonce, \pi_{\text{Auth}}, \star)$ operation at time τ_4 of the local clock of p_2. σ is a valid TAC signature of the state s' and π_{auth} is a valid proof that p_2 knows a secret in $\Psi_{Id}^{(s')}$. τ_4 can be chosen as big as we want. By assumption, condition at line 32 passes. Hence, p_2 broadcasts a ProveReq$(Seq_Prove_i^{(\tau_4, p_2)}, nonce)$ message at line 34.

All the correct processes in $\mathcal{P}$ receive this message. Let us consider $p_c \in \mathcal{P}$ as one of those correct processes that receive the ProveReq message. Because p_2 is correct, Seq_Prove_i was updated at line 68 after receiving the $\mathsf{ccons\_decide}_{\mathsf{Prove}}(Seq\_Prove, \star)$ callback. This callback is received at all the correct processes (agreement property of the Cascading Consensus abstraction). Hence, eventually, $Seq\_Prove_c \geq Seq\_Prove_2^{(\tau_4, p_2)}$. Hence, the condition at line 36 eventually passes at p_c. Then, p_c reaches line 37 at time τ_c. τ_4 can be chosen as big as we want; hence, τ_c, which is causally after τ_4, can also be chosen as large as required. We choose τ_c to be causally after that all the correct processes that are contacted by p_c during the $\mathsf{RequestObjValue}(\Pi)$ operation receive the $\mathsf{Updated}(Id, \star, \star, s')$ callback (this eventually happens due to the Update agreement property). No other $\mathsf{Updated}(Id, \star, \star, \star)$ callback is triggered at those processes. Therefore, and by construction of the $\mathsf{RequestObjValue}$ operation, the $Obj^{(\tau_c, p_c)}$ contains s'. Lines 38 to 43 do not modify the s' value in $Obj^{(\tau_c, p_c)}$ as no more recent value for s' may exist. Therefore, at line 46, p_c sends ProveAns$(SeqProve, Tmp_Obj, \star, \star, nonce)$ to p_2 with $s' \in Tmp\_Obj$. We choose τ_4 large enough such that all other correct processes in $\mathcal{P}$ do the same.

No $\mathsf{Updated}(Id, \star, \star, \star)$ is triggerred at a correct process in Π after $\mathsf{Updated}(Id, \star, \star, s')$ has been triggerred. Hence, no state exists that is more recent than s' for the object Id. More precisely, there exists no such state that can pass the condition at line 50. Furthermore, p_2 waits messages from at least $n_\mathcal{P} - 2t_\mathcal{P} \geq 1$ correct processes before it reaches line 65. Those processes send ProveAns$(SeqProve, Tmp_Obj, \star, \star, nonce)$ messages with $s' \in Tmp\_Obj$. There-

fore, the lines 51 to 56 add s' to $Prove\_Obj_2$ but do not remove it. Hence, when reaching line 65 the p_2's $Prove\_Obj_2$ variable contains s'. We know that p_2 is the only process to invoke the Prove operation, and there are no Byzantine processes in the system. Hence, when p_2 invokes the $\mathsf{ccons\_propose}_{\mathsf{Prove}}(Seq\_Prove, (Prove\_Obj_2, \star, \star, \star, C_{s'}, \star, \star, \star))$ operation, the $\mathsf{ccons\_decide}_{\mathsf{Prove}}(Seq\_Prove, (Prove\_Obj_2, \star, \star, \star, C_{s'}, \star, \star, \star))$ is eventually triggered at all the correct processes in $\mathcal{P}$ (using validity agreement and termination properties of the Cascading Consensus abstraction) with $s' \in Prove\_Obj_2$.

Because p_2 is correct, verifications line 67 to 74 pass. Furthermore, $s' \in Prove\_Obj_2$, therefore, $C_s \neq \emptyset$ and $\sigma \neq \emptyset$ and $\pi_{\text{Auth}} \neq \emptyset$ and $\pi_{\text{SetMem}} \neq \emptyset$. Thus, conditions line 80 and 81 will be successfully verified. Therefore, the $\mathsf{Authorized}(C_{s'}, nonce)$ callback is eventually triggered at each correct process. Hence contradicting the hypothesis and proving the second part of Lemma 9.10 □

Lemma 9.11. Algorithms 11, 12 and 13 fulfill the anonymity property, *i.e.*, let p be a process controlled by a PPT adversary $\mathcal{A}$. $\mathcal{A}$ also controls Byzantine processes in $\mathcal{P}$ and Π. When the callback $\mathsf{Authorized}(C_s, nonce)$ is triggered at p, where C_s is the state of an object identified by Id, then the probability for p to output Id or s is negligible.

Proof. Lemma 9.11 is proved by analyzing the elements given to processes during a Prove operation. The only elements shared with the other processes (and therefore with the adversary) are C_s a commitment to the state s, σ a TAC signature of s by $t_\Pi + 1$ processes in Π, π_{Auth} and π_{SetMem} two ZKPs. By construction, none of those elements reveal any information that can lead to the identification of s or Id. Therefore Lemma 9.11 is proven. □

Theorem 9.1. Algorithms 11, 12 and 13 implement the AAP abstraction

Proof. Theorem 9.1 is proven by Lemma 9.2, Lemma 9.1, Lemma 9.3, Lemma 9.4, Lemma 9.5, Lemma 9.6, Lemma 9.9, Lemma 9.10 and Lemma 9.11. □

9.7 AAP to enable the multi-device authorization feature for PPfDIMSs

The AAP abstraction makes it possible to implement the multi-device authorization capability for a PPfDIMS efficiently. As stated in Section 9.2 and Section 9.3, a PPfDIMS which supports the multi-device authorization capability must implement the Issue, Present and Verify operations along with the Authorize and the Revoke operations. This section assumes we have access to an AAP implementation AAP. The set of processes Π of the AAP implementation is the same as the set Π of the PPfDIMS. It is a set of always-connected processes (except for the

Byzantine ones) that maintain a distributed ledger. Devices in $\mathcal{D}$ can request these processes to modify the ledger if they are authorized.

On the other hand, the set $\mathcal{P}$ of the AAP abstraction cannot be used as it is for the PPfDIMS. This section will use the modification explained at the end of Section 9.5. The PPfDIMS consists of multiple ad hoc $\mathcal{P}$ sets. Each set corresponds to a relationship between a user and a verifier. Additionally, if a user wants to increase its privacy, it can create a new ad hoc set $\mathcal{P}$ for each new connection with the same verifier. In this case, the user must hide their IP address for each connection. To do so, he can use the onion routing protocol, for example. As stated in Section 9.5, the Update operations conducted by the processes in Π impacts all the sets $\mathcal{P}$. We can formalize this by saying that the PPfDIMS uses multiple AAP objects; each AAP object shares the same set Π but is associated with a specific set $\mathcal{P}$. Update operations atomically modify all the AAT objects, whereas Prove operations are specific to each set $\mathcal{P}$. Therefore, processes in Π only store once the state of each object Id. Furthermore, they do not need to act during a Prove operation. They only need to answer the RequestObjValue requests. Hence, they do not need to store information for any $\mathcal{P}$ set.

We build our authorization mechanism on DIDs and DID documents. Each user owns a DID document, which is identified by a DID. Each DID document is associated initially with a manager's public key, which the user owns. In Chapter 10, we modify this claim such that no DID document exists at the initialization, and users create their document using naming algorithms (c.f. Chapter 7).

Each object in the system is a DID document, and the identifier $Id \in \mathcal{ID}$ of the DID document is under the form of a DID. The DID document data model is equivalent to the AAP implementation data model, *i.e.*, DID documents must contain the following fields:

— A managers field: this field is a set of managers' public keys $\mathcal{M}_{Id}$, for each DID document Id this field is initialliazed with one public key, it defines the processes authorized to invoke an Authorize and Revoke operations;
— A provers field: the set of provers' public keys Ψ_{Id}, initialliazed at $\emptyset$, it defines the processes authorized to invoke a Present operation;
— A sequence number: sn_{Id} identifies the version of the object, it is initialized at 0 and monotonically grows; and
— A random identifer: r_{Id} is initialized at 0. It changes with each Authorize or Revoke operation and consists of the public key of a public/secret key pair.

The DID document can contain additional fields for other features, *e.g.*, a field for revoked credentials.

First, we define the Authorize and Revoke operations. As stated in Section 9.4, we do not authorize devices for specific verifiable credentials. We authorize them for DIDs, and DIDs themselves are associated with verifiable credentials. Hence, an authorized device for a DID is

authorized for all the VCs associated with this DID. We refer to a specific DID as $Id \in \mathcal{ID}$, where $\mathcal{ID}$ is the set of identifiers of all the valid DIDs Therefore, we modify the parameters of the Authorize and Revoke operations. The Authorize($Id, \mathsf{pk}_{d_A}, \mathsf{sk}_{d_M}, \mathsf{aux}$) is an operation invoked by a device d_M associated to the key sk_{d_m} which has the manager's right on Id. The operation authorizes the device associated with the public key pk_{d_A} to invoke Present operations for the verifiable credentials associated with Id. Similarly, the Revoke($Id, \mathsf{pk}_{d_A}, \mathsf{sk}_{d_M}, \mathsf{aux}$) operation revokes this right. Additionally, we can define two operations AuthorizeManager and RevokeManager whose arguments are similar to those of the Authorize and Revoke operations. These operations are used to grant and revoke the manager's rights. In the following, the device d_M is assumed to be a manager of Id, *i.e.*, $\mathsf{pk}_{D_M} \in \mathcal{M}_{Id}$. The Authorize, Revoke, AuthorizeManager and RevokeManager use the RequestObjValue operation as defined in Section 9.6.

The Authorize, Revoke, AuthorizeManager and RevokeManager operations are described in Algorithm 14 and Algorithm 15. The algorithm is divided into two parts. The first part (Algorithm 14) describes the operations invoked by the devices. They consist, for a device $d \in \mathcal{D}$, of modifying the state of the DID document to add or remove a public key from the list of authorized public keys (either for the prover or the manager role). Furthermore, these operations are used to build the proofs necessary for the AAP.Update operation. The result is sent to at least one correct process in Π using the *UpdateRequest* message, such that this process can inform other processes in Π that the state of the object Id is modified. For d to be sure that a correct process in Π is reached, d has two options: either it trusts a process in Π to be correct, or it does not. In the first case, the *UpdateRequest* message can be sent to the trusted process. Otherwise, d has to send the *UpdateRequest* message to $t_\Pi + 1$ processes. By assumption, using the second method, d knows at least one correct process will receive its request. When a correct process $p \in \Pi$ receives an *UpdateRequest* message, it verifies the validity of the proofs and invokes the AAP.Update operation. If the update is successful, the process p sends a ResultUpdate(Id, res) message to d with $res =$ `true`. In this case, d triggers an UpdateSuccess() callback. Due to potential contention, the AAP.Update operation may not be considered by processes in Π. This can occur when two different modifications of the same object Id are requested. In this case, p sends a ResultUpdate(Id, res) message to d with $res =$ `false` and d triggers an UpdateFailure() callback. d can then request a new update if it has not been revoked.

To define the operations Issue, Present and Verify, we assume we have access to a Hidden Issuer Anonymous Credential $HIAC$ scheme as defined in Chapter 5. This scheme implements the functions $HIAC$.Setup, $HIAC$.IssuerKeygen, $HIAC$.Sign, $HIAC$.VerifierSetup, $HIAC$.Randomize and $HIAC$.VerifyRandomized. $HIAC$ is assumed to fulfill the EUF-CMA and issuer indistinguishability properties. In this chapter, we only provide a high-level definition of the operations Issue, Present and Verify. A more detailed version of the algorithms is given in Chapter 10. This chapter also integrates additional features to the scheme, such as DID document creation, cre-

```
1  operation AuthorizeProverDevice(Id, pk_A, sk_M, pk_M) is
2      s ← RequestObjValue(Π)[Id];
3      s.Ψ_Id ← s.Ψ_Id ∪ pk_A;                ▷Adds pk_A to the list of authorized prover devices.
4      PrepareUpdate(Id, s, sk_M, pk_M);

5  operation RevokeProverDevice(Id, pk_A, sk_M, pk_M) is
6      s ← RequestObjValue(Π)[Id];
7      s.Ψ_Id ← s.Ψ_Id \ pk_A;                ▷Removes pk_A from the list of authorized prover devices.
8      PrepareUpdate(Id, s, sk_M, pk_M);

9  operation AuthorizeManagerDevice(Id, pk_A, sk_M, pk_M) is
10     s ← RequestObjValue(Π)[Id];
11     s.M_Id ← s.M_Id ∪ pk_A;                ▷Adds pk_A to the list of authorized manager devices.
12     PrepareUpdate(Id, s, sk_M, pk_M);

13 operation RevokeManagerDevice(Id, pk_A, sk_M, pk_M) is
14     s ← RequestObjValue(Π)[Id];
15     s.M_Id ← s.M_Id \ pk_A;                ▷Removes pk_A from the list of authorized manager devices.
16     PrepareUpdate(Id, s, sk_M, pk_M);

17 internal operation PrepareUpdate(Id, s, sk_M, pk_M) is
18     if not pk_M ∈ s.M_Id then return ;
19     r, sk_r ← Auth.Keygen();
20     s.r ← r;
21     π_r ← Auth.Prove(r, sk_r);
22     π_UpdateAuth ← Auth.Prove(pk_M, sk_M);
23     Send UpdateRequest(Id, π_UpdateAuth, s, pk_M, π_r) to a trusted correct process in Π or to t_Π + 1
         processes in Π;
24 when ResultUpdate(Id, res) is received do
25     if res = true then Trigger UpdateSuccess();
26     else
27         Trigger UpdateFailure()
```

Algorithm 14: Authorize, Revoke, AuthorizeManager and RevokeManager operations of a PPfDIMS (code for device $d_i \notin \Pi$).

28 **init:** $To\_Update_i \leftarrow$ A set initialized at $\emptyset$;
29 $Obj_i \leftarrow$ a dictionary of states initialized with the initial value of each DID document;

30 **when** $UpdateRequest(Id, \pi_{\text{UpdateAuth}}, s, \mathsf{pk}, \pi_r)$ **is** received **do**
31 **if** not ($Auth$.Verify$(\mathsf{pk}, \pi_{\text{UpdateAuth}})$ and $\mathsf{pk} \in Obj_i[Id].M_{Id}$ and $Auth$.Verify$(\pi_r, s.r)$) **then** return ;
32 $To\_Update_i \leftarrow To\_Update_i \cup (Id, s, \pi_{\text{UpdateAuth}}, (\mathsf{pk}, \pi_r))$;
33 AAP.Update$(Id, \pi_{\text{UpdateAuth}}, s, (\mathsf{pk}, \pi_r))$;

34 **when** AAP.Updated$(Id, \pi_{\text{UpdateAuth}}, \sigma, s)$ **is** received **do**
35 $Obj_i[Id] \leftarrow s$;
36 **if** $(Id, \pi'_{\text{UpdateAuth}}, s', (\mathsf{pk}', \pi'_r)) \in To\_Update_i$ **then**
37 **if** $s' = s$ **then**
38 $To\_Update_i \leftarrow To\_Update \setminus (Id, s, \pi'_{\text{UpdateAuth}}, (\mathsf{pk}', \pi'_r), p_j)$;
39 Send a ResultUpdate$(Id, \texttt{true}, \sigma, s)$ message to p_j;
40 **else**
41 **if** $s.sn = s'.sn$ **then**
42 $To\_Update_i \leftarrow To\_Update \setminus (Id, s, \pi'_{\text{UpdateAuth}}, (\mathsf{pk}', \pi'_r), p_j)$;
43 Send a ResultUpdate$(Id, \texttt{false}, \emptyset, \emptyset)$ message to p_j; ▷ *The* AAP.Update *operation was not successfull due to concurrency.*

Algorithm 15: Algorithm of a PPfDIMS enabling the multi-device authorization, code for $p_i \in \Pi$.

dential revocation, or key rotation. We define the Issue, Present and Verify operations of the VC scheme such that:

— The Issue operation associates the issued credential to a DID. The value of the DID is, therefore, one of the attributes in att when the $\mathsf{Issue}(att, \mathsf{sk}_I, \mathsf{pk}_d, \mathsf{aux})$ is invoked. However, the device's public key pk_d does not need to be filled, as the authorized devices depend on the DID document's provers field.

— The $\mathsf{Present}(vc, \mathsf{sk}_d, T_v, \mathsf{aux})$ operation is invoked by an authorized device d whose secret key is sk_d and where T_v is the set of trusted issuers of the verifier $v \in \mathcal{V}$. The verifiable credential vc is associated with the DID document identified by Id, *i.e.*, one of the attributes of vc is Id. d is assumed to know a valid proof of agreement σ on the state s of its DID document. The device d is an authorized prover for the DID Id, hence its public key pk_d is in Ψ_{Id}. d builds vp by computing the following elements:
  — $\pi_{\text{auth}} = Auth.\mathsf{Prove}(\mathsf{pk}_d, \mathsf{sk}_d)$, the proof that d knows the secret key sk_d associated to pk_d;
  — cvc is a curated version of vc, where some attributes are left as commitments to the signed values and some other attributes are revealed in clear text. Importantly, the attribute Id associated with the DID linked to vc is left as a commitment in cvc.
  — $(cvc', \pi_I) = HIAC.\mathsf{Randomize}(cvc)$, where cvc' is the randomized version of cvc that also hides the identity of the vc's issuer and π_I is the proof that the issuer of vc is in T_v, *i.e.*, π_I is the proof that the verifier v trusts the issuer of vc;
  — π_{att}, a zero-knowledge proof of the validity of a statement that uses the attributes of cvc' as inputs. For example, this ZKP can be used to prove that the user is over 18 without revealing their date of birth;
  — π_{equal}, a ZKP that the attribute Id in cvc is equal to the DID attribute in σ; and
  — $nonce = \mathsf{H}(cvc' || \pi_{att} || \pi_{\text{equal}} || \pi_I)$, the hash of the proofs that the verifier will verify.

  d sets $vp = (cvc', \pi_{\text{auth}}, \pi_I, \pi_{att}, \pi_{\text{equal}})$, the verifiable presentation that will be transferred to the verifier v. Then, it proves it is authorized to build presentations for vc by invoking the $AAP.\mathsf{Prove}(s, \sigma, nonce, \pi_{\text{auth}}, \mathsf{aux})$ operation. When it receives the $\mathsf{Authorized}(C_s, nonce)$ callback, it can send vp to the verifier. If the verifier is correct, the $\mathsf{Authorized}(C_s, nonce)$ callback is eventually triggered at p. Indeed, in our model, the verifier should not invoke Prove operations unless it is Byzantine. In this case, p can detect the verifier's faulty behavior and interrupt the exchange.

— The $\mathsf{Verify}(vp, T_v, \pi_I, \mathsf{aux})$ operation is invoked by a verifier $v \in \mathcal{P}$ with $vp = (cvc', \pi_{\text{auth}}, \pi_I, \pi_{att}, \pi_{\text{equal}})$. First, v waits until it receives the $\mathsf{Authorized}(C_s, nonce)$ callback. Then, it verifies that $nonce = \mathsf{H}(cvc' || \pi_{att} || \pi_{\text{equal}} || \pi_I)$, meaning that the proving process is authorized and that the $\mathsf{Authorized}(C_s, nonce)$ callback is related to the proof being verified, thus preventing replay attacks. Then, v verifies that

$HIAC.\mathsf{VerifyRandomized}(\star, \mathsf{sk}_v, T_v, cvc', \pi_I) = 1$ and that π_{att}, π_{equal} and π_{att} are valid zero knowledge proofs. If the tests pass, v outputs 1. Otherwise, it outputs 0.

The multi-device authorization scheme presented in this section fulfills the properties stated in Section 9.3. The multi-device authorization capability property is fulfilled thanks to the progress, the Update validity, and the Update termination properties of the AAP abstraction. The Authorize and Revoke order property is fulfilled thanks to the state agreement property of the AAP abstraction. The strong authorization property of the multi-device authorization mechanism is fulfilled thanks to the Update termination, the Append/Remove anti-flickering, and the progress properties of the AAP abstraction. The theft resistance of the multi-device authorization mechanism is fulfilled thanks to the construction of the verifiable presentation. To build a verifiable presentation, a device needs to know an authorized prover's secret key (thanks to the update validity and the Prove validity properties). However, by assumption, the adversary does not know such a secret key. Finally, the privacy-preserving multi-device authorization capability is fulfilled thanks to the anonymity property of the AAP abstraction.

9.8 Discussions

In this chapter, we proposed a multi-device authorization framework for PPfDIMS based on a new abstraction: the Anonymous Agreement Proof abstraction. However, some details were left aside. This section discusses those details. Among them, we will discuss the other usages of the multi-device authorization framework proposed, solutions to potential privacy leaks of the AAP scheme, and different efficiency improvements that can be brought to enhance the implementation of the AAP scheme presented in Section 9.6.

9.8.1 The alternative usages of the multi-device authorization scheme for PPfDIMS

As explained in Section 9.4, the multi-device authorization framework proposed in Section 9.7 is based on a data model initially proposed by Hyperledger Aries' team [19]. This data model was proposed not only to enable multi-device authorization but also for device revocation and key recovery. The device revocation mechanism is already proposed in our framework through the Revoke operation. However, we did not discuss the key recovery mechanism that can benefit from our multi-device authorization mechanism.

A key recovery mechanism must allow users to regain control over their verifiable credentials if they lose all their authorized devices. Generally, it is done through a recovery secret/public key pair. The public key is registered as the recovery key, and the secret key is kept in a safe place. In general, it is advised to use a mechanism such as Shamir secret sharing [88] to divide the recovery secret key into multiple shares and to store each share in a safe place, *e.g.*, a

piece of paper in a safe, a hard drive, a friend's device. The interesting part of such a recovery mechanism is that it can be implemented in our multi-device authorization mechanism without modifications. Indeed, a recovery key in the context of the multi-device authorization scheme presented in Section 9.7 is a secret key associated with a public key listed in the manager set $\mathcal{M}_{Id}$. The only additional requirement is for the recovery key to be used solely for this purpose and to be stored in a dedicated and secure way.

Additionally, our multi-device authorization mechanism enables easy key rotations. We recall that key rotation is an important mechanism that enhances the security of any public key-based cryptographic protocol. Therefore, this mechanism must be implemented for a PPfDIMS to be deployed in a large-scale context. To rotate a key with our multi-device authorization mechanism, a user only has to create a new secret/public key pair on their device, then they add the new public key to the list of authorized provers using its manager's secret key, and it revokes the old public key.

Those alternative usages are represented as operations of a fully functional PPfDIMS in Chapter 10.

9.8.2 Potential improvments of Section 9.6's implementation

Potential anonymity leaks and proposed solutions

A potential privacy leak of the AAP implementation presented in Section 9.6 was left aside. This leak could break the anonymity property of the AAP abstraction or at least decrease the user's privacy if it is not considered. This leak occurs if the proofs of agreement σ do not contain the same number of attributes for different DID documents. In this case, processes in Π know the number of attributes of each DID document. Therefore, they know the number of signed attributes for a given DID document identified by Id. Thus, if a DID document is the only one to have x elements, the invocation of a $AAP.\mathsf{Prove}(\star, \sigma, \star, \star, \star)$ operation where σ is a signature of x attributes identifies the DID document Id as the document used to produce the proof σ. Differences in the number of signed attributes can appear due to different numbers of authorized devices per DID document.

We propose two different methods to mitigate this flaw. Both methods are based on the fact that the user will only reveal a subset of the attributes in σ. Indeed, if multiple devices are authorized for a given DID, the user will only prove knowledge of the secret key of the device it is currently using.

The first method is to set a fixed number of signed attributes. In this case, all the proofs σ are signatures of the same number of elements. A *null* value must be defined if this method is used to fill unused attributes. Furthermore, when proving a statement from any attribute in σ, the prover must also prove that the value of this attribute is not the *null* value. Such inequality proofs can be conducted in zero knowledge using Camenisch and Schoup Zero Knowledge Proof

of inequality of discrete logarithm [174]. This method has two main impracticalities: it limits the maximum number of attributes of a DID Document, and it relies on additional Zero Knowledge Proofs that may decrease the efficiency of the AAP.Prove operation.

The second method is to divide the proof of agreement σ in as many signatures as there are attributes in a DID document, *i.e.*, for a DID document with ℓ attributes, σ is divided in ℓ shares $\sigma = \{\sigma_1, \ldots, \sigma_\ell\}$. Each share $\sigma_i, \forall i \in \{1, \ldots, \ell\}$ is a TAC signature of one of the attributes of the DID document, the identifier of this document (the DID) and the sequence number of the document. Therefore, the user can prove that all the shares are linked to a unique DID document using a Zero Knowledge Proof of equality of exponents (c.f. Chapter 5). This prevents attackers from forging access to verifiable credentials by combining shares from different DID documents or different versions of the same DID document. The main disadvantage of this method is that it forces the user and the verifier to perform additional zero-knowledge proof computations, thus increasing communication and computation complexity. However, it is more versatile than the first solution as it does not limit the number of attributes of a DID document.

9.8.3 Improved view synchronization when the size of $\mathcal{P}$ is small

The view synchronization of the state of the on-ledger objects for the Prove operation as presented in Section 9.6 requires each process in $\mathcal{P}$ to share its entire local view of the objects along with signatures of each state. This design choice was used to simplify the description of the algorithm. However, it leads to a significant message size complexity overhead. We propose several solutions to reduce this complexity. Each solution proposed here reduces the message size complexity but increases the number of messages. Therefore, we expose a tradeoff between those two parameters.

The first solution is to share only the sequence numbers and the random identifiers of each state in the ProveAns message, along with a signature of those parameters by the sender of the ProveAns message. This solution greatly reduces the size of the ProveAns messages. However, the recipient of those messages still needs to request to processes in $\mathcal{P}$ the agreement proofs σ for the states it did not receive with the RequestObjValue operation. Therefore, a second request must be sent to those processes after the recipient of the ProveAns messages received $n_{\mathcal{V}} - t_{\mathcal{V}}$ such messages. Thus, this solution increases the latency by two asynchronous rounds.

The second solution is used when the set $\mathcal{P}$ comprises only two processes. This is the case for PPfDIMS, where $\mathcal{P}$ comprises a prover and a verifier. In this case, we can further reduce the size of messages by using techniques such as hash trees [175] to detect which states are outdated in the view of each process (the verifier and the prover). Processes represent their local view of the DID documents' state under a binary hash tree. Then, they only send the tree's root to the other process. If the roots of the two processes differ, they also share the two children of the root, and so on. Thus, if their views match, the two processes must only share one hashed value

to synchronize. Otherwise, if only one element differs, then the two processes only need to share $\log(|\mathcal{ID}|)$ elements to synchronize their views, where $\mathcal{ID}$ is the set of all identifiers in use.

Yet another method is to modify the RequestObjValue function such that the x last Update operations are also returned. Therefore, processes in $\mathcal{P}$ can only share those x values along with the hash of the Obj set. With high probability and if the parameter x is chosen accordingly, the local view of the processes in $\mathcal{P}$ will differ by less than those x Update. Hence, they only have to share x values to detect which states are outdated in each process' view and to synchronize their views.

9.9 Conclusion

In this chapter, we presented the problem of the multi-device authorization capability applied to PPfDIMSs. This problem is an open problem in the PPfDIMS community, as the only solution proposed to solve it without loss of privacy or security was published as a draft and was not studied for five years [19]. We proposed a novel solution to this problem based on a new abstraction, the Anonymous Agreement Proof abstraction. The AAP abstraction makes it possible for a process $p \in \mathcal{P}$ to prove to other processes in the set $\mathcal{P}$ that a second set of processes Π agreed on the state of an object Id. This proving mechanism implemented by the AAP abstraction additionally hides any identifying information about the object Id, thus preserving the privacy of the processes. We proposed an implementation of the multi-device authorization mechanism for PPfDIMS based on the AAP abstraction. This implementation can also be used to rotate keys and to create backup keys that make it possible to recover the right to create verifiable presentations in case of key loss.

The AAP abstraction is used in Chapter 10 to build a fully functional PPfDIMS.

Chapter 10

A privacy preserving fully distributed IMS framework with (almost) no consensus

This chapter summarizes the contributions of this thesis. It uses them as building blocks of a theoretical PPfDIMS framework. It explains how the main features of a PPfDIMS presented in Chapter 2 can be implemented efficiently, with high privacy preservation guarantees and minimal synchronization requirements. It uses the same tools as the one used in Section 9.7.

