Homomorphic Cryptography and Privacy

Chloé Hébant

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure de Paris

Homomorphic Cryptography and Privacy

Soutenue par
Chloé Hébant
Le 20 Mai 2021

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Dario Catalano University of Catania	*Rapporteur*
Benoît Libert École Normale Supérieure de Lyon	*Rapporteur*
Caroline Fontaine École Normale Supérieure de Paris-Saclay	*Présidente du jury*
Olivier Sanders Orange Labs, Rennes	*Examinateur*
David Pointcheval École Normale Supérieure de Paris	*Directeur de thèse*
Duong Hieu Phan Telecom Paris Institut Polytechnique de Paris	*CoDirecteur de thèse*

ENS ÉCOLE NORMALE SUPÉRIEURE

PSL RESEARCH UNIVERSITY PARIS

Résumé

Avec l'utilisation massive du stockage dématérialisé, l'homomorphisme est devenu l'une des propriétés les plus largement employées en cryptologie. Dans cette thèse, nous allons étudier comment l'utiliser dans des protocoles multi-utilisateurs concrets qui nécessitent non seulement de la confidentialité, mais aussi de l'anonymat, de l'authentification ou encore de la vérifiabilité. Pour cela, nous utilisons des schémas homomorphes de chiffrement, de signature numérique et de preuves à divulgation nulle de connaissances, mais, à chaque fois, nous devrons limiter leurs capacités de malléabilité pour atteindre le niveau de sécurité préalablement défini.

Tout d'abord, l'aspect confidentiel est abordé au travers de l'étude de calculs sur des bases de données externalisées. Être capable d'appliquer des fonctions sur des données chiffrées sans avoir à les télécharger pour les déchiffrer entièrement est permet de profiter de la puissance de calcul du serveur qui est généralement supérieure à celle du client. Cela peut être également indispensable lorsqu'une société sans droit d'accès à une base de données de clients souhaite obtenir le résultat d'un calcul. La quantité d'information apprise ne doit pas être supérieure à celle contenue dans le résultat du calcul. Nous proposons pour cela un schéma de chiffrement décentralisé qui permet d'évaluer des fonctions quadratiques sur les données externalisées tout en ayant un contrôle des opérations grâce à un groupe d'inspecteurs.

Cependant, la confidentialité des données n'est pas toujours la propriété la plus recherchée pour un système car elle ne protège pas l'identité de l'expéditeur. Pour le vote électronique, chaque bulletin chiffré doit être associé à un électeur afin de vérifier que celui-ci était autorisé à voter, mais après la phase de vote, l'anonymat doit être assuré. Pour cela une solution est de mélanger plusieurs fois l'urne de sorte que, au moment du dépouillement, qui correspond au déchiffrement, aucun lien entre le vote et l'électeur ne puisse être fait. C'est le fonctionnement d'un réseau de serveurs-mélangeurs dont nous proposons une nouvelle construction basée sur des signatures linéairement homomorphes avec un coût de vérification de l'urne finale indépendant du nombre de mélanges. Ce protocole est donc d'autant plus efficace que le nombre de mélanges augmente et représente un progrès par rapport aux constructions déjà connues.

Dans certains cas, avoir un anonymat parfait permettrait l'utilisation malveillante d'un système et la cryptologie doit aussi tenir compte de ces abus potentiels. La troisième contribution de cette thèse consiste en la proposition du premier protocole d'accréditation anonyme multi-autorités traçable : un utilisateur demande une accréditation auprès d'une autorité émettrice et peut l'utiliser pour accéder à un système tout en restant anonyme. En cas d'abus, une autorité juge peut lever l'anonymat et retrouver un utilisateur malveillant grâce au traçage. De plus, ce protocole, tout en étant aussi efficace que les précédents pour une seule autorité émettrice, permet d'agréger des accréditations d'autorités émettrices distinctes pour avoir une accréditation de taille optimale .

Mots clés : Cryptographie à Clé Publique, Protocoles Homomorphes, Anonymat, Vote Electronique, Calculs Multipartites

Abstract

With the massive use of dematerialized storage, homomorphism has become one of the most widely used properties in cryptology. In this thesis we will study how to use it in concrete multi-users protocols requiring not only confidentiality but also anonymity, authentication or verifiability. Homomorphic encryption schemes, homomorphic digital signatures and homomorphic zero-knowledge proofs will be used together, but each time restricted to achieve the desired level of security.

First, the confidential aspect is studied for computations on large outsourced databases. Being able to apply functions on encrypted data without having to download and decrypt it entirely may be essential and allows to take advantage of the computational power of the server. This can also be interesting when a third-party company without right-access to the database wants to obtain the result of a computation. However, some guarantees on the learned information need to be taken. To this end, we present a decentralized encryption scheme that allows controlled evaluation of quadratic functions on outsourced data thanks to a group of controllers.

However, sometimes confidentiality of the data is not the most desired property for a system as it does not protect the sender. For electronic voting, each encrypted ballot must be associated with its voter to verify that he is allowed to vote. After the voting phase, anonymity is achieved by shuffling so that, during the count, which corresponds to the decryption, no link between votes and voters can be made. We propose a new construction of mix-network based on linearly homomorphic signatures which allows for the first time a verification which is cost-independent of the number of mix-servers. This scalable mix-net improves the efficiency compared to already known constructions, especially with an increasing number of shuffles.

Nevertheless, with perfect anonymity comes the threat of malicious use of the system. Cryptology must consider these possible abuses and we propose the first multi-authority anonymous credential protocol with traceability property: a user asks a credential issuer for a credential and uses it to access a system while remaining anonymous. In case of abuse, an authority can revoke anonymity and trace a malicious user. The scheme is as efficient as the previously known credential schemes while achieving the multi-credential issuer functionality.

Keywords: Public-Key Cryptography, Homomorphic Protocols, Anonymity, E-voting, Multi-party Computation

Acknowledgments

Lorsque l'on écrit ces lignes, c'est qu'une aventure d'environ 3 ans se termine. On a beau s'imaginer défendre dès le début de la thèse, lorsque l'on arrive au bout de l'écriture du manuscrit, une certaine nostalgie est inévitable. J'ai eu la chance de rencontrer de merveilleuses personnes qui ont su me partager avec finesse leurs connaissances, j'ai eu la chance de travailler dans un cadre bienveillant et chaleureux et finalement j'ai eu la chance de voyager avant que cette pandémie ne nous frappe. Je tiens donc à remercier du fond du coeur tous ceux que j'ai rencontrés, ceux avec qui j'ai travaillé ou simplement ceux avec qui j'ai pu échanger. Si vous lisez cette thèse et que je n'ai pas explicitement cité votre nom, veuillez m'en excuser ! Je suis sûre que vous méritez, vous aussi, des remerciements ne serait-ce que pour avoir lu cette section.

Pour commencer, j'aimerais remercier mes deux directeurs de thèse sans qui tout cela ne serait jamais arrivé et pour qui j'ai énormément de reconnaissance. Tout à commencé grâce à Duong Hieu Phan en 2016 par un projet de master suivi par un stage au Vietnam. Cette première expérience de recherche a été pour moi fantastique, tant humainement que professionnellement. Mon deuxième stage m'a ensuite amené à Paris et m'a permis de rencontrer David Pointcheval. L'expérience fut positive puisqu'elle s'est transformée en thèse. Je vous remercie pour les nombreux conseils, le partage de vos connaissances et le temps que vous m'avez consacré tout au long de mon parcours, votre réactivité pour répondre à mes questions et la clarté de vos réponses. J'espère sincèrement pouvoir continuer à travailler avec vous.

I sincerely thank Benoît Libert and Dario Catalano for being the rapporteurs of my manuscript. It is clearly not the easiest part and I hope you enjoyed the reading. Je remercie également Caroline Fontaine et Olivier Sanders pour avoir accepter de prendre part à mon jury.

Je remercie également tous les membres de l'équipe crypto de l'ENS qui sont déjà partis, qui vont bientôt partir et ceux que je n'ai pas pu d'avantage rencontrer à cause de la pandémie (par ordre alphabétique, mon tact légendaire m'empêchant de trier par préférence). Merci donc à Aisling a true tech-woman model, Aurélien pour tes cours d'escalade, Anca pour être la maman de César dont j'ai temporairement eu la garde, Antoine pour nous avoir entrainés à survivre à des pandémies lors de soirées jeux ... avant la pandémie, Azam, Balthazar pour avoir tenté de ressusciter un cactus mort, Baptiste, Bogdan pour m'avoir montré qu'il était possible de manger très lentement, Brice, Céline pour avoir co-organisé le Working Group avec moi, Damien pour avoir partagé ton frigo, Damuhn, Edouard pour tes conseils en restaurant de burgers, Florian, Geoffroy, Georg, Hoeteck, Hugo, Huy, Jérémy pour nous avoir fait peur en disparaissant temporairement lors d'un voyage à l'étranger, Julia, Léo, Léonard, Louiza pour nos longues conversations téléphoniques, Mélissa pour avoir partagé les logements en dur ou en toile lors des voyages, Michael, Michel pour avoir organisé un immense team building pile à mon arrivée avec l'organisation d'Eurocrypt, Michele, Michele pour être mon fournisseur officiel de conseils en cas d'urgence pour de l'informatique pratique, Paola, Pierre-Alain pour avoir légué un cactus mort à Balthazar, Phong, Pierrick pour être mon fournisseur officiel de plantes d'aquarium, Pooya pour nos discussions de randonnées, Razvan, Romain pour tes extraits de vie qui ont animé de nombreuses fois le labo, Théo, Thierry.

Je tiens également à exprimer toute ma reconnaissance à l'équipe administrative du DI, notamment à Sophie Jaudon, Valérie Mongiat, Linda Boulevart et Lise-Marie Bivard et au SPI,

Ludovic Ricardou et Jacques Beigbeder.

Je souhaite également remercier le service des cartes de l'ENS qui devait beaucoup m'apprécier pour me faire venir aussi souvent en désactivant mon badge, les différentes générations de machines à café, les nombreuses plantes et les aquariums qui égayaient le labo à une certaine époque, les cours d'escalade du mardi ou du jeudi midi, les soirées jeux de sociétés ou de Wii. Je souhaite de tout coeur aux doctorants actuels et futurs de vivre cette expérience de labo pour laquelle j'ai des souvenirs mémorables.

J'aimerais aussi sincèrement remercier Yannick pour m'avoir accueillie virtuellement, pandémie oblige, dans son équipe pour un « stage de fin de thèse » ainsi que son équipe chaleureuse. In particular, I would like to sincerely thank Steve for being a great supervisor and Tim for all your answers to my practical questions. Je souhaite aussi remercier Mathieu et son équipe pour m'avoir accueillie dans les virtual but cultural and social events dans la « bonne » time-zone.

Je souhaite aussi remercier les amis de plus longue date qui ont suivi de près ou de loin mon cheminement jusqu'à cette thèse (par année de rencontre) : Elodie, le Soutien Mental (elles se reconnaitront), Ghislain et Alexia, Florent, Nicolas, Neals, Charline et Thanh.

Pour finir, cette thèse n'aurait pas pu avoir lieu sans le soutien inconditionnel de ma famille. Ils supportent mon hypersensibilité au quotidien et ça ne doit pas être évident. En particulier, je remercie énormément ma soeur, mes parents et bien évidemment Yoan.

Yoan, j'ai essayé mais je n'arrive pas résumer en quelques mots tout ce que tu m'apportes sans en avoir les larmes aux yeux. Je vais donc être brève mais chargée de sentiments: je T'aime.

Contents

Chapter

1 Introduction

Chapter content

Covid-19. The pandemic spreads across the world and teleworking is currently the norm. We log in every morning to check our mailbox, we connect to the VPN to work on the intranet, we save all the files on the cloud and finally, we meet colleagues and family on virtual meetings. We are more than ever dependent on the Internet and all its features. Even children are attending classes from home and people living in dead zones suffer the most from this situation. The Internet is essential and the need to improve privacy is huge in all our uses of electronic devices.

How can we guarantee privacy whereas being connected is more and more essential?

One solution is to study and propose scenarios integrating *privacy by design*. This thesis explores this field and makes its contribution to cloud computation, electronic voting and anonymous authentication.

1.1 Classical Cryptography

Cryptography is a very large field but first, let us start with some historical notions that will be useful for the full understanding of this thesis.

1.1.1 Confidentiality

Historically, the aim of cryptology was to protect confidentiality of communications. The first propositions to hide a message were to encrypt it by transforming it in a secret manner in order to make it incomprehensible. The hidden message can then travel safely to the receiver who applies at his turn secret transformations to recover the original message. A non-encrypted, and thus understandable, message is called *in clear*.

However, Kerckhoff's principles (1883) [Ker83] state that the security must not rely on secret transformations but instead, must rely on public transformations (so that the mechanism can fall in the hands of the adversary) with the use of a key that shall remain secret (must not go in the hands of the adversary).

To that end, an *encryption scheme* is defined as the set of three algorithms. The algorithm EKeygen describes the construction of the key, the algorithm Encrypt details the steps to encrypt. It outputs a hidden message called *ciphertext*. Finally, the algorithm Decrypt allows to recover the original message.

In fact, encryption schemes are divided in two categories depending if the same key is used to both encrypt and decrypt (symmetric encryption) or if the scheme provides two keys (asymmetric encryption): a public one to encrypt so that anyone can encrypt a message with it and a secret one to decrypt so that only the owner of the key can recover the message. This thesis exclusively focuses on the latter case, also called public-key cryptography as it allows a particular user to receive messages from several users while being the only one possessing the tool to decrypt them and thus, being the only one responsible for his security.

Especially, an encryption schemes is said *secure* if an attacker can not deduce one bit of information on a clear from its corresponding ciphertext.

The most famous public-key cryptosystem still in use was developed by Rivest, Shamir and Adleman [RSA78] in 1978. To generate the keys, one needs to choose a large number n product of two primes p and q and two numbers e and d such that $e \cdot d = 1 \mod \phi(n)$. The public key is $(n = pq, e)$ while the secret key is the decomposition (p, q) of n with the private element d. To encrypt a message $m \in \mathbb{Z}_p^*$, one computes the ciphertext $c = m^e \mod n$. Whereas to decrypt, one recovers the message with $c^d = (m^e)^d = m^1 \mod n$.

1.1.2 Authentication

How can Bob be sure he is receiving a message from Alice and not from Eve pretending to be Alice? In our digital world, one needs to be careful about phishing and other attempts of false impersonations. A way to avoid that is to add an authentication process and to check the supposed identity of the user. While confidentiality is mainly achieved with encryption, authentication is mainly achieved thanks to digital signatures.

Similarly to an encryption scheme, a *digital signature* scheme is made up of three algorithms. The algorithm SKeygen generates the signing key (must be kept secret) and the verification key (public). The algorithm Sign creates a signature from a message and the signing key while the algorithm Verif verifies a signature given a message and a verification key. If this last algorithm outputs 1, the signature is said *valid*. We will use these algorithms a lot all along this manuscript, most of the time enhanced with *special* properties.

To illustrate, the RSA signature is composed of the algorithm SKeygen using the same numbers n, p, q, e, d such that $n = pq$ and $ed = 1 \mod \phi(n)$ as for the RSA algorithm EKeygen but this time, the signing key is (p, q, d) and the verification key is (n, e). To sign a message m, one needs to compute $\sigma = \mathcal{H}(m)^d \mod n$ with $\mathcal{H}$ a full domain hash function. Thus, if we compute σ^e, we can verify the signature σ was correct by recovering or not the message as $\sigma^e = (\mathcal{H}(m)^d)^e = \mathcal{H}(m)^1 \mod n$.

Classically, a signature scheme is said *secure* if an attacker cannot forge a new valid signature even if he previously saw valid ones.

1.1.3 Provable Security

The first definition and proof of security dates back to Shannon [Sha49] in 1949. In his article, he defines the notion of *perfect secrecy* in the sense of information theory and gives the example of the One-Time Pad scheme which is perfectly secure. However, this scheme is not convenient as the size of the key needs to be at least equal to the size of the message.

To avoid this limitation, security proofs are no longer necessarily made in the sense of information theory but are most often reductions of mathematical problems for which the complexity is widely accepted.

For example, the RSA encryption scheme is based on the *factoring problem*: given a very large number product of two prime numbers $n = pq$, the goal is to find the two prime factors p and q. The decomposition being part of the secret key of the encryption scheme, breaking the problem directly breaks the security of RSA.

In this manuscript, we will use two other mathematical problems often used in cryptography defined on a group $\mathbb{G}$ generated by g. The Discrete Logarithm (DL) problem is, given an element X of $\mathbb{G}$, to find the exponent x such that $X = g^x$. The other is the Computational Diffie-Hellman (CDH) problem [DH76] (1976): given two elements $X = g^x$ and $Y = g^y$ of $\mathbb{G}$, the goal is to compute $Z = g^{xy}$. For these two problems, the bigger is the number of elements in $\mathbb{G}$, the more difficult is the problem. The security of cryptographic schemes is thus usually defined with a *security parameter* configuring the size of the group. Then, we make the assumption that the problem is difficult and if attacks are improved or found, it is possible to adapt the level of security with the security parameter.

A variant of the CDH problem is the Decisional Diffie-Hellman (DDH) problem: given three elements $X = g^x, Y = g^y$ and Z of a cyclic group, the goal is this time to decide if $Z = g^{xy}$ or not. It is at the core of the El Gamal cryptosystem [ElG84] (1984), the second main public-key encryption scheme that we will use a lot all along this manuscript.

1.2 Advanced Security Goals

The cryptographic world where Alice and Bob are discussing secretly together is not relevant anymore. We live in a connected world where some scenarios imply to deal with thousands of users while still achieving some kind of security. There are users and servers, different access rights for each of them and sometimes there are various authorities at the same time having different abilities. As the number of persons involved increases and the technologies are improving, more specific definitions of security notions are necessary. In this section, we broach the advanced and modern thematic to improve the privacy of users by a better control of his data or guarantees on sensitive systems.

A Multi-User World. Working in a multi-user setting means having to consider different roles for each user. An honest user follows the protocol, an honest-but-curious user follows the protocol but tries to retrieve information from the communications he sees passing by and finally a malicious user tries to actively attack the protocol. If several users try to attack together with a common goal, we talk about collusion but an attacker can also corrupt others. Some schemes allow a limited number of collusions/corruptions, i.e. they are proven safe until this threshold is reached.

Privacy in a World of Information. For the scenarios studied in this manuscript, we will consider Uma, a user who wants to protect her privacy while wishing to take advantage of the technological advances of her time. She therefore uses a cloud to store her data but would like to allow a company to study her records, such as allowing it to do statistical studies of cancers on her medical record but only if the company is forced by the system to recover only this

information and nothing more. Uma would also like to vote electronically at the next election but requests that anonymity is guaranteed as well as the result of the election. Finally, Uma would like to be able to log in anonymously while being properly authenticated to a system. She accepts that, in return, if she misuses and abuses the system, her anonymity may be revoked by a judge.

All of these scenarios require new and more complex security concepts than the one historically used. Cryptology should not be a hindrance to the use of new technological advances.

1.2.1 Decentralization

Decentralized Cryptography is one of the main directions of research in cryptography, especially in a concurrent environment of multi-user applications, where there is no way to trust any authority. Recently, the rise of blockchain's applications also showed the importance of decentralized applications. However, the blockchain mainly addresses the decentralized validation of transactions, but it does not help in decentralizing computations. For the computational purpose, though general solutions can be achieved via multi-party computation, reasonably efficient solutions only exist for a limited number of protocols, as decentralization usually adds design constraints to protocols: in broadcast encryption [FN94], the decentralized protocol in [PPS12] is much less efficient than the underlying original protocol [NNL01]; in attribute-based encryption [SW05], the decentralized scheme [CC09] implies some constraints on the access control policy, that are removed in [LW11], but at the cost of using bilinear groups of composite order with 3 prime factors, etc...

In the last decade, the most active research direction was about computing over encrypted data, with the seminal papers on Fully Homomorphic Encryption (FHE) [Gen09] and on Functional Encryption (FE) [BSW11, GKP+13, GGH+13]. FE was generalized to the case of multi-user setting via the notion of multi-input/multi-client FE [GGG+14, GGJS13, GKL+13]. It is of practical interest to consider the decentralization for FHE and FE without need of trust in any authority. In FE, the question in the multi-client setting was recently addressed by Chotard *et al.* [CDG+18a] for the inner product function and then improved in [ABKW19, CDG+18b], where all the clients agree and contribute to generate the functional decryption keys, there is no need of central authority anymore. Note that, in FE, there are efficient solutions for quadratic functions [Gay16, BCFG17] but actually, only linear function evaluations can be decentralized as none of the methods to decentralize linear schemes seems to apply, and no new method has been proposed so far.

1.2.2 Anonymity

Sometimes confidentiality is not the most desired property for a system as it does not protect the sender or the receiver. In 1981, Chaum wrote the first article [Cha81] proposing a scheme able to hide the participants of a communication.