10.1 Model

The model considered in this chapter is the message passing model presented in Section 4.1.2.

10.2 Building blocks

This section presents the main building blocks used to build a PPfDIMS. Each building block is made implementation independent on purpose. Multiple design choices can be made to implement them. In the following, it is assumed that each system actor (user, issuer, and verifier) owns a DID document identified by a DID. However, DID documents are not created at initialisation.

General purpose Zero Knowledge Proof system In this chapter, we use general purpose zero knowledge proof systems as defined in Chapter 4

The ZKP system has two functions, $ZKP.\mathsf{Prove}(st)$ and $ZKP.\mathsf{Verify}(\pi, st)$. The $ZKP.\mathsf{Prove}(st)$ function takes as input a statement st expressed, for ease of comprehension, with the Camenisch and Stadler notation [78]. This notation is as follows:

$$ZKP\{(s) : st\}.$$

Values between parenthesis (here, s) are hidden values that the verifier does not learn, and the

statement after the semicolon (here, st) is the statement proven. The $ZKP.\mathsf{Verify}(\pi, st)$ function takes as input a proof π and a statement st. The function outputs 1 if the proof π is a valid proof of knowledge of the statement st. It outputs 0 otherwise.

The way this ZKP framework is implemented depends on the usage. Each proof can either be an ad hoc Schnorr like ZKP [176] or a more elaborated zk-SNARKs [177].

Anonymous Credential The main component of any PPDIMS is a signature scheme. Indeed, the distribution of a PPDIMS, be it *fully* distributed or *partially* distributed, comes from the fact that users can present verifiable identity elements to service providers without interacting with the identity provider. Once the signature is issued—and as long as it is valid—users can present it as many times as they want without requiring the action of another party.

Hence, the signature scheme is the most critical building block of any PPfDIMS. Furthermore, it is also the main privacy-sensible part of the system. Information signed is the identity element of users. Therefore, the signature scheme should be designed and chosen carefully.

The requirements exposed in Chapter 2, as well as the study of Hidden Issuer Anonymous Credentials tend to show that anonymous credentials—with or without the issuer-indistinguishability property—present all the properties required to build a PPfDIMS. Such an anonymous credential scheme can implement a fully functional PPfDIMS. For example, the Pointcheval Sanders scheme [84] can be used if issuer-indistinguishability is not required, or the HIAC scheme (c.f. Chapter 5) or its evolution, the Sanders Traoré scheme [31] can be used if this property is required.

However, any anonymous credential must contain two specific attributes to enable the credential revocation and the multi-device feature. The first attribute is a DID Id. This attribute links a credential to its DID, as explained in Chapter 9. The DID can later be linked to devices' public keys. The user can compute a ZKP during the presentation of the credential to prove that the credential is linked to a specific DID and that the device used by the user is an authorized prover for this DID (c.f. Section 9.7). Therefore, the Id attribute in a credential can be used to enable the multi-device capability and should always be one of the attributes signed by an issuer when issuing an anonymous credential.

The second necessary attribute is a random identifier r_{revoc} chosen by the issuer and known to the user. This identifier is meant always to stay hidden, *i.e.*, when the user builds a verifiable presentation, r_{revoc} is left as a commitment. This identifier is only used to revoke credentials. When the issuer or the user wants to revoke a credential, r_{revoc} is added to the issuer's revocation list or the user's revocation list. The implementation of this revocation list is described in Section 10.3. When a user presents a credential, he proves that r_{revoc} is not in either of the two revocation lists.

In the following, we use an abstract anonymous credential scheme that supports the trusted

issuer and the issuer's indistinguishability properties as defined in Chapter 5. The scheme exhibits the following algorithms:

1. $(pp) \leftarrow AC.\mathsf{Setup}(\lambda)$: On input of a security parameter λ, outputs pp, a description of public parameters whose security level depend on λ;
2. $(isk, ipk) \leftarrow AC.\mathsf{IssuerKeyGen}(pp)$: on input of pp, an issuer computes a secret/public key pair (isk, ipk);
3. $(\mathcal{S}, \mathsf{aux}_{\mathcal{S}}) \leftarrow AC.\mathsf{VerifierSetup}(pp, \mathcal{S}')$: on input of pp and a set of ℓ issuers $\mathcal{S}'$, a verifier outputs $\mathcal{S}$ a selection of $\ell \leq \ell'$ trusted issuers in $\mathcal{S}'$. The algorithm also outputs auxiliary information $\mathsf{aux}_{\mathcal{S}}$;
4. $(\sigma) \leftarrow AC.\mathsf{Sign}(pp, M, C, isk)$: on input of pp, a set of k messages $M = \{m_1, \ldots, m_k\}$, a set of commitments to messages $C = \{c_{k+1}, \ldots, c_{k'}\}, k' \geq k$, and an issuer secret key isk, an issuer outputs a signature σ;
5. $(\sigma') \leftarrow AC.\mathsf{Uncommit}(pp, M, C, \sigma, ipk)$: on input of pp, a set of k messages $M = \{m_1, \ldots, m_k, m_{k+1}, \ldots, m_{k'}\}$, a set of commitments to messages $C = \{c_{k+1}, \ldots, c_{k'}\}$, a signature σ and an issuer public key ipk, the algorithm outputs σ', a signature on the set of messages M, where values $\{m_{k+1}, \ldots, m_{k'}\}$ are uncommitted.
6. $(ipk', \pi_{ipk'}, \sigma', \pi_{\sigma'}) \leftarrow AC.\mathsf{Randomize}(pp,\ ipk,\ \mathcal{S},\ \sigma, \mathsf{aux}_{\mathcal{S}}, M, st, r, c)$: on input of pp, a signature σ on the set M of k messages, a public key ipk of the issuer of σ, a set of issuers $\mathcal{S}$, public auxiliary information $\mathsf{aux}_{\mathcal{S}}$, a statement s and two vectors $r, c \in \{0,1\}^k$, a user:
   (a) Produces ipk', a randomized version of ipk;
   (b) Produces a proof $\pi_{ipk'}$ that ipk' is an element of $\mathcal{S}$;
   (c) Produces a randomized signature σ' where if $r_i = 1$, then $m_i \in M$ is revealed, otherwise, if $r_i = 0$, then $m_i \in M$ is left as a commitment;
   (d) Produces $\pi_{\sigma'}$ a proof of knowledge of signature for the messages $m_i\ \forall i \in \{1,\ \ldots, k\}$ such that $c_i = 1$. This ZKP is composed of the statement st, such that the committed messages m_i can be used to prove other statements.

   The algorithm outputs ipk', $\pi_{ipk'}$, and σ'.
7. $\{0,1\} \leftarrow AC.\mathsf{VerifyRandomized}(pp, \mathsf{aux}_{\mathcal{S}}, ipk',\ \pi_{ipk'},\ \sigma',\ M,\ \pi_{\sigma'}, st)$: on input of pp, verifier's auxiliary information $\mathsf{aux}_{\mathcal{S}}$, a randomized issuer key ipk', a proof $\pi_{ipk'}$, a randomized signature σ', a set of messages M, a proof $\pi_{\sigma'}$ and a statement st, a verifier outputs 1 if the proof $\pi_{ipk'}$ is valid, if σ' is a valid signature on the set of messages M signed with the secret key associated with ipk', and if $\pi_{\sigma'}$ is a valid proof of the statement st relative to elements in M and to elements left as commitments in σ'. The algorithm outputs 0 otherwise.

Cryptographic authentication mechanism The third cryptographic building block required is a cryptographic authentication mechanism. This mechanism is the same as the one presented in Section 9.6.1 and can be implemented in numerous ways, *e.g.*, using classical cryptographic signatures or zero-knowledge proofs. Similarly to Section 9.6.1, the authentication mechanism has 3 algorithms, $Auth.\mathsf{KeyGen}$, $Auth.\mathsf{Prove}$ and $Auth.\mathsf{Verify}$. The public and secret keys are created with the algorithm $\mathsf{sk}, \mathsf{pk} \leftarrow Auth.\mathsf{KeyGen}()$. The proof of knowledge of the secret key is created with an algorithm $\pi_{auth} \leftarrow Auth.\mathsf{Prove}(\mathsf{sk}, \mathsf{pk})$. This proof is verified with the algorithm $\{0,1\} \leftarrow Auth.\mathsf{Verify}(\mathsf{pk}, \pi_{auth})$ that outputs 1 only if π_{auth} is a valid proof of knowledge of the secret key sk associated with the public key pk. Alternatively, pk can be a randomizable Pedersen commitment. Thus, the verifier does not learn the actual value of pk when it invokes $Auth.\mathsf{Verify}$.

Ledger The fourth building block required to build a PPfDIMS is a DID-capable distributed ledger. This ledger enables the auxiliary features of a PPfDIMS, namely the revocation of credentials, the multi-device feature, and the key recovery feature. Additionally, a distributed ledger can be used to publish information, *i.e.*, the list of trusted issuers and the type of credentials accepted for a verifier, the type of credentials issued for an issuer, or the type of cryptographic primitives supported.

This ledger is implemented by a message-passing distributed algorithm maintained by processes in Π as defined in Section 4.1.2.

Processes in Π have access to an algorithm AAP implementing the AAP abstraction as defined in Section 9.5. The algorithm is assumed to be implemented using abstractions with low synchronization requirements, such as the Cascading Consensus abstraction. For example, the implementation proposed in Section 9.6 uses solely the Cascading Consensus abstraction and the send/receive primitive. As defined in Section 9.5, the AAP abstraction has two operations, $AAP.\mathsf{Update}$ and $AAP.\mathsf{Prove}$, and three callbacks $AAP.\mathsf{Updated}$, $AAP.\mathsf{Authorized}$ and $\mathsf{NotAuthorized}$. As explained in Section 9.7, we use a specific version of the AAP abstraction where the $AAP.\mathsf{Prove}$ operations concern only two processes, a device d and a verifier V. Those pairs create ad hoc $\mathcal{P}$ sets. We add this as a parameter of the $AAP.\mathsf{Prove}$ operation. We note $AAP.\mathsf{Prove}_{\mathcal{P}}$ the $AAP.\mathsf{Prove}$ operation applied to the set $\mathcal{P}$. Similarly, we note $AAP.\mathsf{Authorized}_{\mathcal{P}}$ the $AAP.\mathsf{Authorized}$ callbacks applied to the set $\mathcal{P}$. On the other hand, the $AAP.\mathsf{Update}$ operations are always invoked and managed by the same set of processes Π. Those operations impact all the potential $\mathcal{P}$ sets in an atomic manner.

Additionally, the $AAP.\mathsf{Authorized}_{\mathcal{P}}$ returns the set of states processes in $\mathcal{P}$ agreed upon. More specifically, the callback is $AAP.\mathsf{Authorized}_{\mathcal{P}}(C_s, nonce, Prove\_Obj)$, where the $Prove\_Obj$ variable is the set of states of each DID document as seen by the user and the verifier in $\mathcal{P}$ for this specific callback. Using the implementation in Section 9.6, this modification to the ab-

straction comes for free, as any correct process computes the $Prove\_Obj$ variable before the $AAP.\mathsf{Authorized}_{\mathcal{P}}$ callback is triggerred.

Each client (users, issuers, and verifiers) can request the invocation of operations on the ledger— *i.e.*, modify the ledger—by sending requests to any node $p \in \Pi$. Correct processes in Π are assumed to always propagate requests according to their prescribed algorithm. However, the threat model we use assumes that there may exist Byzantine processes that do not propagate the requests. Therefore, each client can disseminate its request through each node of the system. Hence, an updated list of all existing nodes is available to the clients. It is also assumed that the clients sign any request they issue. Therefore, no Byzantine node can tamper with a client's request.

Furthermore, clients have access to the $\mathsf{RequestObjValue}(\Pi)$ operation. This operation can be seen as a read operation for clients. Its behavior is defined in Section 9.6.2. This operation returns the state of the ledger to the querying client. Validity of the result is ensured thanks to the assumption on correct processes, either because the querying client trusts a process in Π, or because it requests $t + 1$ processes.

DID and DID document The authentication mechanism, along with the revocation mechanism of our PPfDIMS, is based on DID documents identified by DIDs as specified by the W3C standard [8]. A DID document is used to enable auxiliary features of a PPfDIMS. A DID method (the specification of a DID document and its DID) must specify how an actor can create, read, update, and delete a DID document. In the followings, we do not specify the format of DIDs considered. We assume each DID follows the W3C standard. DID documents can be separated into user DID documents and issuer/verifier DID documents. Each user DID document contains the following fields:

- A DID field DID: links the document to its identifier.
- A type field T: identifies the DID document as a user document.
- A manager's keys field $\mathcal{M}_{Id}$: this field is a set of managers' public keys. For each DID document Id, this field is initialized with one public key; it defines the devices authorized to modify the DID document.
- A prover's keys field Ψ_{Id}: this field is a set of provers' public keys. It is initialized at $\emptyset$. It defines the processes authorized to invoke a $\mathsf{Present}$ operation for the verifiable credentials associated with this DID;
- A sequence number sn_{Id}: identifies the version of the object, it is initialized at 0 and monotonically grows.
- A random identifer r_{Id}: initialized at 0. It changes with each $\mathsf{Authorize}$ or Revoke operation and consists of the public key of a public/secret key pair.
- A revoked credentials field $\mathcal{RC}$: a set of credentials revoked by the user, initialized as an

empty set.

— An information field $\mathcal{AI}$: this field is initialized as an empty set; it contains all the additional information the other system actors may need.

An issuer/verifier DID document contains the following fields:

— A DID field DID: links the document to its identifier.

— A type field T: identifies the DID document as an issuer/verifier document.

— A manager's keys field $\mathcal{M}_{Id}$: this field is a set of managers' public keys. For each DID document Id, this field is initialized with one public key, which defines the devices authorized to modify the DID document.

— A sequence number sn_{Id}: identifies the version of the object, it is initialized at 0 and monotonically grows.

— A revoked credentials field $\mathcal{RC}$: a set of credentials revoked by an issuer, initialized as an empty set.

— An information field $\mathcal{AI}$: this field is initialized as an empty set; it contains all the additional information the other system actors may need. For example, this field can be used to publish service endpoints, type of credentials issued, type of cryptographic schemes supported, type of credential verified, a set of trusted issuers for a specific type of credential, etc.

In the following, we use the AAP abstraction (c.f. Chapter 9) to implement the ledger used to maintain DID documents. Hence, the issuer/verifier DID has a random identifier field and a prover's key field for compatibility. However, those fields are unused and can be discarded by processes in Π.

Naming Algorithm To create DIDs, it is required to have access to a distributed naming algorithm. Two types of naming algorithms can be used: algorithms with the human-choosable property and algorithms without this property. As explained in Chapter 7 and in Section 8.7.1, depending on the properties required, synchronization may or may not be required. More precisely, if the human-choosable property is required, then the naming algorithm requires synchronization, whereas if this property is unnecessary, then the short-naming algorithm can be used. This algorithm does not require synchronization.

The naming algorithm associates a name to a public key. In the followings, when the naming operation is succesfull, the name becomes the DID field of a new DID document, and the public key becomes the only public key in the manager's field of the DID document.

The non-human-chossable naming abstraction used in the followings is the short-naming abstraction $Short\_Naming$ (c.f. Section 8.7.1). This abstraction is executed by processes in Π. It has one operation, $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$, and one callback, $Short\_Naming.\mathsf{Success}(n, \mathsf{pk}, \pi)$. The $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$ operation is used to create

a new DID document, whose DID is a substring of pk. The $\mathsf{Success}(n, \mathsf{pk}, \pi)$ callback is triggerred when the DID document is created after the invocation of the $Short\_Naming.\mathsf{Claim}(\mathsf{pk}, \pi)$ operation, and where n is the name attributed to this DID document. [1] The $Short\_Naming$ algorithm fulfills the *Unicity*, *Short-names*, *Agreement* and *Termination* properties as defined in Section 8.7.1.

Additionnaly, we use in the followings a human-choosable naming abstraction HC_Naming. This abstraction has one operation $HC\_Naming.\mathsf{Claim}(n, \mathsf{pk}, \pi)$ and one callback $HC\_Naming.\mathsf{Success}(n, \mathsf{pk}, \pi)$. The abstraction fulfills the following properties:

- *Validity.* Let a correct $p \in \Pi$ receive the callback $HC\_Naming.\mathsf{Success}(n, \mathsf{pk}, \pi)$, then π is a valid proof of knowledge of the secret key associated to pk.
- *Unicity.* A correct process $p \in \Pi$ receives at most one callback $HC\_Naming.\mathsf{Success}(n, \star, \star)$ for each name n.
- *Termination.* Let a correct process $p \in \Pi$ invoke the $HC\_Naming.\mathsf{Claim}(n, \mathsf{pk}, \pi)$ operation, with π a valid proof of knowledge of the secret key associated to pk. Then the $HC\_Naming.\mathsf{Success}(n, \mathsf{pk}', \pi')$ callback is eventually received at p. [2]
- *Agreement.* If a correct process $p \in \Pi$ receives the callback $HC\_Naming.\mathsf{Success}(n, \mathsf{pk}, \pi)$, then the callback $HC\_Naming.\mathsf{Success}(n, \mathsf{pk}, \pi)$ is triggerred at all correct processes in Π.

We propose an implementation of the HC_Naming abstraction based on the Cascading Consensus abstraction. This implementation uses a dictionnary of cascading consensus algorithms. The dictionnary $CasCons$ contains one cascading consensus for each potential name that may be claimed. We seperate each instance of the naming algorithm such that they can run in parallel. Hence increasing the efficiency of the scheme. The algorithm is presented in Algorithm 16.

```
1 operation HC_Naming.Claim(n, pk, π) is
2 |  if Auth.VerifySig(pk, π) = false then return ;
3 |  if HC_Naming.Success(n, ⋆, ⋆) as already been received then return ;
4 |  CasCons[n].ccons_propose(pk, π).

5 when CasCons[n].ccons_decide(pk, π) do
6 |  if VerifySig(pk, π) = false then return ;
7 |  Trigger HC_Naming.Success(n, pk, π).
```

Algorithm 16: Human-choosable naming algorithm implementation based on the cascading consensus abstraction (code for p_i)

1. The specification of the short-naming abstraction in Section 8.7.1 does not provide such a callback. However, it is easy to modfiy the scheme such that, when $\langle n, \mathsf{pk}, \pi \rangle$ is added to $Names_i$, then the callback is triggerred at p_i.

2. Note that pk′ ad π' may be different from pk and π respectively.

DIDComm and anonymous two parties communications The last building block is a privacy preserving two party communication protocol. The presentation of a credential requires two-party communication between a user and a verifier. This protocol must be conducted anonymously, i.e., this communication should not reveal the user's identity.

To do so, the DIDcomm [10] standard proposes a way to encapsulate messages and use information available in the DID documents of a DIMS to conduct two-party communications. The idea is for the verifier to be constantly connected and available through a publicly known endpoint. In contrast, the user may disconnect and uses privacy-preserving endpoints, *i.e.*, endpoints used only once per presentation. Furthermore, the client should only connect through privacy-preserving networks such as onion routing. The design of such a protocol is out of the scope of this thesis. The interested reader can refer to the W3C standard [10] for more information.

10.3 PPfDIMS Implementation

This section presents an implementation of the main operations necessary to build a PPfDIMS. This implementation considers Allen's minimalization principle: no identifying information related to the user leaks during the presentation of a credential. Furthermore, during a presentation, the user does not reveal any extra information besides the statement they want to prove. This implementation also respects the fully distributed definition stated in Chapter 1. If different entities manage processes in Π, then up to $t < \frac{n}{3}$ processes can crash or act in a Byzantine way, and the system will not be impacted.

We present the PPfDIMS's main operations and explain how to implement them in a privacy-preserving manner while minimalizing the synchronization requirements between large sets of processes. The operations that are required to be implemented are the following:

— CreateDID: This operation, invoked by a user, an issuer, or a verifier, creates a new DID and its associated DID document. The operation takes as input a public key pk and the associated secret key sk. The public key becomes the first authorized device with the manager's right for this DID. This operation can be invoked in two ways. For an issuer or a verifier, the human-choosable property can be used (c.f. Chapter 7); for a user, it should not be used. The operation is associated with a callback DIDCreated(DID, pk). The callback is eventually triggered at any correct process that invokes CreateDID(pk, sk) for the non-human-choosable naming algorithm. Conversely, with the human-choosable algorithm opted-in, the callback is only triggered at a correct process that invokes CreateDID(n, pk) if there is no contention on the name n. If there is contention on the name n, then the name n is attributed to one of the processes that claimed it, and the other process receives the DIDCreationFailure(n, pk) callback. This operation is presented in Algorithm 17 and 18 for a device acting as a user and in Algorithm 19 for a device acting as an issuer or a

verifier.

— AuthorizeProverDevice: This operation authorizes a new device, identified by its public key, to build a verifiable presentation for verifiable credentials associated with a specific DID. The operation takes as input a DID, the public key of the newly authorized device, and proof of knowledge of one of the secret keys associated with one of the manager's public keys of the DID. If the operation is successful, a UpdateSucess callback is triggered at the invoking process. Otherwise, a UpdateFailure callback is triggered at the invoking process. This operation is presented in Algorithm 17 and 18 for a device acting as a user.

— AuthorizeManagerDevice: This operation gives the manager's right to a new device, identified by its public key, for a specific DID. The operation takes as input a DID, the public key of the newly authorized device, and proof of knowledge of one of the secret keys associated with one of the manager's public keys of the DID. If the operation is successful, a UpdateSucess callback is triggered at the invoking process. Otherwise, a UpdateFailure callback is triggered at the invoking process. This operation is presented in Algorithm 17 and 18 for a device acting as a user and in Algorithm 19 for a device acting as an issuer or a verifier.

— RevokeProverDevice: This operation revokes the right from a device, identified by its public key, to build a verifiable presentation for verifiable credentials associated with a specific DID. The operation takes as input a DID, the public key of the newly authorized device, and proof of knowledge of one of the secret keys associated with one of the manager's public keys of the DID. If the operation is successful, a UpdateSucess callback is triggered at the invoking process. Otherwise, a UpdateFailure callback is triggered at the invoking process. This operation is presented in Algorithm 17 and 18 for a device acting as a user.

— RevokeManagerDevice: This operation revokes the manager's right from a device, identified by its public key, for a specific DID. The operation takes as input a DID, the public key of the newly authorized device, and proof of knowledge of one of the secret keys associated with one of the manager's public keys of the DID. If the operation is successful, a UpdateSucess callback is triggered at the invoking process. Otherwise, a UpdateFailure callback is triggered at the invoking process. This operation is presented in Algorithm 17 and 18 for a device acting as a user and in Algorithm 19 for a device acting as an issuer or a verifier.

— RotateManagerKey: This operation rotates the manager key of a device for a specific DID. The operation takes as input the new public key, the old public key, and the associated old secret key. If the operation is successful, a UpdateSucess callback is triggered at the invoking process. Otherwise, a UpdateFailure callback is triggered at the invoking process. This operation is presented in Algorithm 17 and 18 for a device acting as a user and in Algorithm 19 for a device acting as an issuer or a verifier.

— RotateProverKey: This operation rotates the prover key of a device for a specific DID. It takes as input a proof of knowledge of a manager's secret key. If the operation is successful, a UpdateSucess callback is triggered at the invoking process. Otherwise, a UpdateFailure callback is triggered at the invoking process. This operation is presented in Algorithm 17 and 18 for a device acting as a user.

— RevokeCred: revokes a credential based on its random revocation number. A revoked credential can no longer be used to build verifiable presentations. If the operation is successful, a UpdateSucess callback is triggered at the invoking process. Otherwise, a UpdateFailure callback is triggered at the invoking process. This operation is presented in Algorithm 17 and 18 for a device acting as a user and in Algorithm 19 for a device acting as an issuer or a verifier. [3]

— Issue: an operation invoked by an issuer, in collaboration with a user, issues a credential to this user based on the verification of its identity elements. This process links the issued credential to an identifier of the user, its DID, and a random number r_{revoc} used for credential revocation. The operation takes as input a DID DID, a set of attributes M and commitments to attributes C, and a proof π that the user requesting the issuance of a credential has the required attributes. The proof π can either be digital (*e.g.*, another anonymous credential) or a physical proof (*e.g.*, an Identity card). The operation outputs vc, a verifiable credential, and r_{revoc}. We assume the issuer draws a new random number for each new credential issued. This operation is presented in Algorithm 21.

— Present: an operation invoked by an user's device d. Creates a verifiable presentation from a verifiable credential. It takes as input:
  — a verifier V;
  — An anonymous credential vc;
  — The set M of messages signed in vc;
  — The DID DID associated to vc;
  — The revocation number r_{revoc} of vc;
  — The public key pk_I of the issuer of the credential;
  — The set T_V of trusted issuers of a verifier;
  — Auxilary information relative to T_V; [4]
  — A statement st to be proven in zero knowledge;

3. The RevokeCred operation is implemented in Algorithms 18 and 19 using the PrepareUpdate function. This function sends an *UpdateRequest* message to some processes in Π. This update is then shared with the other processes via the Cascading Consensus algorithm. However, the revocation of a credential can be seen as a DenyList which only supports the Append operation. Hence, and as shown in Chapter 6, this RevokeCred operation, which is the equivalent of the Append operation of a DenyList, could have been implemented without using a consensus algorithm. However, we preferred not to add new specificities to the algorithms, as they are already long enough, and because the Cascading Consensus algorithm acts as a BRB algorithm when there is no contention.

4. In the case of the Hidden Issuer Anonymous Credential scheme presented in Chapter 5, the auxiliary information is the verifier's public key vpk.

— A vector r defining the values of M that are revealed to the verifier;
— A vector c defining the values of M that are left as commitment but which are used to prove s;
— The state s of the DID document DID
— A valid proof σ of anonymous agreement on the state s of the DID document DID (c.f., Section 9.5);

The main goals of this operation are to enforce unlinkability, to hide the issuer of the verifiable credential if the issuer-indistinguishability property was opted in, to enforce selective disclosure, to prove that the credential (or the credentials) used is not revoked, and to authenticate the device used to invoke the operation. It outputs vp, a verifiable presentation. This operation uses a ListRevocation function. This function takes as input a list of DID documents and outputs a set of all revocated credentials that concatenates each revocation list of each DID document. This operation is presented in Algorithm 22.

— Verify: this operation, invoked by a verifier, is used to verify that a verifiable presentation is a valid certificate of attributes specified by the verifier, that the verifier trusts the issuer that issued the credential, that the device that built the verifiable presentation is authenticated for this verifiable credential, and that the credential is not revoked. This operation uses the ListRevocation function as defined for the Present operation. This operation is presented in Algorithm 23

The different operations necessary to build a functionnal PPfDIMS are presented in Algorithms 17, 18, 19, 21, 22, and 23. The functions relative to the communication between devices and processes in Π are also given in Algorithm 20.

10.4 Conclusion

This chapter presented a guideline for implementing a fully functional PPfDIMS. The building blocks proposed for the implementation are the main contributions of this thesis. They make it possible to implement a PPfDIMS with all the features that the user, the identity provider, and the service provider may require. Furthermore, the different contributions of this thesis were focused on user privacy and low synchronization requirements. Therefore, the resulting framework also exposes low synchronization requirements, with high privacy and security guarantees.