A user is *anonymous* when his name is either not known or not given. In practice, the anonymity is defined as the property to not be identifiable within a set. Hence, the anonymity notion can be obtained throught an *unlinkability* property. In the next two sections we will see use-cases of anonymity addressed in this thesis.

Electronic voting

There are mainly three ways to construct electronic voting schemes: from blind signatures, from additively homomorphic encryption and from shuffles performed by mix-networks. The first method requires to have interactions during the voting phase: the voter needs to communicate with an authority so that, she can blindly sign his vote. The second method requires to make a

proof at the time of the vote that the ballot is a valid one (e.g. the ballot is the encryption of 0 or 1 but nothing else). The last one is the solution studied in this thesis.

A shuffle of ciphertexts is a set of ciphertexts of the same plaintexts but in a permuted order such that it is not possible to trace back the senders after decryption. It can be used as a building block to anonymously send messages: if several servers perform a shuffle successively, nobody can trace the messages. More precisely, one honest mix-server suffices to mask the order of the ciphertexts even if all the other ones are dishonest. Moreover increasing the number of mix-servers leads to a safer protocol but also increases its cost. The succession of shuffles constitutes the notion of a mix-net protocol introduced by Chaum [Cha81], with applications to anonymous emails, anonymous routing, but also eletronic voting.

Anonymous authentication

In an anonymous credential scheme, a user asks an organization (a credential issuer) for a credential on an attribute, so that he can later claim its possession, even multiple times, but in an anonymous and unlinkable way.

Usually, a credential on one attribute is not enough and the user needs credentials on multiple attributes. Hence the interest in attribute-based anonymous credential schemes (ABC in short): depending on the construction, the user receives one credential per attribute or directly for a set of attributes. One goal is to be able to express relations between attributes (or at least selective disclosure), with *one showing*. As attributes may have different meanings (e.g. a university delivers diploma while a city hall delivers a birth certificat), there should be several credential issuers. Besides multi credential issuers, it can be useful to have a multi-show credential system to allow a user to prove an arbitrary number of times one credential still without breaking anonymity. For that, the showings are required to be unlinkable to each other.

1.2.3 Traceability

Nevertheless, with perfect anonymity comes the threat of malicious use of the system as it is not possible to identify anyone, even in case of misbehavior. Thus, in tracing scheme, one takes advantage of computational anonymity to find guilty members who can be at any level: a guilty sender of an encrypted message, a guilty signer, a guilty verifyer or a guilty authority. Collusions between users with different parts can also help to combine powers.

Here comes the *traitor-tracing* schemes [CFN94, CFNP00] with application to group signatures, group encryption [LYJP14], broadcast encryption, inner-product functional encryption [DPP20] or identity-based encryption [BBP19]. When the number of traitors is not bounded, the scheme is said fully traceable. To identify a culprit, one can add a new authority possessing a secret or one can request public traceability and in such a case, no secret is needed: everyone can find a traitor in case of misbehavior.

However, finding a guilty is not enough. When someone is declared guilty one needs to have guarantees against defamation of users or authorities. In particular, the *exculpability* property ensures that no coalition of authorities can convincingly accuse an innocent user in a group signature scheme. After finding a traitor and having the guarantee of its culpability, one can eventually revoke him but revokability is usually a more difficult property to achieve. When it is not possible, a solution is to combine traceability with decentralization. Hence, even a malicious authority cannot defame a user easily, its ability to judge or revoke being shared among parties.

1.3 Homomorphism

The presented scenarios require to develop new cryptographic tools. Secret sharing techniques allow decentralization by replacing an authority by a group of members. Indeed, thanks to

Shamir [Sha79], it is possible to decompose a secret key s in parts so that $s = \sum_i s_i$, each user $\mathcal{U}_i$ of a group possesses s_i a fragment of the secret and is not able to decrypt a ciphertext alone. However, they can decrypt by playing with the other members of the group which implicitly reconstruct the secret $s = \sum_i s_i$.

All along this manuscript we will use another key ingredient: the homomorphism.

1.3.1 Homomorphic Encryptions

For some generic encryption scheme, if we modify a ciphertext it becomes a completely random string and the content is lost as it is not decryptable anymore even with the secret key. Not being able to manipulate ciphertexts is called *non-malleability*. For a long time, this property ensured the security of encryption schemes: either we know the secret key and we can find the message again or we do not know it and all the operations we can do will not help.

However, with the RSA encryption scheme, if we multiply two encryptions of m_1 and m_2 together, we obtain the ciphertext of the message $m_1 \times m_2$: if $c_1 = m_1^e \bmod n$ and $c_2 = m_2^e \bmod n$ then $c_1 \cdot c_2 = (m_1 \times m_2)^e \bmod n$. A cryptosystem with such property is said to be *multiplicatively* homomorphic as one can operate on the clears by multiplication. With the Paillier cryptosystem [Pai99], if we multiply two encryptions of m_1 and m_2 together, we obtain the ciphertext of the message $m_1 + m_2$. This scheme is thus *additively* homomorphic.

The homomorphism property can be of great interest: if Alice and Bob want to give a common gift to Charlie but none of them want to tell how much he gave. They both encrypt the amount with an additively homomorphic scheme and can, thanks to the property of homomorphism, multiply their two encryptions together and send the result to Charlie. She will be able to decrypt it and the message will correspond directly to the sum of the two amounts.

Another common usage of homomorphism with encryption is for refreshing a ciphertext: by multiplying an encryption of 1 by a ciphertext of m, encrypted with a multiplicatively homomorphic scheme, one can obtain a new encryption of m thanks to the usage of the neutral element 1 for the multiplication. The new ciphertext while still encrypting the same message can be indistinguishable from the original one. A similar relation can be obtained with an encryption of 0 and an additively homomorphic scheme.

Generically, an encryption scheme in which making an operation on ciphertexts is equivalent to encrypting the result of a "twin operation" on the clears is called homomorphic. The seminal paper of Gentry [Gen09] published in 2009 combines for the first time both multiplicative and additive properties, and thus, allows the evaluatation of any function on ciphertexts. This scheme is called *fully homomorphic* encryption (FHE).

1.3.2 Homomorphic Signatures

Similarly to homomorphic encryption, one can define homomorphic signatures. The notion dates back to Rivest in a series of talks [Riv00] and Johnson *et al.* [JMSW02], with notions in [ABC+12].

They can help for computations on certified data. For example, Alice has n grades m_i all individualy signed into σ_i on a remote server. Later, the server is asked to perform an authenticated computation of a mean of the data. Thus, it computes $\sigma = f(\sigma_1, \ldots, \sigma_n)$ possible thanks to homomorphism and $M = f(m_1, \ldots, m_n)$ and publishes the result (M, σ). Then, anyone can check that the server correctly applied f to the data by verifying that σ is a valid signature.

The linearly-homomorphic signatures, that allow to sign vector sub-spaces, were introduced in [BFKW09], with several follow-ups by Boneh and Freeman [BF11b, BF11a]. With a linearly homomorphic signature scheme, from a signature σ_1 on the message m_1 and σ_2 a signature on the message m_2, it is possible to compute for all α, β, the signature $\sigma = \alpha\sigma_1 + \beta\sigma_2$, a valid signature on the message $m = \alpha m_1 + \beta m_2$.

As for encryption, a signature scheme can also be multiplicatively homomorphic: with the RSA signature, if we compute $\sigma_1 \times \sigma_2 = (m_1 \cdot m_2)^d \bmod n$, this is a valid signature of the message $m_1 \cdot m_2$ from two valid signatures σ_1 of m_1 and σ_2 of m_2. Even if this scheme illustrated the homomorphism, it is, as presented, not secure.

1.3.3 Security with Homomorphism

The security of an homomorphic scheme is different and needs to be strengthen. For example, an attacker seeing a signature of a message m, could simply forge the signature of $m+m$, which would break the basic notion of security. He does not know the keys but he is able to compute a valid signature of a new message (here $m+m$).

With the RSA signature scheme above, an attacker can forge any message of his choice: if an attacker asks the signature $\sigma = (m \cdot r^e)^d \bmod n$, this is a valid signature of the message $m \cdot r^e$. However with that, the attacker can forge the signature $\sigma^* = \sigma/r$ which is a valid signature of m while the signer does not know m.

To include the malleability in the security, we will require that it is not possible for an attacker to provide a signature outside the space generated by the signatures already given. A similar change in the security definition is of course needed for all the homomorphic schemes and thus, encryption ones.

1.3.4 Homomorphic Zero-Knowledge Proofs

The last kind of schemes we will intensively use in this thesis is the *zero-knowledge proofs*. They were introduced for the first time by Goldwasser, Micali and Rackoff [GMR85] in 1985.

A zero-knowledge proof is a protocol where a prover knowing a witness w makes a proof that a *statement* $x \in L$ is true to a verifier. The zero-knowledge property implies that the verifier does not learn more than the fact is true or false and nothing about the witness. In 1986, Goldreich, Micali and Wigderson [GMW87] shows that any language in NP possess zero-knowledge proofs.

For example, Bob can make a proof of Diffie-Hellman tuple to Alice: Alice thanks to the proof can verify if the tuple $(X = g^x, Y = g^y, Z)$ is or not a Diffie-Hellman (if $Z = g^{xy}$ or not) but she will not learn any of the exponents involved. The *language* is this case is the set of Diffie-Hellman tuples and the witness of Bob can be the pair of scalars x and y.

The showing of a proof can be interactive or non-interactive if the verifier after having received elements from the prover can check the proof latter without any further interaction with the prover. Moreover, the zero-knowledge proofs is said *of statements* if it proves a relation (as a proof of Diffie-Hellman tuple), or *of knowledge* if it proves the knowledge of a witness: when Bob wants to authenticate himself to a system, he needs to show he knows the password. A non-interactive zero-knowledge proof of knowledge is also called *signature of knowledge*.

With the Groth-Sahai [GS08] scheme, it is possible to combine together different proofs into a unique one. This property can be viewed as an homomorphic property: the combinations of the proofs creates a proof of a relation of statements.

In this thesis, zero-knowledge proofs will be used sometimes enhanced of the homomorphic property to prove that a ciphertext is correctly construted or for authentication.

1.4 Contributions

This thesis studies the usage of the homomorphic property in concrete multi-users protocols requiring not only confidentiality but also anonymity, authentication or verifiability. Homomorphic encryption schemes, homomorphic digital signatures and homomorphic zero-knowledge proofs will be used together to create new protocols, but, each time, restrictions will be applied to limit the possible malleability.

The results presented in this manuscript come from two published papers (co-authored with Duong Hieu Phan and David Pointcheval) and one paper still in reviewing process (co-authored with David Pointcheval).

First, we will present a decentralized encryption scheme that allows the controlled evaluation of quadratic polynomials on outsourced data. Then, we will describe a new method to build mix-networks from linearly homomorphic signatures allowing a verification which is cost-independent of the number of servers. Finally, we will detail a new traceable multi-authority anonymous credential protocol. After defining the security, the schemes are proved to achieve the desired level of security. Below, we detail the contributions.

***Decentralized Evaluation of Quadratic Polynomials on Encrypted Data* [HPP19].**

In this paper, we revisit the Boneh-Goh-Nissim (BGN) [BGN05] cryptosystem, and the Freeman's variant [Fre10], that allow evaluation of quadratic polynomials, or any 2-DNF formula. Whereas the BGN scheme relies on integer factoring for the trapdoor in the composite-order group, and thus possesses only one pair of public/secret keys, the Freeman's scheme can handle multiple users with one general setup that just needs a pairing-based algebraic structure to be defined. We show that it can be efficiently decentralized, with an efficient distributed key generation algorithm, without any trusted dealer, but also efficient distributed decryption and distributed re-encryption, in a threshold setting.

To motivate our work, we focus on some real-life applications of computations on encrypted data, without central authority which only require evaluations of quadratic polynomials so that specific target users can get the result in clear, by running re-encryption in a distributed manner under the keys of the target users.

This paper has been published in the proceedings of the conference ISC in 2019.

***Linearly-Homomorphic Signatures and Scalable Mix-Nets* [HPP20].**

In this article, we propose a new approach for proving correct shuffling of signed ElGamal ciphertexts: the mix-servers can simply randomize individual ballots, namely the ciphertexts, the signatures, and the verification keys, with an additional global proof of constant size. This technique can be seen as an improvement of signatures on randomizable ciphertexts [BFPV11] which however does not allow updates of the verification keys. This previous approach excluded anonymity because of the invariant verification keys.

The computational complexity for each mix-server is linear in the number of ballots and the overhead after each shuffle can be updated to keep it constant-size. Thus, the final proof implies just a constant-size overhead and the verification is also linear in the number of ballots, but independent of the number of rounds of mixing. This leads to a new highly scalable technique.

Our construction relies on Groth-Sahai proofs with pairings [GS08] and a new computational assumption that holds in the generic bilinear group model. We avoid the proof of an explicit permutation on all the ciphertexts (per mixing step) but the appropriate properties of the Mix-Nets are deeply guaranteed using linearly-homomorphic signature schemes with new features, that are of independent interest.

This paper has been published in the proceedings of the conference PKC in 2020.

***Traceable Multi-Authority Anonymous Credentials* [HP20].**

Following the path of aggregate signatures [CL11], our first contribution is the formalization of an aggregate signature scheme with randomizable tags (ART-Sign) for which we propose a practical construction. With such a primitive, two signatures of different messages under different keys can be aggregated only if they are associated to the same tag. In our case, tags will eventually be like pseudonyms, but with some properties which

make them ephemeral (hence EphemerId scheme) and randomizable, even when they are associated to the same user.

However our goal is to obtain a compact ABC system, which is our second contribution: the EphemerId scheme generates keys for users, they will use for authentication. Public keys being randomizable, multiple authentications using the same key will remain unlinkable. In addition, these public keys will be used as (randomizable) tags with the above ARTSign scheme when the credential issuer signs an attribute. Thanks to aggregation, multiple credentials for different attributes and from several credential issuers but under the same tag, and thus the same user, can be combined into a unique compact (constant-size) credential.

About security, whereas there exists a scheme proven in the universal composability (UC) framework [CDHK15], for our constructions we consider a game-based security model for ABC inspired from [FHS19]. As we support different credential issuers, we additionally consider malicious credential issuers, with adaptive corruptions, and collusion with malicious users. However, the keys need to be honestly generated, thus our proofs hold in the certified key setting. As for all the recent ABC schemes, our constructions will rely on signature schemes proven in the bilinear generic group model.

Our last contribution is traceability, in the same vein as group signatures: whereas showings are anonymous, a tracing authority owns tracing keys, and thus is able to link a credential to its owner. In such a case, we also consider malicious tracing authorities, with the non-frameability guarantee. Because the previous constructions ([CL13], [Ver17]) are broken, our scheme is the first traceable attribute-based anonymous credential scheme.

1.5 Organization of the Manuscript

This thesis is organized into seven chapters as follows:

Chapter 1 is the present introduction.

Chapter 2 is a preliminary chapter introducing the notations used in the manuscript. It also provides some definitions and some general notions and presents already existing cryptographic primitives used in the next chapters. In particular, it describes the homomorphic properties we will use a lot.

Chapter 3 presents our decentralized encryption scheme to evaluate quadratic polynomials on encrypted data. It is based on the paper [HPP19].

Chapter 4 is an introduction to linearly homomorphic signatures. In particular, we present two constructions: the first one is a modified already existing one used in our mix-net and the second one is new and used in our anonymous credential scheme. This chapter can be read independently of the previous one.

Chapter 5 presents our scalable mix-net. It is based on the paper [HPP20]. This chapter depends on the previous one.

Chapter 6 presents our anonymous credential scheme. It is based on the paper [HP20]. This chapter depends on the chapter 4.

Chapter 7 concludes this manuscript and expands the scope of the field with open questions.

Chapter

2 Preliminaries

Chapter content

This preliminary chapter aims to fix the notations and to recall the basic notions we will use throughout this thesis. In particular, we recall the homomorphic primitives we will manipulate to provide concrete constructions in the following chapters.

2.1 Notations and Usefull Notions

Sets. The set of all bit strings is denoted by $\{0,1\}^*$, while the set of bit strings of length $n \in \mathbb{N}$ is $\{0,1\}^n$. If $x \in \{0,1\}^*$, its bit-length is denoted $|x|$. If S is a finite set, we denote by $|S|$ the number of elements in S and by $s \xleftarrow{\$} S$ the process of selecting s uniformly at random in the set S.

Groups, Fields. We obtain a *group* by adding to a set a binary operation with good properties. Usually in this thesis, groups will be finite of order p a prime number and denoted with the multiplicative notation. Because a group of prime order is cyclic, an element of $(\mathbb{G}, \cdot)$ can be seen as an element of $\{1, g, \ldots, g^{p-1}\}$ and we will use the abusive notation $\mathbb{G} = \langle g \rangle$ to denote such a group with generator g. Moreover, we assume given $g \in \mathbb{G}$ and $x \in \mathbb{Z}_p$, one can efficiently compute g^x. In other words, the multiplication over $\mathbb{G}$ is an efficient operation.

We denote the set of integers by $\mathbb{Z}$ and the set of non-negative ones by $\mathbb{N}$. For a positive integer n and an integer $x \in \mathbb{Z}$, the reduction of x *modulo* n, denoted $x \bmod n$, is the remainder of the Euclidean division of x by n. We denote by $(\mathbb{Z}_n, +)$ the additive group of integers modulo n and by $(\mathbb{Z}_n, +, \cdot)$ the ring of integers modulo n. Again, p is a prime number in this thesis and, we denote by $\mathbb{Z}_p^* = \mathbb{Z}_p \backslash \{0\}$ the group of units of $\mathbb{Z}_p$ as $\mathbb{Z}_p$ becomes a field in that case.

Vectors, Matrices. The vectors $\mathbf{x} = (x_i)_i$ and matrices $\mathbf{M} = (m_{i,j})_{ij}$ are in bold and the vectors are written as row vectors, with sometimes components separated by commas for clarity: if $\mathbf{x} \xleftarrow{\$} X^n$, $\mathbf{x} = (x_1\ x_2\ \cdots\ x_n) = (x_1, x_2, \cdots, x_n)$.

We denote by $\mathcal{M}_{m,n}(\mathbb{Z}_p)$ the set of matrices on $\mathbb{Z}_p$, of size $m \times n$, and thus m row-vectors of length n. $(\mathcal{M}_{m,n}(\mathbb{Z}_p), +)$ is an Abelian group. When $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$ and $\mathbf{B} \in \mathcal{M}_{n,n'}(\mathbb{Z}_p)$, the matrix product is denoted $\mathbf{A} \times \mathbf{B} \in \mathcal{M}_{m,n'}(\mathbb{Z}_p)$, or just $\mathbf{AB}$ if there is no ambiguity. $(\mathcal{M}_{n,n}(\mathbb{Z}_p), +, \times)$ is a ring, and we denote by $\mathrm{GL}_n(\mathbb{Z}_p) \subset \mathcal{M}_{n,n}(\mathbb{Z}_p) = \mathcal{M}_n(\mathbb{Z}_p)$ the subset of the

invertible matrices of size n (for the above matrix product $\times$), which is also called the *general linear group*.

We will use the tensor product: for two vectors $\mathbf{a} = (a_1, a_2, \cdots, a_n) \in \mathbb{Z}_p^n$ and $\mathbf{b} = (b_1, b_2, \cdots, b_m) \in \mathbb{Z}_p^m$, the tensor product $\mathbf{a} \otimes \mathbf{b}$ is the vector $(a_1\mathbf{b}, \cdots, a_n\mathbf{b}) = (a_1b_1, \cdots, a_1b_m, a_2b_1, \cdots, a_2b_m, \cdots, a_nb_m) \in \mathbb{Z}_p^{mn}$; and for two matrices $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$ and $\mathbf{B} \in \mathcal{M}_{m',n'}(\mathbb{Z}_p)$,

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_m \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{m'} \end{pmatrix} \quad \mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 \\ \mathbf{a}_1 \otimes \mathbf{b}_2 \\ \vdots \\ \mathbf{a}_m \otimes \mathbf{b}_{m'} \end{pmatrix} \in \mathcal{M}_{mm',nn'}(\mathbb{Z}_p).$$

The bilinearity of the tensor product gives:

$$\text{for } \mathbf{A}, \mathbf{A}' \in \mathcal{M}_{m,n}(\mathbb{Z}_p) \text{ and } \mathbf{B}, \mathbf{B}' \in \mathcal{M}_{m',n'}(\mathbb{Z}_p),$$
$$(\mathbf{A} + \mathbf{A}') \otimes (\mathbf{B} + \mathbf{B}') = (\mathbf{A} \otimes \mathbf{B}) + (\mathbf{A} \otimes \mathbf{B}') + (\mathbf{A} \otimes \mathbf{B}') + (\mathbf{A}' \otimes \mathbf{B}')$$

We will also use the following important relation between matrix product and tensor product:

$$\text{for } \mathbf{A} \in \mathcal{M}_{m,k}(\mathbb{Z}_p), \mathbf{A}' \in \mathcal{M}_{k,n}(\mathbb{Z}_p), \mathbf{B} \in \mathcal{M}_{m',k'}(\mathbb{Z}_p), \text{ and } \mathbf{B}' \in \mathcal{M}_{k',n'}(\mathbb{Z}_p),$$
$$(\mathbf{A} \times \mathbf{A}') \otimes (\mathbf{B} \times \mathbf{B}') = (\mathbf{A} \otimes \mathbf{B}) \times (\mathbf{A}' \otimes \mathbf{B}').$$

Projections. In order to continue with matrix properties and linear applications, a *projection* π in a space of dimension n is a linear function such that $\pi \circ \pi = \pi$. Any projection of rank 1 can be represented by the matrix $\mathbf{P} = \mathbf{B}^{-1}\mathbf{U}_n\mathbf{B}$, where $\mathbf{U}_n$ is the canonical projection and $\mathbf{B}$ is the change of basis matrix:

$$\mathbf{U}_n = \begin{pmatrix} 0 & \dots & 0 & 0 \\ & \ddots & & \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_{n-1} \\ \mathbf{b} \end{pmatrix}$$

where $K = \langle \mathbf{p}_1, \dots, \mathbf{p}_{n-1} \rangle$ is the kernel of the projection and $\langle \mathbf{b} \rangle$ the image.