This framework is a guideline for developers. The goal of this thesis is for them to use all or parts of the techniques presented here to propose new PPfDIMS implementations. This implementation effort is ongoing in the WIDE team, where we aim to propose our own fully functional PPfDIMS with low synchronization requirements based on the Rust language.

```
1  operation CreateDID(pk, sk) is
2  |  π ← Auth.Prove(pk, sk);
3  |  Send a message RequestCreateNHCDID(pk, π) to a trusted correct process in Π or to t_Π + 1
   |    processes in Π;
4  operation AuthorizeProverDevice(DID, pk_A, sk_M, pk_M) is
5  |  s ← RequestObjValue(Π)[Id];
6  |  if not pk_M ∈ s.M then return ;
7  |  s.Ψ ← s.Ψ ∪ pk_A;                          ▷Adds pk_A to the list of authorized prover devices.
8  |  PrepareUpdate(DID, s, sk_M, pk_M);
9  operation RevokeProverDevice(DID, pk_A, sk_M, pk_M) is
10 |  s ← RequestObjValue(Π)[Id];
11 |  if not pk_M ∈ s.M then return ;
12 |  s.Ψ ← s.Ψ \ pk_A;                          ▷Removes pk_A from the list of authorized prover devices.
13 |  PrepareUpdate(DID, s, sk_M, pk_M);
14 operation AuthorizeManagerDevice(DID, pk_A, sk_M, pk_M) is
15 |  s ← RequestObjValue(Π)[Id];
16 |  if not pk_M ∈ s.M_DID then return ;
17 |  s.M ← s.M ∪ pk_A;                          ▷Adds pk_A to the list of authorized manager devices.
18 |  PrepareUpdate(DID, s, sk_M, pk_M);
19 operation RevokeManagerDevice(DID, pk_A, sk_M, pk_M) is
20 |  s ← RequestObjValue(Π)[Id];
21 |  s.M ← s.M \ pk_A;                          ▷Removes pk_A from the list of authorized manager devices.
22 |  PrepareUpdate(DID, s, sk_M, pk_M);
23 operation RotateProverKey(DID, pk_new, pk_old, pk_M, sk_M) is
24 |  s ← RequestObjValue(Π)[Id];
25 |  s.Ψ ← s.Ψ ∪ pk_new \ pk_old; ▷Replaces pk_old with pk_new in the list of authorized prover devices.
26 |  PrepareUpdate(DID, s, sk_M, pk_M);
27 operation RotateManagerKey(DID, pk_new, pk_old, sk_M) is
28 |  s ← RequestObjValue(Π)[Id];
29 |  s.M ← s.M ∪ pk_new \ pk_old;      ▷Replaces pk_old with pk_new in the list of authorized manager
   |    devices.
30 |  PrepareUpdate(DID, s, sk_M, pk_M);
```

Algorithm 17: Ledger operations of a PPfDIMS (code for a device $d_i \notin \Pi$ acting as a user) (PART I).

```
31 internal operation PrepareUpdate(DID, s, sk_M, pk_M) is
32     if not pk_M ∈ s.M then return ;
33     r, sk_r ← Auth.Keygen();
34     s.r ← r;
35     π_r ← Auth.Prove(r, sk_r);
36     π_UpdateAuth ← Auth.Prove(pk_M, sk_M);
37     Send UpdateRequest(DID, π_UpdateAuth, s, pk_M, π_r) to a trusted correct process in Π or to
         t + 1 processes in Π;
38 operation RevokeCredential(DID, sk_M, pk_M, r_revoc) is
39     s ← RequestObjValue(Π)[Id];
40     s.RC ← s.RC ∪ r_revoc;                ▷Adds the r_revoc element to the list of revoked credentials.
41     PrepareUpdate(DID, s, sk_M, pk_M);
42 when ResultUpdate(DID, res) is received do
43     if res = true then Trigger UpdateSuccess();
44     else
45         Trigger UpdateFailure()
46 when DIDCreated(DID, pk) is received do
47     Trigger DIDCreated(DID, pk).
```

Algorithm 18: Ledger operations of a PPfDIMS (code for a device $d_i \notin \Pi$ acting as a user) (PART II).

48 **operation** CreateDID$(n, \mathsf{pk}, \mathsf{sk})$ **is**
49 $\quad \pi \leftarrow Auth.\mathsf{Prove}(\mathsf{pk}, \mathsf{sk})$;
50 $\quad$ Send a message $RequestCreate\text{HCDID}(n, \mathsf{pk}, \pi)$ to a trusted correct process in Π or to $t_\Pi + 1$ processes in Π;
51 **operation** AuthorizeManagerDevice$(DID, \mathsf{pk}_A, \mathsf{sk}_M, \mathsf{pk}_M)$ **is**
52 $\quad s \leftarrow \mathsf{RequestObjValue}(\Pi)[Id]$;
53 $\quad$ **if** not $\mathsf{pk}_M \in s.\mathcal{M}$ **then return** ;
54 $\quad s.\mathcal{M} \leftarrow s.\mathcal{M} \cup \mathsf{pk}_A$; ▷*Adds* pk_A *to the list of authorized manager devices.*
55 $\quad \mathsf{PrepareUpdate}(DID, s, \mathsf{sk}_M, \mathsf{pk}_M)$;
56 **operation** RevokeManagerDevice$(DID, \mathsf{pk}_A, \mathsf{sk}_M, \mathsf{pk}_M)$ **is**
57 $\quad s \leftarrow \mathsf{RequestObjValue}(\Pi)[Id]$;
58 $\quad s.\mathcal{M} \leftarrow s.\mathcal{M} \setminus \mathsf{pk}_A$; ▷*Removes* pk_A *from the list of authorized manager devices.*
59 $\quad \mathsf{PrepareUpdate}(DID, s, \mathsf{sk}_M, \mathsf{pk}_M)$;
60 **operation** RotateManagerKey$(DID, \mathsf{pk}_{\text{new}}, \mathsf{pk}_{\text{old}}, \mathsf{sk}_M)$ **is**
61 $\quad s \leftarrow \mathsf{RequestObjValue}(\Pi)[Id]$;
62 $\quad s.\mathcal{M} \leftarrow s.\mathcal{M} \cup \mathsf{pk}_{\text{new}} \setminus \mathsf{pk}_{\text{old}}$; ▷*Replaces* pk_{old} *with* pk_{new} *in the list of authorized manager devices.*
63 $\quad \mathsf{PrepareUpdate}(DID, s, \mathsf{sk}_M, \mathsf{pk}_M)$;
64 **operation** RevokeCredential$(DID, \mathsf{sk}_M, \mathsf{pk}_M, r_{\text{revoc}}))$ **is**
65 $\quad s \leftarrow \mathsf{RequestObjValue}(\Pi)[Id]$;
66 $\quad s.\mathcal{RC} \leftarrow s.\mathcal{RC} \cup r_{\text{revoc}}$; ▷*Adds the* r_{revoc} *element to the list of revoked credentials.*
67 $\quad \mathsf{PrepareUpdate}(DID, s, \mathsf{sk}_M, \mathsf{pk}_M)$;
68 **internal operation** PrepareUpdate$(DID, s, \mathsf{sk}_M, \mathsf{pk}_M)$ **is**
69 $\quad$ **if** not $\mathsf{pk}_M \in s.\mathcal{M}_{DID}$ **then return** ;
70 $\quad r, \mathsf{sk}_r \leftarrow Auth.\mathsf{Keygen}()$;
71 $\quad s.r \leftarrow r$;
72 $\quad \pi_r \leftarrow Auth.\mathsf{Prove}(r, \mathsf{sk}_r)$;
73 $\quad \pi_{\text{UpdateAuth}} \leftarrow Auth.\mathsf{Prove}(\mathsf{pk}_M, \mathsf{sk}_M)$;
74 $\quad$ Send $UpdateRequest(DID, \pi_{\text{UpdateAuth}}, s, \mathsf{pk}_M, \pi_r)$ to a trusted correct process in Π or to t processes in Π;
75 **when** ResultUpdate(DID, res, σ, s) **is** received **do**
76 $\quad$ **if** $res =$ `true` **then Trigger** UpdateSuccess(σ, s);
77 $\quad$ **else**
78 $\quad\quad$ **Trigger** UpdateFailure()
79 **when** $DIDCreated(DID, \mathsf{pk})$ **is** received **do**
80 $\quad$ **Trigger** DIDCreated(DID, pk)
81 **when** $DIDCreationFailure(n, \mathsf{pk})$ **is** received **do**
82 $\quad$ **Trigger** DIDCreationFailure(DID, pk).

Algorithm 19: Ledger operations of a PPfDIMS (code for a device $d_i \notin \Pi$ acting as an issuer or a verifier).

init: $Obj_i \leftarrow$ a dictionnary that records the different Updated callbacks, initially empty;
$To\_Update_i \leftarrow$ A set initialized at $\emptyset$;
$Pending\_Names_i \leftarrow$ A set initialized at emptyset.;
when $UpdateRequest(DID, \pi_{\text{UpdateAuth}}, s, \mathsf{pk}, \pi_r)$ **is** received **from** p_j **do**
  Wait until $Short\_Naming$.Success$(DID, \star, \star)$ or HC_Naming.Success$(DID, \star, \star)$ is received ;
  if not ($Auth$.Verify$(\mathsf{pk}, \pi_{\text{UpdateAuth}})$ and $\mathsf{pk} \in Obj_i[DID].M_{DID}$ and $Auth$.Verify$(\pi_r, s.r)$) **then return** ;
  $To\_Update_i \leftarrow To\_Update_i \cup (DID, s, \pi_{\text{UpdateAuth}}, (\mathsf{pk}, \pi_r), p_j)$;
  AAP.Update$(DID, \pi_{\text{UpdateAuth}}, s, (\mathsf{pk}, \pi_r))$;
when AAP.Updated$(DID, \pi_{\text{UpdateAuth}}, \sigma, s)$ **is** received **do**
  $Obj_i[DID] \leftarrow s$;
  if $(DID, \pi'_{\text{UpdateAuth}}, s', (\mathsf{pk}', \pi'_r), p_j) \in To\_Update$ **then**
    if $s' = s$ **then**
      $To\_Update_i \leftarrow To\_Update \setminus (DID, s, \pi'_{\text{UpdateAuth}}, (\mathsf{pk}', \pi'_r), p_j)$;
      Send a ResultUpdate$(DID, \texttt{true}, \sigma, s)$ message to p_j;
    else
      if $s.sn = s'.sn$ **then**
        $To\_Update_i \leftarrow To\_Update \setminus (DID, s, \pi'_{\text{UpdateAuth}}, (\mathsf{pk}', \pi'_r), p_j)$;
        Send a ResultUpdate$(DID, \texttt{false}, \emptyset, \emptyset)$ message to p_j; ▷ *The* AAP.Update *opration was not successfull due to concurrency.*
when *RequestCreate*NHCDID(pk, π) **is** received **from** p_j **do**
  if not $Auth$.Verify(pk, π) **then return** ;
  $Pending\_Names_i \leftarrow Pending\_Names_i \cup (\mathsf{pk}, \pi, p_j)$;
  $Short\_Naming$.Claim(pk, π)
when $Short\_Naming$.Success(n, pk, π) **is** received **do**
  $Obj_i[n] \leftarrow$ CreateDocument(n, pk);
  if $(\mathsf{pk}, \pi, p_j) \in Pending\_Names_i$ **then**
    $Pending\_Names_i \leftarrow Pending\_Names_i \setminus (\mathsf{pk}, \pi, p_j)$;
    Send a $DIDCreated(n, \mathsf{pk})$ message to p_j;
when *RequestCreate*HCDID(n, pk, π) **is** received **from** p_j **do**
  if not $Auth$.Verify(pk, π) **then return** ;
  $Pending\_Names_i \leftarrow Pending\_Names_i \cup (n, \mathsf{pk}, \pi, p_j)$;
  HC_Naming.Claim(n, pk, π)
when HC_Naming.Success(n, pk, π) **is** received **do**
  $Obj_i[n] \leftarrow$ CreateDocument(n, pk);
  if $(n, \mathsf{pk}', \pi', p_j) \in Pending\_Names_i$ **then**
    $Pending\_Names_i \leftarrow Pending\_Names_i \setminus (n, \mathsf{pk}', \pi', p_j)$;
    if $\mathsf{pk}' = \mathsf{pk}$ and $\pi' = \pi$ **then**
      Send a $DIDCreated(n, \mathsf{pk})$ message to p_j;
    else
      Send a $DIDCreationFailure(n, \mathsf{pk})$ message to p_j.

Algorithm 20: Ledger operations of a PPfDIMS, code for $p_i \in \Pi$.

operation $\mathsf{Issue}(M, C, DID, \pi)$ **is**
  if π is not valid **then return** ;
  $r_{\text{revoc}} \leftarrow_{\$} \mathbb{Z}$;
  $vc \leftarrow AC.\mathsf{Sign}(pp, (M \cup \{DID, r_{\text{revoc}}\}), C, sk_I)$;
  return (vc, r_{revoc}).

Algorithm 21: Credential operations for an issuer I.

operation $\mathsf{Present}(V, vc, M, DID, r_{\text{revoc}}, \mathsf{pk}_I, T_V, \mathsf{aux}_{\mathcal{T}_\mathcal{V}}, st, r, c, s, \sigma)$ **is**
  $(\mathsf{pk}'_I, \pi_{\mathsf{pk}'_I}, vc', \pi_{vc'}) \leftarrow AC.\mathsf{Randomize}(pp, \mathsf{pk}_I, T_V, \sigma, \mathsf{aux}_\mathcal{S}, M, s, r, c)$; ▷*Randomizes the credential to enable unlinkability, selective disclosure and issuer indistinguishability.*
  $C_{DID} \leftarrow \mathsf{Commit}(DID)$;
  $C_{r_{\text{revoc}}} \leftarrow \mathsf{Commit}(r_{\text{revoc}})$;
  $\pi_{\text{auth}} = Auth.\mathsf{Prove}(\mathsf{pk}_d, \mathsf{sk}_d)$ $nonce = \mathsf{H}(vc'||\pi_{vc'}||C_{DID}||C_{r_{\text{revoc}}}||\pi_{\text{EqualAndRev}}||\pi_{\text{auth}}||st)$;
  $AAP.\mathsf{Prove}_{\{V,d\}}(s, \sigma, nonce, \pi_{\text{auth}}, \mathsf{aux})$;
  Wait until $AAP.\mathsf{Authorized}_{\{V,d\}}(C_s, nonce, Prove\_Obj)$ is received; ▷*Proves authentication and waits for the verifier to receive the notification.*
  $rev\_List \leftarrow \mathsf{ListRevocation}(Prove\_Obj)$;
  $\pi_{\text{EqualAndRev}} \leftarrow ZKP.\mathsf{Prove}(\{(r_{\text{revoc}}, DID) : C_{r_{\text{revoc}}} = \mathsf{Commit}(r_{\text{revoc}}) \wedge C_{DID} = \mathsf{Commit}(DID) \wedge r_{\text{revoc}} \in M \wedge C_{DID} \in M \wedge r_{\text{revoc}} \notin rev\_List\})$; ▷*Proves that the commitment given are commitments to the DID and the revoking number associated to the credential, and that the credential is not revoked.*
  return $vp = (vc', \mathsf{pk}'_I, \pi_{\mathsf{pk}'_I}, \pi_{vc'}, C_{DID}, C_{r_{\text{revoc}}}, \pi_{\text{EqualAndRev}}, \pi_{\text{auth}}, st)$.

Algorithm 22: Credential operations for a user's device d.

operation $\mathsf{Verify}(vp)$ **is**
  $(vc', ipk', \pi_{ipk'}, \pi_{vc'}, C_{DID}, C_{r_{\text{revoc}}}, \pi_{\text{EqualAndRev}}, \pi_{\text{auth}}, st) = vp$;
  $nonce = \mathsf{H}(vc'||\pi_{vc'}||C_{DID}||C_{r_{\text{revoc}}}||\pi_{\text{EqualAndRev}}||\pi_{\text{auth}}||st)$;
  Wait until $AAP.\mathsf{Authorized}_{\{V,d\}}(C_s, nonce, Prove\_Obj)$ is received; ▷*Verifies authentication.*
  if not $Auth.\mathsf{verify}(\pi_{\text{auth}})$ **then return** 0;
  if not $AC.\mathsf{Verify}(pp, \mathsf{aux}_{T_V}, \mathsf{pk}'_I, \pi_{\mathsf{pk}'_I}, vc', M, \pi_{vc'}, st)$ **then return** 0;
  $rev\_List \leftarrow \mathsf{ListRevocation}(Prove\_Obj)$;
  if not $ZKP.\mathsf{Verify}(\pi_{\text{EqualAndRev}}, st)$ **then return** 0;
  return 1.

Algorithm 23: Credential operations for a verifier V.

Chapter 11

A step back on political and philosophical implications of PPfDIMSs

This chapter aims to take a step back on Identity Management Systems from a political and philosophical perspective. It results from personal reflection and does not reflect the thoughts of the co-authors who participated in this thesis or the organisms that financed it.

This thesis provides numerous tools to implement privacy-preserving and fully distributed IMS. However, an IMS is more than just a technical construction. It aims at providing a tool for individuals to prove their identity elements. This type of tool does not need to be digitalized to work. In our everyday lives, we already use national identity cards, passports, or driving licenses to prove our identity. Therefore, the question arises: Why should we use a digitalized IMS?

The answer to this question can seem simple: We want to efficiently and securely prove our identity on the Internet. However, this answer needs to be completed, as many IMSs also aim to prove an individual's identity in the real world. We have seen this in Europe with the COVID-19 vaccine certificate. This use case will also be extended by the eIDAS 2.0 regulation [178]. The problem with online and offline authentication of individuals is that it uses highly sensitive data. This data can be used for mass surveillance and/or targeted surveillance of citizens. Therefore, special considerations must be taken when managing individuals' identity elements.

The first obvious concern with digital identity is that it increases the potential surface of attacks on individual identity. Indeed, like any other digital system, implementations are always flawed to some extent. A security breach may exist that can be exploited to harm the system. In our context, and if digitalized IMSs are widely spread among citizens, a single breach may harm a large proportion of the population. Unlike other systems, however, IMSs already exist in a non-digitalized form. Therefore, this new paradigm will inevitably create new breaches for an already existing service. Hence, this digitalized form of IMS must be well-defined and limited to reduce the potential surface of attack. Furthermore, PPfDIMSs use many assumptions, like anonymous peer-to-peer communications. If those assumptions are not met or flawed, the user's privacy is at risk. Side channel reidentification attacks are a concern for any PPfDIMS, and it

is not sure that this issue can be addressed in a real-world and widely spread implementation.

The second concern is the difference between the privacy guarantees of a system advertised by a regulator and the actual privacy guarantees provided by the system. Even if we assume implementations are perfect, privacy preservation may not be perfect. Indeed, privacy clashes with some aspects that regulators, legislators, or private companies may require. For example, one of the aspects of PPfDIMSs we did not tackle in this thesis is accountability. Indeed, an organization may need to be able to revoke the privacy of individuals in specific cases. If such an accountability feature is implemented in the system, then privacy is no longer guaranteed by technical aspects but by organizational aspects. Therefore, individuals who think their privacy is preserved can act without consideration. It will lead them to present information they would not have presented if they did not think their privacy was at risk. Therefore, a PPfIMS may be more impactful than a non-privacy preserving system. Furthermore, some organizations will likely advertise that they use highly privacy-preserving technologies, *e.g.*, hidden issuer anonymous credentials, while they only use them for limited use cases. In the same way, as with accountability, users will think their privacy is preserved, whereas it is not. Therefore, they will disclose information they did not intend to and would not have disclosed if they did not think their privacy was preserved.

The third concern is related to the rebound effect. If an efficient and easy-to-use IMS is deployed, services and individuals will be more likely to use it. They may use it more than a non-digitalized IMS (national identity card, passport). Therefore, even if the IMS protects the user's privacy, this user will use it more than a regular, non-digitalized IMS. Thus, its personal information can be recorded at more services. Therefore, each user's fingerprint may increase. Related to the previous point, if users think their privacy is preserved, they may increase this phenomenon. Therefore, even if we assume high privacy guarantees for each presentation, the user's presentation will become easier to intersect, ultimately leading to reidentification attacks that may have a tremendous impact.

The fourth concern is a direct continuation of the previous one. As stated, if an efficient digital IMS is released and regulatory power passes legislation to encourage its usage, it does not mean individuals will benefit from using it. It is even possible that people will not want to use it. However, it can be a real advantage for public and private services, as it may reduce the time to verify an individual's identity. Therefore, we may see a wide adoption of those technologies at the benefit of the services and the detriment of the individuals. Thus, each security and privacy concern raised in this chapter may be inevitable for individuals who may disagree with their usage. Furthermore, the added privacy (that may not be that high, as explained before) does not provide the user with additional liberty or political power. We can use one of Berlan's statements [179], which says there has been a shift in the definition of liberty between ancient Greece and modern democracies. He says that modern democracies' definition of liberty is the

privacy preservation of individuals in their private spaces. In contrast, liberty in ancient Greek democracies was about the ability to choose the rules for the city. With this definition, the privacy brought by PPfDIMS slightly increases individuals' liberty but does not increase its possibility of ruling. In fact, and as we discussed in the previous paragraphs, it even decreases one's ability to influence political decisions by increasing the ability of the state to control individual behaviors and by requesting more identity element presentations than what is required with regular IMSs. Furthermore, and due to potential privacy leaks that we already discussed, PPfDIMs may reduce privacy, liberty, and influence on political decisions at the same time. Thus, a non-regulated or badly-regulated PPfDIMS adoption would only negatively impact individuals. This fourth concern can be summarized by a quote from the preface to the third edition of "*La société du spectacle*" [180] : "*C'est cette volonté de modernisation et d'unification du spectacle, liée à tous les autres aspects de la simplification de la société, qui a conduit en 1989 la bureaucratie russe à se convertir soudain, comme un seul homme, à la présente idéologie de la démocratie : c'est à dire la liberté dictatoriale du Marché, tempérée par la reconnaissance des Droits de l'homme spectateur.*" The quote implies that individuals become spectators of a show they have no impact on, where simplification of the society, or "frictionless" systems, are adopted to reduce the burden of administrative work for citizens, at the cost of their political power.

A fifth concern is the accessibility of the deployed system. There will inevitably be individuals who will not be able to access a digitalized identity. Be it because they do not have a powerful enough smartphone, their condition prevents them from requesting the issuance of new digitalized credentials, or because they do not understand how to use their smartphone or computers. This concern is already a problem with digitalizing public and private services. In France, around 15% of the population do not have access to or do not know how to use the Internet [181]. Digitalization of identities will increase the boundary between those who can use the Internet and those who cannot. Even if interoperability is proposed between digitalized and regular IMSs, digitalization often reduces the regular access to services for those who do not know how to use the Internet. For example, in France, the digitalization of public services reduced the ease of access to public services for people who do not know how to use the Internet [182]. Therefore, we have to be careful when deploying a PPfDIMS. Hence, a PPfDIMS respecting Allen's recommendation, namely the access property, may reduce the accessibility to the system for specific individuals.

None of these concerns (except the first one) can be solved using technical solutions. There are internal issues that any digitalized IMS incorporates. Modifying how we present and prove our identity will inevitably impact society overall. To conclude, even if we exposed many points against the widespread of digitalized IMSs in this section, such systems may be interesting in specific, chosen, and well-delimited cases. Therefore, any implementation and deployment of such technology should be regulated by a democratic power. Indeed, and as stated earlier, a state or

a private company will always tend to abuse its power. Therefore, such deployment should be managed by an independent entity without interest in global or individual surveillance. To this end, the new eIDAS 2.0 regulation [178] and its application in national laws should be carefully studied to limit the weaknesses a digitalized IMS will inevitably bring.

Conclusion

This thesis formalized tools and algorithms to build Privacy Preserving fully Distributed Identity Management Systems (PPfDIMS), a new paradigm in Identity Management Systems (IMS) that focuses on user privacy and availability. An IMS aims to provide authorization and authentication means to users. Authorization is when a user presents proof of their identity to a verifier. Authentication is the process of proving that an identity proof describes a given user. A PPfDIMS aims to provide authorization and authentication while respecting principles stated by Christopher Allen [1]. To summarize those principles, an IMS should focus on user consent, privacy, and control over its identity elements.

In this thesis, we presented PPfDIMSs and the different problems that have to be solved to implement such a system, namely: a scheme to certify one's identity elements while preserving privacy, a way to publish information in a distributed manner, a distributed key management mechanism, a mechanism to authorize multiple devices to use credentials while enabling strong authentication guarantees, a revocation mechanism, and a naming algorithm.

We studied those problems formally. First, we highlighted a weakness in state-of-the-art anonymous credential schemes. Prior to our work (and concurrently with Bobolz et al. paper [26]), anonymous credential schemes made it mandatory to reveal the issuer of a credential for the verification process to be computable. We proposed a new scheme, Hidden Issuer Anonymous Credential, that makes it possible to hide the issuer of a credential while convincing the verifier that it trusts this issuer. This work was presented in Chapter 5. In this chapter, we presented a formal definition of this new type of anonymous credential, proposed an implementation of this scheme, and compared this scheme to the state of the art, theoretically and experimentally.

We then focussed on the distributed system part of PPfDIMSs. First, in Chapters 6 and 7 we formally defined the three main distributed objects used in PPfDIMSs and analyzed their consensus number: the AllowList, the DenyList, and the Namespace object (or Identifier System object). This analysis led to the following results. An AllowList does not need synchronization to be implemented, the Namespace object needs total synchronization of the processes of the system, and a DenyList needs synchronization between subset of the processes of the system to be implemented. We also highlighted special cases, like AllowList which support the Remove operation and which behave like a DenyList. Furthermore, Namespace objects that do not provide a specific property, namely the human-choosable property, probabilistically do not require synchronization to be implemented.

This formal study of the distributed objects used in PPfDIMSs was then used to build a

PPfDIMS with minimal synchronization requirements. We showed that the revocation of credentials of a PPfDIMS can be implemented using a DenyList. Therefore, processes that invoke the Append operation or revoke credentials do not have to use a consensus algorithm. Furthermore, other auxiliary features of a PPfDIMS that we highlighted can be implemented using an AllowList that supports the Append and the Remove operation. However, only authorized processes for a specific DID document can invoke such operations. A DID Document can only be modified by the devices of the individual who owns it. Moreover, two DID documents can be concurrently modified. Hence, only the devices belonging to a specific individual must synchronize to modify on-ledger DID documents.

In Chapter 8, we proposed a new consensus algorithm, the Cascading Consensus, built specifically for this purpose. This algorithm, based on the Context Adaptive Cooperation (CAC) abstraction, detects if concurrent operations are invoked and lets processes that invoke those operations synchronize and solve the contention. We also designed two naming algorithms based on the CAC abstraction. The first algorithm is meant for individuals who would likely not share their DIDs. This algorithm does not provide the human-choosable property and is solely based on the CAC abstraction. Contrary to other non-human-choosable naming algorithms, this algorithm makes it possible to reduce the entropy of names. The second is based on the Cascading Consensus algorithm and is used when the human-choosable property is desirable.

This thesis's developments are summarized in Chapter 10 in which we give a high-level implementation of a PPfDIMS that completely preserves users' privacy and proposes all the desirable properties that such a system should implement.

Perspectives and ongoing works

In this thesis, we focused on analyzing the different properties of a PPfDIMS and proposing solutions to implement them in a privacy-preserving manner with minimal synchronization requirements. However, some issues still need to be addressed.

First, in Chapter 5, we highlighted a weakness inherent to Hidden Issuer Anonymous Credential schemes. Indeed, the choice of the set of trusted issuers for a verifier can impact the privacy guarantees of such schemes. To solve this problem, we proposed a new framework, the checker, that would inform the user of the potential privacy risks by analyzing the set of trusted issuers of a verifier. The design and implementation of this checker is a research problem that requires in-depth analysis. This is an ongoing work.

Second, we proposed in Chapter 10 a high-level implementation of a PPfDIMS, but we did not implement it. It would be interesting to have an implementation of this framework to analyze its efficiency and to verify that the proposed framework is usable by individuals. Furthermore, except for the Hidden Issuer Anonymous Credential scheme, we did not implement the tools

presented in this thesis. This would be an interesting work validating the claimed efficiency gains and usability of the algorithms. This is an ongoing work.

Third, as stated in Chapter 2, we did not tackle the accountability problem. Accountability is the possibility for an authority to revoke a user's privacy in case of legitimate suspicions that this user misbehaved. We can see that this property of a PPfDIMS is in direct contradiction with privacy preservation, as an actor, usually a state, can revoke this property. Papers that addressed the trade-off between accountability have already been published [183, 35]. Thus, we preferred to focus on privacy preservation rather than accountability in this thesis. However, implementing a PPfDIMS that allows users to access state services would likely require such a feature.

Fourth, we did not consider biometrics as an authentication means. Indeed, biometrics features can be seen as the best authentication means, as they are transportable and directly related to the user. For example, if we need a recovery mechanism, then the multi-device framework presented in Chapter 9 would still be necessary. Furthermore, Biometric based identification requires special hardaware and is based on additional security assumptions. Finally, the two approaches can be complementary. A PPfDIMS can compose asymmetric cryptography-based authentication with biometric-based authentication, thus improving security.

Finally, a real impact study on the wide use of PPfDIMS is necessary before such a system is deployed. Indeed, we pointed out several points in Chapter 11 that may not be solvable, as they arise from the use of any identity management system. Hence, a sociological, philosophical, and political study should be conducted to know if the trade-off between the benefits of a PPfDIMS and the risks it creates is sufficiently secure for wide adoption of such systems. A recent paper [184] assesses the issues discussed in Chapter 11. They also study the reaction of users to "Self-Sovereign Identity" solutions. They conclude that PPfDIMS do not provide more control to users. Using this type of technology even seems to decrease the sensation of control for users.

BIBLIOGRAPHY

[1] C. Allen, *The Path To Self Sovereign Identity*, Publication Title: Physical Review Volume: 91, Apr. 2016.

[2] *Compatible Time-Sharing System*, en, Page Version ID: 1211080905, Feb. 2024, URL: https://en.wikipedia.org/w/index.php?title=Compatible_Time-Sharing_System&oldid=1211080905 (visited on 03/21/2024).

[3] MD Sadek Ferdous, Farida Chowdhury, and Madini O. Alassafi, *In Search of Self-Sovereign Identity Leveraging Blockchain Technology*, 2019, URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8776589 (visited on 04/09/2024).

[4] Matthew Davie, Dan Gisolfi, Daniel Hardman, John Jordan, Darrell O'Donnell, and Drummond Reed, « The Trust over IP Stack », *in*: *IEEE Communications Standards Magazine* 3.*4* (Dec. 2019), Conference Name: IEEE Communications Standards Magazine, pp. 46–51, ISSN: 2471-2833, DOI: 10.1109/MCOMSTD.001.1900029, URL: https://ieeexplore.ieee.org/document/9031548 (visited on 03/11/2024).

[5] W3C, *Verifiable Credentials Data Model v1.1*, 2022.

[6] David Chaum, « Security without Identification: Transaction Systems to Make Big Brother Obsolete », *in*: *Commun. ACM* 28.*10* (Oct. 1985), pp. 1030–1044, ISSN: 0001-0782, DOI: 10.1145/4372.4373.

[7] Christian Lundkvist, Rouven Heck, Joel Torstensson, Zac Mitton, and Michael Sena, « Uport: A platform for self-sovereign identity », *in*: *URL: https://whitepaper. uport. me/uPort_ whitepaper_DRAFT20170221. pdf* 128 (2017), p. 214.

[8] W3C Decentralized Identifier Working Group, *Decentralized Identifiers (DIDs) v1.0 Core architecture, data model, and representation*, tech. rep., Published: (https://www.w3.org/TR/did-core/), Aug. 2021.

[9] *DID Specification Registries*, URL: https://w3c.github.io/did-spec-registries/#did-methods (visited on 06/04/2024).

[10] W3C, *DIDComm Messaging v2.1 Editor's Draft*, 2023.

[11] *Veramo - A JavaScript Framework for Verifiable Data | Performant and modular APIs for Verifiable Data and SSI*, en, URL: https://veramo.io/ (visited on 06/05/2024).

[12] Andrew Tobin, « Sovrin: What Goes on the Ledger? », en, *in*: (2018).

[13] Jolocom, *Self Sovereign Identity $\ne$ Blockchain*, Dec. 2021, URL: https://web.archive.org/web/20231210204245/https://jolocom.io/blog/dezentrale-identitaten-not-blockchain-2/ (visited on 03/19/2024).

[14] *SSI-on-Blockchain is Objectively a Bad Thing*, en, July 2022, URL: https://weh.wtf/ssi.html (visited on 03/19/2024).

[15] Daniel Bosk, Davide Frey, Mathieu Gestin, and Guillaume Piolle, « Hidden Issuer Anonymous Credential », *in*: *Proc. Priv. Enhancing Technol.* 2022.*4* (2022), pp. 571–607.

[16] Wilcox Zooko, *Names: Decentralized, Secure, Human-Meaningful: Choose Two*, 2001, URL: https://pestilenz.org/~bauerm/names/distnames.html (visited on 04/16/2024).

[17] Aaron Swartz, *Squaring the Triangle: Secure, Decentralized, Human-Readable Names (Aaron Swartz's Raw Thought)*, 2011, URL: http://www.aaronsw.com/weblog/squarezooko (visited on 04/16/2024).

[18] Alexander Mühle, Andreas Grüner, Tatiana Gayvoronskaya, and Christoph Meinel, « A survey on essential components of a self-sovereign identity », *in*: *Computer Science Review* 30 (Nov. 2018), pp. 80–86, DOI: 10.1016/j.cosrev.2018.10.002.

[19] Hyperledger, *Aries DKMS*, en, 2019, URL: https://github.com/hyperledger/aries-rfcs/blob/main/concepts/0051-dkms/dkms-v4.md (visited on 03/19/2024).

[20] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis, « Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers », *in*: *NDSS*, The Internet Society, 2019.

[21] Davide Frey, Matthieu Gestin, and Michel Raynal, « The Synchronization Power (Consensus Number) of Access-Control Objects: the Case of AllowList and DenyList », *in*: *Proc. 37th Int'l Symposium on Distributed Computing (DISC'23)*, vol. 281, LIPICs, 2023, 21:1–21:23.

[22] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson, « Impossibility of Distributed Consensus with One Faulty Process », *in*: *J. ACM* 32 (1985), pp. 374–382.

[23] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi, « The Consensus Number of a Cryptocurrency », *in*: *Distributed Computing* 35 (2022), pp. 1–15.

[24] Alex Auvolat, Davide Frey, Michel Raynal, and François Taïani, « Money Transfer Made Simple: a Specification, a Generic Algorithm, and its Proof », *in*: *Bulletin of EATCS* 132 (2020), pp. 22–43.

[25] Gabriel Bracha and Sam Toueg, « Asynchronous Consensus and Broadcast Protocols », *in*: *J. ACM* 32.*4* (1985), pp. 824–840.

[26] Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin, « Issuer-Hiding Attribute-Based Credentials », *in*: *Cryptology and Network Security*, ed. by Mauro Conti, Marc Stevens, and Stephan Krenn, Cham: Springer International Publishing, 2021, pp. 158–178, ISBN: 978-3-030-92548-2.

[27] Maurice Herlihy, « Wait-Free Synchronization », *in*: *ACM Trans. Program. Lang. Syst.* 13.*1* (1991), pp. 124–149.

[28] Ronald L. Rivest, Adi Shamir, and Yael Tauman, « How to Leak a Secret », *in*: *Advances in Cryptology — ASIACRYPT 2001*, ed. by Colin Boyd, Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565.

[29] David Chaum and Eugène van Heyst, « Group Signatures », en, *in*: *Advances in Cryptology — EUROCRYPT '91*, ed. by Donald W. Davies, Berlin, Heidelberg: Springer, 1991, pp. 257–265, ISBN: 978-3-540-46416-7, DOI: 10.1007/3-540-46416-6_22.

[30] Aisling Connolly, Pascal Lafourcade, and Octavio Perez Kempner, « Improved Constructions of Anonymous Credentials from Structure-Preserving Signatures on Equivalence Classes », en, *in*: *Public-Key Cryptography – PKC 2022*, ed. by Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, Cham: Springer International Publishing, 2022, pp. 409–438, ISBN: 978-3-030-97121-2, DOI: 10.1007/978-3-030-97121-2_15.

[31] Olivier Sanders and Jacques Traoré, « Compact Issuer-Hiding Authentication, Application to Anonymous Credential », *in*: *Proceedings on Privacy Enhancing Technologies* (2024), ISSN: 2299-0984, URL: https://petsymposium.org/popets/2024/popets-2024-0097.php (visited on 07/02/2024).

[32] Elaine Barker, *Recommendation for Key Management: Part 1 – General*, en, tech. rep. NIST Special Publication (SP) 800-57 Part 1 Rev. 5, National Institute of Standards and Technology, May 2020, DOI: 10.6028/NIST.SP.800-57pt1r5, URL: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final (visited on 06/26/2024).

[33] AnonCred, *AnonCreds Specification*, URL: https://hyperledger.github.io/anoncreds-spec/#requirements-notation-and-conventions (visited on 03/19/2024).

[34] Jan Camenisch and Anna Lysyanskaya, « An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation », *in*: *Advances in Cryptology — EUROCRYPT 2001*, ed. by Birgit Pfitzmann, Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 93–118.

[35] Joakim Brorsson, Bernardo David, Lorenzo Gentile, Elena Pagnin, and Paul Stankovski Wagner, « PAPR: Publicly Auditable Privacy Revocation for Anonymous Credentials », en, *in*: *Topics in Cryptology – CT-RSA 2023*, ed. by Mike Rosulek, Cham: Springer International Publishing, 2023, pp. 163–190, ISBN: 978-3-031-30872-7, DOI: 10.1007/978-3-031-30872-7_7.

[36] D W Chadwick, « Understanding X.500 », *in*: (1994), URL: https://sec.cs.kent.ac.uk/x500book/ (visited on 03/14/2024).

[37] *The GNU Privacy Handbook*, URL: https://www.gnupg.org/gph/en/manual.html#AEN385 (visited on 03/18/2024).

[38] Carl M. Ellison, « Establishing identity without certification authorities », *in*: *6th USENIX security symposium* (1996), URL: https://www.usenix.org/legacy/publications/library/proceedings/sec96/full_papers/ellison/index.html (visited on 03/13/2024).

[39] Kim Cameron, « The laws of identity », *in*: *Microsoft Corp* 12 (2005), pp. 8–11.

[40] Andrew Orlowski, *Do Androids Dream of Electric Single Sign-Ons?*, en, 2001, URL: https://www.theregister.com/2001/10/24/do_androids_dream_of_electric/ (visited on 03/18/2024).

[41] John Markoff, « TECHNOLOGY; Microsoft Has Quietly Shelved Its Internet 'Persona' Service », en-US, *in*: *The New York Times* (Apr. 2002), ISSN: 0362-4331, URL: https://www.nytimes.com/2002/04/11/business/technology-microsoft-has-quietly-shelved-its-internet-persona-service.html (visited on 03/18/2024).

[42] Mansour Alsaleh and Carlisle Adams, « Enhancing Consumer Privacy in the Liberty Alliance Identity Federation and Web Services Frameworks », en, *in*: *Privacy Enhancing Technologies*, ed. by George Danezis and Philippe Golle, Berlin, Heidelberg: Springer, 2006, pp. 59–77, ISBN: 978-3-540-68793-1, DOI: 10.1007/11957454_4.

[43] Marc Le maitre, « The identity web », *in*: (2002).

[44] Ken Jordan, Jan Hauser, and Steven Foster, « The Augmented Social Network: Building identity and trust into the next-generation Internet », en, *in*: *First Monday* 8.*8* (Aug. 2003), ISSN: 13960466, DOI: 10.5210/fm.v8i8.1068, URL: http://journals.uic.edu/ojs/index.php/fm/article/view/1068 (visited on 03/18/2024).

[45] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and M. Chuck, *OpenID Connect Core 1.0*, tech. rep., Nov. 2014, URL: https://openid.net/specs/openid-connect-core-1_0.html#toc.

[46] *OpenID*, en, Page Version ID: 1221433932, Apr. 2024, URL: https://en.wikipedia.org/w/index.php?title=OpenID&oldid=1221433932 (visited on 06/06/2024).

[47] Cyrille CHAUSSON, *France Connect: an ID federation system to simplify administrative processes*, 2015.

[48] Microsoft Corporation, *Microsoft's Vision for an Identity Metasystem*, en-us, 2005, URL: https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/ms996422(v=msdn.10) (visited on 03/11/2024).

[49] kexugit, *Security Briefs: A First Look at InfoCard*, en-us, Oct. 2006, URL: https://learn.microsoft.com/en-us/archive/msdn-magazine/2006/april/security-briefs-a-first-look-at-infocard (visited on 06/06/2024).

[50] *eIDAS regulation*, 2016.

[51] David W. Chadwick, « Federated Identity Management », en, *in*: *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures*, ed. by Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, Berlin, Heidelberg: Springer, 2009, pp. 96–120, ISBN: 978-3-642-03829-7, DOI: 10.1007/978-3-642-03829-7_3, URL: https://doi.org/10.1007/978-3-642-03829-7_3 (visited on 06/03/2024).

[52] Drummond Reed, Jason Law, and Daniel Hardman, « The Technical Foundations of Sovrin », en, *in*: (2016).

[53] Michael Boyd, *Hyperledger Indy Plenum's overview*, Published: (https://hyperledger-indy.readthedocs.io/projects/plenum/en/latest/main.html), Nov. 2018.

[54] *Swiss digital identity law approved by parliament lower house | Biometric Update*, en-US, Mar. 2024, URL: https://www.biometricupdate.com/202403/swiss-digital-identity-law-approved-by-parliament-lower-house (visited on 03/20/2024).

[55] *Proposition de RÈGLEMENT DU PARLEMENT EUROPÉEN ET DU CONSEIL modifiant le règlement (UE) nº 910/2014 en ce qui concerne l'établissement d'un cadre européen relatif à une identité numérique*, fr, 2021, URL: https://eur-lex.europa.eu/legal-content/FR/TXT/?uri=CELEX%3A52021PC0281 (visited on 06/03/2024).

[56] Satoshi Nakamoto, *Bitcoin: A Peer-toPeer Electronic Cash System*, tech. rep., Mar. 2009, URL: https://bitcoin.org/bitcoin.pdf.

[57] Vitalik Buterin, *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.* 2014.

[58] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza, « Zerocash: Decentralized Anonymous Payments from Bitcoin », *in*: *2014 IEEE Symposium on Security and Privacy*, ISSN: 2375-1207, May 2014, pp. 459–474, DOI: 10.1109/SP.2014.36.

[59] *All cryptocurrencies - Coin Market Capitalization list of all Crypto Currencies and prices | CryptoRank.io*, en, URL: https://cryptorank.io/all-coins-list?page=25 (visited on 06/04/2024).

[60] *Jolocom white paper v2.1, A Decentralized, Open Soure Solution For Digital Identity and Access Management*, tech. rep., Dec. 2019.

[61] *Self-Sovereign Identity for more Freedom and Privacy*, en-US, URL: https://selfkey.org/ (visited on 06/04/2024).

[62] Bob Reid, *EverID Whitepaper*, 2018.

[63] Karl Marx, *Das Kapital. Kritik der politischen Ökonomie. Buch I: Der Produktionsprocess des Kapitals.* Otto Meissner, 1867.

[64] David Graeber, *Debt: The first five thousand years*, 2011.

[65] *SnailMail DID method Specification*, URL: https://xn--3n8h.amy.gy/ (visited on 06/05/2024).

[66] *Peer DID Method Specification*, URL: https://identity.foundation/peer-did-method-spec/index.html (visited on 06/05/2024).

[67] *Papers/whitepapers/KERI_WP_2.x.web.pdf at master · SmithSamuelM/Papers*, en, URL: https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/KERI_WP_2.x.web.pdf (visited on 06/04/2024).

[68] *GNUnet*, URL: https://www.gnunet.org/en/gns.html (visited on 06/04/2024).

[69] *Distributed hash table*, en, Page Version ID: 1222986598, May 2024, URL: https://en.wikipedia.org/w/index.php?title=Distributed_hash_table&oldid=1222986598 (visited on 06/05/2024).

[70] *SSB DID Method (did:ssb) v0.1*, URL: https://viewer.scuttlebot.io/&5Bne/slGKH/i1361qemVlNBElWInSUfntlWvMXaD4M4=.sha256?hl=zQmdh4Ya6WasmjnS4UMn5ot6k5tbCypy1oyhhdJ6yB6MjfT (visited on 06/05/2024).

[71] *InterPlanetary File System*, en, Page Version ID: 1224580830, May 2024, URL: https://en.wikipedia.org/w/index.php?title=InterPlanetary_File_System&oldid=1224580830 (visited on 06/04/2024).

[72] *Holochain | Distributed app framework with P2P networking*, en, URL: https://holochain.org/ (visited on 06/04/2024).

[73] Gabriel Bracha, « Asynchronous Byzantine agreement protocols », *in*: *Information and Computation* 75 (1987), pp. 130–143.

[74] Maurice P Herlihy and Jeannette M Wing, « Linearizability: A correctness condition for concurrent objects », *in*: *ACM Transactions on Programming Languages and Systems* 12.*3* (1990), pp. 463–492.

[75] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit, « Atomic Snapshots of Shared Memory », *in*: *JACM* 40.*4* (Sept. 1993), pp. 873–890, ISSN: 0004-5411, DOI: 10.1145/153724.153741, URL: https://doi.org/10.1145/153724.153741.

[76] Steven D. Galbraith, *Mathematics of Public Key Cryptography*, Cambridge University Press, 2012, DOI: 10.1017/CBO9781139012843.

[77] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart, « Pairings for cryptographers », *in*: *Discrete Applied Mathematics* 156.*16* (2008), pp. 3113–3121, ISSN: 0166-218X.

[78] Jan Camenisch and Markus Stadler, « Efficient group signature schemes for large groups », *in*: *Advances in Cryptology — CRYPTO '97*, ed. by Burton S. Kaliski, Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 410–424.

[79] David Chaum and Torben Pryds Pedersen, « Wallet Databases with Observers », *in*: *Advances in Cryptology — CRYPTO' 92*, ed. by Ernest F. Brickell, Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 89–105, ISBN: 978-3-540-48071-6.

[80] Amos Fiat and Adi Shamir, « How To Prove Yourself: Practical Solutions to Identification and Signature Problems », *in*: *Advances in Cryptology — CRYPTO' 86*, ed. by Andrew M. Odlyzko, Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.

[81] David Bernhard, Olivier Pereira, and Bogdan Warinschi, « How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios », *in*: *Advances in Cryptology – ASIACRYPT 2012*, ed. by Xiaoyun Wang and Kazue Sako, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 626–643.

[82] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest, « A Digital Signature Scheme Secure Against Adaptive Chosen Message Attack », *in*: *Discrete Algorithms and Complexity*, ed. by David S. Johnson, Takao Nishizeki, Akihiro Nozaki, and Herbert S. Wilf, Academic Press, 1987, pp. 287–310, ISBN: 978-0-12-386870-1.

[83] David Chaum, « Blind Signatures for Untraceable Payments », *in*: *Advances in Cryptology*, 1983, pp. 199–203, ISBN: 978-1-4757-0602-4.

[84] David Pointcheval and Olivier Sanders, « Short Randomizable Signatures », *in*: *Topics in Cryptology - CT-RSA 2016*, ed. by Kazue Sako, Cham: Springer International Publishing, 2016, pp. 111–126, ISBN: 978-3-319-29485-8.

[85] Jan Camenisch and Anna Lysyanskaya, « A Signature Scheme with Efficient Protocols », *in*: *SCN*, vol. 2576, Lecture Notes in Computer Science, Springer, 2002, pp. 268–289.

[86] Jens Groth and Amit Sahai, « Efficient Non-interactive Proof Systems for Bilinear Groups », *in*: *Electron. Colloquium Comput. Complex.* TR07-053 (2007).

[87] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung, « How to share a function securely », *in*: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, event-place: Montreal, Quebec, Canada, New York, NY, USA: Association for Computing Machinery, 1994, pp. 522–533, ISBN: 0-89791-663-8, DOI: 10.1145/195058.195405, URL: https://doi.org/10.1145/195058.195405.

[88] Adi Shamir, « How to Share a Secret », *in*: *Commun. ACM* 22.*11* (Nov. 1979), Place: New York, NY, USA Publisher: Association for Computing Machinery, pp. 612–613, ISSN: 0001-0782, DOI: 10.1145/359168.359176, URL: https://doi.org/10.1145/359168.359176.

[89] Fré Vercauteren, Benger, Bernhard, Catalano, Charlemagne, Conti, Cubaleska, Fernando, Fiore, Galbraith, Galindo, Hermans, Iovino Vincenzo, Jager, Kohlweiss, Libert, Lindner, Loehr, Lynch, Moloney, Ouafi, Pinkas, Polach, Di Raimondo, Rückert, Schneider, Singh, Smart, Stam, Vercauteren, Villar Santos, and Williams, *Final Report on Main Computational Assumptions in Cryptography*, tech. rep., Jan. 2013.

[90] Zhiyi Zhang, Michał Król, Alberto Sonnino, Lixia Zhang, and Etienne Rivière, « EL PASSO: Efficient and Lightweight Privacy-preserving Single Sign On », *in*: *Proceedings on Privacy Enhancing Technologies*, vol. 2, 2021, pp. 70–87, DOI: 10.2478/popets-2021-0018.

[91] The Sovrin foundation, *Sovrin: A Protocol and Token for Self-Sovereign Identity and Decentralized Trust*, tech. rep., Jan. 2018.

[92] Md. Sadek Ferdous, Gethin Norman, and Ron Poet, « Mathematical Modelling of Identity, Identity Management and Other Related Topics », *in*: *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, event-place: Glasgow, Scotland, UK, New York, NY, USA: Association for Computing Machinery, 2014, pp. 9–16, ISBN: 978-1-4503-3033-6, DOI: 10.1145/2659651.2659729, URL: https://doi.org/10.1145/2659651.2659729.

[93] *SelfKey white paper*, tech. rep., Sept. 2017.

[94] Jan Camenisch and Anna Lysyanskaya, « Signature Schemes and Anonymous Credentials from Bilinear Maps », *in*: *Advances in Cryptology – CRYPTO 2004*, ed. by Matt Franklin, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 56–72, ISBN: 978-3-540-28628-8.

[95] Foteini Baldimtsi and Anna Lysyanskaya, « Anonymous credentials light », *in*: *Proceedings of the ACM Conference on Computer and Communications Security*, Nov. 2013, pp. 1087–1098, DOI: 10.1145/2508859.2516687.

[96] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford, « Scalable Bias-Resistant Distributed Randomness », *in*: *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 444–460, DOI: 10.1109/SP.2017.45.

[97] Serguei Popov, « On a decentralized trustless pseudo-random number generation algorithm », *in*: *Journal of Mathematical Cryptology* 11.*1* (2017), pp. 37–43, DOI: doi:10.1515/jmc-2016-0019, URL: https://doi.org/10.1515/jmc-2016-0019.

[98] Jan Camenisch and Anna Lysyanskaya, « Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials », *in*: *Advances in Cryptology — CRYPTO 2002*, ed. by Moti Yung, Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 61–76, ISBN: 978-3-540-45708-4.

[99] Christina Garman, Matthew Green, and Ian Miers, *Decentralized Anonymous Credentials*, Published: Cryptology ePrint Archive, Report 2013/622, 2013.

[100] Yevgeniy Dodis, Aggelos Kiayias, Antonio Nicolosi, and Victor Shoup, « Anonymous Identification in Ad Hoc Groups », *in*: *Advances in Cryptology - EUROCRYPT 2004*, ed. by Christian Cachin and Jan L. Camenisch, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 609–626, ISBN: 978-3-540-24676-3.

[101] Lidong Chen, « Oblivious signatures », *in*: *Computer Security — ESORICS 94*, ed. by Dieter Gollmann, Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 161–172, ISBN: 978-3-540-49034-0.

[102] Raylin Tso, « A new way to generate a ring: Universal ring signature », *in*: *Computers and Mathematics with Applications* 65.*9* (2013), pp. 1350–1359, ISSN: 0898-1221.

[103] Jens Groth, « Efficient Fully Structure-Preserving Signatures for Large Messages », *in*: *Advances in Cryptology – ASIACRYPT 2015*, ed. by Tetsu Iwata and Jung Hee Cheon, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 239–259, ISBN: 978-3-662-48797-6.

[104] Josh Benaloh and Michael de Mare, « One-Way Accumulators: A Decentralized Alternative to Digital Signatures », *in*: *EUROCRYPT '93*, 1994, pp. 274–285, ISBN: 978-3-540-48285-7.

[105] Niko Barić and Birgit Pfitzmann, « Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees », *in*: *Advances in Cryptology — EUROCRYPT '97*, ed. by Walter Fumy, Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 480–494, ISBN: 978-3-540-69053-5.

[106] Lan Nguyen, « Accumulators from Bilinear Pairings and Applications », *in*: *Topics in Cryptology – CT-RSA 2005*, ed. by Alfred Menezes, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 275–292, ISBN: 978-3-540-30574-3.

[107] Daniel Benarroch, Matteo Campanelli, Dario Fiore, and Dimitris Kolonelos, « Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular », *in*: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 1255.

[108] Jan Camenisch, Rafik Chaabouni, and abhi shelat abhi, « Efficient Protocols for Set Membership and Range Proofs », *in*: *Advances in Cryptology - ASIACRYPT 2008*, ed. by Josef Pieprzyk, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 234–252, ISBN: 978-3-540-89255-7.

[109] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott, « Constructing Elliptic Curves with Prescribed Embedding Degrees », *in*: *Security in Communication Networks*, ed. by Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 257–267, ISBN: 978-3-540-36413-9.

[110] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente, « An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials », *in*: *Public Key Cryptography – PKC 2009*, ed. by Stanis\law Jarecki and Gene Tsudik, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 481–500, ISBN: 978-3-642-00468-1.

[111] Omid Mir, Balthazar Bauer, Scott Griffy, Anna Lysyanskaya, and Daniel Slamanig, « Aggregate Signatures with Versatile Randomization and Issuer-Hiding Multi-Authority Anonymous Credentials », *in*: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 30–44, ISBN: 9798400700507, DOI: 10.1145/3576915.3623203, URL: https://doi.org/10.1145/3576915.3623203 (visited on 07/02/2024).

[112] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi, « The Consensus Number of a Cryptocurrency », *in*: PODC '19, 2019, pp. 307–316, ISBN: 978-1-4503-6217-7, DOI: 10.1145/3293611.3331589, URL: https://doi.org/10.1145/3293611.3331589.

[113] Orestis Alpos, Christian Cachin, Giorgia Azzurra Marson, and Luca Zanolini, « On the Synchronization Power of Token Smart Contracts », *in*: *Proc. 41st IEEE Int'l Conference on Distributed Computing Systems (ICDCS'21)*, IEEE, 2021, pp. 640–651.

[114] Gaby G. Dagher, Praneeth Babu Marella, Matea Milojkovic, and Jordan Mohler, « BroncoVote: Secure Voting System using Ethereum's Blockchain », *in*: *ICISSP*, 2018.

[115] Nick Johnson, Shayan Escandari, and Brantly Millegan, *Ethereum Name Service Documentation*, Published: (https://docs.ens.domains/).

[116] Harry A. Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan, « An Empirical Study of Namecoin and Lessons for Decentralized Namespace Design », *in*: *Workshop on the Economics of Information Security*, 2015.

[117] Miguel Castro and Barbara Liskov, « Practical Byzantine Fault Tolerance », *in*: OSDI '99 also ACM TOCS 2002, Place: New Orleans, Louisiana, USA, 1999, pp. 173–186, ISBN: 1-880446-39-1.

[118] Leslie Lamport, « The Part-Time Parliament », *in*: *ACM Trans. Comput. Syst.* 16.*2* (1998), pp. 133–169.

[119] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma, « RBFT: Redundant Byzantine Fault Tolerance », *in*: *IEEE 33rd International Conference on Distributed Computing Systems*, 2013, pp. 297–306, DOI: 10.1109/ICDCS.2013.53.

[120] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos, « Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular », *in*: *Financial Cryptography and Data Security*, Springer Berlin Heidelberg, 2021.

[121] Keren Censor-Hillel, Erez Petrank, and Shahar Timnat, « Help! », *in*: PODC '15, 2015, pp. 241–250, ISBN: 978-1-4503-3617-8, DOI: 10.1145/2767386.2767415, URL: https://doi.org/10.1145/2767386.2767415.

[122] Leslie Lamport, « Time, Clocks and the Ordering of Events in a Distributed System », *in*: *Communications of the ACM 21, (7), 558-565* (July 1978), URL: https://www.microsoft.com/en-us/research/publication/time-clocks-ordering-events-distributed-system/.

[123] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han, « SoK: Sharding on Blockchain », *in*: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 41–61, ISBN: 978-1-4503-6732-5, DOI: 10.1145/3318041.3355457.

[124] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, and Rüdiger Reischuk, « Renaming in an asynchronous environment », *in*: *Journal of the ACM (JACM)* 37.*3* (1990), Publisher: ACM New York, NY, USA, pp. 524–548.

[125] Armando Castañeda and Michel Raynal, « On the Consensus Number of Non-adaptive Perfect Renaming », *in*: *Networked Systems: First International Conference, NETYS 2013, Marrakech, Morocco, May 2-4, 2013, Revised Selected Papers*, Springer, 2013, pp. 1–12.

[126] *Domain Name System Security Extensions*, en, Page Version ID: 1226085388, May 2024, URL: https://en.wikipedia.org/w/index.php?title=Domain_Name_System_Security_Extensions&oldid=1226085388 (visited on 06/12/2024).