Given two projections π_1 and π_2 of rank 1, that are represented by $\mathbf{P}_1 = \mathbf{B}_1^{-1}\mathbf{U}_n\mathbf{B}_1$ and $\mathbf{P}_2 = \mathbf{B}_2^{-1}\mathbf{U}_n\mathbf{B}_2$, respectively, the tensor product $\pi = \pi_1 \otimes \pi_2$ is represented by $\mathbf{P} = \mathbf{P}_1 \otimes \mathbf{P}_2$, that is equal to

$$\begin{aligned}(\mathbf{B}_1^{-1}\mathbf{U}_n\mathbf{B}_1) \otimes (\mathbf{B}_2^{-1}\mathbf{U}_n\mathbf{B}_2) &= (\mathbf{B}_1^{-1} \otimes \mathbf{B}_2^{-1}) \times (\mathbf{U}_n \otimes \mathbf{U}_n) \times (\mathbf{B}_1 \otimes \mathbf{B}_2) \\ &= (\mathbf{B}_1 \otimes \mathbf{B}_2)^{-1} \times \mathbf{U}_{n^2} \times (\mathbf{B}_1 \otimes \mathbf{B}_2).\end{aligned}$$

The associated change of basis matrix is thus $\mathbf{B} = \mathbf{B}_1 \otimes \mathbf{B}_2$. In dimension 2:

$$\mathbf{B}_1 = \begin{pmatrix} \mathbf{p}_1 \\ \mathbf{b}_1 \end{pmatrix} \text{ and } \mathbf{B}_2 = \begin{pmatrix} \mathbf{p}_2 \\ \mathbf{b}_2 \end{pmatrix}, \quad \text{then } \mathbf{B} = \mathbf{B}_1 \otimes \mathbf{B}_2 = \begin{pmatrix} \mathbf{p}_1 \otimes \mathbf{p}_2 \\ \mathbf{p}_1 \otimes \mathbf{b}_2 \\ \mathbf{b}_1 \otimes \mathbf{p}_2 \\ \mathbf{b}_1 \otimes \mathbf{b}_2 \end{pmatrix},$$

hence, the image of $\pi = \pi_1 \otimes \pi_2$ is spanned by $\mathbf{b} = \mathbf{b}_1 \otimes \mathbf{b}_2$, while $\{\mathbf{p}_1 \otimes \mathbf{p}_2, \mathbf{p}_1 \otimes \mathbf{b}_2, \mathbf{b}_1 \otimes \mathbf{p}_2\}$ is a basis of the kernel. But, as explained below, $\mathbf{p}_1 \otimes \mathbf{r}_2 + \mathbf{r}_1 \otimes \mathbf{p}_2$, for $\mathbf{r}_1, \mathbf{r}_2 \xleftarrow{\$} \mathbb{Z}_p^2$, provides a uniform sampling in $\ker(\pi)$:

$$\begin{aligned}\ker(\pi) &= \{(x_1 + x_2) \cdot \mathbf{p}_1 \otimes \mathbf{p}_2 + y_2 \cdot \mathbf{p}_1 \otimes \mathbf{b}_2 + y_1 \cdot \mathbf{b}_1 \otimes \mathbf{p}_2, x_1, x_2, y_1, y_2 \in \mathbb{Z}_p\} \\ &= \{\mathbf{p}_1 \otimes (x_2 \cdot \mathbf{p}_2 + y_2 \cdot \mathbf{b}_2) + (x_1 \cdot \mathbf{p}_1 + y_1 \cdot \mathbf{b}_1) \otimes \mathbf{p}_2, x_1, x_2, y_1, y_2 \in \mathbb{Z}_p\}.\end{aligned}$$

Cryptographic Notations. All along this manuscript, κ will be the security parameter. In all the public-key cryptographic primitives, keys will implicitly include the global parameters and secret keys will include the public keys.

Bilinear Group Setting. A *bilinear group generator* $\mathcal{G}$ is an algorithm that takes a security parameter κ as input, and outputs a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ such that $\mathbb{G}_1 = \langle g \rangle$ and $\mathbb{G}_2 = \langle \mathfrak{g} \rangle$ are cyclic groups of prime order p (a κ-bit prime integer), and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an *admissible pairing*:

- e is bilinear: for all $a, b \in \mathbb{Z}_p$, $e(g^a, \mathfrak{g}^b) = e(g, \mathfrak{g})^{ab}$;
- e is efficiently computable (in polynomial-time in κ);
- e is non-degenerated: $e(g, \mathfrak{g}) \neq 1$.

Furthermore, the bilinear setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ is said *symmetric* when $\mathbb{G}_1 = \mathbb{G}_2$ and *asymmetric* when $\mathbb{G}_1 \neq \mathbb{G}_2$ and of

- Type 1: if there exist two efficiently computable isomorphisms from $\mathbb{G}_1$ to $\mathbb{G}_2$ and conversely from $\mathbb{G}_2$ to $\mathbb{G}_1$;
- Type 2: if there only exists an efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$;
- Type 3: if there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

In this thesis, we will only use asymmetric pairing of type 3 and denote in Fraktur font elements of $\mathbb{G}_2$ for the sake of clarity.

2.2 Provable Security

Correctly evaluate the security of a system is challenging and fundamental. Especially because it is always possible to attack a system: it suffices to try all the possible secrets until finding the correct one. This method is called the brute-force attack. However, it takes time and sometimes more efficient attacks can be found. Moreover knowing precisely the powers of an attacker is often impossible: what does he intend to do, how much power or how long does he have? Therefore, the goal becomes to modelize the interactions between some adversary and the system so that a real attacker would have less power than the idealized/considered one. This is usually achieved by upper bounding the data he has access to and formalized by an *experiment* representing the desired security as a game between an attacker and an implicit challenger playing as wanted by the system.

With the current computational power available, one considers an attack to be infeasible if it requires 2^{128} computational steps to break the system and one says that a cryptosystem provides κ bits of security if it requires 2^{κ} elementary operations to be broken. Hence, the goal becomes to prove that the attacker winning the experiment has to perform at least this number of operations.

After having defined the attacker and the *security model* we want to achieve, the proof usually consists in hybrid games where the first one corresponds to the interactions between the attacker and the system and each step of the proof shows that this intermediate experiment is equivalent to the next one until the final experiment corresponds to the desired security or a difficult mathematical problem. Usually at the end, one can tell if the attacker can break the system if he is also able to break this well-known hard problem.

For that, it can be easier to make proofs in the Random Oracle Model (ROM) [BR93] in which hash functions used in a scheme are considered as an ideal random oracle. Hence, the randomness is considered as perfect.

Another possible model to help in the study of the security is the Generic Group Model (GGM) [Sho97]: if the attacker has access to group elements, then he can just make linear combinations of them and nothing else. For example, if the attacker knows A and B such that $A = g^a$ and $B = g^b$ then the attacker can obtain $C = g^{\alpha a+\beta b}$ for any α and β but cannot have $D = g^d$ with d not in relation with a and b. In fact, this corresponds to an idealization of the group structure as the attacker can not exploit any special structure of the representation of the group elements. Similarly, the Generic Bilinear Group Model (GBGM) adds the idealization of the pairing operation.

Finally, security proofs in the Common Reference String Model [Dam00] assume that a string crs, known by all the participants before the begining of the scheme, exists and was honestly generated.

Of course, the best model is the Standard Model where no idealization of the system is made but difficult to achieve.

2.3 Computational Assumptions

As introduced in the previous chapter, in an asymmetric bilinear setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$, or just in a simple group $\mathbb{G}$, we can make assumptions that will help in the establishment of the security proofs of the cryptographic schemes.

The most famous one is the Discrete Logarithm (DL) Assumption:

Definition 1 — Discrete Logarithm (DL) Assumption

In a group $\mathbb{G}$ of prime order p, it states that for any generator g, given $y = g^x$, it is computationally hard to recover x.

There exists a variant of the DL assumption when having two groups $\mathbb{G}_1, \mathbb{G}_2$:

Definition 2 — Symmetric External Discrete Logarithm (SEDL) Assumption

In groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p, it states that for any generators g and $\mathfrak{g}$ of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, given $f = g^x$ and $\mathfrak{f} = \mathfrak{g}^x$, it is computationally hard to recover x.

The second most famous assumption is the Decisional Diffie-Hellman (DDH) assumption:

Definition 3 — Decisional Diffie-Hellman (DDH) Assumption

In a group $\mathbb{G}$ of prime order p, it states that for any generator g, the two following distributions are computationally indistinguishable:

$$\mathcal{D}_{\mathsf{dh}}(g) = \{(g, g^x, h, h^x); h \xleftarrow{\$} \mathbb{G}, x, \xleftarrow{\$} \mathbb{Z}_p\}$$
$$\mathcal{D}^4_{\$}(g) = \{(g, g^x, h, h^y); h \xleftarrow{\$} \mathbb{G}, x, y, \xleftarrow{\$} \mathbb{Z}_p\}.$$

More precisely, the advantage $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A})$ of an adversary $\mathcal{A}$ against the Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ is defined by:

$$\Pr\left[\mathcal{A}(g, g^x, g^y, g^{xy}) = 1 | x, y \xleftarrow{\$} \mathbb{Z}_p\right] - \Pr\left[\mathcal{A}(g, g^x, g^y, g^z) = 1 | x, y, z \xleftarrow{\$} \mathbb{Z}_p\right].$$

The DDH problem in $\mathbb{G}$ is said (t, ε)-hard if for any advantage $\mathcal{A}$ running within time t, its advantage $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{A})$ is bounded by ε. We also denote by $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$ the best advantage any adversary can get within time t.

It is well-known, using an hybrid argument, or the random-self-reducibility, that the DDH assumption implies the Decisional Multi Diffie-Hellman (DMDH) assumption:

Definition 4 — Decisional Multi Diffie-Hellman (DMDH) Assumption

In a group $\mathbb{G}$ of prime order p, it states that for any generator g and for any constant $n \in \mathbb{N}$, the two following distributions are computationally indistinguishable:

$$\mathcal{D}^n_{\mathsf{mdh}}(g) = \{(g, (g^{x_i})_i, h, (h^{x_i})_i); h \xleftarrow{\$} \mathbb{G}, (x_i)_i \xleftarrow{\$} \mathbb{Z}_p^n\}$$
$$\mathcal{D}^{2n+2}_{\$}(g) = \{(g, (g^{x_i})_i, h, (h^{y_i})_i); h \xleftarrow{\$} \mathbb{G}, (x_i)_i, (y_i)_i \xleftarrow{\$} \mathbb{Z}_p^n\}.$$

2.4 Cryptographic Primitives

In this section we introduce the homomorphic blocks at the core of this thesis. Each time the primitive will be presented in its simplest form with its corresponding security definition and, after that, homomorphic properties will be utilized to enhance it. Hence, the notations will be fixed and already existing constructions given.

The first part focuses on the definition of an encryption scheme then, the second one describes a signature scheme and the last one concerns zero-knowledge proofs.

2.4.1 (Homomorphic) Encryption

We begin with the most famous crytographic primitive: encryption. In this thesis we only use Public-Key Encryption schemes:

Definition 5 — Public-Key Encryption Scheme

A public-key encryption scheme consists of the following algorithms:

$\mathsf{EKeygen}(\kappa)$**:** Given a security parameter κ, it outputs the public key pk with the associated private key sk;

$\mathsf{Encrypt}(\mathsf{pk}, m)$**:** Given a message m and a public key pk, it outputs the ciphertext C;

$\mathsf{Decrypt}(\mathsf{sk}, C)$**:** Given a ciphertext C and a secret key sk, it outputs a message m.

A user with the pair $(\mathsf{sk}, \mathsf{pk})$ of secret-public keys can publish pk to everyone likely to send him a message. However, sk needs to be stored in a safe place.

A public-key encryption scheme if said *correct* is for all message m and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{EKeygen}(\kappa)$,

$$\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{pk}, m)) = m.$$

Example 6 (El Gamal Cryptosystem)**.** To illustrate, the El Gamal encryption scheme (1984) is composed of the three algorithms $(\mathsf{EKeygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ defined on a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order p with:

El Gamal Encryption Scheme [EIG84]

$\mathsf{EKeygen}(\kappa)$**:** Given a security parameter κ, it chooses $x \xleftarrow{\$} \mathbb{Z}_p$ and outputs the public key $\mathsf{pk} = g^{-x}$ and the private key $\mathsf{sk} = x$.

$\mathsf{Encrypt}(\mathsf{pk}, m)$**:** To encrypt a message $M \in \mathbb{G}$ using public key pk, it chooses $r \xleftarrow{\$} \mathbb{Z}_p$ and outputs the ciphertext $C = (M \cdot \mathsf{pk}^r, g^r) \in \mathbb{G}^2$.

$\mathsf{Decrypt}(\mathsf{sk}, C)$**:** Given $C = (c_1, c_2)$ and the private key sk, it computes $c_1 \cdot c_2^{\mathsf{sk}}$.

The scheme is correct as $c_1 \cdot c_2^{\mathsf{sk}} = M \cdot \mathsf{pk}^r \cdot (g^r)^{\mathsf{sk}} = M \cdot g^{-\mathsf{sk}r + r\mathsf{sk}} = M$.

Security. The security of a public encryption scheme can be defined by different levels (from the weakest to the strongest):

IND-CPA: the older is the semantic security, *a.k.a.* Indistinguishability Under Chosen-Plaintext Attacks. Informally, it means that an attacker can not find which message is encrypted after receiving a ciphertext of one of the two messages of its choice;

IND-CCA: the Indistinguishability Under adaptive Chosen-Ciphertext Attacks is similar to the previous scenario except that the attacker has access to an oracle answering the decryption of encrypted messages. The only restriction is that the attacker can not request the decryption of the challenge ciphertext.

$$\mathsf{IND\text{-}CCA} \Rightarrow \mathsf{IND\text{-}CPA}$$

Let us now formally define the semantic security for a public-key encryption scheme. The attack is done in two steps, and so the adversary outputs a state s to resume the process in the second step:

Definition 7 — IND-CPA

Let $\mathcal{E} = (\mathsf{EKeygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be an encryption scheme. Let us denote $\mathsf{Exp}_{\mathcal{E}}^{\mathsf{ind\text{-}cpa}\text{-}b}(\mathcal{A})$ the experiment defined by:

$\mathsf{Exp}_{\mathcal{E}}^{\mathsf{ind\text{-}cpa}\text{-}b}(\mathcal{A})$**:**
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{EKeygen}(\kappa)$
$(s, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk})$
$C \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b)$
$b' \leftarrow \mathcal{A}(s, C)$
return b'

The advantage $\mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind\text{-}cpa}}(\mathcal{A})$ of an adversary $\mathcal{A}$ against *indistinguishability under chosen plaintext attacks* (IND-CPA) is

$$\Pr[\mathsf{Exp}_{\mathcal{E}}^{\mathsf{ind\text{-}cpa}\text{-}1}(\mathcal{A}) = 1] - \Pr[\mathsf{Exp}_{\mathcal{E}}^{\mathsf{ind\text{-}cpa}\text{-}0}(\mathcal{A}) = 1].$$

An encryption scheme $\mathcal{E}$ is said $(t, \varepsilon) - \mathsf{IND\text{-}CPA}$ if for any adversary $\mathcal{A}$ running within time t, its advantage $\mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind\text{-}cpa}}(\mathcal{A})$ is bounded by ε and we denote by $\mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind\text{-}cpa}}(t)$ the best advantage any adversary $\mathcal{A}$ can get within time t.

For example, the El Gamal cryptosystem is IND-CPA secure under the DDH assumption.

Homomorphic Encryption Scheme

An encryption scheme is said *partially homomorphic* or homomorphic within a group $(G, \star)$ if one can define an additional algorithm taking as input two ciphertexts C on implicit message m and C' on implicit message m' and producing a ciphertext C'' on the message $m \star m'$. For a multiplicative group $(G, \cdot)$, this algorithm is usually denoted $\mathsf{Multiply}(C, C')$ whereas for an additive group $(G, +)$, it is denoted $\mathsf{Add}(C, C')$.

If the scheme is at the same time additively (meaning having an Add algorithm as defined above) and multiplicatively (meaning having an Multiply algorithm as defined above) homomorphic then, one can evaluate any function on ciphertexts and the scheme is said *fully homomorphic*.

Example 8 (El Gamal Cryptosystem)**.** The El Gamal encryption scheme as presented in Example 6 is homomorphic in $(\mathbb{G}, \cdot)$ as one can define an algorithm Multiply:

Multiply(C, C'): Given two El Gamal ciphertexts $C = (c_1, c_2)$ and $C' = (c'_1, c'_2)$, it outputs

$$C'' = (c_1 \cdot c'_1, c_2 \cdot c'_2).$$

This works as:

$$C'' = ((M \cdot \mathsf{pk}^r) \cdot (M' \cdot \mathsf{pk}^{r'}), g^r \cdot g^{r'}) = ((M \cdot M') \cdot \mathsf{pk}^{r+r'}, g^{r+r'}) = \mathsf{Encrypt}(\mathsf{pk}, M \cdot M').$$

However, by changing the message space from $\mathbb{G}$ to $\mathbb{Z}_p$ and by defining the encryption of $m \in \mathbb{Z}_p$ by $\mathsf{Encrypt}(\mathsf{pk}, m) = (g^m \cdot \mathsf{pk}^r, g^r)$. The scheme becomes homomorphic in $(\mathbb{Z}_p, +)$:

Add(C, C'): Given two El Gamal ciphertexts $C = (c_1, c_2)$ and $C' = (c'_1, c'_2)$, it outputs

$$C'' = (c_1 \cdot c'_1, c_2 \cdot c'_2).$$

We will often use this El Gamal version in this manuscript.

Security. An homomorphic encryption scheme can be IND-CPA secure however, IND-CCA2 can not be achieved. Indeed, the attacker chooses two messages m_0, m_1 and sends them to the challenger, the challenger chooses $b \xleftarrow{\$} \{0, 1\}$ and encrypts m_b into C_b. The attacker receiving C_b simply computes $C = C_b \times C_b$ and asks the decryption to obtain $m = 2 \cdot m_b$. Finding the right $b' = b$ is then easy.

Freeman Cryptosystem.

To evaluate 2-DNF formulae on encrypted data, Boneh-Goh-Nissim described a cryptosystem [BGN05] in 2005 that supports additions, one multiplication layer, and additions again. They used a bilinear map on a composite-order group and the secret key is the factorization of the order of the group. Unfortunately, composite-order groups require huge orders, since the factorization must be difficult, with costly pairing evaluations.

In order to improve the efficiency on this cryptosystem, Freeman in [Fre10] proposed a system on prime-order groups, using a similar property of noise that can be removed, with the general definition of subgroup decision problem:

Freeman's Cryptosystem [Fre10]

EKeygen(κ): Given a security parameter κ, it generates $(G, H, G_T, p, g, \mathfrak{h}, e) \leftarrow \mathcal{G}(\kappa)$, two subgroups $G_1 \subset G, H_1 \subset H$ and three homomorphisms π_1, π_2, π_T such that G_1, H_1 are contained in the kernels of π_1, π_2 respectively and $e(\pi_1(g), \pi_2(\mathfrak{h})) = \pi_T(e(g, \mathfrak{h}))$. Finally, it outputs the public key $\mathsf{pk} = (G, G_1, H, H_1, G_T, e, g, \mathfrak{h})$ and the private key $\mathsf{sk} = (\pi_1, \pi_2, \pi_T)$;

Encrypt(pk, m): To encrypt a message m using public key pk, one picks $g_1 \xleftarrow{\$} G_1$ and $\mathfrak{h}_1 \xleftarrow{\$} H_1$, and outputs the ciphertext $(C_A, C_B) = (g^m \cdot g_1, \mathfrak{h}^m \cdot \mathfrak{h}_1) \in G \times H$;

Decrypt(sk, C): Given $C \in G$ (resp. $\in H$ or $\in G_T$), it outputs $m \leftarrow \log_{\pi_1(g)}(\pi_1(C))$ (resp. $\log_{\pi_2(\mathfrak{h})}(\pi_2(C))$ or $\log_{\pi_T(e(g,\mathfrak{h}))}(\pi_T(C))$).