[127] Buterin Vitalic and Gavin Wood, *Ethereum whitepaper*, 2014.

[128] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui, « (Almost) All Objects Are Universal in Message Passing Systems », en, *in*: *Distributed Computing*, ed. by Pierre Fraigniaud, Berlin, Heidelberg: Springer, 2005, pp. 184–198, ISBN: 978-3-540-32075-3, DOI: 10.1007/11561927_15.

[129] Md Sadek Ferdous, Audun Jøsang, Kuldeep Singh, and Ravishankar Borgaonkar, « Security Usability of Petname Systems », *in*: *Identity and Privacy in the Internet Age*, 2009, pp. 44–59.

[130] Timothé Albouy, Davide Frey, Mathieu Gestin, Michel Raynal, and François Taïani, *Context Adaptive Cooperation*, _eprint: 2311.08776, 2023, URL: https://arxiv.org/abs/2311.08776.

[131] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease, « The Byzantine Generals Problem », *in*: *ACM Trans. Program. Lang. Syst.* 4.*3* (1982), pp. 382–401.

[132] Marshall Pease, Robert Shostak, and Leslie Lamport, « Reaching Agreement in the Presence of Faults », *in*: *J. ACM* 27 (1980), pp. 228–234.

[133] Leslie Lamport, « Fast Paxos », *in*: *Distributed Computing* 19 (2006), pp. 79–103.

[134] Hagit Attiya and Jennifer L. Welch, *Distributed computing - fundamentals, simulations, and advanced topics*, 2nd ed., Wiley, 2004.

[135] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues, *Reliable and secure distributed programming*, 1st ed., Springer, 2011.

[136] Michel Raynal, *Fault-tolerant message-passing distributed systems: an algorithmic approach*, Springer, 2018.

[137] Bracha and Toueg, « Resilient Consensus Protocols », *in*: *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC'83)*, 1983, pp. 12–26.

[138] Z. Bouzid, A. Mostéfaoui, and M. Raynal, « Minimal Synchrony for Byzantine Consensus », *in*: *Proc. 34th ACM Symposium on Principles of Distributed Computing (PODC'15)*, ACM, 2015, pp. 461–470.

[139] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg, « The Weakest Failure Detector for Solving Consensus », *in*: *J. ACM* 43.*4* (1996), pp. 685–722.

[140] Damien Imbs and Michel Raynal, « Trading off \empht-Resilience for Efficiency in Asynchronous Byzantine Reliable Broadcast », *in*: *Parallel Processing Letters* 26 (2016), 1650017:1–1650017:8.

[141] Achour Mostéfaoui and Michel Raynal, « Low cost consensus-based Atomic Broadcast », *in*: *Proc. 2000 Pacific Rim International Symposium on Dependable Computing (PRDC'00)*, IEEE Computer Society, 2000, pp. 45–52.

[142] Yee Jiun Song and Robbert van Renesse, « Bosco: One-Step Byzantine Asynchronous Consensus », *in*: *Proc. 22nd Int'l Symposium on Distributed Computing (DISC'08)*, vol. 5218, Lecture Notes in Computer Science, Springer, 2008, pp. 438–450.

[143] Piotr Zielinski, « Optimistically Terminating Consensus: All Asynchronous Consensus Protocols in One Framework », *in*: *Proc. 5th Int'l Symposium on Parallel and Distributed Computing (ISPDC'06)*, IEEE Computer Society, 2006, pp. 24–33.

[144] Jakub Sliwinski, Yann Vonlanthen, and Roger Wattenhofer, « Consensus on Demand », *in*: *Proc. 24th Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'22)*, vol. 13751, 2022, pp. 299–313.

[145] Lefteris Kokoris-Kogias, Alberto Sonnino, and George Danezis, « Cuttlefish: Expressive Fast Path Blockchains with FastUnlock », *in*: *CoRR* abs/2309.12715 (2023), arXiv: 2309.12715, DOI: 10.48550/ARXIV.2309.12715, URL: https://doi.org/10.48550/arXiv.2309.12715.

[146] Leslie Lamport, « A Fast Mutual Exclusion Algorithm », *in*: *ACM Trans. Comput. Syst.* 5.*1* (1987), pp. 1–11.

[147] Gadi Taubenfeld, « Contention-sensitive data structures and algorithms », *in*: *Theor. Comput. Sci.* 677 (2017), pp. 41–55.

[148] Michel Raynal, *Concurrent programming: Algorithms, principles and foundations*, Springer, 2013.

[149] Gadi Taubenfeld, *Synchronization algorithms and concurrent programming*, Pearson Education/Prentice Hall, 2006.

[150] Francisco Vilar Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal, « Consensus in One Communication Step », *in*: *Proc. 6th International Conference on Parallel Computing Technologies (PaCT'01)*, vol. 2127, Lecture Notes in Computer Science, Springer, 2001, pp. 42–50.

[151] Jean-Philippe Martin and Lorenzo Alvisi, « Fast Byzantine Consensus », *in*: *IEEE Trans. Dependable Secur. Comput.* 3.*3* (2006), pp. 202–215.

[152] Fernando Pedone and André Schiper, « Optimistic atomic broadcast: a pragmatic viewpoint », *in*: *Theor. Comput. Sci.* 291.*1* (2003), pp. 79–101.

[153] Petr Kuznetsov, Andrei Tonkin, and Yan Zang, « Revisiting optimal resilience of fast Byzantine consensus », *in*: *Proc. 40th ACM Symposium on Principles of Distributed Computing (PODC'21)*, 2021, pp. 343–353.

[154] H. Attiya and J. L. Welch, « Brief Announcement: Multi-Valued Connected Consensus: A New Perspective on Crusader Agreement and Adopt-Commit », *in*: *DISC'23, Lipics*, vol. 281, 2023, 36:1–36:7.

[155] Danny Dolev, « The Byzantine Generals Strike Again », *in*: *J. Algorithms* 3.*1* (1982), pp. 14–30.

[156] Pesech Feldman and Silvio Micali, « An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement », *in*: *SIAM J. Comput.* 26.*4* (1997), pp. 873–933.

[157] Eli Gafni, « Round-by-Round Fault Detectors: Unifying Synchrony and Asynchrony (Extended Abstract) », *in*: *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC'98)*, ACM, 1998, pp. 143–152.

[158] Michael Ben-Or, « Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols », *in*: *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC'83)*, 1983, pp. 27–30.

[159] Bret Victor, *Tripphrases*, 2008, URL: `http://worrydream.com/tripphrase/`.

[160] Daniel Shawcross Wilkerson, « A Proposal for Proquints: Identifiers that are Readable, Spellable, and Pronounceable », *in*: *CoRR* abs/0901.4016 (2009), arXiv: 0901.4016, URL: `http://arxiv.org/abs/09011.4016`.

[161] Danny Dolev, Cynthia Dwork, and Larry J. Stockmeyer, « On the minimal synchronism needed for distributed consensus », *in*: *J. ACM* 34.*1* (1987), pp. 77–97.

[162] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer, « Consensus in the presence of partial synchrony », *in*: *J. ACM* 35.*2* (1988), pp. 288–323, DOI: `10.1145/42282.42283`, URL: `https://doi.org/10.1145/42282.42283`.

[163] Rachid Guerraoui, « Indulgent algorithms », *in*: *Proc. 19th Int'l Symposium on Distributed Computing (DISC'00)*, vol. 281, ACM Press, 2000, pp. 298–298.

[164] Rachid Guerraoui and Michel Raynal, « The alpha of indulgent consensus », *in*: *The Computer Journal*, vol. 50, 2007, pp. 53–67.

[165] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal, « Signature-Free Asynchronous Binary Byzantine Consensus with t>n/3, O(N2) Messages, and O(1) Expected Time », *in*: *J. ACM* (2015).

[166] Sebastian Pape, « A Survey on Non-transferable Anonymous Credentials », en, *in*: *The Future of Identity in the Information Society*, ed. by Vashek Matyáš, Simone Fischer-Hübner, Daniel Cvrček, and Petr Švenda, Berlin, Heidelberg: Springer, 2009, pp. 107–118, ISBN: 978-3-642-03315-5, DOI: 10.1007/978-3-642-03315-5_8.

[167] Russell Impagliazzo and Sara Miner More, « Anonymous credentials with biometrically-enforced non-transferability », *in*: *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, WPES '03, New York, NY, USA: Association for Computing Machinery, Oct. 2003, pp. 60–71, ISBN: 978-1-58113-776-7, DOI: 10.1145/1005140.1005150, URL: https://dl.acm.org/doi/10.1145/1005140.1005150 (visited on 03/19/2024).

[168] Marina Blanton and William M. P. Hudelson, « Biometric-Based Non-transferable Anonymous Credentials », en, *in*: *Information and Communications Security*, ed. by Sihan Qing, Chris J. Mitchell, and Guilin Wang, Berlin, Heidelberg: Springer, 2009, pp. 165–180, ISBN: 978-3-642-11145-7, DOI: 10.1007/978-3-642-11145-7_14.

[169] *DIF Wallet Security WG*, en, 2023, URL: https://github.com/decentralized-identity/org/blob/master/Org%20documents/WG%20documents/DIF_Wallet_Security_WG_Charter_20210616.pdf (visited on 03/19/2024).

[170] Abylay Satybaldy, Md. Sadek Ferdous, and Mariusz Nowostawski, « A Taxonomy of Challenges for Self-Sovereign Identity Systems », *in*: *IEEE Access* 12 (2024), Conference Name: IEEE Access, pp. 16151–16177, ISSN: 2169-3536, DOI: 10.1109/ACCESS.2024.3357940, URL: https://ieeexplore.ieee.org/abstract/document/10413448 (visited on 03/13/2024).

[171] *Hyperledger Aries Distributed Key Management System*, 2019.

[172] Timothé Albouy, Emmanuelle Anceaume, Davide Frey, Mathieu Gestin, Arthur Rauch, Michel Raynal, and François Taïani, *Asynchronous BFT Asset Transfer: Quasi-Anonymous, Light, and Consensus-Free*, _eprint: 2405.18072, 2024, URL: https://arxiv.org/abs/2405.18072.

[173] Mathieu Baudet, Alberto Sonnino, Mahimna Kelkar, and George Danezis, « Zef: Low-latency, Scalable, Private Payments », *in*: *CoRR* abs/2201.05671 (2022).

[174] Jan Camenisch and Victor Shoup, « Practical Verifiable Encryption and Decryption of Discrete Logarithms », en, *in*: *Advances in Cryptology - CRYPTO 2003*, ed. by Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, and Dan Boneh, vol. 2729, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 126–144, ISBN: 978-3-540-40674-7 978-3-540-45146-4, DOI: 10.1007/978-3-540-45146-4_8, URL: http://link.springer.com/10.1007/978-3-540-45146-4_8 (visited on 09/03/2024).

[175] Ralph C. Merkle, « A Digital Signature Based on a Conventional Encryption Function », *in*: *Advances in Cryptology — CRYPTO '87*, ed. by Carl Pomerance, Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378.

[176] C. P. Schnorr, « Efficient Identification and Signatures for Smart Cards », *in*: *Advances in Cryptology — CRYPTO' 89 Proceedings*, ed. by Gilles Brassard, New York, NY: Springer New York, 1990, pp. 239–252.

[177] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer, « From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again », *in*: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, New York, NY, USA: Association for Computing Machinery, Jan. 2012, pp. 326–349, ISBN: 978-1-4503-1115-1, DOI: 10.1145/2090236.2090263, URL: https://dl.acm.org/doi/10.1145/2090236.2090263 (visited on 09/12/2024).

[178] *Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity*, en, 2021, URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0281 (visited on 07/17/2024).

[179] Aurelien Berlan, *Terre et liberté. La quête d'autonomie contre le fantasme de délivrance - Aurélien Berlan*, fr, 2021, URL: https://www.decitre.fr/livres/terre-et-liberte-9791095432302.html (visited on 07/17/2024).

[180] Guy Debord, *La société du spectacle*, fr, troisième édition, 1992.

[181] *Fracture numérique : l'illectronisme touche plus de 15% de la population | vie-publique.fr*, fr, June 2023, URL: https://www.vie-publique.fr/en-bref/290057-fracture-numerique-lillectronisme-touche-plus-de-15-de-la-population (visited on 07/18/2024).

[182] Cours des comptes, *L'accès aux services publics dans les territoires ruraux*, fr, 2019.

[183] Michael Backes, Jan Camenisch, and Dieter Sommer, « Anonymous yet accountable access control », *in*: *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, WPES '05, New York, NY, USA: Association for Computing Machinery, Nov. 2005, pp. 40–46, ISBN: 978-1-59593-228-0, DOI: 10.1145/1102199.1102208, URL: https://dl.acm.org/doi/10.1145/1102199.1102208 (visited on 10/02/2024).

[184] Nicholas Martin and Frederik Metzger, « The chimera of control: Self-sovereign identity, data control, and user perceptions », *in*: *Human Technology* 20 (Sept. 2024), pp. 183–223, DOI: 10.14254/1795-6889.2024.20-2.1.

[185] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf, « Pseudonym Systems », *in*: *Selected Areas in Cryptography*, ed. by Howard Heys and Carlisle Adams, Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 184–199, ISBN: 978-3-540-46513-3.

[186] Torben P. Pedersen, « Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing », *in*: *CRYPTO*, vol. 576, Lecture Notes in Computer Science, Springer, 1991, pp. 129–140.

Appendix A

E-vote system implementation using a DenyList object

In this section, we show that DenyList objects can provide upper bounds on the consensus number of a complex objects. As an example, we study an e-vote system. An e-vote system must comply with the same properties as an "in-person" voting. An "in-person" voting system must ensure four security properties, two for the organizers and two for the voters. First, the organizers of the vote must ensure that each person who votes has the right to do so. Second, each voter must vote only once. Third, a voter must verify that their vote is considered in the final count. Fourth, an optional property is voter anonymity. Depending on the type of vote, the voter may want to hide their identity.

We want to design a distributed e-vote system, where a voter can submit their vote to multiple different voting servers—while ensuring the unicity of their vote. We assume each server is a process of the distributed infrastructure. The voters act as clients, submitting vote requests to the servers. We assume the "right to vote" property is ensured using tokens. The Token is a one-time-used pseudonym that links a vote to a voter. Users obtains their Tokens from an issuer. All the voting servers trust this issuer. Neither the voters nor the issuer has access to the e-vote object—except if one of the actors assumes multiple roles. Using these specifications, we define the e-vote object type as follows:

Definition A.1. The e-vote object type supports two operations: Vote(*Token*, v) and VoteCount(). *Token* is the voter's identifier, and v is the ballot. Moreover, these operations support three mandatory properties and one optional property:

1. (Vote Validity) A Vote(*Token*, v) invoked at time t is valid if:
   — *Token* is a valid token issued by an issuer trusted by the voting servers; and
   — No valid Vote(*Token*, v') operation was invoked at time $t' < t$, where $v' \neq v$.
2. (VoteCount validity) A VoteCount() operation returns the set of valid Vote operations invoked.
3. (optional - Anonymity) A *Token* does not link a vote to its voter identity, even if the voting servers and the issuers can collude.

In the following, we analyze an e-vote system based on signatures. The issuer issues a signature to the voter. The message of the signature is a nonce. The tuple (signature, nonce) is used as a token. When a voter issues a vote request, the server verifies the signature's validity and proceeds to vote if the signature is valid.

Adding anonymity to a signature-based e-vote system can be easily achieved using blind signatures [83]. A blind signature algorithm is a digital signature scheme where the issuer does not learn the value it signs. Because the signed value is usually a nonce, the issuer does not need to verify the value—a value chosen maliciously will not grant the voter more privileges than expected.

Formally, a Blind signature algorithm is defined by the tuple (Setup, Commit, Sign, Uncommit, Verify), where Setup creates the common values of the scheme (generators, shared randomness, etc...) and secret/public key pairs for all issuers. The public keys are shared with all the participants in the system. Commit is a commitment scheme that is hiding and binding; it outputs a commitment to a value—the nonce—randomly chosen by the user. The Sign algorithm takes as input a commitment to a nonce and the secret key of an issuer. It outputs a signature on the commitment. The Uncommit algorithm takes a signature on a commitment and the issuer's public key as input and outputs the same signature on the uncommitted message (the original message). Finally, the Verify algorithm outputs 1 if the uncommitted signature is a valid signature by an issuer on the message m.

Algorithm 24 provides a wait-free implementation of an e-vote system using any signature scheme, one k-DenyList object, and one SWMR atomic snapshot object. Here, the value of k corresponds to the number of voting servers, and $k{=}|\Pi_V| = |\Pi_M|$. The idea of the algorithm is to use the Append operation to state that a token has already been used to cast a vote. In order to obtain a wait-free implementation, we use a helper value [121] stored in the Atomic Snapshot object AS-prevote. The vote operation is conducted as follows: the voting server V communicates the vote it will cast in the AS-prevote object. Then, V conducts a Prove(*Token*) operation to prove that the Token has not yet been used. Then, V invokes an Append(*Token*) operation and waits until the Append is effective—the do-while loop in line 8 to 10. Finally, V uses the Read operation to verify that it is the only process that proposed a vote for this specific Token. If it is the case, the vote is added to the vote array—the AS-vote array. Otherwise, the vote is added to the vote array only if the other servers that voted using the same Token proposed the same ballot in line 4.

Other implementations can be proposed in the case of two concurrent transactions requested by the same voter—to different servers—with different values. For example, it is possible to modify the algorithm presented in Algorithm 24 using a deterministic function to choose one value among all the potential votes. This modification does not impact the properties of this implementation.

We now provide an informal proof of the linearizability of this implementation. The anti-flickering property of the k-DenyList object ensures the termination of the while loop. Therefore, the implementation is wait-free. The same property ensures that the *vote-values* variables are the same for all voters with the same Token, thus ensuring the unicity of the vote. The proof of authorization of the vote is ensured by the signature verification in line 1. The anonymity property is also fulfilled if a blind signature scheme is used. Therefore, the use of anonymous DenyList objects is not required. Hence, we can conclude that the consensus number of the k-DenyList object type is an upper bound on the consensus number of an e-vote system.

Shared variables:
k-dlist ← k-DenyList object;
AS-prevote ← Atomic Snapshot object, initially $\{\emptyset\}^k$
AS-vote ← Atomic Snapshot object, initially $\{\emptyset\}^k$
Operation Vote((*signature*, *pk*, *token*, v) **is**:
1: **If** Verify(*signature*, *token*, *pk*) $\neq 1$ **then**:
2: **Return** false;
3: AS-prevote.Update((*token*, v), p);
4: ret ← k-dlist.Prove(*token*)
5: **If** ret = false **then**:
6: **Return** false;
7: k-dlist.Append(*token*);
8: **Do**:
9: ret ← k-dlist.Prove(*token*);
10: **While** ret ≠ false; 11: *votes* ← k-dlist.Read();
12: *client-votes* ← all values in *votes* where token is *token*;
13: *voters* ← all processes in *client-votes*;
14: *vote-values* ← all values in AS-prevote
‘ where token is *token* and processes that added the value are in *voters*;
15: **If** *vote-values* = $\{v\}^l, \forall\ l \geq 1$ **then**:
16: previous-votes ← AS-vote.SNAPSHOT()[p];
17: AS-vote.Update(previous-votes ∪ (token, v, voters), p);
18: **Return** true;
19: **Else return** false;
Operation VoteCount() **is**:
20: votes ← all values in AS-vote.Snapshot().
Only one occurrence of each tuple (token, v, voters) is kept;
21: **Return** votes;

Algorithm 24: Implementation of an e-vote object using one k-DenyList object and Atomic Snapshots

It is also possible to build a wait-free implementation of a k-consensus object using one e-vote object. This implementation is presented in Algorithm 25. The idea of this algorithm is that each process will try to vote with the same Token. Because only one vote can be accepted, the e-vote object will only consider the first voter. Ultimately, every process will see the same value in the vote object. This value is the result of the consensus.

Each invocation is a sequence of a finite number of local operations and e-vote object accesses, which are assumed atomic. Therefore, each process terminates the PROPOSE operation in a finite number of its own steps. A vote can only be taken into account if it was proposed by some process, which enforces the validity property. The agreement property comes from the unicity of the vote. Finally, the non-trivial property of the consensus object is ensured because the decided value is any value v chosen by the winning process.

Shared variables:
  vote-obj ← e-vote object where the only authorized token is 0;
  σ ← signature on 0 by a trusted issuer;
Operation Propose(v) **is**:
1: vote-obj.Vote((σ, 0), v);
2: {(winner-token, value)} ← vote-obj.VoteCount();
3: **Return** value;

Algorithm 25: Implementation of a k-consensus object using one e-vote object

The consensus number of a blind-signature-based e-vote system is bounded on one side by the consensus number of a k-consensus object and the other side by the consensus number of a k-DenyList object. Hence, the blind-signature-based e-vote object type has consensus number k.

Appendix B

Possible Additional Properties to the HIAC scheme

We presented the general hidden issuer Anonymous Credential scheme in 5.7.1. in this section we will present alternative properties we can achieve with the same signature, by modifying it slightly.

B.1 Non Transferable Signature

It can be useful for the verifier to ensure that a given credential was actually issued to the user who is using it, and not to someone else. Because of the unlinkability property, this cannot be straightforward. For example, the easiest way to obtain this property consists in adding an element in the message that is linked to its user. However, this would break the unlinkability property. We offer a way to make our signature non transferable without this drawback.

The idea consists in discouraging transfers by relying on an external escrow where the user stores some valuable item, or money. Sharing a credential would also imply sharing the valuable in the escrow. This is called *PKI-assured non-transferability* [185].

We propose to adapt this property to our scheme. The user will put something valuable inside an escrow, and lock it in place with a private key. The user then uses this private key instead of the random value R_y in step 4 (**Sign** algorithm). He then proves to the issuer, that the value he gives to it (which in our protocol was named $u = Y_1^{R_y} g_1^R$) is a commitment to a key that can open the escrow. The value of this escrow will depend on the requirements of the issuer. Afterward, the process is the same, except for the protocol **VerifyRandomized**. We add an interactive ZKP verification to the original algorithm:

$$\text{ZKPoK}\{(R_y r_u r_{(u,y)}) : h_y = g_2^{R_y r_u r_{(u,y)}}\}$$

This allows the verifier to ensure that the user knows the blinding elements, in particular the non-transferable element R_y. If all the issuers follow the requirements of the verifier, then the credentials are non transferable. This technique can also be used to link credentials between themselves. A verifier can verify that different credentials embed the same R_y's values. It proves

that they were issued to the same user.

This modification can be added to the PS signature scheme to enable non-transferability property.

B.2 Signature on Commitments and One-Show Credential

An interesting feature to have in a credential scheme is optional one-show property. In this configuration, a credential can only be used once. It can be particularly useful in e-vote systems for example.

To enhance this property, the easiest way is for the issuer to sign committed message. If the issuer's key is used for this purpose only, the message can represent a nonce, that will identify the credential. Before showing the credential, the user will un-commit the message, while keeping the signature structure. Thus, the verifier will see the nonce, and will be able to add it to a shared ledger. Any message with a nonce already added to the ledger is thus revoked. With this protocol, the user gets one-show property, without loosing any other security properties.

To adapt this principle to our scheme, we use a modified PS signature on committed message protocol. In fact, to achieve this property we only modify the **Sign** algorithm:

Sign(pp, isk_I, u, u', ϕ): on input of a setup parameter pp,, an issuer's secret key $isk_I = (x_I, y_I)$, two values given by the user $u = (Y_1^{(I)})^{\mathsf{H}(m)R_y} \cdot g_1^R$ and $u' = (Y_1^{(I)})^{\mathsf{H}(\mathsf{H}(m))R_y} \cdot g_1^R$, where R_y and R are secret random values, and m is the message, and ϕ a ZKPoK of u, $\phi : NI-ZKPoK\{(\mathsf{H}(m)R_y, R) : u = Y_1^{(I)\mathsf{H}(m)R_y} \cdot g_1^R \wedge u' = Y_1^{(I)\mathsf{H}(\mathsf{H}(m))R_y} \cdot g_1^R\}$. Initially, the issuer outputs a signature $\sigma^\Delta = (h_1, h_2, \sigma_1^\Delta, \sigma_2^\Delta)$, where $h_1 = g_1^{r_{(I,1)}}$, $h_2 = g_1^{r_{(I,2)}}$, $\sigma_1^\Delta = (g_1^{x_I} \cdot u)^{r_{(I,1)}}$, $\sigma_2^\Delta = (g_1^{x_I} \cdot u')^{r_{(I,2)}}$, with $r_{(I,1)}, r_{(I,2)} \leftarrow\!\!\$\, \mathbb{Z}_p^*$. The user then uncommits the signature σ^Δ into $\sigma = (h_1, h_2, \sigma_1 = \sigma_1^\Delta \cdot h_1^{-R}, \sigma_2 = \sigma_2^\Delta \cdot h_2^{-R})$. The user stores σ and R_y.

With this transformation, the other parts of the protocol are not modified, and the issuer does not learn the message signed.

Appendix C

HIAC's Proofs

C.1 Assumptions

Our signature scheme relies on several complexity assumptions:

Definition C.1. [Discrete Logarithm (DL) assumption] Given $h \in \mathbb{G}$, with $\mathbb{G}$ a group generated by g, no PPT adversary can find x such that $h = g^x$ with non negligible probability.

Definition C.2. [Computational Diffie Hellman (CDH) Assumption] Given $(g, g^x, g^y) \in \mathbb{G}$, with $\mathbb{G}$ a group generated by g, no PPT adversary can find g^{xy} with non negligible probability.

Definition C.3. [Bilinear Diffie Hellman(BDH) Assumption] Given a type 3 bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, with $\langle g_1 \rangle = \mathbb{G}_1$, $\langle g_2 \rangle = \mathbb{G}_2$ and $(g_i^a, g_j^b, g_k^c), \forall i, j, k \in \{1, 2\}$, no PPT adversary can find $e(g_1, g_2)^{abc}$ with non negligible probability.

Definition C.4. [Decisional Diffie Hellman (DDH) Assumption] Given $(g, g^x, g^y, h) \in \mathbb{G}$, with $\mathbb{G}$ a group generated by g, no PPT adversary can decide with non negligible probability if $h = g^{xy}$.

C.2 Aggregator Correctness

We prove in this subsection that the aggregator presented in Section 5.7.1 is correct.

Proof. We prove the correctness of the aggregator by developing the verification equation with values produced following the entire protocol:

$$\begin{aligned} e((W)'_{(i)}, C') &= e\left(g_1^{\mathsf{sk}\ r_2 \frac{1}{x_l}}, g_2^{x_l r_1}\right) \\ &= e(g_1, g_2)^{\mathsf{sk}\ r_2\ r_1} \\ &= e\left(g_1^{\mathsf{sk}}, g_2^{r_1\ r_2}\right) \\ &= e(g_1^{sk}, h) \end{aligned}$$

□

C.3 Aggregator Collision-Freedom

We want to prove that the aggregator scheme proposed in 5.7.1 is collision-free. We restate Theorem 5.2:

Theorem 5.2. ***CollisionFreedomGame***$(\mathcal{A}, \mathcal{C})$*, for a challenger* $\mathcal{C}$*, and an adversary* $\mathcal{A}$ *:*

- ***Setup****:* $\mathcal{C}$ *runs the aggregator* ***Setup*** *algorithm, chooses a set* $\mathcal{S}$*, runs the* ***Gen*** *algorithm to build a commitment to* $\mathcal{S}$*, and the associated verification key.* $\mathcal{A}$ *is given the aggregator's public information.*
- ***Output****:* $\mathcal{A}$ *outputs* C^* *a commitment to* s^* *and* π_s^**. The game outputs* 1 *if* $s^* \notin \mathcal{S}$*, and* ***Verify****(*$pp, \mathsf{Agg}, \mathsf{aux}, \mathsf{sk}, C^*, \pi_s^*$*) = 1, or* 0 *otherwise.*

An Aggregator is said to be element-indistinguishable if for a negligible ϵ*, the probability for a* PPT *Adversary* $\mathcal{A}$ *can to win the* ***CollisionfreedomGame*** *is:*

$$\Pr[\textbf{\textit{CollisionfreedomGame}}(\mathcal{A}, \mathcal{C}) = 1] \leq \epsilon$$

We will prove Theorem 5.2 by contradiction, *i.e.*, we assume the adversary is able to find a tuple $(C^*, \pi_s^* = ((W)'^*_{(l)}, h^*))$ such that:

$$e((W)'^*_{(l)}, C^*) = e(g_1^{\mathsf{sk}}, h^*))$$

We will process as follow:

1. Representation of the elements;
2. Proof that $(W)'^*_{(l)}$ is a combination of different $(W)_i$ $i \in \{1, \cdots, k\}$;
3. Proof that $(W)'^*_{(l)}$ is composed on only one $(W)_i$ $i \in \{1, \cdots, k\}$; and
4. Proof of Theorem 5.2.