Freeman's scheme also has homomorphic properties:

Add(C, C'): Given two ciphertexts C and C' of G (resp. of H or of G_T), it chooses $g_1 \in G_1$ and $\mathfrak{h}_1 \in H_1$ and outputs the ciphertext $C'' = C \cdot C' \cdot g_1$ (resp. $C'' = C \cdot C' \cdot h_1$ or $C'' = C \cdot C' \cdot e(g_1, \mathfrak{h}) \cdot e(g, \mathfrak{h}_1)$);

Multiply(C, C')**:** Given a ciphertext $C \in G$ and $C' \in H$, it outputs the ciphertext $C'' = e(C, C') \cdot e(g_1, \mathfrak{h}) \cdot e(g, \mathfrak{h}_1)$.

This scheme is secure under the subgroup decision problem which informally means that an attacker can not distinguish elements of the subgroup $G_1 \subset G$ from elements of G and as well as elements of the subgroup $H_1 \subset H$ from elements of H.

2.4.2 (Homomorphic) Signature

After having described the encryption schemes with their homomorphic variant, we will now present the signature schemes following the same structure. First, we provide the general definition with an example and the security requirement. Then, we extend the signature schemes to their homomorphic version.

In their seminal article [DH76], Diffie and Hellman described a digital signature scheme as a "digital phenomenon with the same properties as a written signature. It must be easy for anyone to recognize the signature as authentic, but impossible for anyone other than the legitimate signer to produce it":

Definition 9 — Digital Signature Scheme

A signature scheme consists of the following algorithms:

SKeygen(κ)**:** Given a security parameter κ, it outputs the (public) verification key vk with the associated (private) signing key sk;

Sign(sk, m)**:** Given a signing key and a message m, it outputs the signature σ;

VerifSign(vk, m, σ)**:** Given a verification key vk, a message m and a signature σ, it outputs 1 if σ is a valid signature relative to vk, and 0 otherwise.

A signature scheme is said *correct* is for all message m and for all $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SKeygen}(\kappa)$,

$$\mathsf{VerifSign}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1.$$

Example 10 (Schnorr Signature Scheme)**.** Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p and $\mathcal{H} : \{0,1\}^* \times \mathbb{G} \to \mathbb{Z}_p$ be a hash function. The Schnorr signature scheme (1989) is composed of the three algorithms $(\mathsf{SKeygen}, \mathsf{Sign}, \mathsf{VerifSign})$ with:

Schnorr Signature Scheme [Sch90]

SKeygen(κ)**:** Given a security parameter κ, it outputs the signing key $\mathsf{sk} = x \xleftarrow{\$} \mathbb{Z}_p^*$ and the verification key $\mathsf{vk} = g^x$;

Sign(sk, m)**:** Given a signing key sk and a message $m \in \{0,1\}^*$, it chooses $a \xleftarrow{\$} \mathbb{Z}_p$ and computes $r = g^a$, $c = \mathcal{H}(m, r)$ and $s = a + cx \bmod p$. The signature is then $\sigma = (s, c)$;

VerifSign(vk, m, σ)**:** Given a verification key vk, a message m and a signature $\sigma = (s, c)$, it computes $r = g^s \cdot \mathsf{vk}^{-c}$ and outputs 1 if $c = \mathcal{H}(m, r)$, and 0 otherwise.

The scheme is correct as $\mathcal{H}(m, r) = \mathcal{H}(m, g^s \cdot \mathsf{vk}^{-c}) = \mathcal{H}(m, g^{s-xc}) = \mathcal{H}(m, g^a)$.

Security. The security of a signature scheme was defined for the first time by Goldwasser *et al.* [GMR88] and is called the *unforgeability*. As for encryption, one can distinguished different levels of attackers (from the weakest scheme to the safest):

EUF: a scheme is said to be Existentially Unforgeable (EUF) if an attacker can not succeed in forging the signature of one message, even not of his choice;

SUF: a scheme is said to be Strong Unforgeable (SUF) if an attacker can not succeed in forging the signature of one message, even not of his choice;

Let us now formally define the universal unforgeability for a signature scheme:

Definition 11 — EUF-CMA

Let $\Sigma = (\mathsf{SKeygen}, \mathsf{Sign}, \mathsf{VerifSign})$ be a signature scheme and $\mathcal{A}$ an attacker against it having access to the signing oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$. Let us denote $\mathsf{Exp}^{\text{euf-cma}}(1^\kappa)$ the experiment defined by:

$\mathsf{Exp}^{\textbf{euf-cma}}(1^\kappa)$**:**
$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SKeygen}(\kappa)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{vk})$
return m^* "new" $\wedge\ \mathsf{VerifSign}(\mathsf{vk}, m^*, \sigma^*)$

The advantage $\mathsf{Adv}^{\text{euf-cma}}(\mathcal{A})$ of an adversary $\mathcal{A}$ against *existential unforgeability under chosen-message attacks* is $\mathsf{Adv}^{\text{euf-cma}}(\mathcal{A}) = |\Pr[\mathsf{Exp}^{\text{euf-cma}}(1^\kappa) = 1]|$. A signature scheme Σ is said $t-\mathsf{EUF\text{-}CMA}$ secure if for any adversary $\mathcal{A}$ running in time t polynomial in κ and making a number of signing queries also polynomial in κ, its advantage $\mathsf{Adv}^{\text{euf-cma}}\mathcal{E}(\mathcal{A})$ is negligible.

By replacing m^* "new" by (m^*, σ^*) "new" in the definition above, we obtain the formal definition of the Strong Unforgeability.

Homomorphic Signature Schemes

In this part, we will provide the informal notions of the homomorphic signature schemes as a more detailed presentation of the specific linearly homomorphic signatures will be done in the Chapter 4. The concept dates back to Desmedt [Des93] in 1993.

Similarly to encryption, a signature scheme is said *homomorphic* in $(G, \cdot)$ if one can define an additional algorithm:

$\mathsf{MultiplySign}(\mathsf{vk}, (\omega_i, m_i, \sigma_i)_{i=1}^{\ell})$**:** Given a public key vk and ℓ tuples of weights ω_i and signed messages m_i in σ_i, it outputs a signature σ on the message $m = \prod_{i=1}^{\ell} m_i^{\omega_i}$.

Example 12 (Libert *et al.* [LPJY13] Signature Scheme)**.** In their article, Libert *et al.* [LPJY13] introduce a new signature scheme proven in the standard model with messages in $\mathcal{M} \in \mathbb{G}^n$, for a cyclic group $(\mathbb{G}, \times)$ of prime order p and some $n \in \mathsf{poly}(\kappa)$:

One-Time Linearly Homomorphic Signature Scheme [LPJY13]

$\mathsf{SKeygen}(\kappa, n)$**:** Given a security parameter κ and the dimension n, it generates $(\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathcal{G}(\kappa)$ a symmetric bilinear setting of prime order p. Then, it choses generators $h, g_z, g_r, h_z \xleftarrow{\$} \mathbb{G}$. For $i = 1$ to n, it picks $\chi_i, \gamma_i, \delta_i \xleftarrow{\$} \mathbb{Z}_p$ and computes $g_i = g_z^{\chi_i} g_r^{\gamma_i}, h_i = h_z^{\chi_i} h^{\delta_i}$. The signing key is $\mathsf{sk} = \{\chi_i, \gamma_i, \delta_i\}_{i=1}^n$ and the verification key is $\mathsf{vk} = (g_z, h_r, h_z, h, \{g_i, h_i\}_{i=1}^n)$;

$\mathsf{Sign}(\mathsf{sk}, \boldsymbol{M})$**:** Given a signing key $\mathsf{sk} = \{\chi_i, \gamma_i, \delta_i\}_{i=1}^n$ and a vector-message $\boldsymbol{M} = (M_i)_i \in$

$\mathbb{G}^n$, it outputs the signature $\sigma = (z, r, u)$ with

$$z = \prod_{i=1}^{n} M_i^{-\chi_i}, \qquad r = \prod_{i=1}^{n} M_i^{-\gamma_i}, \qquad u = \prod_{i=1}^{n} M_i^{-\delta_i};$$

VerifSign($\mathsf{vk}, \boldsymbol{M}, \sigma$)**:** Given a verification key vk, a vector-message $\boldsymbol{M} = (M_i)_i$ and a signature σ, it outputs 1 if $\boldsymbol{M} \neq 1_{\mathbb{G}_T^n}$ and $\sigma = (z, r, u)$ satisfies

$$1_{\mathbb{G}_T} = e(g_z, z) \cdot e(g_r, r) \cdot \prod_{i=1}^{n} e(g_i, M_i), \qquad 1_{\mathbb{G}_T} = e(h_z, z) \cdot e(h, u) \cdot \prod_{i=1}^{n} e(h_i, M_i).$$

Their scheme was inspired by the one-time structure-preserving signature of Abe *et al.*[AFG+10] and is multiplicatively homomorphic as one can define a MultiplySign algorithm:

MultiplySign($\mathsf{vk}, (\omega_i, \boldsymbol{M}_i, \sigma_i)_{i=1}^{\ell}$)**:** Given a public key vk and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signed messages $\boldsymbol{M}_i$ in $\sigma_i = (z_i, r_i, u_i)$, it outputs the signature $\sigma = (z, r, u)$ with:

$$z = \prod_{i=1}^{\ell} z_i^{\omega_i}, \qquad r = \prod_{i=1}^{\ell} r_i^{\omega_i}, \qquad u = \prod_{i=1}^{\ell} u_i^{\omega_i};$$

The output of this algorithm is a valid signature on the vector $\boldsymbol{M} = \prod_{i=1}^{\ell} \boldsymbol{M}_i^{\omega_i}$. In fact, as for El Gamal, the signature scheme can be seen as a linearly homomorphic one as the message can be $\mathsf{m} = (m_1, \ldots, m_n) \in \mathbb{Z}_p^n$ such that $\boldsymbol{M} = (g^{m_1}, \ldots, g^{m_n})$.

Security. Similarly to homomorphic encryption, the security of homomorphic signatures needs to be specified. Indeed, if one can evaluate any function of previously seen signatures of messages, an attacker can forged any message of his choice. Informally, the security will require that an attacker can not forge a message outside an authorized space.

For example, the Libert *et al.* scheme defined above considers that a signature $\sigma = \sigma_1^{\alpha} \cdot \sigma_2^{\beta}$ is not a forgery if the attacker previously asked for the signatures σ_1 and σ_2. The attacker needs to provide a signature outside any linear combination of previously asked signatures.

Aggregate Signature Schemes

In 2001, Boneh, Lynn and Shacham propose a new signature scheme [BLS01] that was extended firstly in 2007 by Bellare, Namprempre and Neven [BNN07] and then, in 2018 by Boneh, Drijvers and Neven [BDN18]. These extensions allow aggregation of multiple messages signed by multiple users. Hence, their signature is constant-size. Moreover, the verification is also constant-time when the same message is signed by all the users (multi-signature). Since this is the most interesting case for us, we focus on it.

BLS Aggregate Signature Scheme

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p and $\mathcal{H} : \{0,1\}^* \to \mathbb{G}^*$ be a full-domain hash function. The BLS signature scheme is composed of the three algorithms (SKeygen, Sign, VerifSign) with:

Boneh-Lynn-Shacham Signature Scheme [BLS01]

SKeygen(κ)**:** Given a security parameter κ, it chooses $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(\mathsf{sk}, \mathsf{vk} = \mathfrak{g}^{\mathsf{sk}})$;

Sign(sk, m)**:** Given a signing key sk and a message $m \in \{0,1\}^*$, it outputs the signature

$\sigma = \mathcal{H}(m)^{\mathsf{sk}}$;

VerifSign(vk, m, σ)**:** Given a verification key vk, a message m and a signature σ, it outputs 1 if and only if $(g, \mathsf{vk}, \mathcal{H}(m), \sigma)$ is a Diffie-Hellman tuple.

Given N tuples $(\mathsf{vk}_i, \sigma_i, m_i)$, it is possible to aggregate the signatures into a unique one $\sigma = \prod \sigma_i$. To verify σ, one needs to check that:

$$e(g, \sigma) = \prod_i e(\mathsf{vk}_i, \mathcal{H}(m_i)).$$

However, this scheme is sensitive to the rogue public-key attack if all the message m_i are the same. In 2018, Boneh, Drijvers and Neven proposed a scheme to solve this issue.

BDN Multi-Signature Scheme

Let $\mathsf{MSparam} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e) \leftarrow \mathcal{G}(\kappa)$ be an asymmetric pairing setting and let $\mathcal{H}_0 : \{0,1\}^* \to \mathbb{G}_1$ and $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$ be two full-domain hash functions.

Boneh-Drijvers-Neven Multi-Signature Scheme [BDN18]

MSKeygen$(\mathsf{MSparam})$**:** Given the global parameter $\mathsf{MSparam}$, it chooses $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(\mathsf{sk}, \mathsf{vk} = \mathfrak{g}^{\mathsf{sk}})$;

MSKeyAgg$(\{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\})$**:** Given N verification keys vk_i, it outputs the aggregated verification key $\mathsf{avk} = \prod_{i=1}^{N} \mathsf{vk}_i^{\mathcal{H}_1(\mathsf{vk}_i, \{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\})}$;

MSSign$(\{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\}, \mathsf{sk}_i, m)$**:** Given N verification keys vk_i, a signing key sk_i and a message m, it outputs $\sigma_i = \mathcal{H}_0(m)^{\mathsf{sk}_i}$.
From all the individual signatures σ_i, any combiner (who can be one of the signers) computes the multi-signature $\mathsf{msig} = \prod_{j=1}^{N} \sigma_j^{\mathcal{H}_1(\mathsf{vk}_j, \{\mathsf{vk}_1, \ldots, \mathsf{vk}_N\})}$;

MSVerif$(\mathsf{avk}, m, \mathsf{msig})$**:** Given a verification key avk, a message m and a signature msig, it outputs 1 if and only if $e(g, \mathsf{msig}) = e(\mathcal{H}_0(m), \mathsf{avk})$.

Since the aggregated verification key can be precomputed, verification just consists of two pairing evaluations.

2.4.3 (Homomorphic) Proof

A zero-knowledge proof (ZK) [GMR85] is a protocol between two players: a Prover and a Verifier where the prover needs to convince the verifier of a given statement without revealing any other information that the truth of this statement. The prover knowing a witness w and a statement s provides a proof π that the verifier, given the statement s, accepts or rejects.

Informally, a zero-knowledge proof must satisfy the three properties:

Completeness: if the prover and the verifier follow the protocol, the verifier always accepts;

Soundness: a false proof is rejected with a probability of at least 1/2;

Zero-knowledge: whatever the verifier learns, he could have learned by himself without any interaction with the prover.

Moreover, the zero-knowledge proof is said non-interactive (NIZK) if the prover simply sends a proof π and, later, the verifier can verify π without any more information from the prover.

To achieve that, the classical method uses commitment schemes: the prover first commits some random values and receives a challenge from the verifier, then the prover sends his response.

A commitment scheme ensures that a user committing a value m such that $\mathsf{Com} = \mathsf{Commit}(m)$ reveals no information about m (hiding property) but prevents the user from modifying m later: there is only one way to open the commitment (binding property).

In particular, a commitment scheme is said *perfectly hiding* (resp. *perfectly blinding*) if the hiding property (resp. blinding property) is true independently of the power of the attacker and computationally if it relies on a computational assumption.

Example 13. To prove $(g, h, A = g^x, B = h^x)$ is a Diffie-Hellman tuple with a Schnorr-like zero-knowledge proof, the prover first chooses $r \xleftarrow{\$} \mathbb{Z}_p$ and sends the commitments $U = g^r$ and $V = h^r$ to the verifier that answers a challenge $c \in \mathbb{Z}_p$. The prover constructs its response $s = r - x \cdot c \bmod p$ and the verifier checks whether both $U = g^s \cdot A^c$ and $V = h^s \cdot B^c$ hold.

To make the proof non-interactive, one can use the Fiat-Shamir heuristic with c generated by a hash function on the statement (g, h, A, B) and commitment (U, V). The proof eventually consists of (c, s). From this proof, one can compute the candidates for (U, V), and check whether the hash value gives back c.

Groth-Sahai Proofs

Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e) \leftarrow \mathcal{G}(\kappa)$ be an asymmetric pairing setting. We recall the Groth-Sahai methodology [GS08] to prove a Diffie-Hellman tuple in $\mathbb{G}_2$ (because it will be used in $\mathbb{G}_2$ later). As the Groth-Sahai proofs are randomizable [BCC+09], we will also present how to randomize a Diffie-Hellman proof of $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ and how to update it into a new one for $(\mathfrak{g}, \mathfrak{g}'', \mathfrak{A}, \mathfrak{A}'')$.

First, we set a tuple $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2}) \in \mathbb{G}_1^4$, such that $(v_{1,1}, v_{1,2}, v_{2,1}, g \times v_{2,2})$ is not a Diffie-Hellman tuple. Then, given a Diffie-Hellman tuple $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ in $\mathbb{G}_2$, knowing the witness $\alpha \in \mathbb{Z}_p$ such that $\mathfrak{A} = \mathfrak{g}^\alpha$ and $\mathfrak{A}' = \mathfrak{g}'^\alpha$, one first commits α:

$$\mathsf{Com} = (c = v_{2,1}^\alpha \cdot v_{1,1}^\mu, d = v_{2,2}^\alpha \cdot g^\alpha \cdot v_{1,2}^\mu)$$

for a random $\mu \xleftarrow{\$} \mathbb{Z}_p$, and one sets $\Theta = \mathfrak{g}^\mu$ and $\Psi = \mathfrak{A}^\mu$, which satisfy:

$$e(c, \mathfrak{g}) = e(v_{2,1}, \mathfrak{g}') \cdot e(v_{1,1}, \Theta) \qquad e(d, \mathfrak{g}) = e(v_{2,2} \cdot g, \mathfrak{g}') \cdot e(v_{1,2}, \Theta)$$
$$e(c, \mathfrak{A}) = e(v_{2,1}, \mathfrak{A}') \cdot e(v_{1,1}, \Psi) \qquad e(d, \mathfrak{A}) = e(v_{2,2} \cdot g, \mathfrak{A}') \cdot e(v_{1,2}, \Psi)$$

The proof $\mathsf{proof} = (\mathsf{Com}, \Theta, \Psi)$, when it satisfies the above relations, guarantees that $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ is a Diffie-Hellman tuple.

Security. This proof is zero-knowledge, under the DDH assumption in $\mathbb{G}_1$: by switching $(v_{1,1}, v_{1,2}, v_{2,1}, g \times v_{2,2})$ into a Diffie-Hellman tuple, one can simulate the proof, as the commitment becomes perfectly hiding.

Efficiency. To verify the proof, instead of checking the four equations independently, one can apply a batch verification [BFI+10], and pack them in a unique one with random scalars $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \xleftarrow{\$} \mathbb{Z}_p$:

$$e(c^{x_{1,1}} d^{x_{1,2}}, \mathfrak{g}^{x_{2,1}} \mathfrak{A}^{x_{2,2}}) = e(v_{2,1}^{x_{1,1}} (v_{2,2} \cdot g)^{x_{1,2}}, \mathfrak{g}'^{x_{2,1}} \mathfrak{A}'^{x_{2,2}}) \times e(v_{1,1}^{x_{1,1}} v_{1,2}^{x_{1,2}}, \Theta^{x_{2,1}} \Psi^{x_{2,2}})$$

One thus just has 3 pairing evaluations.