C.3.1 Representation of the Elements

Figure C.1 is a game representation of Theorem 5.2.

Remark C.1. Probability of success of a game i is written as $\Pr[S_i]$. Success is obtained when a game returns 1. the adversary has a negligible advantage in $\mathsf{Game}_1\mathcal{A}$ if $\Pr[S_1] \leq \epsilon$, where ϵ is negligible.

For the ease of the proof, we set a notation for each element produced by the adversary :

$$C^* = g_2^{s_x}$$

$Theorem - 6 - Game_1\mathcal{A}$

1 : $(x_1, ..., x_k) \leftarrow_\$ (\mathbb{Z}_p^*)^k$;
2 : $\mathrm{PK}_1 = (X_1^{(1)}, ..., X_1^{(k)}) = (g_1^{x_1}, ..., g_1^{x_k})$
3 : $\mathrm{PK}_2 = (X_2^{(1)}, ..., X_2^{(k)}) = (g_2^{x_1}, ..., g_2^{x_k})$
4 : $\bar{\mathrm{PK}}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
5 : $(W, \mathsf{sk}) \leftarrow_\$ \mathbf{Gen}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1)$
6 : $(h^*, C^*, (W)'^*_{(l)}, \eta) \leftarrow_\$ \mathcal{A}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1, W)$;
7 : $a_1 := e((W_y)I^*, \mathsf{Comm}(s^*))$
8 : $a_2 := e(g_1^{\mathsf{sk}}, h^*)$
9 : If $a_1 = a_2 \wedge X_2^{(I)*} \neq g_2^{x_l \eta}, \forall i \in \{1, ..., k\}$:
10 : **return** 1;
11 : Else:
12 : **return** 0;

Figure C.1 – Theorem 5.2 game representation. PK_1 is the set of aggregated values in $\mathbb{G}_1$ and PK_2 is the set of aggregated values in $\mathbb{G}_2$. η is a randomizing value, known by the adversary.

$$\begin{aligned} \pi_s^* =& ((W)'^*_{(l)}, h^*) \\ =& (g_1^{s_w}, g_2^{s_h}) \end{aligned} \tag{C.1}$$

We develop the pairings a_1 and a_2 from Figure C.1, using notation from Equation (C.1).

$$\begin{aligned} a_1 =& e((W)^*_{(l)}, C^*) \\ =& e(g_1^{s_w}, g_2^{s_x}) \\ =& g_T^{s_w s_x} \end{aligned} \tag{C.2}$$

$$\begin{aligned} a_2 =& e(g_1^{\mathsf{sk}}, h^*) \\ =& e(g_1^{\mathsf{sk}}, g_2^{s_h}) \\ =& g_T^{s_h \mathsf{sk}} \end{aligned} \tag{C.3}$$

From the Equation $a_1 = a_2$, the Equation (C.2) and the Equation (C.3), we can write:

$$\begin{aligned} & a_1 = a_2 \\ \Leftrightarrow\ & g_T^{s_x s_w} = g_T^{s_h \mathsf{sk}} \\ \Rightarrow\ & s_x s_w \equiv s_h \mathsf{sk} \pmod{p-1} \end{aligned} \tag{C.4}$$

We rewrite $\mathsf{Game}_1\mathcal{A}$, using Equation (C.4) development. This game is presented in Figure C.2

$Theorem - 6 - Game_2\mathcal{A}$

1 : $(x_1, ..., x_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
2 : $\mathrm{PK}_1 = (X_1^{(1)}, ..., X_1^{(k)}) = (g_1^{x_1}, ..., g_1^{x_k})$
3 : $\mathrm{PK}_2 = (X_2^{(1)}, ..., X_2^{(k)}) = (g_2^{x_1}, ..., g_2^{x_k})$
4 : $\bar{\mathrm{PK}}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
5 : $(W, \mathsf{sk}) \leftarrow\!\!\$\ \mathbf{Gen}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1)$
6 : $(g_2^{s_h}, g_1^{s_w}, g_2^{s_x}, \eta) \leftarrow\!\!\$\ \mathcal{A}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1, W)$;
7 : $a'_1 := s_w s_x$
8 : $a'_2 := s_h \mathsf{sk}$
9 : If $a'_1 \equiv a'_2 \pmod{p-1} \wedge g_2^{s_x} \neq g_2^{x_i \eta}, \forall i \in \{1, ..., k\}$:
10 : **return** 1;
11 : Else:
12 : **return** 0;

Figure C.2 – Theorem 5.2 game - notation modifications

$\mathsf{Game}_1\mathcal{A}$ and $\mathsf{Game}_2\mathcal{A}$ are exactly the same games, with notation modification, this implies that $\Pr[S_1] = \Pr[S_2]$.

C.3.2 Proof that $(W)'^*_{(l)}$ is a Combination of Different $(W)_i$ $i \in \{1, \cdots, k\}$

The goal is to prove that $(W)'^*_{(l)}$ is a composition of $(W)_i$. Informally, it is possible to understand this statement as the fact that the adversary needs to output an element in $\mathbb{G}_1$ containing sk. Under CDH assumption, the only way he can achieve this is by using a combination of $(W)_i$.

First, we state that under type-3 bilinear pairing assumption, finding a morphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ is as hard as solving the DL problem [89]. Then, from the equation $a'_1 = a'_2$, we know that $s_w s_x$ depends on sk. And sk is only disclosed as an exponent in $\mathbb{G}_1$. Thus, under DL assumption, the adversary has to include sk inside s_w. In other words $s_w = \mathsf{sk}\zeta$ for some ζ. Because sk is only known to the adversary in $(W)_i$, under CDH assumption we can deduct that $(W)'^*_{(l)}$ is a linear combination of $(W)_i$ values.

From the above discussion, s_w is composed of W values. Therefore, we can represent s_w as follow:

$$s_w = \mathsf{sk} \sum_{i=1}^{k} \left(\frac{b_i}{x_i} \right) \tag{C.5}$$

Where the $b_i \in \mathbb{Z}_p, 1 \leq i \leq k$, are factors known and chosen by the adversary, and $\sum_{i=1}^{k} (|b_i|) \geq 1$.

Furthermore, we know that $\frac{s_w s_x}{s_h} = \mathsf{sk}$. Let us assume $x_1, \cdots, x_k = \mathsf{sk}$. In this case, $\deg(s_w) =$

Theorem $-$ 6 $-$ *Game*$_3\mathcal{A}$

$(x_1, ..., x_k) \leftarrow\$ (\mathbb{Z}_p^*)^k$;
$\mathrm{PK}_1 = (X_1^{(1)}, ..., X_1^{(k)}) = (g_1^{x_1}, ..., g_1^{x_k})$
$\mathrm{PK}_2 = (X_2^{(1)}, ..., X_2^{(k)}) = (g_2^{x_1}, ..., g_2^{x_k})$
$\bar{\mathrm{PK}}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
$(W, \mathsf{sk}) \leftarrow\$ \mathbf{Gen}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1)$
$(g_2^{s_h}, g_1^{\mathsf{sk}\sum_{i=1}^{k}\left(\frac{b_i}{x_i}\right)}$
$\quad, g_2^{s_x}, \eta) \leftarrow\$ \mathcal{A}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1, W)$;
$a_1' := s_x \mathsf{sk} \sum_{i=1}^{k}\left(\frac{b_i}{x_i}\right)$
$a_2' := s_h \mathsf{sk}$
If $a_1' = a_2' \wedge g_2^{s_y} \neq g_2^{x_i \eta}, \forall i \in \{1, ..., k\}$:
$\quad$ **return** 1;
Else:
$\quad$ **return** 0;

Figure C.3 – s_w representation game

0 in term of values unknown by the adversary, *i.e.*, $\mathsf{sk}, x_1, \cdots, x_k$ (from Equation (C.5)). We thus face an easier problem : $\frac{s_x}{s_h} = \mathsf{sk}$. Solving the first problem implies being able to solve the second one. Furthermore, s_x and s_h are outputed by the adversary in $\mathbb{G}_2$, this mean that they are buil using (g_2, g_2^{sk}). From the equation $\frac{s_x}{s_h} = \mathsf{sk}$ we know that s_h should be of degree 0, or lower. However, getting $\deg(s_h) < 0$ is directly equivalent to the Diffie Hellman inversion problem in this case, which is equivalent to CDH problem [76]. Thus $\deg(s_h) = 0$, in other world, the adversary knows the discrete logarithm of $g_2^{s_h}$. We can thus conclude that $\deg(s_x) = 1$.

According to this discussion, we write a new game, presented in Figure C.3. We call this new game: $\mathsf{Game}_3\mathcal{A}$.

The transition between $\mathsf{Game}_3\mathcal{A}$ and $\mathsf{Game}_2\mathcal{A}$ is a failure based transition, where the failure event is the probability for the adversary to find a solution to the CDH problem or the DL problem. This means :

$$|\Pr[S_3] - \Pr[S_2]| \leq \epsilon_{\mathrm{CDH}} + \epsilon_{\mathrm{DL}}$$

C.3.3 Proof that $(W)'^{*}_{(l)}$ is Composed of Only One $(W)_i$ $i \in \{1, \cdots, k\}$

If we rewrite the equation $a'_1 = a'_2$ from $\mathsf{Game}_3\mathcal{A}$, we obtain:

$$s_x \mathsf{sk} \sum_{i=1}^{k} \left(\frac{b_i}{x_i}\right) = s_h \mathsf{sk}$$
$$\Leftrightarrow s_x \sum_{i=1}^{k} \left(\frac{b_i}{x_i}\right) = s_h$$

We need to study the value $\frac{s_x}{s_h}$. We want to prove that no adversary can output $\frac{s_x}{s_h} = \frac{1}{\sum_{i=1}^{k}\left(\frac{b_i}{x_i}\right)}$ unless $b_i = 0$ for $(k-1)$ i's. Or, in other words, that $\frac{s_x}{s_h} = \frac{x_I}{b_I}, 1 \leq I \leq k$, and $b_I \in \mathbb{Z}^*_p$.

Another way to explain this expression is to say that the Adversary has to use exactly one witness to build the $(W)^*_I$ value.

Lemma C.1. Given $(g_2, g_2^{x_1}, ..., g_2^{x_k})$, no adversary can output, with non negligible probability, $g_2^{s_x}$ and $g_2^{s_h}$ such that $\frac{s_x}{s_h} = \frac{1}{\sum_{i=1}^{k}\left(\frac{b_i}{x_i}\right)}$, and more than one $b_i \neq 0, \forall i \in \{1, ..., k\}$.

We will prove Lemma C.1 by contradiction, proving that finding such values $g_2^{s_h}$ and $g_2^{s_x}$ is at least as hard as solving the CDH problem.

Proof. Hypothesis 1 : Given $(g_2, g_2^{x_1}, ..., g_2^{x_k})$, an adversary can output, with non negligible probability, $g_2^{s_x}$ and $g_2^{s_h}$ such that $\frac{s_x}{s_h} = \frac{1}{\sum_{i=1}^{k}\left(\frac{b_i}{x_i}\right)}$, and more than one $b_i \neq 0, \forall i \in \{1, \cdots, k\}$.

We know from the previous step of the proof that $\deg(s_h) = 0$, thus, we only study the values taken by s_x, assuming s_h is a value known by the adversary, independent of the variables $x_i, \forall i \in \{1, \cdots, k\}$.

We study the case where they are only two variables, *i.e.*, we give to the adversary $(g_2, g_2^{x_1}, g_2^{x_2})$. We will prove that CDH problem can be reduced to this simplified version of the problem. We write the restricted version of the game:

Game C.1. Let $\mathbb{G}_2$ be a group, let g_2 be a generator of this group, and let x_1, x_2 be two elements in $\mathbb{Z}^*_p$. Given $(g_2, g_2^{x_1}, g_2^{x_2})$, output $g_2^{\frac{x_1 x_2}{x_1 b_2 + x_2 b_1}}$, for some $b_1, b_2 \in \mathbb{Z}^*_p$.

Remark C.2. It is easy to reduce Lemma C.1 to Game C.1, by adding ad hoc values $x_3 = 1, \cdots, = x_k = 1$ to the construction of Game C.1.

If the adversary has a non negligible probability of success in Game C.1, then he has access to an algorithm A, that given (g, g^{x_1}, g^{x_2}), outputs with non negligible probability $g^{\frac{x_1 x_2}{x_1 b_1 + x_2 b_2}}$, for any values b_1, b_2, independent from x_1 and x_2. Given this algorithm, the adversary can query: $A(g, g^{1+x_1}, g^{1-x_1}) = g^{\frac{(1+x_1)(1-x_1)}{(1+x_1)+(1-x_1)}} = g^{\frac{1-x_1^2}{2}}$. Lets call this value a. Then, the adversary is able to extract $g^{x^2} = a^{-2} \cdot g$. Thus, the algorithm A can solve the Diffie Hellman squaring problem, which is equivalent to CDH problem [76]. Therefore, under CDH assumption, no PPT adversary

$Theorem - 6 - Game_5\mathcal{A}$

$(x_1, ..., x_k) \leftarrow\$ (\mathbb{Z}_p^*)^k$;
$\mathrm{PK}_1 = (X_1^{(1)}, ..., X_1^{(k)}) = (g_1^{x_1}, ..., g_1^{x_k})$
$\mathrm{PK}_2 = (X_2^{(1)}, ..., X_2^{(k)}) = (g_2^{x_1}, ..., g_2^{x_k})$
$\bar{\mathrm{PK}}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
$(W, \mathsf{sk}) \leftarrow\$ \mathbf{Gen}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1)$
$(g_2^{s_h}, I, g_1^{\mathsf{sk}\frac{b_I}{x_I}}$
$\quad , g_2^{s_x}, \eta) \leftarrow\$ \mathcal{A}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1, W)$;
$a'_1 := \mathsf{sk}s_x b_I \frac{b_I}{x_I}$
$a'_2 := \mathsf{sk}s_h$
If $a'_1 = a'_2 \wedge g_2^{s_x} \neq g_2^{x_i \eta}, \forall i \in \{1, \cdots, k\}$:
$\quad$ **return** 1;
Else:
$\quad$ **return** 0;

Figure C.4 – Theorem 5.2 game with reduction of the s_w value

can solve Game C.1 with probability greater than ϵ_{CDH}, with ϵ_{CDH} the probability to solve the CDH problem. By extension, hypothesis 1 is contradicted. Thus Lemma C.1 is proven. □

C.3.4 Proof of Theorem 5.2

We have $\frac{s_x}{s_h} = \frac{x_I}{b_I}, 1 \leq I \leq k$, and $b_I \in \mathbb{Z}_p^*$.

We rewrite $\mathsf{Game}_3\mathcal{A}$ using Lemma C.1. This new game is provided in Figure C.4. We call this new game: $\mathsf{Game}_4\mathcal{A}$.

The transition between $\mathsf{Game}_4\mathcal{A}$ and $\mathsf{Game}_3\mathcal{A}$ is a failure based transition, where the failure event is the probability for the adversary to find a solution to the CDH problem. This means :

$$\begin{aligned} |\Pr[S_4] - \Pr[S_3]| &\leq F \\ &\leq \epsilon_{\mathrm{CDH}} \end{aligned}$$

We develop again $a'_1 = a'_2$ with values from $\mathsf{Game}_5\mathcal{A}$:

$$\begin{aligned} \mathsf{sk}s_x \frac{b_I}{x_I} &= \mathsf{sk}s_h \prod_{j=1}^{k}(x_j) \\ \Leftrightarrow s_x &= x_I \frac{s_h}{b_I} \end{aligned}$$

$Theorem - 6 - Game_6\mathcal{A}$

1 : $(x_1, ..., x_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
2 : $\mathrm{PK}_1 = (X_1^{(1)}, ..., X_1^{(k)}) = (g_1^{x_1}, ..., g_1^{x_k})$
3 : $\mathrm{PK}_2 = (X_2^{(1)}, ..., X_2^{(k)}) = (g_2^{x_1}, ..., g_2^{x_k})$
4 : $\bar{\mathrm{PK}}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
5 : $(W, \mathsf{sk}) \leftarrow\!\!\$\ \mathbf{Gen}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1)$
6 : $(g_2^{s_x}, \eta') \leftarrow\!\!\$\ \mathcal{A}(\mathrm{PK}_1, \mathrm{PK}_2, \bar{\mathrm{PK}}_1, W)$;
7 : If $g_2^{s_x} = g_2^{x_I \eta'}, \forall I \in \{1, ..., k\}$
8 : $\quad \wedge g_2^{s_x} \neq g_2^{x_I \eta'}, \forall I \in \{1, ..., k\}$:
9 : $\quad$ **return** 1;
10 : Else:
11 : $\quad$ **return** 0;

Figure C.5 – Theorem 5.2 game with devellopped a'_1 and a'_2

We write a final game, directly developing a'_1 and a'_2, with $\frac{s_h}{b_I} = \eta'$. This game is presented in Figure C.5

This last transition is a notation modification, thus $\Pr[S_5] = \Pr[S_4]$.

And the probability of success of $\mathsf{Game}_6\mathcal{A}$ is obviously 0, as the condition of success is a contradiction.

Now we can evaluate probability of success of $\mathsf{Game}_1\mathcal{A}$:

$$|\Pr[S_5] - \Pr[S_1]| \leq 2 \cdot \epsilon_{\mathrm{CDH}} + \epsilon_{\mathrm{DL}}$$

And, as $\Pr[S_5] = 0$:

$$\Pr[S_1] \leq 2 \cdot \epsilon_{\mathrm{CDH}} + \epsilon_{\mathrm{DL}} \tag{C.6}$$

Probability of success of $\mathsf{Game}_1\mathcal{A}$ is negligible. Therefore our aggregator construction is collision free.

C.4 Proof of Element-Indistinguishability

Theorem 5.3. *The aggregator scheme presented in Section 5.7.1 is element indistinguishable. Given an aggregator* Agg_x*, the set of secret values accumulated in the aggregator* $(x_1, \cdots, x_k\}$*,* $k \geq 2$*, a set-membership proof* π_s*, and a randomized commitment* C' *to an element of the set* $\mathcal{S}$*, no* PPT *adversary can find with non negligible probability the element* s *the user committed to.*

The element indistinguishable property states that no malicious verifier can, with non negligible probability, decide which elements the user commits to. In order to prove this statement, we need a preliminary lemma:

Lemma C.2. Given $(g, g^{x_1}, \cdots, g^{x_n}) \in \mathbb{Z}_p^n$. Under DDH assumption, no PPT adversary is able to distinguish between non linear expression of the variables with non negligible probability.

Proof. We will prove Lemma C.2 by contradiction.

We assume that, given $(g, g^{x_1}, \cdots, g^{x_n}, h)$, the adversary (malicious verifier) is able to decide with non negligible probability if $h \stackrel{?}{=} g^{x_1+\cdots+x_ix_j+\cdots+x_n}$, for $i, j \in \{1, \cdots, n\}$. If we set $x_k = 0, \forall k \neq \{i, j\}$, then this decision the adversary is able to make is equivalent to find, with non negligible probability, a solution to the DDH problem. Under DDH assumption, this has negligible probability to occur. This implies that Lemma C.2 is true. □

Given this lemma, it is possible to prove Theorem 5.3:

Proof. Let define element indistinguishability with two elements x_1 and x_2. With the bilinear map e, and the elements he is given, the adversary is able to compute the tuple $(g_T, g_T^{r_1a}, g_T^{r_2b}, g_T^{r_1r_2}, g_T^{r_1r_2ab})$. He must decide, with non negligible probability whether $(a = x_1 \wedge b = x_2) \vee (a = x_2 \wedge b = x_1)$.

The order of $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ is prime, thus every element of these groups is a generator, except from the neutral element. Therefore, every elements presented in Table C.2 will be seen, from the adversary point of view, as elements chosen uniformly at random in one of these groups.

Furthermore the value $g_T^{r_1r_2ab}$ is symmetrical. The adversary knows this value, but cannot use it in any way to distinguish between a and b. And $g_T^{r_1r_2}$ does not depend on a nor b.

Therefore, the adversary has to combine and compare the different element he is given to be able to distinguish between a and b.

However, we can argue, using Lemma C.2, that the adversary is not able to combine the values $(g_T, g_T^{r_1a}, g_T^{r_2b}, g_T^{r_1r_2})$ to take its decision.

Thus the adversary is not able to decide whether $(a = x_1 \wedge b = x_2) \vee (a = x_2 \wedge b = x_1)$ under DDH assumption. Thus our aggregator implementation is element indistinguishable. □

C.5 Signature Correctness

We prove in this subsection that the signature presented in Section 5.7.1 is correct.

Proof. First, the aggregator is correct, thus, the verification of an honestly built aggregator will always output 1. Then, let $\sigma' = (h'_1, h'_2, \sigma'_1, \sigma'_2)$ be a randomized signature built following the

above protocol, let $(X^{(I)'}, Y_2^{(I)'})$ be commitments to trusted issuer's keys, and let $vsk = (\mathsf{sk}_x, \mathsf{sk}_y)$ be a secret verifier key. We can verify that the signature is correct:

$$\begin{aligned}
&e(h_1^{'}, X^{(i)'} Y_2^{(i)'\mathsf{H}(m)}) \\
&\qquad = e(g_1^{r_{I,1} r_u'}, g_2^{x_I r_u + r_u y_I R_y \mathsf{H}(m)}) \\
&\qquad = e(g_1, g_2)^{r_{(I,1)} r_u' r_u (x_I + y_I R_y \mathsf{H}(m))} \\
&\qquad = e(g_1^{r_{(I,1)} r_u' r_u (x_I + y_I R_y \mathsf{H}(m))}, g_2) \\
&\qquad = e(\sigma_1', g_2)
\end{aligned}$$

$$\begin{aligned}
&e(h_2^{'}, X^{(i)'} Y_2^{(i)'\mathsf{H}(\mathsf{H}(m))}) \\
&\qquad = e(g_1^{r_{(I,2)} r_u^{''}}, g_2^{x_I r_u + r_u y_I R_y \mathsf{H}(\mathsf{H}(m))}) \\
&\qquad = e(g_1, g_2)^{r_{(I,2)} r_u^{''} r_u (x_I + y_I R_y \mathsf{H}(\mathsf{H}(m)))} \\
&\qquad = e(g_1^{r_{(I,2)} r_u^{''} r_u (x_I + y_I R_y \mathsf{H}(\mathsf{H}(m)))}, g_2) \\
&\qquad = e(\sigma_2', g_2)
\end{aligned}$$

□

C.6 EUF-CMA Proof

We prove that the signature scheme presented in Section 5.7.1 has the EUF-CMA property. Here is a summary of the steps of the proof:

1. We use Appendix C.3 proof to represent the commitment the adversary makes to the issuer's keys;
2. We present the elements of the signature as polynomials, as a function of the elements published by the verifier and the issuers;
3. We reduce these polynomials; and
4. We analyze the reduced polynomials, and we prove that if the adversary is able to output such signature, either he is also able to invert a hash funcion, or he is using an already queried signature.

As a preliminary, we give the notations we will use to discuss a forged signature. This notation is used for a potential σ^* value, which is a forged signature on a message m^* never queried to one of the verifier's trusted issuer's signing oracle. We note this signature:

$EUF - CMA_1\mathcal{A}$

1 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \text{KeyGen}(1^\lambda)$;
2 : $(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}\mathsf{Sign}(\mathsf{sk},\bullet)}(\mathsf{pk})$;
3 : If $\text{Verify}(pk, m^*, \sigma^*) = 1$ and $m^* \notin M$:
4 : **return** 1;
5 : Else:
6 : **return** 0;

Figure C.6 – EUF-CMA game. M represents the already queried messages.

$$\begin{aligned} \sigma^* &= (h_1^*, h_2^*, \sigma_1^*, \sigma_2^*) \\ &= (g_1^{s_1}, g_1^{s_2}, g_1^{s_{\sigma_1}}, g_1^{s_{\sigma_2}}) \end{aligned} \tag{C.7}$$

Theorem C.1. Given the tuple $(g_1, Y_1, Y_2, X, \bar{X_1}, \bar{Y_1})$, and given access to a signing oracle, that can sign messages on behalf of all issuers, no adversary can output, with non negligible probability, a signature $(h_1^*, h_2^*, \sigma_1^*, \sigma_2^*, X^{(I_1)*}, Y_2^{(I_2)*})$ that was not queried to the signing oracle, and that verifies:

$$e(h_1^*, X^{(I_1)*}(Y_2^{(I_2)*})^{\mathsf{H}(m^*)}) = e(\sigma_1^*, g_2)$$
$$e(h_2^*, X^{(I_1)*}(Y_2^{(I_2)*})^{\mathsf{H}(\mathsf{H}(m^*))}) = e(\sigma_2^*, g_2)$$

Remark C.3. Theorem C.1 represents commitment to x and y issuer's keys as independent. Indeed, the two aggregator, and therefore the two associated set-membership proofs are independent. At this point of the proof, we can not know if I_1 and I_2 are equal or different. We will see later that they are indeed equal.

Proof. We will prove Theorem C.1 by contradiction.

We present in Figure C.6 the EUF-CMA property as a game, for a generic signature scheme. We define M as the set of messages already queried to the oracle. $\mathcal{O}\mathsf{Sign}(\mathsf{sk}, \bullet)$ represents the access to the signing oracle. Any query to the oracle outputs k different signature, issued by the k different trusted issuers.

The advantage of the adversary in the EUF-CMA game is:

$$\mathsf{Adv}_{\Sigma}^{\mathsf{eufcma}}(\mathcal{A}) = |\Pr[S_1] - (p-1)^{-1}| \tag{C.8}$$

Where $\Pr[S_i] = \Pr_i \mathcal{A} = 1$.

$EUF - CMA_2\mathcal{A}$

1 : $SK_x = (x_1, ..., x_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
2 : $SK_y = (y_1, ..., y_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
3 : $X = (X^{(1)}, \cdots, X^{(k)}) = (g_2^{x_1}, \cdots, g_2^{x_k})$
4 : $Y_1 = (Y_1^{(1)}, \cdots, Y_1^{(k)}) = (g_1^{y_1}, \cdots, g_1^{y_k})$
5 : $Y_2 = (Y_2^{(1)}, \cdots, Y_2^{(k)}) = (g_2^{y_1}, \cdots, g_2^{y_k})$
6 : $\bar{X}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
7 : $\bar{Y}_1 = (\bar{Y}_1^{(1)}, \cdots, \bar{Y}_1^{(k)}) = (g_1^{\frac{1}{y_1}}, \cdots, g_1^{\frac{1}{y_k}})$
8 : $PK = \{X, \bar{X}_1, Y_1, \bar{Y}_1, Y_2\}$
9 : $(m^*, h_1^*, h_2^*, \sigma_1^*, \sigma_2^*,$
10 : $\quad X^{(I_1)*}, Y^{(I_2)*}) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}\mathsf{Sign}(\mathrm{SK_x}, \mathrm{SK_y}, \bullet)}(PK)$;
11 : $a_1 = e(h_1^*, X^{(I_1)*}(Y_2^{(I_2)*})^{\mathsf{H}(m)^*})$
12 : $a_2 = e(\sigma_1^*, g_2)$
13 : $a_3 = e(h_2^*, X^{(I_1)*}(Y_2^{(I_2)*})^{\mathsf{H}(\mathsf{H}(m))^*})$
14 : $a_4 = e(\sigma_2^*, g_2)$
15 : If $a_1 = a_2 \wedge a_3 = a_4 \wedge m^* \notin M$:
16 : $\quad$ **return** 1;
17 : Else:
18 : $\quad$ **return** 0;

Figure C.7 – EUF-CMA game - notation modifications. M represents the already queried messages.

We modify the generic game presented in Figure C.6 with the values of our protocol. The new game is presented in Figure C.7.

We rewrite this game, using the result of Theorem 5.2, *i.e.*, $X^{(I_1)*} = g_2^{x_{I_1}\eta_x}$ and $Y_2^{(I_2)*} = g_2^{y_{I_2}\eta_y}$. Where I_1 and I_2 represent two verifier's trusted issuers, and η_x and η_y represent two random values known by the adversary. We also use the notations from Equation (C.7). We assume the adversary makes n queries to the Signing Oracles. This new game is represented in Figure C.8.