Updating the Diffie-Hellman proof. The interesting property of Groth-Sahai proofs is that it is possible from a Diffie-Hellman proof for $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ to generate the Diffie-Hellman proof for $(\mathfrak{g}, \mathfrak{g}'' = \mathfrak{g}'^{\alpha'}, \mathfrak{A}, \mathfrak{A}'' = \mathfrak{A}'^{\alpha'})$ just knowing the incremental witness α', whereas the new witness shoud be $\alpha\alpha'$, but is unknown to the prover: from the Diffie-Hellman proof $\mathsf{proof} = (\mathsf{Com}, \Theta, \Psi)$ for $(\mathfrak{g}, \mathfrak{g}', \mathfrak{A}, \mathfrak{A}')$ where

$$\mathsf{Com} = (c = v_{2,1}^{\alpha} v_{1,1}^{\mu}, d = v_{2,2}^{\alpha} v_{1,2}^{\mu} g^{\alpha}) \qquad \Theta = \mathfrak{g}^{\mu} \qquad \Psi = \mathfrak{A}^{\mu}$$

one can compute the proof proof' for $(\mathfrak{g}, \mathfrak{g}'' = \mathfrak{g}'^{\alpha'}, \mathfrak{A}, \mathfrak{A}'' = \mathfrak{A}'^{\alpha'})$, with $\mu' \xleftarrow{\$} \mathbb{Z}_p$ and:

$$\mathsf{Com}' = (c^{\alpha'} \cdot v_{1,1}^{\mu'}, d^{\alpha'} \cdot v_{1,2}^{\mu'}) \qquad \Theta' = \Theta^{\alpha'} \cdot \mathfrak{g}^{\mu'} \qquad \Psi' = \Psi^{\alpha'} \cdot \mathfrak{A}^{\mu'}$$

One implicitly updates α into $\alpha\alpha'$ and μ into $\alpha'\mu + \mu'$. In particular, one can remark that if $\alpha' = 1$, this simply randomizes the original proof.

Chapter

3 Decentralized Evaluation of Quadratic Polynomials on Encrypted Data

This chapter is based on the paper [HPP19] published in the proceedings of the International Conference on Information Security, ISC 2019.

Chapter content

In 2005, Boneh, Goh and Nissim [BGN05] proposed a nice solution for quadratic polynomials evaluation. However, their solution relies on a composite-order elliptic curve and thus on the hardness of the integer factoring. This possibly leads to a distributed solution, but that is highly inefficient. Indeed, no efficient multi-party generation of distributed RSA modulus is known, except for 2 parties. Catalano and Fiore [CF15] introduced an efficient technique to transform a linearly-homomorphic encryption into a scheme able to evaluate quadratic operations on ciphertexts. They are able to support decryption of a large plaintext space after the multiplication. However, as in Kawai *et al.* [KMH+19] which used this technique to perform proxy re-encryption, they only consider a subclass of degree-2 polynomials where the number of additions of degree-2 terms is bounded by a constant. This is not enough for most of the applications and we do not try to decentralize these limited protocols.

In 2010, Freeman [Fre10] proposed a conversion from composite-order groups to prime-order groups for the purpose of improving the efficiency. In Section 3.1, we show Freeman's conversion allows multi-user setting, since a common setup can handle several keys and we show it

is well-suited for distributed evaluation of 2-DNF formulae. However, it is not enough to have an efficient distributed setup. One also needs to distribute any use of the private keys in the construction: for decryption and re-encryption. Unfortunately, Freeman's generic description with projection matrices does not directly allow the design of a decentralized scheme, *i.e.*, with efficient distributed (threshold) decryption without any trusted dealer. We thus specify in Section 3.2 particular projections, with well-chosen private and public keys. Finally, in Section 3.3, we propose two more applications that are related to the group testing and the consistency model in machine learning.

Related Work. In a previous and independent work, Attrapadung *et al.* [AHM+18] proposed an efficient two-level homomorphic encryption in prime-order groups. They put forward a new approach that avoids Freeman's transformation from BGN encryption. Interestingly, our work shows this scheme falls into Freeman's framework because their construction is similar to the simplified non-decentralized version of our scheme which is obtained from BGN via a Freeman transformation with a particular choice of projections. The concrete implementations in [AHM+18] show that such a scheme is quite efficient, which applies to our construction, and even to the distributed construction as each server, for a partial decryption, essentially has to perform a decryption with its share. In another unpublished work [CPRT18], Culnane *et al.* considered a universally verifiable MPC protocol in which one of the two steps is to distribute the key generation in somewhat homomorphic cryptosystems. However, as we mentioned above, Freeman's generic description with projection matrices, as considered in [CPRT18], does not lead to an efficient distributed decryption. In short, our result bridges the gap between the objective of decentralization as in [CPRT18] and the efficiency goal as in [AHM+18].

3.1 Freeman's Approach

3.1.1 Notations

In this section, for a generator g of a cyclic group $\mathbb{G} = \langle g \rangle$, we use the implicit representation $[a]$ of any element $h = g^a \in \mathbb{G}$ and by extension we will use the "bracket" notations, which makes use of the matrix properties over the exponents defined in Chapter 2, that are scalars in $\mathbb{Z}_p$ when $\mathbb{G}$ is a cyclic group of order p. See Figure 3.1 for more details about the "brackets".

- For $x \in \mathbb{Z}_p, \mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$: $[x] = g^x$, $[\mathbf{A}] = g^{\mathbf{A}} = (g^{a_{ij}})_{ij}$
- For $x \in \mathbb{Z}_p, \mathbf{A}, \mathbf{B} \in \mathcal{M}_{m,n}(\mathbb{Z}_p), \mathbf{X} \in \mathcal{M}_{n,n'}(\mathbb{Z}_p), \mathbf{Y} \in \mathcal{M}_{m',m}(\mathbb{Z}_p)$:

  $$x \cdot [\mathbf{A}] = g^{x\mathbf{A}} \qquad [\mathbf{A}] \cdot \mathbf{X} = g^{\mathbf{AX}} \qquad \mathbf{Y} \cdot [\mathbf{A}] = g^{\mathbf{YA}} \qquad [\mathbf{A}] \boldsymbol{+} [\mathbf{B}] = [\mathbf{A} + \mathbf{B}]$$

- For $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$, $\mathbf{B} \in \mathcal{M}_{m',n'}(\mathbb{Z}_p)$: $[\mathbf{A}]_1 \bullet [\mathbf{B}]_2 = [\mathbf{A} \otimes \mathbf{B}]_T$

Figure 3.1: Bracket Notations

In case of bilinear groups, we also define, for $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{Z}_p)$ and $\mathbf{B} \in \mathcal{M}_{m',n'}(\mathbb{Z}_p)$, $[\mathbf{A}]_1 \bullet [\mathbf{B}]_2 = [\mathbf{A} \otimes \mathbf{B}]_T$, which can be evaluated with pairing operations between $\mathbb{G}_1$ and $\mathbb{G}_2$ group elements.

3.1.2 Freeman's Scheme with Projections

The main goal of Freeman's approach was to generalize the BGN cryptosystem to any hard-subgroup problems, which allows applications to prime-order groups under the classical Decisional Diffie-Hellman or Decisional Linear assumptions, with high gain in efficiency.

We now present a variant of Freeman's cryptosystem allowing multiple users, without the twin ciphertexts (in G and H). Since we will work in groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, the algorithms $\mathsf{EKeygen}, \mathsf{Encrypt}$ and $\mathsf{Decrypt}$ will take a sub-script s to precise the group $\mathbb{G}_s$ in which they operate, but the Setup is common.

Multi-User Freeman's Cryptosystem

$\mathsf{Setup}(\kappa)$**:** Given a security parameter κ, it runs and outputs $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathcal{G}(\kappa)$.

$\mathsf{EKeygen}_s(\mathsf{param})$**:** For $s \in \{1,2\}$, it chooses $\mathbf{B}_s \xleftarrow{\$} \mathrm{GL}_2(\mathbb{Z}_p)$, let $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$ and $\mathbf{p}_s \in \ker(\mathbf{P}_s) \setminus \{\mathbf{0}\}$, and outputs the public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$ and the private key $\mathsf{sk}_s = \mathbf{P}_s$.

From $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{EKeygen}_1(\mathsf{param})$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{EKeygen}_2(\mathsf{param})$, one can consider $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{sk}_T = (\mathsf{sk}_1, \mathsf{sk}_2)$, which are associated public and private keys in $\mathbb{G}_T$, as we explain below.

$\mathsf{Encrypt}_s(\mathsf{pk}_s, m, A_s)$**:** For $s \in \{1,2\}$, to encrypt a message $m \in \mathbb{Z}_p$ using public key pk_s and $A_s = [\mathbf{a}]_s \in \mathbb{G}_s^2$, it chooses $r \xleftarrow{\$} \mathbb{Z}_p$ and outputs the ciphertext $C_s = (m \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \in \mathbb{G}_s^2 \times \mathbb{G}_s^2$.
For $s = T$, with $A_s = ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2)$, it sets $[\mathbf{a}]_T = [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2 \in \mathbb{G}_T^4$, chooses $[\mathbf{r}_1]_1 \xleftarrow{\$} \mathbb{G}_1^2$, $[\mathbf{r}_2]_2 \xleftarrow{\$} \mathbb{G}_2^2$, and outputs $C_T = (m \cdot [\mathbf{a}]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2, [\mathbf{a}]_T) \in \mathbb{G}_T^4 \times \mathbb{G}_T^4$.

$\mathsf{Decrypt}_s(\mathsf{sk}_s, C_s)$**:** For $s \in \{1,2\}$, given $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ and $\mathsf{sk}_s = \mathbf{P}_s$, it lets $C'_s = ([\mathbf{c}_{s,1}]_s \cdot \mathbf{P}_s, [\mathbf{c}_{s,2}]_s \cdot \mathbf{P}_s)$.
For $s = T$, it computes $C'_T = ([\mathbf{c}_{T,1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2), [\mathbf{c}_{T,2}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2))$.
In both cases, it outputs the logarithm of the first component of $\mathbf{c}'_{s,1}$ in base the first component of $\mathbf{c}'_{s,2}$.

With the algorithms defined above, we have three encryption schemes $\mathcal{E}_s : (\mathsf{Setup}, \mathsf{EKeygen}_s, \mathsf{Encrypt}_s, \mathsf{Decrypt}_s)$ for $s = 1, 2$ or T, with a common Setup.

Remark. We note that in Freeman's cryptosystem, ciphertexts contain encryptions of m in both $\mathbb{G}_1$ and $\mathbb{G}_2$ to allow any kind of additions and multiplication. But one could focus on one ciphertext only when the formula to be evaluated is known.

Proposition 14. *For $s \in \{1, 2, T\}$, $\mathcal{E}_s$ is correct.*

Proof. For $s = 1, 2$:

$$
\begin{aligned}
[\mathbf{c}_{s,2}]_s \cdot \mathbf{P}_s &= [\mathbf{a}]_s \cdot \mathbf{P}_s = [\mathbf{a}\mathbf{P}_s]_s \\
[\mathbf{c}_{s,1}]_s \cdot \mathbf{P}_s &= (m \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s) \cdot \mathbf{P}_s = m \cdot [\mathbf{a}]_s \cdot \mathbf{P}_s + r \cdot [\mathbf{p}_s]_s \cdot \mathbf{P}_s \\
&= m \cdot [\mathbf{a}\mathbf{P}_s]_s + r \cdot [\mathbf{p}_s\mathbf{P}_s]_s = m \cdot [\mathbf{a}\mathbf{P}_s]_s + r \cdot [\mathbf{0}]_s = m \cdot [\mathbf{a}\mathbf{P}_s]_s
\end{aligned}
$$

For $s = T$:

$$
\begin{aligned}
&[\mathbf{c}_{T,2}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = [\mathbf{a}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T \\
&[\mathbf{c}_{T,1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = (m \cdot [\mathbf{a}]_T \bullet [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 \bullet [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2) \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \\
&= m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T \bullet ([\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2) \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \bullet ([\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2) \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \\
&= m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T \bullet [\mathbf{p}_1 \otimes \mathbf{r}_2]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \bullet [\mathbf{r}_1 \otimes \mathbf{p}_2]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \\
&= m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T \bullet [\mathbf{p}_1\mathbf{P}_1 \otimes \mathbf{r}_2\mathbf{P}_2]_T \bullet [\mathbf{r}_1\mathbf{P}_1 \otimes \mathbf{p}_2\mathbf{P}_2]_T \\
&= m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T \bullet [\mathbf{0} \otimes \mathbf{r}_2\mathbf{P}_2]_T \bullet [\mathbf{r}_1\mathbf{P}_1 \otimes \mathbf{0}]_T = m \cdot [\mathbf{a}(\mathbf{P}_1 \otimes \mathbf{P}_2)]_T
\end{aligned}
$$

In both cases, $C'_{s,1} = [\mathbf{c}'_{s,1}]_s = m \cdot [\mathbf{c}'_{s,2}]_s = m \cdot C'_{s,2}$. Whatever the size of the vectors, one discrete logarithm computation is enough to extract m. □

As already explained above, the encryption process masks the message by an element in the kernel of a certain projection. The secret key is the corresponding projection $\mathbf{P}_s$ which later removes this mask. In the Decrypt algorithm, C'_s is a Diffie-Hellman tuple (whatever the group under consideration), the discrete logarithm of one component is enough to decrypt, since the plaintext is the common exponent.

One can note that matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ are drawn independently, so the keys in $\mathbb{G}_1$ and $\mathbb{G}_2$ are independent. For any pair of keys $(\mathsf{pk}_1 = [\mathbf{p}_1]_1, \mathsf{pk}_2 = [\mathbf{p}_2]_2)$, one can implicitly define a public key for the target group. To decrypt in the target group, both private keys $\mathsf{sk}_1 = \mathbf{P}_1$ and $\mathsf{sk}_2 = \mathbf{P}_2$ are needed. Actually, one just needs $\mathbf{P}_1 \otimes \mathbf{P}_2$ to decrypt: $C'_T = ([\mathbf{c}_{T,1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2), [\mathbf{c}_{T,2}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2))$, but $\mathbf{P}_1 \otimes \mathbf{P}_2$ and $(\mathbf{P}_1, \mathbf{P}_2)$ contain the same information and the latter is more compact.

3.1.3 Homomorphic Properties

As BGN, Freeman cryptosystem also allows additions, one multiplication layer, and additions: we detail the homomorphic functions below.

Add(C_s, C'_s): Given two ciphertexts $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s), C' = ([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s)$ in one of $\mathbb{G}_1^2 \times \mathbb{G}_1^2, \mathbb{G}_2^2 \times \mathbb{G}_2^2, \mathbb{G}_T^4 \times \mathbb{G}_T^4$, if $[\mathbf{c}_{s,2}]_s = [\mathbf{c}'_{s,2}]_s$, it outputs $([\mathbf{c}_{s,1}]_s \bullet [\mathbf{c}'_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$, otherwise it outputs $\perp$.

Multiply(C_1, C_2): Given two ciphertexts $C_1 = ([\mathbf{c}_{1,1}]_1, [\mathbf{c}_{1,2}]_1) \in \mathbb{G}_1^2 \times \mathbb{G}_1^2$ and $C_2 = ([\mathbf{c}_{2,1}]_2, [\mathbf{c}_{2,2}]_2) \in \mathbb{G}_2^2 \times \mathbb{G}_2^2$, it outputs $C_T = ([\mathbf{c}_{1,1}]_1 \bullet [\mathbf{c}_{2,1}]_2, [\mathbf{c}_{1,2}]_1 \bullet [\mathbf{c}_{2,2}]_2) \in \mathbb{G}_T^4 \times \mathbb{G}_T^4$.

Randomize$_s(\mathsf{pk}_s, C_s)$: Given a ciphertext $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$, for $s \in \{1, 2\}$ and a public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$, it chooses $\alpha, r \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(\alpha \cdot ([\mathbf{c}_{s,1}]_s \bullet r \cdot [\mathbf{p}_s]_s), \alpha \cdot [\mathbf{c}_{s,2}]_s)$; while for $s = T$ and a public key $\mathsf{pk}_T = ([\mathbf{p}_1]_1, [\mathbf{p}_2]_2)$, it chooses $\alpha \xleftarrow{\$} \mathbb{Z}_p, [\mathbf{r}_1]_1 \xleftarrow{\$} \mathbb{G}_1^2$ and $[\mathbf{r}_2]_2 \xleftarrow{\$} \mathbb{G}_2^2$, and outputs $(\alpha \cdot ([\mathbf{c}_{T,1}]_T \bullet [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 \bullet [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2), \alpha \cdot [\mathbf{c}_{T,2}]_T)$.

Instead of performing a systematic randomization of ciphertexts as proposed by Freeman each time an Add or a Multiply is computed, we create a specific function Randomize usable at any time, when more privacy is required.

Let us check the correctness of the three homomorphic functions:

Proposition 15. Add *and* Multiply *are correct.*

Proof. Let us first consider the addition operations:

- For $s = 1, 2$:

$$\begin{aligned}
&\mathsf{Add}(\mathsf{Encrypt}_s(\mathsf{pk}_s, m, [\mathbf{a}]_s; r), \mathsf{Encrypt}_s(\mathsf{pk}_s, m', [\mathbf{a}]_s; r')) \\
&= ([m\mathbf{a} + r\mathbf{p}_s]_s \cdot [m'\mathbf{a} + r'\mathbf{p}_s]_s, [\mathbf{a}]_s) = ([(m+m')\mathbf{a} + (r+r')\mathbf{p}_s]_s, [\mathbf{a}]_s) \\
&= \mathsf{Encrypt}_s(\mathsf{pk}_s, m+m', [\mathbf{a}]_s; r+r')
\end{aligned}$$

- For $s = T$:

$$\begin{aligned}
&\mathsf{Add}(\mathsf{Encrypt}_T(\mathsf{pk}_T, m, ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); \mathbf{r}_1, \mathbf{r}_2), \\
&\qquad \mathsf{Encrypt}_T(\mathsf{pk}_T, m', ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); \mathbf{r}'_1, \mathbf{r}'_2)) \\
&= ([m([\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) + \mathbf{r}_1 \otimes \mathbf{p}_2 + \mathbf{p}_1 \otimes \mathbf{r}_2]_T \cdot \\
&\qquad [m'([\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) + \mathbf{r}'_1 \otimes \mathbf{p}_2 + \mathbf{p}_1 \otimes \mathbf{r}'_2]_T, [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) \\
&= ([(m+m')([\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) + (\mathbf{r}_1 + \mathbf{r}'_1) \otimes \mathbf{p}_2 + \mathbf{p}_1 \otimes (\mathbf{r}_2 + \mathbf{r}'_2)]_T, [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) \\
&= \mathsf{Encrypt}_T(\mathsf{pk}_T, m+m', ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); \mathbf{r}_1 + \mathbf{r}'_1, \mathbf{r}_2 + \mathbf{r}'_2)
\end{aligned}$$

About multiplication, we can see that

$$\begin{aligned}
&\mathsf{Multiply}(\mathsf{Encrypt}_1(\mathsf{pk}_1, m_1, [\mathbf{a}_1]_1; r_1), \mathsf{Encrypt}_2(\mathsf{pk}_2, m_2, [\mathbf{a}_2]_2; r_2)) \\
&= ([m_1\mathbf{a}_1 + r_1\mathbf{p}_1]_1 \cdot [m_2\mathbf{a}_2 + r_2\mathbf{p}_2]_2, [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) \\
&= ([(m_1\mathbf{a}_1 + r_1\mathbf{p}_1) \otimes (m_2\mathbf{a}_2 + r_2\mathbf{p}_2)]_T, [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) \\
&= ([m_1\mathbf{a}_1 \otimes m_2\mathbf{a}_2 + m_1\mathbf{a}_1 \otimes r_2\mathbf{p}_2 + r_1\mathbf{p}_1 \otimes m_2\mathbf{a}_2 + r_1\mathbf{p}_1 \otimes r_2\mathbf{p}_2]_T, [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) \\
&= ([m_1\mathbf{a}_1 \otimes m_2\mathbf{a}_2 + m_1\mathbf{a}_1 \otimes r_2\mathbf{p}_2 + r_1\mathbf{p}_1 \otimes (m_2\mathbf{a}_2 + r_2\mathbf{p}_2)]_T, [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) \\
&= ([m_1m_2(\mathbf{a}_1 \otimes \mathbf{a}_2) + \mathbf{p}_1 \otimes (r_1m_2\mathbf{a}_2 + r_1r_2\mathbf{p}_2) + (r_2m_1\mathbf{a}_1) \otimes \mathbf{p}_2]_T, [\mathbf{a}_1]_1 \bullet [\mathbf{a}_2]_2) \\
&= \mathsf{Encrypt}_T(\mathsf{pk}_T, m_1m_2, ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); m_1r_2\mathbf{a}_1, m_2r_1\mathbf{a}_2 + r_1r_2\mathbf{p}_2)
\end{aligned}$$

□

Proposition 16. *For $s \in \{1, 2, T\}$, $\mathsf{Randomize}_s$ is correct, with $\alpha = 1$.*

Proof. For $s \in \{1, 2\}$:

$$\begin{aligned}
&\mathsf{Randomize}_s(\mathsf{pk}_s, \mathsf{Encrypt}_s(\mathsf{pk}_s, m, [\mathbf{a}]_s; r), \alpha, r') \\
&= ([\alpha(m\mathbf{a} + r\mathbf{p}_s + r'\mathbf{p}_s)]_s, [\alpha\mathbf{a}]_s) = ([m(\alpha\mathbf{a}) + \alpha(r+r')\mathbf{p}_s]_s, [\alpha\mathbf{a}]_s) \\
&= \mathsf{Encrypt}_s(\mathsf{pk}_s, m, [\alpha\mathbf{a}]_s; \alpha(r+r'))
\end{aligned}$$

Since r' is uniformly distributed, the mask of the first component of the ciphertext is uniformly distributed, as in a fresh ciphertext, while with $\alpha = 1$, the basis in the second component remains unchanged. In addition, the random α also randomizes the basis $[\alpha\mathbf{a}]_s$, in the second component of the ciphertext, but computationally only, under the DDH assumption in $\mathbb{G}_s$.