The difference between probability of success of the game $\mathsf{EUF} - \mathsf{CMA}_3\mathcal{A}$ and the game $\mathsf{EUF} - \mathsf{CMA}_2\mathcal{A}$ is the probability for the Adversary to solve the problem defined in Theorem 5.2:

$$|\Pr[S_3] - \Pr[S_2]| \leq 2 \cdot \epsilon_{\mathrm{CDH}} + \epsilon_{\mathrm{DL}}$$

We develop the equations in line 13 from the game $\mathsf{EUF} - \mathsf{CMA}_3\mathcal{A}$. The goal is to represent these equations as polynomials as a function of public values given to the adversary. With $h^* = \mathsf{H}(m^*)$:

$EUF - CMA_3\mathcal{A}$

1 : $SK_x = (x_1, ..., x_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
2 : $SK_y = (y_1, ..., y_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
3 : $X = (X^{(1)}, ..., X^{(k)}) = (g_2^{x_1}, ..., g_2^{x_k})$
4 : $Y_1 = (Y_1^{(1)}, \cdots, Y_1^{(k)}) = (g_1^{y_1}, \cdots, g_1^{y_k})$
5 : $Y_2 = (Y_2^{(1)}, \cdots, Y_2^{(k)}) = (g_2^{y_1}, \cdots, g_2^{y_k})$
6 : $\bar{X}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
7 : $\bar{Y}_1 = (\bar{Y}_1^{(1)}, \cdots, \bar{Y}_1^{(k)}) = (g_1^{\frac{1}{y_1}}, \cdots, g_1^{\frac{1}{y_k}})$
8 : $PK = \{X, \bar{X}_1, Y_1, \bar{Y}_1, Y_2\}$
9 : $(m^*, g_1^{s_1}, g_1^{s_2}, g_1^{s_{\sigma_1}}, g_1^{s_{\sigma_1}}, X^{(I_1)*} = g_2^{x_{I_1}\eta_x},$
10 : $\quad Y^{(I_2)*} = g_2^{y_{I_2}\eta_y}, \eta_x, \eta_y,) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}\mathsf{Sign}(\mathrm{SK_x}, \mathrm{SK_y}, \bullet)}(PK)$;
11 : $a_1 = e(g_1^{s_1}, g_2^{x_{I_1}\eta_x} g_2^{\mathsf{H}(m^*) y_{I_2}\eta_y})$
12 : $a_2 = e(g_1^{s_{\sigma_1}}, g_2)$
13 : $a_3 = e(g_1^{s_2}, g_2^{x_{I_1}\eta_x} g_2^{y_{I_2}\mathsf{H}(\mathsf{H}(m^*))\eta_y})$
14 : $a_4 = e(g_1^{s_{\sigma_2}}, g_2)$
15 : If $a_1 = a_2 \wedge a_3 = a_4 \wedge m^* \notin M$:
16 : $\quad$ **return** 1;
17 : Else:
18 : $\quad$ **return** 0;

Figure C.8 – EUF-CMA game - Theorem 5.2 modifications

$$
\begin{aligned}
&\begin{cases} a_1 = a_2 \\ a_3 = a_4 \end{cases} \\
\Leftrightarrow &\begin{cases} e(g_1^{s_1}, g_2^{x_{I_1}\eta_x} g_2^{h^* y_{I_2}\eta_y}) = e(g_1^{s_{\sigma_1}}, g_2) \\ e(g_1^{s_2}, g_2^{x_{I_1}\eta_x} g_2^{y_{I_2}\mathsf{H}(h^*)\eta_y}) = e(g_1^{s_{\sigma_2}}, g_2) \end{cases} \\
\Leftrightarrow &\begin{cases} g_T^{s_1(x_{I_1}\eta_x + h^* y_{I_2}\eta_y)} = g_T^{s_{\sigma_1}} \\ g_T^{s_2(x_{I_1}\eta_x + y_{I_2}\mathsf{H}(h^*)\eta_y)} = g_T^{s_{\sigma_2}} \end{cases} \\
\Rightarrow &\begin{cases} s_{\sigma_1} = s_1(x_{I_1}\eta_x + y_{I_2} h^* \eta_y) \pmod{p-1} \\ s_{\sigma_2} = s_2(x_{I_1}\eta_x + y_{I_2}\mathsf{H}(h^*)\eta_y) \pmod{p-1} \end{cases}
\end{aligned} \tag{C.9}
$$

The adversary outputs $g_1^{s_1}, g_1^{s_{\sigma_1}}, g_1^{s_2}, g_1^{s_{\sigma_2}}$ in $\mathbb{G}_1$. Thus these elements must be computed using element in $\mathbb{G}_1$ – because there is no efficiently computable morphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ and from $\mathbb{G}_T$ to $\mathbb{G}_1$, under type-3 pairing assumption. The elements known to the adversary in $\mathbb{G}_1$ after n queries to the Signing Oracles, are:

$$
\begin{aligned}
E = \Big(& g_1, Y_1, \bar{X}_1, \bar{Y}_1, \{g_1^{(r_{(I_j,1)})_j}\}_{j=1}^n, \{g_1^{(r_{I_j,2})_j}\}_{j=1}^n, \\
& \left\{ g_1^{r_{I_j,1}(x_{I_j} + h_j y_{I_j}(R_y)_j)} \right\}_{j=1}^n, \\
& \left\{ g_1^{r_{I_j,2}(x_{I_j} + \mathsf{H}(h_j) y_{I_j}(R_y)_j)} \right\}_{j=1}^n \Big).
\end{aligned}
$$

In our case we want to express s_1, s_2, s_{σ_1} and s_{σ_2} as polynomials, as a function of the values unknown to the adversary. First, we represent an ad hoc element $g_1^{z_\alpha}$ in the form of a polynomial as a function of the values unknown to the adversary under DL assumption.

We then represent each signature's elements (*i.e.*, $s_1, s_2, s_{\sigma_1}, s_{\sigma_2}$) in the same way as we did with the ad hock element $g_1^{z_\alpha}$.

We assume the adversary makes n queries to the Signing Oracles. Therefore, he receives n signatures from the issuer I_1, and n signatures from the issuer I_2. The variable's factors potentially added by the adversary are represented by a $11 \times n$ matrix α, and an additional factor w_α.

Here is the representation of $g_1^{z_\alpha}$:

$$\begin{aligned} g_1^{z_\alpha} =& g_1^{w_\alpha} \cdot \prod_{j=1}^{k} \left((\bar{X}_1^{(j)})^{\alpha_{(1,j)}} \cdot (\bar{Y}_1^{(j)})^{\alpha_{(2,j)}} \cdot (Y_1^{(j)})^{\alpha_{(3,j)}} \right) \\ & \cdot \prod_{i=1}^{n} \Big(h_{(1,I_1)})_i^{\alpha_{(4,i)}} \cdot (h_{(1,I_2)})_i^{\alpha_{(5,i)}} \cdot (h_{(2,I_1)})_i^{\alpha_{(6,i)}} \\ & \cdot (h_{(2,I_2)})_i^{\alpha_{(7,i)}} \cdot (\sigma_{(1,I_1)})_i^{\alpha_{(8,i)}} \cdot (\sigma_{(2,I_1)})_i^{\alpha_{(9,i)}} \\ & \cdot (\sigma_{(1,I_2)})_i^{\alpha_{(10,i)}} \cdot (\sigma_{(2,I_2)})_i^{\alpha_{(11,i)}} \Big) \end{aligned}$$

$$\begin{aligned} \Rightarrow z_\alpha =& w_\alpha + \sum_{j=1}^{k} \left(\alpha_{(1,j)} \frac{1}{x_j} + \alpha_{(1,j)} \frac{1}{y_j} + \alpha_{(3,j)} y_j \right) \\ & + \sum_{i=1}^{n} \Big((r_{(I_1,1)})_i \alpha_{(4,i)} + (r_{(I_2,1)})_i \alpha_{(5,i)} \\ & + (r_{(I_1,2)})_i \alpha_{(6,i)} + (r_{(I_2,2)})_i \alpha_{(7,i)} \\ & + (r_{(I_1,2)})_i \alpha_{(8,i)} (x_{I_1} + R_y y_{I_1} h_i) \\ & + (r_{(I_1,2)})_i \alpha_{(9,i)} (x_{I_1} + R_y y_{I_1} \mathsf{H}(h_i)) \\ & + (r_{(I_2,1)})_i \alpha_{(10,i)} (x_{I_2} + R_y y_{I_2} h_i) \\ & + (r_{(I_2,2)})_i \alpha_{(11,i)} (x_{I_2} + R_y y_{I_2} \mathsf{H}(h_i)) \Big) \\ & \pmod{p-1} \end{aligned} \tag{C.10}$$

Remark C.4. If $n > k$, columns 1 to 3 of matrix α are shorter than the other columns. We assume these columns are filed with zeroes to match the size of the matrix. If $k > n$ then it is the columns from 4 to 11 which are filled with zeroes.

Furthermore, we assume that all R_y values are similar. This does not change the validity of the proof, and it simplifies the notations.

Lemma C.3. The probability for the adversary to output an element in $\mathbb{G}_1$ different from the z_α representation is inferior or equal to the probability of winning in one instance of the CDH game.

Proof. We prove this lemma by contradiction. We assume the adversary is able to output something different from representation presented in Equation (C.10). This is equivalent to say the adversary has access to an Oracle $O(g_1, g_1^{z_1}, ..., g_1^{z_{3k}}, g_1^{z_{3k+1}}, ..., g_1^{z_{3k+1+8n}})$. With

$z_i, i \in \{1, ..., 3k+1+8n\}$, the variables unknown to the adversary. This oracle outputs with non negligible probability an element $g_1^{z_\alpha + z_a z_b}$, where $a, b \in \{1, ..., 3k+1+8n\}$. This oracle allows the adversary to output $g_1^{z_a f(z_b)}$, with f a polynomial function, and $\deg f_(z_b) \neq 0$. Thus, for any given CDH problem $(g_1, g_1^{z_a}, g_1^{z_b})$, an adversary with such an oracle can set all variables to 0, except z_a and z_b, and the oracle solves an instance of the CDH problem. □

We represents s_1, s_2, s_{σ_1}, and s_{σ_2} in the same way we represented z_α in Equation (C.10). Then, we replace s_1, s_2, s_{σ_1}, and s_{σ_2} in Equation (C.9) with this new representation.

Factor matrices of s_1, s_2, s_{σ_1}, and s_{σ_2} representations (*i.e.*, the α matrices in Equation (C.10)) will be written respectively a, b, c and d. To simplify this discussion, we will assume the transformations applied to the first part of Equation (C.9) can be applied symmetrically to the second part of the equation.

Let us take:

$$s_{\sigma_1} = s_1(x_{I_1}\eta_x + y_{I_2}h^*\eta_y) \tag{C.11}$$

If s_{σ_1} and s_1 validates Equation (C.11), then either $s_{\sigma_1} = 0$ and $s_1 = 0$, or s_{σ_1} and s_1 depends on the variables $x_{I_1}, x_{I_2}, y_{I_1}, y_{I_2}, (r_{(I_1,1)})_i, (r_{(I_1,2)})_i, (r_{(I_2,1)})_i$, and $(r_{(I_2,2)})_i$, $\forall i \in \{1, \cdots, n\}$. However $s_{\sigma_1} = 0$ or $s_1 = 0$ is a rejecting condition, thus s_{σ_1} and s_1 depends on the variables of the equation. Under DL and CDH assumptions, the adversary has a negligible probability to output a value a, b, c or d that depends on these variables. Therefore, we can reduce s_{σ_1} and s_1 to:

$$\begin{aligned} s_1 = \sum_{i=1}^{n} \Big(& (r_{(I_1,1)})_i a_{(4,i)} + (r_{(I_2,1)})_i a_{(5,i)} \\ & + (r_{(I_1,2)})_i a_{(6,i)} + (r_{(I_2,2)})_i a_{(7,i)} \Big) \end{aligned} \tag{C.12}$$

$$\begin{aligned} s_{\sigma_1} = \sum_{i=1}^{n} \Big(& (r_{(I_1,1)})_i c_{(8,i)}(x_{I_1} + R_y y_{I_1} h_i) \\ & + (r_{(I_1,2)})_i c_{(9,i)}(x_{I_1} + R_y y_{I_1} \mathsf{H}(h_i)) \\ & + (r_{(I_2,1)})_i c_{(10,i)}(x_{I_2} + R_y y_{I_2} h_i) \\ & + (r_{(I_2,2)})_i c_{(11,i)}(x_{I_2} + R_y y_{I_2} \mathsf{H}(h_i)) \Big) \end{aligned} \tag{C.13}$$

Now that we simplified representations of s_1 and s_{σ_1} we want to study the possibility for I_1 and I_2 to represent two distinct issuers.

We see that right side of Equation (C.11) contains elements $(r_{(I_1,1)})_i$ and $(r_{(I_1,2)})_i$ multiplied by

y_{I_2}. At the same time, there are also elements $(r_{(I_2,1)})_i$ and $(r_{(I_2,2)})_i$ multiplied by x_{I_1}. These terms does not exist on the left side of the equation (in s_{σ_1}). We can deduce that, either the whole equation is reduced to zero, which is impossible since $g_1^{s_1} \neq 1_{\mathbb{G}_1}$, or, $I_1 = I_2 = I$. This means that all elements come from only one issuer.

We represent Equation (C.11) with these new representations of s_{σ_1} and s_1.

Remark C.5. In the Equation (C.14), c'_1 is the concatenation of c_8 and c_{10}, $c'_1 = \binom{c_8}{c_{10}}$. c'_2 is the concatenation of row c_9 and row c_{11}, $c'_2 = \binom{c_9}{c_{11}}$. a'_1 is the concatenation of row a_4 and row a_5, $a'_1 = \binom{a_4}{a_5}$. And a'_2 is the concatenation of row a_6 and row a_7, $a'_2 = \binom{a_6}{a_7}$.

$$
\begin{aligned}
&\sum_{i=1}^{2n} \Big((r_{(I,1)})_i c'_{(1,i)} (x_I + R_y y_I h_i) \\
&+ (r_{(I,2)})_i c'_{(2,i)} (x_I + R_y y_{I_1} \mathsf{H}(h_i)) \Big) \\
&= \sum_{i=1}^{2n} \Big((r_{(I,1)})_i a'_{(1,i)} \\
&+ (r_{(I,2)})_i a'_{(2,i)} \Big) (x_I \eta_x + y_I h^* \eta_y) \\
\Leftrightarrow &\sum_{i=1}^{2n} \Big(x_I ((r_{(I,1)})_i (c'_{(1,i)} - a'_{(1,i)} \eta_x) \\
&+ (r_{(I,2)})_i (c'_{(2,i)} - a'_{(2,i)} \eta_x)) \\
&+ y_I ((r_{(I,1)})_i (R_y c'_{(1,i)} h_i - a'_{(1,i)} \eta_y h^*) \\
&+ (r_{(I,2)})_i (R_y c'_{(2,i)} \mathsf{H}(h_i) - a'_{(2,i)} \eta_y h^*) \Big) \\
&= 0
\end{aligned}
\tag{C.14}
$$

In Equation (C.14), factors a' and c' have negligible probability to be function of the variables (*i.e.*, r, x and y), as this would imply the adversary is able to break DL or CDH problem, from the Lemma C.3 result. This directly implies that each part of the above sum is independently equal to zero.

We obtain the system of equations:

$$
\begin{cases}
c'_{(1,1)} - a'_{(1,1)}\eta_x = 0 \\
\qquad\qquad \dots \\
c'_{(1,2n)} - a'_{(1,2n)}\eta_x = 0 \\
c'_{(2,1)} - a'_{(2,1)}\eta_x = 0 \\
\qquad\qquad \dots \\
c'_{(2,2n)} - a'_{(2,2n)}\eta_x = 0 \\
R_y c'_{(1,1)} h_1 - a'_{(1,1)}\eta_y h^* = 0 \\
\qquad\qquad \dots \\
R_y c'_{(1,2n)} h_{2n} - a'_{(1,2n)}\eta_y h^* = 0 \\
R_y c'_{(2,1)} \mathsf{H}(h_1) - a'_{(2,1)}\eta_y h^* = 0 \\
\qquad\qquad \dots \\
R_{I,y,2n} c'_{2,2n} \mathsf{H}(h_{2n}) - a'_{2,2n}\eta_y h^* = 0
\end{cases}
\tag{C.15}
$$

We write : $A'_1 = \begin{pmatrix} a'_{(1,1)} & \dots & 0 \\ & \ddots & \\ 0 & \dots & a'_{(1,2n)} \end{pmatrix}$; $A'_2 = \begin{pmatrix} a'_{(2,1)} & \dots & 0 \\ & \ddots & \\ 0 & \dots & a'_{(2,2n)} \end{pmatrix}$; $C'_1 = \begin{pmatrix} c'_{(1,1)} & \dots & 0 \\ & \ddots & \\ 0 & \dots & c'_{(1,2n)} \end{pmatrix}$; $C'_2 = \begin{pmatrix} c'_{(2,1)} & \dots & 0 \\ & \ddots & \\ 0 & \dots & c'_{(2,2n)} \end{pmatrix}$; $D'_1 = \begin{pmatrix} d'_{(1,1)} & \dots & 0 \\ & \ddots & \\ 0 & \dots & d'_{(1,2n)} \end{pmatrix}$; $D'_2 = \begin{pmatrix} d'_{(2,1)} & \dots & 0 \\ & \ddots & \\ 0 & \dots & d'_{(2,2n)} \end{pmatrix}$; $R_y = \begin{pmatrix} R_y & \dots & 0 \\ & \ddots & \\ 0 & \dots & R_y \end{pmatrix}$; $H = \begin{pmatrix} h_1 & \dots & 0 \\ & \ddots & \\ 0 & \dots & h_{2n} \end{pmatrix}$; and $\mathsf{H}(H) = \begin{pmatrix} \mathsf{H}(h_1) & \dots & 0 \\ & \ddots & \\ 0 & \dots & \mathsf{H}(h_{2n}) \end{pmatrix}$.

We only use diagonal matrices, thus, each matrix is inversible, and the product between two matrices is commutative. We now have the linear system:

$$\begin{cases} C'_1 - A'_1\eta_x = 0 \\ C'_2 - A'_2\eta_x = 0 \\ R_y C'_1 H - A'_1\eta_y h^* = 0 \\ R_y C'_2 \mathsf{H}(H) - A'_2\eta_y h^* = 0 \end{cases} \tag{C.16}$$

$$\Leftrightarrow \begin{cases} C'_1 = A'_1\eta_x \\ C'_2 = A'_2\eta_x \\ R_y C'_1 H = A'_1\eta_y h^* \\ R_y C'_2 \mathsf{H}(H) = A'_2\eta_y h^* \end{cases} \tag{C.17}$$

$$\Leftrightarrow \begin{cases} \frac{1}{\eta_x} C'_1 = A'_1 \\ \frac{1}{\eta_x} C'_2 = A'_2 \\ R_y C'_1 H = \frac{1}{\eta_x} C'_1\eta_y h^* \\ R_y C'_2 \mathsf{H}(H) = \frac{1}{\eta_x} C'_2\eta_y h^* \end{cases} \tag{C.18}$$

We assume all elements on the diagonal of C'_1 and C'_2 are different from 0:

$$\begin{aligned} &\Leftrightarrow \begin{cases} \frac{1}{\eta_x} C'_1 = A'_1 \\ \frac{1}{\eta_x} C'_2 = A'_2 \\ R_y H = \frac{1}{\eta_x}\eta_y h^* I \\ R_y \mathsf{H}(H) = \frac{1}{\eta_x}\eta_y h^* I \end{cases} \qquad \Leftrightarrow \begin{cases} \frac{1}{\eta_x} C'_1 = A'_1 \\ \frac{1}{\eta_x} C'_2 = A'_2 \\ \frac{\eta_x}{\eta_y} R_y H = h^* I \\ \frac{\eta_x}{\eta_y} R_y \mathsf{H}(H) = h^* I \end{cases} \\ &\Leftrightarrow \mathsf{H}(H) = H \end{aligned} \tag{C.19}$$

This last equation is impossible with a collision resistant hash function. This means that our hypothesis, is wrong, *i.e.*, $C'_1 = 0$ or $C'_2 = 0$.

Remark C.6. We proved here that there is at least one element equal to 0 in C'_1 or C'_2, but we can then reduce the matrices to be of size $2n - 1 \times 2n - 1$, and have the same result, until C'_1 or C'_2 is totally reduced to 0.

Remark C.7. From Theorem 5.2, we know that η_x and η_y are different from zero. This implies that $(C'_1 = 0) \Rightarrow (A'_1 = 0)$, and $(C'_2 = 0) \Rightarrow (A'_2 = 0)$. From this, we deduce that C'_1 and C'_2 can't be reduced to 0 at the same time, as this would imply that $s_1 = 0$, which is a rejecting condition for the verifier.

Symmetrically, we deduce the same thing from second line of Equation (C.9):

$$\begin{cases} \frac{1}{\eta_x} D'_1 = B'_1 \\ \frac{1}{\eta_x} D'_2 = B'_2 \\ R_y D'_1 H = \frac{1}{\eta_x} D'_1 \eta_y \mathsf{H}(h^* I) \\ R_y D'_2 \mathsf{H}(H) = \frac{1}{\eta_x} D'_2 \eta_y \mathsf{H}(h^* I) \end{cases} \tag{C.20}$$

(C.21)

Thus $D'_1 = 0$ or $D'_2 = 0$, and D'_1 and D'_2 cannot be both reduced to 0 at the same time.

There are four cases to study: $(C'_1, D'_1) = (0,0)$, $(C'_1, D'_2) = (0,0)$, $(C'_2, D'_1) = (0,0)$, and $(C'_2, D'_2) = (0,0)$.

First, $(C'_1, D'_1) = (0,0)$ and $(C'_2, D'_2) = (0,0)$ lead back to $H = \mathsf{H}(H)$, which is impossible.

Then, case $(C'_2, D'_1) = (0,0)$ gives us the system:

$$\begin{cases} \frac{\eta_x}{\eta_y} R_y H = h^* I \\ \frac{\eta_x}{\eta_y} R_y \mathsf{H}(H) = \mathsf{H}(h^* I) \end{cases} \tag{C.22}$$

If we apply the hash function, we obtain:

$$\mathsf{H}(\frac{\eta_x}{\eta_y} R_y H) = \frac{\eta_x}{\eta_y} R_y \mathsf{H}(H)$$

There are two cases:

1. $\frac{\eta_y}{\eta_x} = R_y$
2. $\frac{\eta_y}{\eta_x} \neq R_y$

Case 1 tells us that $h^* I = H$. In this case, the signature is a signature on an already signed message. Case 2 implies that the adversary is able to find a collision in a collision resistant hash function. In the Random Oracle Model, this has a negligible probability to occure.

Finally, case $(C'_1, D'_2) = (0,0)$ gives us the system:

$$\begin{cases} \frac{\eta_x}{\eta_y} R_y \mathsf{H}(H) = h^* I \\ \frac{\eta_x}{\eta_y} R_y H = \mathsf{H}(h^*) I \end{cases} \tag{C.23}$$

If we apply the hash function, we obtain:

$EUF - CMA_4\mathcal{A}$

1 : $(x_1, ..., x_k) \leftarrow_\$ (\mathbb{Z}_p^*)^k$;
2 : $(y_1, ..., y_k) \leftarrow_\$ (\mathbb{Z}_p^*)^k$;
3 : $X = (X^{(1)}, ..., X^{(k)}) = (g_2^{x_1}, ..., g_2^{x_k})$
4 : $Y_1 = (Y_1^{(1)}, \cdots, Y_1^{(k)}) = (g_1^{y_1}, \cdots, g_1^{y_k})$
5 : $Y_2 = (Y_2^{(1)}, \cdots, Y_2^{(k)}) = (g_2^{y_1}, \cdots, g_2^{y_k})$
6 : $\bar{X}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
7 : $\bar{Y}_1 = (\bar{Y}_1^{(1)}, \cdots, \bar{Y}_1^{(k)}) = (g_1^{\frac{1}{y_1}}, \cdots, g_1^{\frac{1}{y_k}})$
8 : $PK = \{X, \bar{X}_1, Y_1, \bar{Y}_1, Y_2\}$
9 : $(\eta_x, \eta_y, R_y, h*, H) \leftarrow_\$ \mathcal{A}(PK)$;
10 : $t_1 = \frac{\eta_x}{\eta_y} R_y H$
11 : $t_2 = \frac{eta_x}{\eta_y} R_y \mathsf{H}(H)$
12 : $t_3 = \mathsf{H}(\frac{\eta_x}{\eta_y} R_y \mathsf{H}(H))$
13 : $t_4 = \mathsf{H}(\frac{\eta_x}{\eta_y} R_y H)$
14 : $t_5 = \mathsf{H}(H)$
15 : If $(t_1 = t_3 \vee t_2 = t_4 \vee H = t_5 \vee H = h^* I) \wedge m^* \notin M$:
16 : **return** 1;
17 : Else:
18 : **return** 0;

Figure C.9 – EUF-CMA game - reduction

$$\mathsf{H}(\frac{\eta_x}{\eta_y} R_y \mathsf{H}(H)) = \frac{\eta_x}{\eta_y} R_y H$$

This is equivalent for adversary to find a collision in a collision resistant hash function.

The study of the four cases imply that succeeding in $(\mathsf{EUF} - \mathsf{CMA}_3\mathcal{A})$ implies that either the adversary is able to find a collision in a collision resistant hash function, or the message signed was already queried to the singing oracle. We represent this as a new game transition. This transition is represented in Figure C.9.

The transition between $\mathsf{EUF} - \mathsf{CMA}_3\mathcal{A}$ and $\mathsf{EUF} - \mathsf{CMA}_4\mathcal{A}$ is a failure based transition, where the failure is equivalent to the probability for the adversary to break Lemma C.3.

$$\begin{aligned} |\Pr[S_3] - \Pr[S_2]| &\leq F \\ &\leq \epsilon_{\text{CDH}} \end{aligned}$$

$EUF - CMA_5\mathcal{A}$

1 : $(x_1, ..., x_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
2 : $(y_1, ..., y_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
3 : $X = (X_1, ..., X_k) = (g_2^{x_1}, ..., g_2^{x_k})$
4 : $Y_1 = (Y_1^{(1)}, \cdots, Y_1^{(k)}) = (g_1^{y_1}, \cdots, g_1^{y_k})$
5 : $Y_2 = (Y_2^{(1)}, \cdots, Y_2^{(k)}) = (g_2^{y_1}, \cdots, g_2^{y_k})$
6 : $\bar{X}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
7 : $\bar{Y}_1 = (\bar{Y}_1^{(1)}, \cdots, \bar{Y}_1^{(k)}) = (g_1^{\frac{1}{y_1}}, \cdots, g_1^{\frac{1}{y_k}})$
8 : $PK = \{X, \bar{X}_1, Y_1, \bar{Y}_1, Y_2\}$
9 : $(\eta_x, \eta_y, R_y, h*, H) \leftarrow\!\!\$\ \mathcal{A}(PK)$;
10 : $t_1 = \frac{\eta_x}{\eta_y} R_y H$
11 : $t_2 = \frac{eta_x}{\eta_y} R_y \mathsf{H}(H)$
12 : $t_3 \leftarrow\!\!\$\ \{1, \cdots, p-1\}$
13 : $t_4 \leftarrow\!\!\$\ \{1, \cdots, p-1\}$
14 : $t_5 \leftarrow\!\!\$\ \{1, \cdots, p-1\}$
15 : If $(t_1 = t_3 \vee t_2 = t_4 \vee H = t_5 \vee H = h^*I) \wedge m^* \notin M$:
16 : **return** 1;
17 : Else:
18 : **return** 0;

Figure C.10 – EUF-CMA game - Random Oracle Model

In the Random Oracle Model, we represent the hash function as a Random Oracle. We build a new transition presented in Figure C.10. The new game is called $\mathsf{EUF} - \mathsf{CMA}_5\mathcal{A}$.

This transition is a representation of $\mathsf{EUF} - \mathsf{CMA}_4\mathcal{A}$ in the Random Oracle model. Thus $\Pr[S_5] = \Pr[S_4]$.

We develop the success condition of the game presented in Figure C.10:

$$\begin{aligned} & (t_1 = t_3 \vee t_2 = t_4 \vee H = t_5 \vee H = h^*I) \wedge h^* \notin M \\ \Leftrightarrow & (t_1 = t_3 \wedge h^* \notin M) \vee (t_2 = t_4 \wedge h^* \notin M) \\ & \vee (H = t_5 \wedge h^* \notin M) \vee (H = h^*I \wedge h^* \notin M) \end{aligned} \tag{C.24}$$

We need to analyze each condition individually. The first three conditions are true if the values t_1 or t_2 chosen by the adversary are equal to a number, chosen uniformly at random in $\{1, \cdots, p-1\}$. The probability for this condition to be true is $(p-1)^{-1}$. Furthermore, the last condition is clearly a contradiction. It asks h^* to be at the same time different and equal to some previously signed message. We can represent the condition presented in Equation (C.24) as:

$$\Leftrightarrow((p-1)^{-1}) \vee ((p-1)^{-1}) \vee ((p-1)^{-1}) \vee (0)$$

We can deduce that, the probability for an adversary to succeed in $\mathsf{Game}_4\mathcal{A}$ is $3 \cdot p^{-1}$. Now, we can deduce the probability for a PPT adversary to forges the signature scheme:

$$\begin{aligned} |\Pr[S_5] - \Pr[S_1]| &= |\Pr[S_4] - \Pr[S3]| + |\Pr[S_3] - \Pr[S_2]| \\ &\quad + |\Pr[S_2] - \Pr[S_1]| \\ &\leq 3 \cdot \epsilon_{\text{CDH}} + \epsilon_{\text{DL}} \end{aligned}$$

And $\Pr[S_5] = 3 \cdot (p-1)^{-1}$. Thus :

$$\Pr[S_1] \leq 3 \cdot (p-1)^{-1} + 3 \cdot \epsilon_{\text{CDH}} + \epsilon_{\text{DL}}$$

Finally, we deduce that :

$$\begin{aligned} \mathsf{Adv}_{\Sigma}^{\mathsf{eufcma}}(\mathcal{A}) &= |\Pr[S_1] - (p-1)^{-1}| \\ &\leq 2 \cdot (p-1)^{-1} + 3 \cdot \epsilon_{\text{CDH}} + \epsilon_{\text{DL}} \end{aligned}$$

If the order p is sufficiently large, this value is negligible. Therefore, the adversary has a negligible probability of forging this scheme, even with multiple access to the signing oracles. In conclusion, our signature scheme has EUF-CMA property. □

C.7 Issuer-Indistinguishability proof

In this section, we will prove that the issuer of a randomized signature is indistinguishable among the set of trusted issuers of a malicious verifier. We assume the adversary (in this case, the Malicious Verifier) keeps track of multiple **VerifyRandomized** transactions. We also assume the adversary can collude with *all* the issuers, and therefore knows all their private keys, and all (non randomized) issued signatures.

We define an issuer Indistinguishability game, represented in Figure C.11. The adversary succeeds in this game if he is able to distinguish between a re-randomized signature, issued by the issuer I_1, and a signature issued different issuer I_2.

$HiddenIssuerGame_1\mathcal{A}$

1 : $pp \leftarrow_\$ \mathcal{C}(1^\lambda)$
2 : $(\mathsf{sk}_{I_1}, \mathsf{pk}_{I_1}) =$**IssuerKeygen(pp)**
3 : $(\mathsf{sk}_{I_2}, \mathsf{pk}_{I_2}) =$**IssuerKeygen(pp)**
4 : $(vpk, vsk) \leftarrow_\$$ **VerifierSetup**$(pp, \{I_1, I_2\})$
5 : $m \leftarrow_\$ \mathbb{Z}_p^*$
6 : $u \leftarrow_\$ \mathbb{Z}_p^*$
7 : $\sigma_1 =$ **Sign**$(pp, m, \mathsf{sk}_{I_1}, u)$
8 : $\sigma_2 =$ **Sign**$(pp, m, \mathsf{sk}_{I_2}, u)$
9 : $a_1 = (X^{(I_1)'}, Y^{(I_1)'}, \pi_x, \pi_y, \sigma_1') =$
10 : **Randomize**$_R(pp, \mathsf{pk}_{I_1}, vpk,$
11 : $\{I_1, I_2\}, \sigma_1, u)$
12 : $a_2 = (X^{(I_1)''}, Y^{(I_1)''}, \pi_x', \pi_y', \sigma_1'') =$
13 : **Randomize**$'_R(pp, \mathsf{pk}_{I_1}, vpk,$
14 : $\{I_1, I_2\}, \sigma_1, u)$
15 : $b \leftarrow_\$ \{0, 1\}$
16 : if $b = 1$:
17 : $a_2 = (X^{(I_1)'''}, Y^{(I_1)'''}, \pi_x'', \pi_y'', \sigma_1''') =$
18 : **Randomize**$'_R(pp, \mathsf{pk}_{I_2}, vpk,$
19 : $\{I_1, I_2\}, \sigma_2, u)$
20 : $c \leftarrow_\$ \mathcal{A}(m, a_1, a_2, \mathsf{sk}_{I_1}, \mathsf{sk}_{I_2}, \mathsf{pk}_{I_1}, \mathsf{pk}_{I_2}, vpk, vsk, \sigma_1, \sigma_2)$
21 : If $c = b$:
22 : **return** 1;
23 : Else:
24 : **return** 0;

Figure C.11 – Issuer indistinguishability game, where the randomize algorithm uses as random values the R given in subscript.

Remark C.8. The R subscript is here to allows us to fix the random variables of the algorithm **Randomize**. This concerns the values $r_u, r_u', r_u'', r_{(u,x)}$, and $r_{(u,y)}$.

Theorem 5.6. *The signature scheme presented in 5.7.1 is issuer indistinguishable. No* PPT *adversary can win the game presented in Figure C.11 with probability greater than* $\frac{1}{2} + \epsilon$*, where* ϵ *is a negligible value.*

The game presented in Figure C.11 can be modified to represent the signature built for the second issuer as a modification of the first signature. This modification is given in Figure C.12. The transition between these two games is a notation difference. Indeed, the resulting elements are the same in both cases. Therefore, the probability of success of the second game is the same as the probability of success of the first game.

$HiddenIssuerGame_2\mathcal{A}$

1 : $pp \leftarrow_\$ \mathcal{C}(1^\lambda)$
2 : $(\mathsf{sk}_{I_1}, \mathsf{pk}_{I_1}) =$**IssuerKeygen(pp)**
3 : $(\mathsf{sk}_{I_2}, \mathsf{pk}_{I_2}) =$**IssuerKeygen(pp)**
4 : $(vpk, vsk) \leftarrow_\$$ **VerifierSetup**$(pp, \{I_1, I_2\})$
5 : $m \leftarrow_\$ \mathbb{Z}_p^*$
6 : $u \leftarrow_\$ \mathbb{Z}_p^*$
7 : $\sigma_1 =$ **Sign**$(pp, m, \mathsf{sk}_{I_1}, u)$
8 : $\sigma_2 =$ **Sign**$(pp, m, \mathsf{sk}_{I_2}, u)$
9 : $a_1 = (X^{(I_1)'}, Y^{(I_1)'}, \pi_x, \pi_y, \sigma_1') =$
10 : **Randomize**$_R(pp, \mathsf{pk}_{I_1}, vpk,$
11 : $\{I_1, I_2\}, \sigma_1, u)$
12 : $a_2 = (X^{(I_1)''}, Y^{(I_1)''}, \pi_x', \pi_y', \sigma_1'', \sigma_2'') =$
13 : **Randomize**$'_R(pp, \mathsf{pk}_{I_1}, vpk,$
14 : $\{I_1, I_2\}, \sigma_1, u)$
15 : $b \leftarrow_\$ \{0, 1\}$
16 : if $b = 1$:
17 : $a_2 = ((X^{(I_1)''})^{\frac{x_2}{x_1}}, (Y^{(I_1)''}))^{\frac{y_2}{y_1}}, (W_x)_{I_1}^{''\frac{x_1}{x_2}}, (W_y)_{I_1}^{''\frac{y_1}{y_2}},$
18 : $h_x'', h_y'', \sigma_1^{''\frac{x_2+y_2 R_y m}{x_1+y_1 R_y m}}, \sigma_2^{''\frac{x_2+y_2 R_y \mathsf{H}(m)}{x_1+y_1 R_y \mathsf{H}(m)}}, h_1'', h_2'')$
19 : $c \leftarrow_\$ \mathcal{A}(m, a_1, a_2, \mathsf{sk}_{I_1}, \mathsf{sk}_{I_2}, \mathsf{pk}_{I_1}, \mathsf{pk}_{I_2}, vpk, vsk, \sigma_1, \sigma_2)$
20 : If $c = b$:
21 : **return** 1;
22 : Else:
23 : **return** 0;

Figure C.12 – Issuer indistinguishability game, where the randomize algorithm uses as random values the R given in subscript

The difference between the two possible signatures given to the adversary only depends on x_1, y_1, x_2, y_2, m, R_y, and the random elements added by the user.

We want to compare the re-randomized signature (case $b = 0$), which we will call σ_0, and the signature with the modified issuer (case $b = 1$), which we will call σ_1.

σ_0 and σ_1 are exclusively composed of elements expressed in $\mathbb{G}_1$ and in $\mathbb{G}_2$. This means that the adversary can combine them using the bilinear pairing e. Therefore, from one randomized signature, the adversary is able to compute 40 different combinations in $\mathbb{G}_T$. We represent all these combinations in Table C.1 and in Table C.2. These combinations are represented in term of variables (random elements added by the user), and in term of distinguishing elements. It is to say the elements that distinguish a message signed by one issuer, and the other. These variables and distinguishing elements are $r_u, r'_u, r''_u, r_{(u,x)}, r_{(u,y)}$, x_1, y_1, x_2, y_2, m, R, and R_y.

	g_2	h_x	h_y	$X^{(1)}$	$Y^{(1)}$
g_1	1	$r_u r_{u,x}$	$R_y r_u r_{u,y}$	r_u	$R_y r_u$
h_1	r'_u	$r_u r'_u r_{u,x}$	$R_y r_u r'_u r_{u,y}$	$r_u r'_u$	$R_y r_u r'_u$
h_2	r''_u	$r_u r''_u r_{u,x}$	$R_y r_u r''_u r_{u,y}$	$r_u r''_u$	$R_y r_u r''_u$
W_x	$r_{u,x}$	$r_u r_{u,x}^2$	$R_y r_u r_{u,x} r_{u,y}$	$r_u r_{u,x}$	$R_y r_u r_{u,x}$
W_y	$r_{u,y}$	$r_u r_{u,x} r_{u,y}$	$R_y r_u r_{u,y}^2$	$r_u r_{u,y}$	$R_y r_u r_{u,y}$
σ_1	$r_u r'_u$	$r_u^2 r'_u r_{u,x}$	$R_y r_u^2 r'_u r_{u,y}$	$r_u^2 r'_u$	$R_y r_u^2 r'_u$
σ_2	$r_u r''_u$	$r_u^2 r''_u r_{u,x}$	$R_y r_u^2 r''_u r_{u,y}$	$r_u^2 r''_u$	$R_y r_u^2 r''_u$
u	$(R_y + R)$	$r_u r_{u,x}(R_y + R)$	$R_y r_u r_{u,y}(R_y + R)$	$r_u(R_y + R)$	$R_y r_u(R_y + R)$

Table C.1 – Resulting elements of the rerandomized signature σ_0 after application of the bilinear pairing.

The order of $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ is prime, thus every element of these groups is a generator, except from the neutral element. Therefore, every elements presented in Table C.2 will be seen, from the adversary point of view, as elements chosen uniformly at random in one of these groups. This implies, under DL assumption, that a single element cannot be used by an adversary to decide whether the issuer of a credential is I_1 or I_2. Therefore, the only way an adversary could identify the issuer of a signature would be by comparing elements of the signature.

We can distinguish two types of elements in Table C.1 and in Table C.2: the elements which are function of the variables x_1, y_1, x_2, y_2, and the elements which are not. These variables are the one that the adversary can use to identify the actual issuer of the signature. Using only elements without variables would not allow him to distinguish the issuer, as these elements are equivalent in σ_0 and in σ_1.

In the 40 elements list characterizing the re-randomized signature σ_0, we can see that there are two cells of the tabular that matches with two other cells. These are the elements $e(g_1, h_x) = e(W_x, X^{(1)})$ and $e(g_1, h_y) = e(W_y, Y^{(1)})$. In the σ_1 representation, these matching elements does not depend on the distinguishing variables, and are represented in the same way in σ_0.

	g_2	h_x	h_y
g_1	1	$r_u r_{u,x}$	$R_y r_u r_{u,y}$
h_1	r'_u	$r_u r'_u r_{u,x}$	$R_y r_u r'_u r_{u,y}$
h_2	r''_u	$r_u r''_u r_{u,x}$	$R_y r_u r''_u r_{u,y}$
W_x	$r_{u,x}\frac{x_1}{x_2}$	$r_u r^2_{u,x}\frac{x_1}{x_2}$	$R_y r_u r_{u,x} r_{u,y}\frac{x_1}{x_2}$
W_y	$r_{u,y}\frac{y_1}{y_2}$	$r_u r_{u,x} r_{u,y}\frac{y_1}{y_2}$	$R_y r_u r^2_{u,y}\frac{y_1}{y_2}$
σ_1	$r_u r'_u\frac{x_2+R_y y_2 m}{x_1+R_y y_1 m}$	$r^2_u r'_u r_{u,x}\frac{x_2+R_y y_2 m}{x_1+R_y y_1 m}$	$R_y r^2_u r'_u r_{u,y}\frac{x_2+R_y y_2 m}{x_1+R_y y_1 m}$
σ_2	$r_u r''_u\frac{x_2+R_y y_2 \mathsf{H}(m)}{x_1+R_y y_1 \mathsf{H}(m)}$	$r^2_u r''_u r_{u,x}\frac{x_2+R_y y_2 \mathsf{H}(m)}{x_1+R_y y_1 \mathsf{H}(m)}$	$R_y r^2_u r''_u r_{u,y}\frac{x_2+R_y y_2 \mathsf{H}(m)}{x_1+R_y y_1 \mathsf{H}(m)}$
u	(R_y+R)	$r_u r_{u,x}(R_y+R)$	$R_y r_u r_{u,y}(R_y+R)$
	$X^{(1)}$	$Y^{(1)}$	
g_1	$r_u\frac{x_2}{x_1}$	$R_y r_u\frac{y_2}{y_1}$	
h_1	$r_u r'_u\frac{x_2}{x_1}$	$R_y r_u r'_u\frac{y_2}{y_1}$	
h_2	$r_u r''_u\frac{x_2}{x_1}$	$R_y r_u r''_u\frac{y_2}{y_1}$	
W_x	$r_u r_{u,x}$	$R_y r_u r_{u,x}\frac{y_2}{y_1}\frac{x_1}{x_2}$	
W_y	$r_u r_{u,y}\frac{x_2}{x_1}\frac{y_1}{y_2}$	$R_y r_u r_{u,y}$	
σ_1	$r^2_u r'_u\frac{x_2}{x_1}\frac{x_2+R_y y_2 m}{x_1+R_y y_1 m}$	$R_y r^2_u r'_u\frac{y_2}{y_1}\frac{x_2+R_y y_2 m}{x_1+R_y y_1 m}$	
σ_2	$r^2_u r''_u\frac{x_2}{x_1}\frac{x_2+R_y y_2 \mathsf{H}(m)}{x_1+R_y y_1 \mathsf{H}(m)}$	$R_y r^2_u r''_u\frac{y_2}{y_1}\frac{x_2+R_y y_2 \mathsf{H}(m)}{x_1+R_y y_1 \mathsf{H}(m)}$	
u	$r_u(R_y+R)\frac{x_2}{x_1}$	$R_y r_u(R_y+R)\frac{y_2}{y_1}$	

Table C.2 – Resulting elements of the signature with modified issuer σ_1 after application of the bilinear pairing.

Furthermore, the aggregator's integrity verification algorithm ensures that the different values $e((W_x)_i, X^{(i)}), \forall i \in \{1, \cdots, k\}$ are equal. This means that the adversary cannot distinguish the issuer using this method.

In these tables, there are two other matching expressions. These are the one that are supposed to match in the **VerifyRandomized** algorithm. But these equations are symmetrical in term of distinguishing values, thus they do not distinguish σ_0 and σ_1. We could think, from the representation we made in Table C.1, that $e(\sigma_1, g_2)$ should match with $e(h_1, X^{(I)})$, but σ_1 is factor of an element not represented here for clarity. This element is $(x_1 + R_y y_1 m)$. Indeed, the values match, but the adversary would still need to use the value $e(h_1, Y^{(I)})$ to be able to decide anything.

We can verify, with an algorithm, that each element the adversary as access to in $\mathbb{G}_T$ is distinct from every other one, by at least one random secret element. Or more precisely, there are no linear relationship that links the different element given to the adversary. The only exception are the four cases discussed above, and they do not allow the adversary to distinguish between both cases (*i.e.*, I_1 issued the credential or I_2 issued the credential).

All elements given to the adversary are disjoint in term of the random elements $(R_y, r_{u,x}, r_{u,y}, r_u, r'_u, r''_u, R_y + R)$. $(R_y + R)$ is the element given to the adversary during the

Sign algorithm. This element can be seen as random in $\mathbb{Z}_p$ because of the addition of R which is only used once. It is a Pedersen commitment, which is totally hiding [186].

We know that the adversary can only distinguish between linear expression of the given elements (Lemma C.2). All the elements the adversary has access to are chosen uniformally at random from the adversary point of view, and one by one disjoint in term of random variables. This means that all linear expression of these elements are function of at least one random element. This implies that the adversary cannot distinguish the variation induced by the modification of the issuer unless he is able to break DL assumption or DDH assumption.

Therefore, the adversary is not able to distinguish between σ_0 and σ_1, under DL assumption and DDH assumption. Our scheme is issuer indistinguishable.

Remark C.9. The adversary model takes into account potential replay attack. However two randomized signatures use two set of newly generated random elements. From the above proof, we know that a replay attack would give to the adversary elements that would not help the adversary to build linear relationships between the elements. Thus the scheme is also secure against replay attacks.

C.8 Interactive Protocol

In this section, we give proofs of correctness, collision freedom, and element indistinguishability for the interactive version of HIAC discussed in Section 5.7.2.

C.8.1 Correctness

Lemma C.4. The interactive version of the aggregator presented in Section 5.7.2 keeps it correct.

Proof. We want to prove that the value output $(W)'_I$ of the interactive protocol *Commitment reveal exchange* in Figure 5.1 is the same as the one outputted by the original **Randomize**

algorithm, i.e $(W)'_I = g_1^{r_1 \mathsf{sk}(\frac{1}{X_I})}$. We have:

$$\begin{aligned}
(W)'_I &= (W)''_I \cdot (C''_1)^{-\frac{1}{r_d}} \\
&= (W)''_I \cdot \left((C'_1)^{\frac{1}{r_b}} \cdot (C'_2)^{\frac{1}{r_c}}\right)^{-\frac{1}{r_d}} \\
&= (W)''_I \cdot \left((C_1)^{\mathsf{sk}\frac{1}{r_b} r_d} \cdot (C_2)^{\frac{1}{r_c} r_d}\right)^{-\frac{1}{r_d}} \\
&= (W)''_I \cdot (C_1)^{-\mathsf{sk}\frac{1}{r_b}} \cdot (C_2)^{-\frac{1}{r_c}} \\
&= (W)''_I \cdot g_1^{-r_1 r_b \mathsf{sk}\frac{1}{r_b}} \cdot g_1^{-r_c r_a \frac{1}{r_c}} \\
&= (W)''_I \cdot g_1^{-r_1 \mathsf{sk}} \cdot g_1^{-r_a} \\
&= (W)_I^{r_1} \cdot g_1^{r_a} \cdot g_1^{-r_1 \mathsf{sk}} \cdot g_1^{-r_a} \\
&= g_1^{r_1 \mathsf{sk}(1+\frac{1}{X_I})} \cdot g_1^{r_a} \cdot g_1^{-r_1 \mathsf{sk}} \cdot g_1^{-r_a} \\
&= g_1^{r_1 \mathsf{sk}(\frac{1}{X_I})}
\end{aligned}$$

□

C.8.2 Collision Freedom

Lemma C.5. The interactive version of the aggregator presented in Section 5.7.2 keeps it collision free.

Proof. The goal is to show that the secret sk value keeps the scheme unforgeable, even if the scheme is modified accordingly to Section 5.7.2. To do so, we will analyze a modified version of Theorem 5.2, and use the result of the original theorem, along with the proof that the transformation does not give an advantage to the adversary (in this case, malicious user).

We represent Lemma C.5 as a game, presented in Figure C.13.

This new game is ultimately equivalent to the game used in Theorem 5.2. Indeed, there is the same number of unknown (potentially harmful) elements. And the success condition is also equivalent. We only need to analyze the resulting $(W)^*_I (C^*_1)^{''-(\frac{1}{r_d})}$ value.

Firstly, thanks to the proof of knowledge of the elements $(C_1)^*$ and $(C_2)^*$, we know that these elements cannot contain any of the given (Agg, X_1, X_2) under DL assumption. Secondly, under CDH assumption, and because of the r_d value used by the verifier at the end of the protocol, C''_1 must be a linear combination of C'_1 and C'_2. Otherwise, the resulting value would depend on r_d, and this value did not even exist when the adversary computed the values $(\mathsf{Comm}'(s)(W)''_I, \pi_s)$. Thus if $(W)^*_I (C^*_1)^{''-(\frac{1}{r_d})}$ depends on r_d then the probability for the verification algorithm to output 1 is negligible (in fact, this probability is p^{-1}). Always under CDH assumption, C''_1 only contains sk, r_c and elements known to the adversary. This means that we can represent

$\mathsf{Game}_1\mathcal{A}$

$(x_1, ..., x_k) \leftarrow\!\!\$\ (\mathbb{Z}_p^*)^k$;
$X_1 = (X_1^{(1)}, ..., X_1^{(k)}) = (g_1^{x_1}, ..., g_1^{x_k})$
$X_2 = (X^{(1)}, ..., X^{(k)}) = (g_2^{x_1}, ..., g_2^{x_k})$
$\bar{X}_1 = (\bar{X}_1^{(1)}, \cdots, \bar{X}_1^{(k)}) = (g_1^{\frac{1}{x_1}}, \cdots, g_1^{\frac{1}{x_k}})$
$(W, \mathsf{sk}) \leftarrow\!\!\$\ \mathbf{Gen}(X_1,\ X_2,\ \bar{X}_1)$
$C_\sigma^* = ((C_1^*), (C_2^*),$
$\qquad h^*, (W)_I^*, X_2^{(I)*}) \leftarrow\!\!\$\ \mathcal{A}(X_1, X_2, \bar{X}_1, W)$;
$r_c \leftarrow\!\!\$\ \mathbb{Z}_p^*$
$(C_1^*)' = (C_1^*)_y^{r_c \mathsf{sk}}$
$(C_2^*)' = (C_2^*)_y^{r_c}$
$(C_1^*)'' \leftarrow \mathcal{A}(C_\sigma^*, \mathrm{pg}, X_1, X_2, (C_2^*)', (C_1^*)')$
If $e((W)_I^*(C_1^*)^{''-(\frac{1}{r_d})}, X_2^{(I)*}) = e(T_1^{\mathsf{sk}}, h^*) \wedge$
$\quad X_2^{(I)*} \neq g_2^{x_I \eta}, \forall i \in \{1, ..., k\}$:
$\quad$ **return** 1;
Else:
$\quad$ **return** 0;

Figure C.13 – Collision freedom game for the interactive protocol of Section 5.7.2

$(W)_I^*(C_1^*)^{''-(\frac{1}{r_d})}$ as $g_1^{s_w} g_1^{s_1 \mathsf{sk}} g_1^{s_2}$, with s_1 and s_2 value known to the adversary.

We use the same idea as in Appendix C.3, we know that $g_1^{s_w} g_1^{s_1 \mathsf{sk}} g_1^{s_2} = g_1^{\mathsf{sk}\gamma}$, for some γ independent from sk. At this point, the only difference between the game presented in Figure C.13 and Appendix C.3's proof is that the adversary is given the extra values $g_1^{skr_d}$ and $g_1^{r_d}$. We use Appendix C.3 proof again and more precisely Appendix C.3.2 proof. The setup we have here is exactly the same as the one used in the simplified game of this proof. Thus the conclusion is the same, *i.e.*, γ is a composition of $(W)_i$ values.

We can resume Appendix C.3's proof without further modification, we will get the same result, *i.e.*, $X_2^{(I)*} = g_2^{x_I \eta}$. □

C.8.3 Indistinguishability of the Signature with Commitment Reveal Exchange

Lemma C.6. The interactive version of the aggregator presented in Section 5.7.2 keeps it element indistinguishable.

Proof. We want to prove that the indistinguishability property is not affected by the modification made in Section 5.7.2. This transformation can be proven to keep issuer indistinguishability property and unlinkability property, using the theorem proved in Appendix C.7.

Indeed, the transformation makes the user share three more values, C_1, C_2, and C_1''. We will prove these values indistinguishable in the same way we did in Appendix C.7. We know that $C_1 = g_1^{r_1 r_b}$ and $C_2 = g_1^{r_a r_c}$. From the ZKP ϕ_2 , we also know that $C_1' = C_1^{s_1}$ and $C_2' = C_2^{s_2}$, for some unknown values s_1 and s_2. This implies that $C_1'' = C_1^{\frac{1}{r_b} s_1} C_2^{\frac{1}{r_c} s_2} = g_1^{r_{(u,x)} s_1 + r_a s_2}$. Any new request from the adversary would lead to a new commitment value, with new random values. This implies that the adversary has no advantage in requesting multiple commitments.

The values disclosed by C_1 and C_2 are randomized with elements used only once. The adversary cannot extract any information from them. The value disclosed by C_1'' can be considered as a value chosen uniformly at random in $\mathbb{G}_1$ because of the addition of r_a, used only there. Thus, C_1'' does not disclose information about the issuer nor the user. There is no linear relationship between C_1, C_2, C_1'' and the other values of the scheme. Thus the modification made in Section 5.7.2 keeps the indistinguishability property. □

COLLEGE DOCTORAL BRETAGNE

MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Systèmes de Gestion de l'Identité totalement distribués et respectant la vie privée

Mot clés : Cryptographie, Systèmes Distribués, Respect de la vie privée, Identité Auto Souveraine

Résumé : Dans cette thèse, nous nous intéressons aux systèmes de gestion d'identité totalement distribués respectant la vie privée. Ces systèmes ont pour but de permettre à un utilisateur de s'authentifier et d'être autorisé par un fournisseur de services, tout en ne lui révélant que les informations strictement nécessaires. De plus, ces systèmes doivent être résilients à la présence de processus malveillant. Dans ce contexte, nous nous intéressons à deux points. D'abord, aux certificats anonymes et à leur propriétés de respect de la vie privée. Nous identifions un manque qui réduit cette propriété dans l'état de l'art, et nous le corrigeons grâce à un nouveau type de signature : les certificats anonymes à émetteurs cachés. Ensuite, nous nous intéressons aux algorithmes distribués utilisés pour les propriétés annexes des systèmes de gestion d'identité distribués, notamment pour la révocation de certificats, ou la gestion de clés publiques. Nous analysons formellement ces problèmes, notamment du point de vue de leur *consensus number*. Ces analyses nous permettent finalement de proposer des algorithmes pour implémenter un système de gestion de l'identité totalement distribué qui nécessite une synchronisation réduite. En d'autres termes, un système où l'utilisation d'algorithmes de consensus est réduite au minimum.

Title: Privacy Preserving and fully Distributed Identity Management Systems

Keywords: Cryptography, Distributed Systems, Privacy, Self-Sovereign Identity

Abstract: This thesis focuses on privacy preserving and fully distributed identity management systems. These systems aim to allow a user to authenticate and be authorized by a service provider while only revealing strictly necessary information. In addition, these systems must be resilient to the presence of malicious processes. In this context, we are interested in two points. Firstly, anonymous credentials and their privacy properties. We identify a shortcoming that reduces this property in state of the art, and we correct it with a new type of signature: hidden issuer anonymous credentials. Next, we look at the distributed algorithms used for the auxiliary properties of distributed identity management systems, in particular for certificate revocation and public key management. We analyze these problems formally, particularly from the point of view of their *consensus number*. Finally, these analyses allow us to propose algorithms for implementing a fully distributed identity management system that requires reduced synchronization. In other words, a system where the use of consensus algorithms is reduced to a minimum.