For $s = T$:

$$\begin{aligned}
&\mathsf{Randomize}_T(\mathsf{pk}_T, \mathsf{Encrypt}_T(\mathsf{pk}_T, m, ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2); r), \alpha, \mathbf{r}'_1, \mathbf{r}'_2) \\
&= (\alpha \cdot (m \cdot [\mathbf{a}]_T \bullet [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 \bullet [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2 \bullet [\mathbf{p}_1]_1 \bullet [\mathbf{r}'_2]_2 \bullet [\mathbf{r}'_1]_1 \bullet [\mathbf{p}_2]_2, [\mathbf{a}]_T), \\
&([\alpha\mathbf{a}_1]_1, [\alpha\mathbf{a}_2]_2)) \\
&= (\alpha \cdot (m \cdot [\mathbf{a}]_T \bullet [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2 + \mathbf{r}'_2]_2 \bullet [\mathbf{r}_1 + \mathbf{r}'_1]_1 \bullet [\mathbf{p}_2]_2), ([\alpha\mathbf{a}_1]_1 \bullet [\alpha\mathbf{a}_2]_2)) \\
&= \mathsf{Encrypt}_T(\mathsf{pk}_T, m, ([\alpha\mathbf{a}_1]_1, [\alpha\mathbf{a}_2]_2); \alpha(\mathbf{r}_2 + \mathbf{r}'_1), \alpha(\mathbf{r}_2 + \mathbf{r}'_2))
\end{aligned}$$

Again, since $\mathbf{r}'_1$ and $\mathbf{r}'_2$ are uniformly distributed, the mask of the first component of the ciphertext is uniformly distributed, as in a fresh ciphertext. In addition, the random α randomizes the basis in the second component of the ciphertext, but computationally only, under the DDH assumption in both $\mathbb{G}_1$ and $\mathbb{G}_2$. □

3.1.4 Security Properties

Theorem 17. *For $s \in \{1,2\}$, $\mathcal{E}_s$ is* IND-CPA *under the* DDH *assumption in $\mathbb{G}_s$: for any adversary $\mathcal{A}$ running within time t, $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_s}(\mathcal{A}) \leq 2 \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$.*

Proof. We denote by $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_s}(\mathcal{A})$ the advantage of $\mathcal{A}$ against $\mathcal{E}_s$. We assume the running time of $\mathcal{A}$ is bounded by t.

Game $\mathbf{G}_0$: In this first game, the simulator plays the role of the challenger in the experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_s}(\mathcal{A})$, where $b = 0$:
$\mathcal{S}(\kappa)$:

- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\kappa)$
- $(\mathsf{sk}_s, \mathsf{pk}_s) \leftarrow \mathsf{EKeygen}_s(\mathsf{param})$
- $m_0, m_1, [\mathbf{a}]_s \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s)$
- $C_s = (m_0 \cdot [\mathbf{a}]_s \bullet r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \leftarrow \mathsf{Encrypt}_s(\mathsf{pk}_s, m_0, [\mathbf{a}]_s)$
- $b' \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s, C_s)$

We are interested in the event E: $b' = 1$. By definition,

$$\Pr_{\mathbf{G}_0}[E] = \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}}(\mathcal{A}) = 1\right].$$

Game $\mathbf{G}_1$: Now the simulator takes as input a Diffie-Hellman tuple $([\mathbf{p}]_s, [\mathbf{r}]_s)$, with $\mathbf{r} = r \cdot \mathbf{p}$ for some scalar r, and emulates $\mathsf{EKeygen}_s$ and $\mathsf{Encrypt}_s$ by defining $\mathsf{pk}_s \leftarrow [\mathbf{p}]_s$ and $C_s \leftarrow (m_0 \cdot [\mathbf{a}]_s \bullet [\mathbf{r}]_s, [\mathbf{a}]_s)$. Thanks to the Diffie-Hellman tuple, this game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_1}[E] = \Pr_{\mathbf{G}_0}[E]$.

Game $\mathbf{G}_2$: The simulator now receives a random tuple $([\mathbf{p}]_s, [\mathbf{r}]_s)$: $\Pr_{\mathbf{G}_2}[E] - \Pr_{\mathbf{G}_1}[E] \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$.

Game $\mathbf{G}_3$: The simulator still receives a random tuple $([\mathbf{p}]_s, [\mathbf{r}]_s)$, but generates $C_s \leftarrow (m_1 \cdot [\mathbf{a}]_s \bullet [\mathbf{r}]_s, [\mathbf{a}]_s)$. Thanks to the random mask $[\mathbf{r}]_s$, this game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_3}[E] = \Pr_{\mathbf{G}_2}[E]$.

Game $\mathbf{G}_4$: The simulator now receives a Diffie-Hellman tuple $([\mathbf{p}]_s, [\mathbf{r}]_s)$, with $\mathbf{r} = r \cdot \mathbf{p}$ for some scalar r: $\Pr_{\mathbf{G}_4}[E] - \Pr_{\mathbf{G}_3}[E] \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$.

Game $\mathbf{G}_5$: In this game, the simulator can perfectly emulate the challenger in the experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A})$, where $b = 1$: This game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_5}[E] = \Pr_{\mathbf{G}_4}[E]$.

One can note, that in this last game, $\Pr_{\mathbf{G}_5}[E] = \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right]$, hence

$$\Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right] \leq 2 \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t),$$

which concludes the proof. □

Corollary 18. *$\mathcal{E}_T$ is* IND-CPA *under the* DDH *assumptions in $\mathbb{G}_1$ or $\mathbb{G}_2$. More precisely, for any adversary $\mathcal{A}$ running within time t,*

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_T}(\mathcal{A}) \leq 2 \times \min\{\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t + t_m + t_e), \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + t_m + t_e)\},$$

where t_m is the time for one multiplication and t_e the time for one encryption.

Proof. The semantic security for ciphertexts in $\mathbb{G}_T$ comes from the fact that:

$$\begin{aligned}&\mathsf{Encrypt}_T(\mathsf{pk}_T, m, ([\mathbf{a}_1]_1, [\mathbf{a}_2]_2))\\ &= \mathsf{Multiply}(\mathsf{Encrypt}_1(\mathsf{pk}_1, m, [\mathbf{a}_1]_1), \mathsf{Encrypt}_2(\mathsf{pk}_2, 1, [\mathbf{a}_2]_2))\\ &= \mathsf{Multiply}(\mathsf{Encrypt}_1(\mathsf{pk}_1, 1, [\mathbf{a}_1]_1), \mathsf{Encrypt}_2(\mathsf{pk}_2, m, [\mathbf{a}_2]_2))\end{aligned}$$

Indeed, with this relation, each ciphertext in $\mathbb{G}_1$ can be transformed into a ciphertext in $\mathbb{G}_T$ (idem with a ciphertext in $\mathbb{G}_2$). Let $\mathcal{A}$ be an adversary against IND-CPA of $\mathcal{E}_T$, in $\mathbb{G}_T$.

Game $\mathbf{G}_0$: In the first game, the simulator plays the role of the challenger in the experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_T}(\mathcal{A})$, where $b = 0$:
$\mathcal{S}(\kappa)$:

- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\kappa)$
- $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{EKeygen}_1(\mathsf{param})$,$(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{EKeygen}_2(\mathsf{param})$
- $m_0, m_1, [\mathbf{a}]_1, [\mathbf{a}]_2 \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2))$
- $C_T = \mathsf{Encrypt}_T((\mathsf{pk}_1, \mathsf{pk}_2), m_0, ([\mathbf{a}]_1, [\mathbf{a}]_2))$
- $\beta \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2), C_T)$

We are interested in the event E: $b' = 1$. By definition,

$$\Pr_{\mathbf{G}_0}[E] = \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_T}(\mathcal{A}) = 1\right].$$

Game $\mathbf{G}_1$: The simulator interacts with a challenger in $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_1}(\mathcal{A})$, where $b = 0$. It thus first receives $\mathsf{param}, \mathsf{pk}_1$ from that challenger, generates pk_2 by himself to provide ($\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$) to the adversary. The latter sends back $(m_0, m_1, [\mathbf{a}]_1, [\mathbf{a}]_2)$, from which it sends $(m_0, m_1, [\mathbf{a}]_1)$ to the challenger. It gets back $C_1 = \mathsf{Encrypt}_1(\mathsf{pk}_1, m_0, [\mathbf{a}]_1)$. It can compute the ciphertext $C_T = \mathsf{Multiply}(C_1, \mathsf{Encrypt}_2(\mathsf{pk}_2, 1, [\mathbf{a}_2]_2))$, to be sent to the adversary. This game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_1}[E] = \Pr_{\mathbf{G}_0}[E]$.

Game $\mathbf{G}_2$: The simulator interacts with a challenger in $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_1}(\mathcal{A})$, where $b = 1$:

$$\Pr_{\mathbf{G}_2}[E] - \Pr_{\mathbf{G}_1}[E] \leq \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_1}(t + t_m + t_e),$$

where t_m is the time for one multiplication and t_e the time for one encryption.

Game $\mathbf{G}_3$: In this final game, the simulator plays the role of the challenger in the experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_T}(\mathcal{A})$, where $b = 1$. This game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_3}[E] = \Pr_{\mathbf{G}_2}[E]$.

One can note, that in this last game, $\Pr_{\mathbf{G}_3}[E] = \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_T}(\mathcal{A}) = 1\right]$, hence

$$\Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_T}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_T}(\mathcal{A}) = 1\right] \leq \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_T}(t + t_m + t_e),$$

which concludes the proof, since it works exactly the same way for $\mathbb{G}_2$. □

3.1.5 Re-Encryption

We have three efficient encryption schemes able to compute homomorphic operations and supporting multiple users. When a result should be available to a unique user, a classical technique called proxy re-encryption [BBS98] is to re-encrypt to this target user: this is a virtual decryption followed by encryption under the new key. With Freeman's approach, and our formalism, this is just a change of basis in the exponents: we can re-encrypt a message encrypted under a private key pk^a into another encryption for a private key pk^b by using a special secret key called re-encryption key $\mathsf{rk}^{a\to b}$.

Below we describe $\mathsf{REKeygen}_s$ that creates the re-encryption key from the secret keys and $\mathsf{REEncrypt}_s$ the function to re-encrypt a ciphertext, but under a different basis.

$\mathsf{REKeygen}_s(\mathsf{sk}_s^a, \mathsf{sk}_s^b)$**:** For $s = 1, 2$, from two different secret keys $\mathsf{sk}_s^a = \mathbf{P}_s$ and $\mathsf{sk}_s^b = \mathbf{P}'_s$ associated respectively to the two public keys pk_s^a and pk_s^b, compute $\mathbf{B}_s, \mathbf{B}'_s \in \mathrm{GL}_2(\mathbb{Z}_p)$ such that $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}\mathbf{B}_s$ and $\mathbf{P}'_s = {\mathbf{B}'_s}^{-1}\mathbf{U}\mathbf{B}'_s$ and output $\mathsf{rk}_s^{a\to b} = \mathbf{R}_s = \mathbf{B}_s^{-1}\mathbf{B}'_s$ the secret re-encryption key. From the re-encryption keys $\mathsf{rk}_1^{a\to b} = \mathbf{R}_1 \leftarrow \mathsf{REKeygen}_1(\mathsf{sk}_1^a, \mathsf{sk}_1^b)$ and $\mathsf{rk}_2^{a\to b} = \mathbf{R}_2 \leftarrow \mathsf{REKeygen}_2(\mathsf{sk}_2^a, \mathsf{sk}_2^b)$, we will consider $\mathsf{rk}_T^{a\to b} = (\mathsf{rk}_1^{a\to b}, \mathsf{rk}_2^{a\to b})$, as the matrix $\mathbf{R}_T$ actually is $\mathbf{R}_1 \otimes \mathbf{R}_2$.

$\mathsf{REEncrypt}_s(\mathsf{rk}_s^{a\to b}, C_s)$**:** To re-encrypt a ciphertext $C = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$:

- for $s = 1, 2$, output $([\mathbf{c}_{s,1}]_s \cdot \mathsf{rk}_s^{a\to b}, [\mathbf{c}_{s,2}]_s \cdot \mathsf{rk}_s^{a\to b})$;
- for $s = T$, output $([\mathbf{c}_{T,1}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}), [\mathbf{c}_{T,2}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}))$.

We stress that the basis $\mathbf{a}$ is modified with the re-encryption process, into $\mathbf{a}\mathbf{R}_s$ or $\mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2)$, which could leak some information about the re-encryption key. But as explained above, the randomization process can provide a new ciphertext that computationally hides it, under DDH assumptions. However, this requires this basis $\mathbf{a}$ to be part of the ciphertext as it cannot be a constant.

Correctness of Re-Encryption. The correctness of the re-encryption is based on a change of basis that transforms an element in the kernel of $\mathbf{P}_s$ in an element in the kernel of $\mathbf{P}'_s$: let $\mathbf{p} \in \ker(\mathbf{P}_s)$ and $\mathbf{p}' \in \ker(\mathbf{P}'_s)$ because $\ker(\mathbf{P}_s)$ and $\ker(\mathbf{P}'_s)$ are of dimension 1 in $\mathbb{Z}_p^2$, there exist $a, b, k \in \mathbb{Z}_p$, such that $\mathbf{p} = k \cdot (a, b)$ and $a', b', k' \in \mathbb{Z}_p$, such that $\mathbf{p}' = k' \cdot (a', b')$. We have:

$$\mathbf{p} \cdot \mathsf{rk} = \mathbf{p} \cdot \mathbf{B}^{-1}\mathbf{B}' = k(1,0)\mathbf{B}' = k(a', b') \Rightarrow \mathbf{p} \cdot \mathsf{rk} = kk'^{-1}\mathbf{p}' = r'\mathbf{p}'$$

for some $r' \in \mathbb{Z}_p$ and with that, the correctness follows, where $\mathsf{rk}_s^{a\to b}$ is denoted $\mathbf{R}_s$: for $s \in \{1, 2\}$,

$$\begin{aligned}
&\mathsf{REEncrypt}_s(\mathsf{rk}_s^{a\to b}, \mathsf{Encrypt}_s(\mathsf{pk}_s^a, m, \mathbf{a}, r)) = ([\mathbf{c}_{s,1}]_s \cdot \mathsf{rk}_s^{a\to b}, [\mathbf{c}_{s,2}]_s \cdot \mathsf{rk}_s^{a\to b}) \\
&= ([m\mathbf{a}\mathbf{R}_s + r\mathbf{p}_s\mathbf{R}_s]_s, [\mathbf{a}\mathbf{R}_s]_s) = ([m\mathbf{a}\mathbf{R}_s + rr'\mathbf{p}'_s]_s, [\mathbf{a}\mathbf{R}_s]_s) \\
&= \mathsf{Encrypt}_s(\mathsf{pk}_s^b, m, \mathbf{a}\mathbf{R}_s; rr')
\end{aligned}$$

For $s = T$,

$$\begin{aligned}
&\mathsf{REEncrypt}_T(\mathsf{rk}_T^{a\to b}, \mathsf{Encrypt}_T(\mathsf{pk}_T^a, m, \mathbf{a}; \mathbf{r}_1, \mathbf{r}_2)) \\
&= ([\mathbf{c}_{T,1}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b}), [\mathbf{c}_{T,2}]_T \cdot (\mathsf{rk}_1^{a\to b} \otimes \mathsf{rk}_2^{a\to b})) \\
&= ([(m\mathbf{a} + \mathbf{p}_1 \otimes \mathbf{r}_2 + \mathbf{r}_1 \otimes \mathbf{p}_2) \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T, [\mathbf{a} \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T) \\
&= ([m\mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2) + \mathbf{p}_1\mathbf{R}_1 \otimes \mathbf{r}_2\mathbf{R}_2 + \mathbf{r}_1\mathbf{R}_1 \otimes \mathbf{p}_2\mathbf{R}_2]_T, [\mathbf{a} \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T) \\
&= ([m\mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2) + \mathbf{r}'_1\mathbf{p}'_1 \otimes \mathbf{r}_2\mathbf{R}_2 + \mathbf{r}_1\mathbf{R}_1 \otimes \mathbf{r}'_2\mathbf{p}'_2]_T, [\mathbf{a} \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T) \\
&= ([m\mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2) + \mathbf{p}'_1 \otimes \mathbf{r}'_1\mathbf{r}_2\mathbf{R}_2 + \mathbf{r}'_2\mathbf{r}_1\mathbf{R}_1 \otimes \mathbf{p}'_2]_T, [\mathbf{a} \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)]_T) \\
&= \mathsf{Encrypt}_T(\mathsf{pk}_T^b, m, \mathbf{a}(\mathbf{R}_1 \otimes \mathbf{R}_2); \mathbf{r}'_2\mathbf{r}_1\mathbf{R}_1, \mathbf{r}'_1\mathbf{r}_2\mathbf{R}_2)
\end{aligned}$$

3.1.6 Verifiability

When a ciphertext is randomized or re-encrypted by a third party, one may want to be sure the content is kept unchanged. Verifiability is thus an important property we can efficiently achieve, with classical zero-knowledge proofs of discrete logarithm relations *à la* Schnorr. Such linear proofs of existence for k scalars that satisfy linear relations generally consist of a commitment c, a challenge $e \in \mathbb{Z}_p$ and the response $r \in \mathbb{Z}_p^k$ (see Preliminaries 2.4.3). The non-interactive variant just contains e and r, and thus $k+1$ scalars.

Example 19. Let $\mathbf{M} \in \mathcal{M}_2(\mathbb{Z}_p)$ and $([\mathbf{x}]_s, [\mathbf{y}]_s), ([\mathbf{x}']_s, [\mathbf{y}']_s) \in \mathbb{G}_s^2$. We will make the zero-knowledge proof of existence of $\mathbf{M}$ such that both $[\mathbf{y}]_s = [\mathbf{x}]_s \cdot \mathbf{M}$ and $[\mathbf{y}']_s = [\mathbf{x}']_s \cdot \mathbf{M}$, where $[\mathbf{x}]_s$, $[\mathbf{y}]_s$, $[\mathbf{x}']_s$ and $[\mathbf{y}']_s$ are public, but the prover knows $\mathbf{M}$. This is the classical zero-knowledge proof of equality of discrete logarithms with matrices.

The prover chooses $\mathbf{M}' \xleftarrow{\$} \mathcal{M}_2(\mathbb{Z}_p)$ and sends the commitments $[\mathbf{c}]_s = [\mathbf{x}]_s \cdot \mathbf{M}'$ and $[\mathbf{c}']_s = [\mathbf{x}']_s \cdot \mathbf{M}'$ to the verifier that answers a challenge $e \in \mathbb{Z}_p$. The prover constructs its response $\mathbf{R} = \mathbf{M}' - e\mathbf{M}$ in $\mathcal{M}_2(\mathbb{Z}_p)$ and the verifier checks whether both $[\mathbf{c}]_s = [\mathbf{x}]_s \cdot \mathbf{R} + e[\mathbf{y}]_s$ and $[\mathbf{c}']_s = [\mathbf{x}']_s \cdot \mathbf{R} + e[\mathbf{y}']_s$, in $\mathbb{G}_s^2$. To make the proof non-interactive, one can use the Fiat-Shamir heuristic with e generated by a hash function (modeled as a random oracle) on the statement $([\mathbf{x}]_s, [\mathbf{y}]_s), ([\mathbf{x}']_s, [\mathbf{y}']_s)$ and commitments $([\mathbf{c}]_s, [\mathbf{c}']_s)$. The proof eventually consists of $(e, \mathbf{R})$. From this proof, one can compute the candidates for $([\mathbf{c}]_s, [\mathbf{c}']_s)$, and check whether the hash value gives back e.

Before entering into the details of the relations to be proven, for each function of our encryption scheme, we rewrite the $\mathsf{EKeygen}_s$ and $\mathsf{REKeygen}_s$ algorithms to prepare the verifiability of $\mathsf{Decrypt}_s$ and $\mathsf{REEncrypt}_s$. These new $\mathsf{EKeygen}_s$ and $\mathsf{REKeygen}_s$ algorithms consist of the original $\mathsf{EKeygen}_s$ and $\mathsf{REKeygen}_s$ but with more elements in the output: they both output a public version of the produced secret key plus a zero-knowledge proof of the correctness of the keys. This significantly simplifies the relations to be proven afterwards for $\mathsf{Decrypt}_s$ and $\mathsf{REEncrypt}_s$. At the end of this section, we prove that adding those elements do not compromise the security of the encryption scheme.

$\mathsf{EKeygen}_s$ for Verifiability.

While the secret key is the projection $\mathbf{P}_s$, the verification key vsk_s consists of $[\mathbf{P}_s]_s$:

> $\mathsf{EKeygen}_s(\mathsf{param})$: For $s \in \{1,2\}$. It chooses $\mathbf{B}_s \xleftarrow{\$} \mathrm{GL}_2(\mathbb{Z}_p)$, lets $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}_2\mathbf{B}_s$ and $\mathbf{p}_s \in \ker(\mathbf{P}_s) \setminus \{\mathbf{0}\}$ and outputs the public key $\mathsf{pk}_s = [\mathbf{p}_s]_s$, the private key $\mathsf{sk}_s = \mathbf{P}_s$ and $\mathsf{vsk}_s = [\mathbf{P}_s]_s$ a verifiable public version of the secret key with the proof π_s:
>
> $$\{\exists \mathsf{sk}_s \in \mathcal{M}_2(\mathbb{Z}_p), \mathsf{vsk}_s \neq [\mathbf{0}]_s \wedge \mathsf{vsk}_s = [\mathbf{1}]_s \cdot \mathsf{sk}_s \wedge \mathsf{pk}_s \cdot \mathsf{sk}_s = [\mathbf{0}]_s\}.$$

The proof π_s guarantees that all the keys are well-formed: vsk_s is the exponentiation of a 2×2-matrix sk_s, for which the discrete logarithm of pk_s is in the kernel. Hence, sk_s is not full rank, and $\mathsf{vsk}_s \neq [\mathbf{0}]_s$ proves that sk_s is of dimension 1: a projection. As a consequence, π_s consists of 5 scalars of $\mathbb{Z}_p$, using the above non-interactive zero-knowledge technique *à la* Schnorr.

From $(\mathsf{vsk}_1, \mathsf{vsk}_2)$, we consider $\mathsf{vsk}_T = \mathsf{vsk}_1 \bullet \mathsf{vsk}_2$. It satisfies $\mathsf{vsk}_T = [\mathbf{P}_1 \otimes \mathbf{P}_2]_T$ if $(\mathsf{vsk}_1, \mathsf{vsk}_2) = ([\mathbf{P}_1]_1, [\mathbf{P}_2]_2)$.

$\mathsf{REKeygen}_s$ for Verifiability.

As above, while the secret re-encryption key is an invertible change of basis matrix $\mathsf{rk}_s^{a\to b}$, the verification key $\mathsf{vrk}_s^{a\to b}$ consists of $[\mathsf{rk}_s^{a\to b}]_s$. However, in order to prove the matrix $\mathsf{rk}_s^{a\to b}$ is

invertible, one can show it is non-zero, and not of rank 1, which would mean that $\mathsf{vrk}_s^{a\to b}$ would consist of a Diffie-Hellman tuple:

$\mathsf{REKeygen}_s(\mathsf{sk}_s^a, \mathsf{sk}_s^b)$**:** For $s = 1, 2$, from two different secret keys $\mathsf{sk}_s^a = \mathbf{P}_s$ and $\mathsf{sk}_s^b = \mathbf{P}'_s$ associated respectively to the two public keys pk_s^a and pk_s^b, it computes $\mathbf{B}_s, \mathbf{B}'_s \in \mathcal{M}_2(\mathbb{Z}_p)^2$ such that $\mathbf{P}_s = \mathbf{B}_s^{-1}\mathbf{U}\mathbf{B}_s$ and $\mathbf{P}'_s = {\mathbf{B}'_s}^{-1}\mathbf{U}\mathbf{B}'_s$. Let $\mathsf{rk}_s^{a\to b} = \mathbf{R}_s^{a\to b} = \mathbf{B}_s^{-1}\mathbf{B}'_s$ be the secret re-encryption key, $\mathsf{vrk}_s^{a\to b} = [\mathsf{rk}_s^{a\to b}]_s$ be a verifiable public version of the re-encryption key and $[r']_s = \lambda \cdot [r_{12}]_s$ where λ is such that $r_{21} = \lambda \cdot r_{11}$ (with $r_{11}, r_{12}, r_{21}, r_{22}$ the components of $\mathsf{rk}_s^{a\to b}$, and $\pi_s^{a\to b}$:

$$\{\exists \mathsf{rk}_s^{a\to b} \in \mathcal{M}_2(\mathbb{Z}_p), \exists \lambda \in \mathbb{Z}_p,$$
$$\mathsf{vrk}_s \neq [\mathbf{0}]_s \wedge \mathsf{vrk}_s^{a\to b} = [\mathbf{1}]_s \cdot \mathsf{rk}_s^{a\to b} \wedge \mathsf{pk}_s^b = \mathsf{pk}_s^a \cdot \mathsf{rk}_s^{a\to b}$$
$$\wedge [r']_s = \lambda \cdot [r_{12}]_s \wedge [r_{21}]_s = \lambda \cdot [r_{11}]_s \wedge [r']_s \neq [r_{22}]_s\}$$

It outputs $(\mathsf{rk}_s^{a\to b}, \mathsf{vrk}_s^{a\to b}, [r']_s, \pi_s^{a\to b})$.

The proof $\pi_s^{a\to b}$ guarantees that $\mathsf{vrk}_s^{a\to b}$ is well-formed and, since in $\mathcal{M}_2(\mathbb{Z}_p)$, the matrices are $\mathbf{0}$, or of rank 1 as a projection, or invertible: $\pi_s^{a\to b}$ first checks it is not $\mathbf{0}$, and then not of rank 1 either, as $\mathsf{vrk}_s^{a\to b}$ is not a Diffie-Hellman tuple.

The two checks $\mathsf{vrk}_s^{a\to b} \neq [\mathbf{0}]_s$ and $[r']_s \neq [r_{22}]_s$ are just simple verifications, thus $\pi_s^{a\to b}$ needs 6 scalars of $\mathbb{Z}_p$ as a proof *à la* Schnorr.

Similarly as for vsk_s, from $(\mathsf{vrk}_1, \mathsf{vrk}_2)$, we consider $\mathsf{vrk}_T = \mathsf{vrk}_1 \bullet \mathsf{vrk}_2$. So that, $\mathsf{vrk}_T = [\mathbf{R}_1 \otimes \mathbf{R}_2]_T$ if $(\mathsf{vrk}_1, \mathsf{vrk}_2) = ([\mathbf{R}_1]_1, [\mathbf{R}_2]_2)$.

Now, we explain for each function, the relations to be proven:

The function $\mathsf{Randomize}_s$**.**

It takes a ciphertext $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ encrypted with a public key pk_s and produces a ciphertext $C'_s = ([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s)$ such that:

- for $s \in \{1, 2\}$ and $\mathsf{pk}_s = [\mathbf{p}_s]_s$, it exists $\alpha, r \in \mathbb{Z}_p$ such that:

$$[\mathbf{c}'_{s,1}]_s = \alpha \cdot ([\mathbf{c}_{s,1}]_s + r \cdot [\mathbf{p}_s]_s) \wedge [\mathbf{c}'_{s,2}]_s = \alpha \cdot [\mathbf{c}_{s,2}]_s$$

- for $s = T$ and $\mathsf{pk}_T = ([\mathbf{p}_1]_1, [\mathbf{p}_2]_2)$, it exists $\alpha \in \mathbb{Z}_p, \mathbf{r}_1, \mathbf{r}_2 \in \mathbb{Z}_p^2$ such that:

$$[\mathbf{c}'_{T,1}]_T = \alpha \cdot ([\mathbf{c}_{T,1}]_T + [\mathbf{p}_1]_1 \bullet [\mathbf{r}_2]_2 + [\mathbf{r}_1]_1 \bullet [\mathbf{p}_2]_2) \wedge [\mathbf{c}'_{T,2}]_T = \alpha \cdot [\mathbf{c}_{T,2}]_T$$

These relations are equivalent to the linear relations:

- for $s \in \{1, 2\}$, it exists $\alpha, r \in \mathbb{Z}_p$ such that:

$$[\mathbf{c}'_{s,1}]_s = \alpha \cdot [\mathbf{c}_{s,1}]_s + r \cdot [\mathbf{p}_s]_s \wedge [\mathbf{c}'_{s,2}]_s = \alpha \cdot [\mathbf{c}_{s,2}]_s$$

- for $s = T$, it exists $\alpha \in \mathbb{Z}_p, \mathbf{r}_1, \mathbf{r}_2 \in \mathbb{Z}_p^2$ such that:

$$[\mathbf{c}'_{T,1}]_T = \alpha \cdot [\mathbf{c}_{T,1}]_T + [\mathbf{p}_1]_T \cdot \mathbf{r}_2 + \mathbf{r}_1 \cdot [\mathbf{p}_2]_T \wedge [\mathbf{c}'_{T,2}]_T = \alpha \cdot [\mathbf{c}_{T,2}]_T$$

These proofs consist of 3 scalars of $\mathbb{Z}_p$ for $s \in \{1, 2\}$, and 6 scalars of $\mathbb{Z}_p$ for $s = T$.

The functions Add **and** $\mathsf{Multiply}$**.**

They are public and deterministic thus everyone can check the operations.

The function $\mathsf{Decrypt}_s$.

It takes a ciphertext $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ encrypted with a public key pk_s and produces its decryption m such that:

- for $s \in \{1,2\}$ and $\mathsf{pk}_s = [\mathbf{p}_s]_s$, it exists $\mathsf{sk}_s = \mathbf{P}_s \in \mathcal{M}_2(\mathbb{Z}_p)$ such that:

$$[\mathbf{p}_s]_s \cdot \mathbf{P}_s = [\mathbf{0}]_s \wedge \mathbf{P}_s \neq \mathbf{0} \wedge [\mathbf{c}_{s,1}]_s \cdot \mathbf{P}_s = m \cdot [\mathbf{c}_{s,2}]_s \cdot \mathbf{P}_s$$

- for $s = T$ and $\mathsf{pk}_T = ([\mathbf{p}_1]_1, [\mathbf{p}_2]_2)$, it exists $\mathsf{sk}_T = (\mathbf{P}_1, \mathbf{P}_2) \in \mathcal{M}_2(\mathbb{Z}_p)^2$ such that:

$$\begin{aligned}&[\mathbf{p}_1]_1 \cdot \mathbf{P}_1 = [\mathbf{0}]_1 \wedge [\mathbf{p}_2]_2 \cdot \mathbf{P}_2 = [\mathbf{0}]_2 \wedge \mathbf{P}_1 \neq \mathbf{0} \wedge \mathbf{P}_2 \neq \mathbf{0}\\ &\wedge [\mathbf{c}_{T,1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = m \cdot [\mathbf{c}_{T,2}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2)\end{aligned}$$

Instead of proving these relations, the prover will use vsk_s for $s \in \{1,2,T\}$ produced by $\mathsf{EKeygen}_s$ for verifiability and will make the proof of the relations:

- for $s \in \{1,2\}$, it exists $\mathsf{sk}_s = \mathbf{P}_s \in \mathcal{M}_2(\mathbb{Z}_p)$ such that:

$$[\mathsf{vsk}_s]_s = [\mathbf{1}]_s \cdot \mathbf{P}_s \wedge ([\mathbf{c}_{s,1}]_s - m \cdot [\mathbf{c}_{s,2}]_s) \cdot \mathbf{P}_s = [\mathbf{0}]_s$$

- for $s = T$, it exists $\mathsf{sk}_T = (\mathbf{P}_1, \mathbf{P}_2) \in \mathcal{M}_2(\mathbb{Z}_p)^2$ such that:

$$[\mathsf{vsk}_T]_T = [\mathbf{1}]_T \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) \wedge ([\mathbf{c}_{T,1}]_T - m \cdot [\mathbf{c}_{T,2}]_T) \cdot (\mathbf{P}_1 \otimes \mathbf{P}_2) = [\mathbf{0}]_T$$

The linear proofs consist of 5 scalars of $\mathbb{Z}_p$ for $s \in \{1,2\}$ and 17 scalars of $\mathbb{Z}_p$ for $s = T$.

The function $\mathsf{REEncrypt}_s$.

It takes a ciphertext $C_s = ([\mathbf{c}_{s,1}]_s, [\mathbf{c}_{s,2}]_s)$ encrypted with a public key pk_s^a and produces a ciphertext $C'_s = ([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s)$ encrypted with a public key pk_s^b such that:

- for $s \in \{1,2\}$, it knows $\mathsf{rk}_s^{a\to b} = \mathbf{R}_s \in \mathrm{GL}_2(\mathbb{Z}_p)$ such that:

$$([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s) = ([\mathbf{c}_{s,1}]_s \cdot \mathbf{R}_s, [\mathbf{c}_{s,2}]_s \cdot \mathbf{R}_s) \wedge \mathsf{pk}_s^b = \mathsf{pk}_s^a \cdot \mathbf{R}_s$$

- for $s = T$, $\mathsf{pk}_T^a = (\mathsf{pk}_1^a, \mathsf{pk}_2^a)$, $\mathsf{pk}_T^b = (\mathsf{pk}_1^b, \mathsf{pk}_2^b)$ and $\mathsf{vrk}_T = ([\mathbf{R}_1]_1 \bullet [\mathbf{R}_2]_2)$, it knows $\mathsf{rk}_T^{a\to b} = (\mathbf{R}_1, \mathbf{R}_2) \in \mathrm{GL}_2(\mathbb{Z}_p)^2$ such that:

$$\begin{aligned}&([\mathbf{c}'_{T,1}]_T, [\mathbf{c}'_{T,2}]_T) = ([\mathbf{c}_{T,1}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2), [\mathbf{c}_{T,2}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2))\\ &\wedge \mathsf{pk}_T^b = (\mathsf{pk}_1^b, \mathsf{pk}_2^b) = (\mathsf{pk}_1^a \cdot \mathbf{R}_1, \mathsf{pk}_2^a \cdot \mathbf{R}_2)\end{aligned}$$

Instead of proving these relations, the prover will use vrk_s for $s \in \{1,2,T\}$ produced by $\mathsf{REKeygen}_s$ for verifiability and will make the proof of the relations below:

- for $s \in \{1,2\}$, it knows $\mathsf{rk}_s^{a\to b} = \mathbf{R}_s \in \mathcal{M}_2(\mathbb{Z}_p)$ such that:

$$([\mathbf{c}'_{s,1}]_s, [\mathbf{c}'_{s,2}]_s) = ([\mathbf{c}_{s,1}]_s \cdot \mathbf{R}_s, [\mathbf{c}_{s,2}]_s \cdot \mathbf{R}_s) \wedge \mathsf{vrk}_s^{a\to b} = [\mathbf{1}]_s \cdot \mathbf{R}_s$$

- for $s = T$, it knows $\mathsf{rk}_T^{a\to b} = (\mathbf{R}_1 \otimes \mathbf{R}_2) \in \mathcal{M}_4(\mathbb{Z}_p)$ such that:

$$\begin{aligned}&([\mathbf{c}'_{T,1}]_T, [\mathbf{c}'_{T,2}]_T) = ([\mathbf{c}_{T,1}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2), [\mathbf{c}_{T,2}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2))\\ &\wedge \mathsf{vrk}_T^{a\to b} = [\mathbf{1}]_T \cdot (\mathbf{R}_1 \otimes \mathbf{R}_2)\end{aligned}$$

This proof needs 5 scalars of $\mathbb{Z}_p$ for $s \in \{1,2\}$ and 17 scalars of $\mathbb{Z}_p$ for $s = T$.

Proposition 20. *For $s \in \{1,2\}$, $\mathcal{E}_s$ with verifiability is still secure. More precisely, for any adversary $\mathcal{A}$ running within time t,*

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_s}(\mathcal{A}) \leq 4 \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t).$$

Proof. The modified $\mathsf{EKeygen}_s$ also outputs vsk_s and a zero-knowledge proof π_s. This implies that some games need to be added before the first game in the security proof of $\mathcal{E}_s$ for Theorem 17:

Game $\mathbf{G}_0$: In the first game, the simulator plays the role of the challenger in the experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_s}(\mathcal{A})$, where $b = 0$:
$\mathcal{S}(\kappa)$:

- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\kappa)$
- $(\mathsf{sk}_s, \mathsf{pk}_s, \mathsf{vsk}_s, \pi_s) \leftarrow \mathsf{EKeygen}_s(\mathsf{param})$
- $m_0, m_1, [\mathbf{a}]_s \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s)$
- $C_s = (m_0 \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \leftarrow \mathsf{Encrypt}_s(\mathsf{pk}_s, m_0, [\mathbf{a}]_s)$
- $b' \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s, C_s)$

We are interested in the event E: $b' = 1$. By definition,

$$\Pr_{\mathbf{G}_0}[E] = \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right].$$

Game $\mathbf{G}_1$: The first modification is to replace π_s by its simulation, possible thanks to the zero-knowledge property. This game is statistically indistinguishable from the previous one, under the statistical zero-knowledge property of the proof *à la* Schnorr in the Random Oracle Model.

Game $\mathbf{G}_2$: Now the simulator takes as input a Diffie-Hellman tuple $([\mathbf{a}]_s, [\mathbf{b}]_s)$, with $\mathbf{b} = r \cdot \mathbf{a}$ for some scalar r, and emulates $\mathsf{EKeygen}_s$ by defining vsk_s the matrix defined by the two vectors $([\mathbf{a}]_s, [\mathbf{b}]_s)$. Thanks to the Diffie-Hellman tuple this corresponds to the matrix of a projection, and thus this game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_2}[E] = \Pr_{\mathbf{G}_1}[E]$.

Game $\mathbf{G}_3$: The simulator now receives a random tuple $([\mathbf{a}]_s, [\mathbf{b}]_s)$: $\Pr_{\mathbf{G}_3}[E] - \Pr_{\mathbf{G}_2}[E] \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$. In this game, there is no information in vsk_s anymore and the zero-knowledge proofs are simulated. In the original proof, sk_s is never used, thus we can plug the games from the original proof here. To finish the proof we need to unravel the games of vsk_s and π_s in order to have:

Game $\mathbf{G}_4$: $\mathcal{S}(\kappa)$:

- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\kappa)$
- $(\mathsf{sk}_s, \mathsf{pk}_s, \mathsf{vsk}_s, \pi_s) \leftarrow \mathsf{EKeygen}_s(\mathsf{param})$
- $m_0, m_1, [\mathbf{a}]_s \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s)$
- $C_s = (m_1 \cdot [\mathbf{a}]_s + r \cdot [\mathbf{p}_s]_s, [\mathbf{a}]_s) \leftarrow \mathsf{Encrypt}_s(\mathsf{pk}_s, m_0, [\mathbf{a}]_s)$
- $b' \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{pk}_s, C_s)$

the experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A})$.

Hence, we have:

$$\Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_s}(\mathcal{A}) = 1\right] \leq 4 \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t).$$

□

Corollary 21. *$\mathcal{E}_T$ with verifiability is still secure.*

Proof. Similarly to the previous proof, the zero-knowledge proofs are replaced by their simulations. Then, vsk_1 and vsk_2 are replaced by random matrices in $\mathcal{M}_2(\mathbb{G}_1)$ and $\mathcal{M}_2(\mathbb{G}_2)$ respectively. Thus, vsk_T is also a random matrix. □

3.1.7 Distributed Decryption

When a third-party performs the decryption, it is important to be able to prove the correct decryption, which consists of zero-knowledge proofs, as described in the previous Section 3.1.6. However, this is even better if the decryption process can be distributed among several servers, under the assumption that only a small fraction of them can be corrupted or under the control of an adversary.

To decrypt a ciphertext in $\mathbb{G}_s$ with $s \in \{1, 2\}$, one needs to compute $([\mathbf{c}_{s,1}]_s \cdot \mathsf{sk}_s, [\mathbf{c}_{s,2}]_s \cdot \mathsf{sk}_s)$. In a Shamir's like manner [Sha79], one can perform a t-out-of-n threshold secret sharing by distributing sk_s such that $\mathsf{sk}_s = \sum_{i \in I} \lambda_{I,i} \mathsf{sk}_{s,i}$ with $I \subset \{1, \ldots, n\}$ a subset of t users, and for all $i \in I$, $\lambda_{I,i} \in \mathbb{Z}_p$ and $\mathsf{sk}_{s,i}$ is the secret key of the party P_i.

For $s = T$ and with just the distribution of sk_1 and sk_2, it is also possible to perform a distributed decryption, using the relation $\mathsf{sk}_1 \otimes \mathsf{sk}_2 = (\mathsf{sk}_1 \otimes \mathbf{1}) \times (\mathbf{1} \otimes \mathsf{sk}_2)$. One can thus make a two round decryption, first in $\mathbb{G}_1$ and then in $\mathbb{G}_2$.

Remark. Because the operations to decrypt or re-encrypt are the same, one can make distributed re-encryption in the same vein: in our applications, computations will be performed on data encrypted under a *controller*'s key, where the *controller* is actually a pool of controllers with a distributed decryption key. The latter will be used to re-encrypt the result under the targer end-user's key.

However, in this scheme, the secret key must be a projection matrix, which is not easy to generate at random: for this key generation algorithm, a trusted dealer is required, which is not ideal when nobody is trusted. This is the goal of the rest of the chapter, to show that we can optimize this generic construction, and distribute everything without any trusted dealer.

3.2 Optimized Version

We presented the translation of Freeman's approach with projection matrices. This indeed leads to a public-key encryption scheme that can evaluate quadratic polynomials in $\mathbb{Z}_p$, under the DDH assumption. However, because the secret key must be a projection matrix, the distributed generation, while possible, is not as efficient as one can expect. We thus now propose a particular instantiation of projections, which allows very compact keys and ciphertexts.

3.2.1 Instantiation

While in the generic transformation of Freeman, the secret key belongs to the whole projection matrix space, our particular instantiation of projections means that the secret key will belong to a proper sub-space of the projection matrix space. In addition, this will allow to generate keys in a distributed manner, without any trusted dealer.

Indeed, it is possible to reduce by a factor two the size of the keys: for $s \in \{1, 2\}$, the secret key is just one scalar and the public key one group element in $\mathbb{G}_s$. For the keys, we will consider orthogonal projections on $\langle (1, x) \rangle$, for any $x \in \mathbb{Z}_p$. Thus, sk_s can simply be described by $x \in \mathbb{Z}_p$, which is enough to define the projection. The public key pk_s can simply be described by $g_s^{-x} \in \mathbb{G}_s$, which is enough to define (g_s^{-x}, g_s), as $(-x, 1)$ is a vector in the kernel of the projection, to add noise that the secret key will be able to remove.

More precisely, we can describe our optimized encryption schemes, for $s \in \{1, 2, T\}$, as $\mathcal{E}_s : (\mathsf{Setup}, \mathsf{EKeygen}_s, \mathsf{Encrypt}_s, \mathsf{Decrypt}_s)$ with a common Setup (as the index s indicates the group, in this section, elements of $\mathbb{G}_2$ will not be denoted in Fraktur font):

$\mathsf{Setup}(\kappa)$**:** Given a security parameter κ, it runs and outputs

$$\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathcal{G}(\kappa).$$

$\mathsf{EKeygen}_s(\mathsf{param})$**:** For $s \in \{1, 2\}$. It chooses $x_s \xleftarrow{\$} \mathbb{Z}_p$ and outputs the public key $\mathsf{pk}_s = g_s^{-x_s}$ and the private key $\mathsf{sk}_s = x_s$. From $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{EKeygen}_1(\mathsf{param})$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{EKeygen}_2(\mathsf{param})$, one can consider $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{sk}_T = (\mathsf{sk}_1, \mathsf{sk}_2)$, which are associated public and private keys in $\mathbb{G}_T$.

$\mathsf{Encrypt}_s(\mathsf{pk}_s, m)$**:** For $s \in \{1, 2\}$, to encrypt a message $m \in \mathbb{Z}_p$ using public key pk_s, it chooses $r \xleftarrow{\$} \mathbb{Z}_p$ and outputs the ciphertext

$$C_s = (c_{s,1} = g_s^m \cdot \mathsf{pk}_s^r, c_{s,2} = g_s^r) \in \mathbb{G}_s^2.$$

For $s = T$, to encrypt a message $m \in \mathbb{Z}_p$ using public key $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$, it chooses $r_{11}, r_{12}, r_{21}, r_{22} \xleftarrow{\$} \mathbb{Z}_p^4$ and outputs the ciphertext

$$C_T = \begin{pmatrix} c_{T,1} &= e(g_1, g_2)^m \cdot e(g_1, \mathsf{pk}_2)^{r_{11}} \cdot e(\mathsf{pk}_1, g_2)^{r_{21}}, \\ c_{T,2} &= e(g_1, g_2)^{r_{11}} \cdot e(\mathsf{pk}_1, g_2)^{r_{22}}, \\ c_{T,3} &= e(g_1, \mathsf{pk}_2)^{r_{12}} \cdot e(g_1, g_2)^{r_{21}}, \\ c_{T,4} &= e(g_1, g_2)^{r_{12}+r_{22}} \end{pmatrix} \in \mathbb{G}_T^4$$

$\mathsf{Decrypt}_s(\mathsf{sk}_s, C_s)$**:** For $s \in \{1, 2\}$, given $C_s = (c_{s,1}, c_{s,2})$ and the private key sk_s, it computes $d = c_{s,1} \cdot c_{s,2}^{\mathsf{sk}_s}$ and outputs the logarithm of d in basis g_s. For $s = T$, given $C_T = (c_{T,1}, c_{T,2}, c_{T,3}, c_{T,4})$ and $\mathsf{sk}_T = (\mathsf{sk}_1, \mathsf{sk}_2)$, it computes $d = c_{T,1} \cdot c_{T,2}^{\mathsf{sk}_2} \cdot c_{T,3}^{\mathsf{sk}_1} \cdot c_{T,4}^{\mathsf{sk}_1 \cdot \mathsf{sk}_2}$ and outputs the logarithm of d in basis $e(g_1, g_2)$.

In $\mathbb{G}_1$ and $\mathbb{G}_2$, this is actually the classical ElGamal encryption. We essentially extend it to $\mathbb{G}_T$, to handle quadratic operations:

$\mathsf{Add}(C_s, C'_s)$ just consists of the component-wise product in $\mathbb{G}_s$;

$\mathsf{Multiply}(C_1, C_2)$ for $C_1 = (c_{1,1} = g_1^{m_1} \cdot \mathsf{pk}_1^{r_1}, c_{1,2} = g_1^{r_1}) \in \mathbb{G}_1^2$ and $C_2 = (c_{2,1} = g_2^{m_2} \cdot \mathsf{pk}_2^{r_2}, c_{2,2} = g_2^{r_2}) \in \mathbb{G}_2^2$, consists of the tensor product:

$$C_T = (e(c_{1,1}, c_{2,1}), e(c_{1,1}, c_{2,2}), e(c_{1,2}, c_{2,1}), e(c_{1,2}, c_{2,2})) \in \mathbb{G}_T^4$$

$\mathsf{Randomize}_s(\mathsf{pk}_s, C_s)$ is, as usual, the addition of a random ciphertext of 0 in the same group $\mathbb{G}_s$. For $s \in \{1, 2\}$: Given a ciphertext $C_s = (c_{s,1}, c_{s,2})$ with its public key pk_s, it chooses $r \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(c_{s,1} \cdot \mathsf{pk}_s^r, c_{s,2} \cdot g_s^r)$; while for $s = T$, a public key pk_T and a ciphertext $(c_{T,1}, c_{T,2}, c_{T,3}, c_{T,4})$, it chooses $r'_{11}, r'_{12}, r'_{21}, r'_{22} \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(c_{T,1} \cdot e(g_1, \mathsf{pk}_2)^{r'_{11}} \cdot e(\mathsf{pk}_1, g_2)^{r'_{21}}, c_{T,2} \cdot e(g_1, g_2)^{r'_{11}} \cdot e(\mathsf{pk}_1, g_2)^{r'_{22}}, c_{T,3} \cdot e(g_1, \mathsf{pk}_2)^{r'_{12}} \cdot e(g_1, g_2)^{r'_{21}}, c_{T,4} \cdot e(g_1, g_2)^{r'_{12}+r'_{22}})$.

3.2.2 Security Properties

Whereas the correctness directly comes from the correctness of Freeman's construction, presented in the Section 3.1, and verification is straightforward, the semantic security comes from the classical ElGamal encryption security, under the DDH assumptions, for the basic schemes in $\mathbb{G}_1$ and $\mathbb{G}_2$:

Theorem 22. *For $s \in \{1,2\}$, $\mathcal{E}_s$ is* IND-CPA *under the* DDH *assumption in $\mathbb{G}_s$: for any adversary $\mathcal{A}$ running within time t, $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_s}(\mathcal{A}) \leq 2 \times \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_s}(t)$.*

Corollary 23. *$\mathcal{E}_T$ is* IND-CPA *under the* DDH *assumptions in $\mathbb{G}_1$ or $\mathbb{G}_2$.*

Proof. The semantic security for ciphertexts in $\mathbb{G}_T$ comes from the fact that:

$$\begin{aligned}\mathsf{Encrypt}_T(\mathsf{pk}_T, m) &= \mathsf{Multiply}(\mathsf{Encrypt}_1(\mathsf{pk}_1, m), \mathsf{Encrypt}_2(\mathsf{pk}_2, 1))\\ &= \mathsf{Multiply}(\mathsf{Encrypt}_1(\mathsf{pk}_1, 1), \mathsf{Encrypt}_2(\mathsf{pk}_2, m))\end{aligned}$$

Indeed, with this relation, each ciphertext in $\mathbb{G}_1$ can be transformed into a ciphertext in $\mathbb{G}_T$ (idem with a ciphertext in $\mathbb{G}_2$). Let $\mathcal{A}$ be an adversary against IND-CPA of $\mathcal{E}_T$, in $\mathbb{G}_T$.

Game $\mathbf{G}_0$: In the first game, the simulator plays the role of the challenger in the experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_T}(\mathcal{A})$, where $b = 0$:

- $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e) \leftarrow \mathsf{Setup}(\kappa)$
- $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{EKeygen}_1(\mathsf{param})$,$(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{EKeygen}_2(\mathsf{param})$
- $m_0, m_1 \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2))$; $C_T = \mathsf{Encrypt}_T((\mathsf{pk}_1, \mathsf{pk}_2), m_0)$
- $\beta \leftarrow \mathcal{A}(\mathsf{param}, (\mathsf{pk}_1, \mathsf{pk}_2), C_T)$

We are interested in the event E: $b' = 1$. By definition,

$$\Pr_{\mathbf{G}_0}[E] = \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_T}(\mathcal{A}) = 1\right].$$

Game $\mathbf{G}_1$: The simulator interacts with a challenger in $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathcal{E}_1}(\mathcal{A})$, where $b = 0$. It thus first receives $\mathsf{param}, \mathsf{pk}_1$ from that challenger, generates pk_2 by himself to provide $(\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2))$ to the adversary. The latter sends back (m_0, m_1) the simulators forwards to the challenger. It gets back $C_1 = \mathsf{Encrypt}_1(\mathsf{pk}_1, m_0)$. It can compute $C_T = \mathsf{Multiply}(C_1, \mathsf{Encrypt}_2(\mathsf{pk}_2, 1))$, to be sent to the adversary. This game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_1}[E] = \Pr_{\mathbf{G}_0}[E]$.

Game $\mathbf{G}_2$: The simulator interacts with a challenger in $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_1}(\mathcal{A})$, where $b = 1$:

$$\Pr_{\mathbf{G}_2}[E] - \Pr_{\mathbf{G}_1}[E] \leq \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_1}(t + 4 \cdot t_p + 4 \cdot t_e),$$

where t_p is the time for one pairing and t_e the time for one exponentiation.

Game $\mathbf{G}_3$: In this final game, the simulator plays the role of the challenger in $\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_T}(\mathcal{A})$, where $b = 1$. This game is perfectly indistinguishable from the previous one: $\Pr_{\mathbf{G}_3}[E] = \Pr_{\mathbf{G}_2}[E]$.

One can note, that in this last game, $\Pr_{\mathbf{G}_3}[E] = \Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathcal{E}_T}(\mathcal{A}) = 1\right]$, hence

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_T}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathcal{E}_1}(t + 4 \cdot t_p + 4 \cdot t_e),$$

which concludes the proof, since it works exactly the same way for $\mathbb{G}_2$. □

We stress that the security of $\mathcal{E}_T$ only requires the DDH assumption in one of the two groups, and not the SXDH assumption (which means that the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$).

3.2.3 Decentralized Homomorphic Encryption

Our main motivation was a decentralized key generation and a distributed decryption in order to be able to compute on encrypted data so that nobody can decrypt intermediate values but the result can be provided in clear to a target user. We now show that our optimized construction allows both decentralized key generation without a trusted dealer and distributed decryption. They are both quite efficient. We also show this is possible to do proxy re-encryption in a distributed way, without any leakage of information.

Decentralized Key Generation

In fact, a classical decentralized t-out-of-n threshold secret sharing allows to generate the shares of a random element and it seems hard (if one expects efficiency) to use it to generate the shares of a structured matrix, such as projections required in the generic construction, because its elements are not independently random. In our specific construction, the secret keys in $\mathbb{G}_1$ and $\mathbb{G}_2$ are now one scalar and one can perform a classical t-out-of-n threshold secret sharing: each player i generates a random polynomial P_i of degree $t-1$ in $\mathbb{Z}_p[X]$, privately sends $x_{i,j} = P_i(j)$ to player j, and publishes $g_s^{-P_i(0)}$; each player i then aggregates the values into $\mathsf{sk}_i = \sum_j x_{j,i} = P(i)$, for $P = \sum_j P_j$, which leads to a share of $x = P(0)$, and the public key is the product of all the public values.

Distributed Decryption

In order to decrypt $C_s = (c_{s,1}, c_{s,2})$ in $\mathbb{G}_1$ or $\mathbb{G}_2$, each player in a sub-set of t players sends its contribution $c_{s,2}^{\mathsf{sk}_i}$, that can be multiplied with Lagrange coefficients as exponents to obtain the mask $c_{s,2}^{\mathsf{sk}} = \mathsf{pk}_s^{-r}$. To decrypt $C_T = (c_{T,1}, c_{T,2}, c_{T,3}, c_{T,4})$ in $\mathbb{G}_T$, one can first use the shares of sk_1 to compute $c_{T,3}^{\mathsf{sk}_1}$ and $c_{T,4}^{\mathsf{sk}_1}$, and then the shares of sk_2 to compute $c_{T,2}^{\mathsf{sk}_2}$ and $c_{T,4}^{\mathsf{sk}_1 \cdot \mathsf{sk}_2}$. Under the DDH assumptions in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, one can show that the intermediate values $c_{s,2}^{\mathsf{sk}_i}$, or $c_{T,3}^{\mathsf{sk}_1}$, $c_{T,4}^{\mathsf{sk}_1}$, $c_{T,2}^{\mathsf{sk}_2}$, and $c_{T,4}^{\mathsf{sk}_1 \cdot \mathsf{sk}_2}$ do not leak more than the decryption itself. Of course, classical verifiable secret sharing techniques can be used, for both the decentralized generation and the distributed decryption. This can allow, with simple Schnorr-like proofs of Diffie-Hellman tuples, universal verifiability.

Distributed Re-encryption

Besides a distributed decryption, when outsourcing some computations on private information, a distributed authority may want to re-encrypt the encrypted result to a specific user, so that the latter can get the result in clear, and nobody else. More precisely, we assume the input data were encrypted under the keys pk_1, pk_2, and $\mathsf{pk}_T = (\mathsf{pk}_1, \mathsf{pk}_2)$, which leads, after quadratic evaluations, to a resulting ciphertext under the key pk_T, for which the distributed authorities, knowing a t-out-of-n additive secret sharing $(\mathsf{sk}_{1,i}, \mathsf{sk}_{2,i})_i$ of $(\mathsf{sk}_1, \mathsf{sk}_2)$, will re-encrypt under $\mathsf{PK}_T = (\mathsf{PK}_1, \mathsf{PK}_2)$ for the target user. Of course, such a re-encryption can be performed using multi-party computation, but we will show an efficient way to do it.

We start with the re-encryption of $c_s = (c_{s,1} = g_s^m \cdot \mathsf{pk}_s^r, c_{s,2} = g_s^r)$: player i chooses $r_i' \xleftarrow{\$} \mathbb{Z}_p$, computes $\alpha_i = c_{s,2}^{\mathsf{sk}_{s,i}} \cdot \mathsf{PK}_s^{r_i'}$ and $\beta_i = g_s^{r_i'}$, and outputs (α_i, β_i). Then, anybody can compute, for the appropriate Lagrange coefficients λ_i's,

$$C_s = (C_{s,1} = c_{s,1} \times \prod \alpha_i^{\lambda_i} = g_s^m \mathsf{pk}_s^r g_s^{r \cdot \mathsf{sk}_s} \cdot \mathsf{PK}_s^{r'} = g_s^m \cdot \mathsf{PK}_s^{r'}, C_{s,2} = \prod \beta_i^{\lambda_i} = g_s^{r'})$$

with $r' = \sum \lambda_i r_i'$, where the sum is on the t members available.

For $s = T$, given a ciphertext $c_T = (c_{T,1}, c_{T,2}, c_{T,3}, c_{T,4})$, player i chooses $u_i \xleftarrow{\$} \mathbb{Z}_p$, and first computes and sends $\alpha_{3,i} = c_{T,4}^{\mathsf{sk}_{1,i}} \cdot e(g_1, g_2)^{-u_i}$. With a linear combination for the appropriate

Lagrange coefficients λ_i's, anybody can compute, $\alpha_3 = \prod \alpha_{3,i}^{\lambda_i} = c_{T,4}^{\mathsf{sk}_1} \cdot e(g_1, g_2)^{-u}$, with implicit $u = \sum \lambda_i u_i$. Then each player i chooses $r'_{11,i}, r'_{12,i}, r'_{21,i}, r'_{22,i}, v_i \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$\alpha_{1,i} = c_{T,2}^{\mathsf{sk}_{2,i}} \cdot e(\mathsf{PK}_1, g_2)^{r'_{21,i}} \qquad \beta_i = e(g_1, g_2)^{r'_{11,i}+u_i} \cdot e(\mathsf{PK}_1, g_2)^{r'_{22,i}}$$
$$\alpha_{2,i} = c_{T,3}^{\mathsf{sk}_{1,i}} \cdot e(g_1, \mathsf{PK}_2)^{r'_{11,i}} \qquad \gamma_i = e(g_1, \mathsf{PK}_2)^{r'_{12,i}} \cdot e(g_1, g_2)^{r'_{21,i}+v_i}$$
$$\alpha_{4,i} = \alpha_3^{\mathsf{sk}_{2,i}} \cdot e(\mathsf{PK}_1, g_2)^{v_i} \qquad \delta_i = e(g_1, g_2)^{r'_{12,i}+r'_{22,i}}$$

Again, with linear combinations for the appropriate Lagrange coefficients λ_i's, anybody can compute, with $r'_{jk} = \sum \lambda_i r'_{jk,i}$, for $j, k \in \{1, 2\}$, and $v = \sum \lambda_i v_i$:

$$\alpha_1 = c_{T,2}^{\mathsf{sk}_2} \cdot e(\mathsf{PK}_1, g_2)^{r'_{21}} \qquad C_{T,2} = e(g_1, g_2)^{r'_{11}+u} \cdot e(\mathsf{PK}_1, g_2)^{r'_{22}}$$
$$\alpha_2 = c_{T,3}^{\mathsf{sk}_1} \cdot e(g_1, \mathsf{PK}_2)^{r'_{11}} \qquad C_{T,3} = e(g_1, \mathsf{PK}_2)^{r'_{12}} \cdot e(g_1, g_2)^{r'_{21}+v}$$
$$\alpha_4 = c_{T,4}^{\mathsf{sk}_1\mathsf{sk}_2} \cdot e(g_1, \mathsf{PK}_2)^{u} \cdot e(\mathsf{PK}_1, g_2)^{v} \qquad C_{T,4} = e(g_1, g_2)^{r'_{12}+r'_{22}}$$

Then, $C_{T,1} = c_{T,1} \times \alpha_1\alpha_2\alpha_4 = e(g_1, g_2)^m \cdot e(g_1, \mathsf{PK}_2)^{r'_{11}+u} \cdot e(\mathsf{PK}_1, g_2)^{r'_{21}+v}$, so that the ciphertext $C_T = (C_{T,1}, C_{T,2}, C_{T,3}, C_{T,4})$ is a re-encryption of c_T under PK_T.

For random scalars, the re-encryption algorithms (which is just a one-round protocol in $\mathbb{G}_1$ and $\mathbb{G}^2$, but 2-round in $\mathbb{G}_T$) generate new ciphertexts under appropriate keys that look perfectly fresh. In addition, one can claim:

Theorem 24. *The above distributed protocols for re-encryption do not leak additional information than the outputs of the non-distributed algorithms.*

Proof. The goal of this proof is to show that the distributed protocol to re-encrypt a ciphertext under PK_s does not leak more information than a direct encryption under PK_s. For $s \in \{1, 2\}$, one is given $c_s = \mathsf{Encrypt}_s(m, \mathsf{pk}_s; r) = (c_{s,1}, c_{s,2})$ and $C_s = \mathsf{Encrypt}_s(m, \mathsf{PK}_s; R) = (C_{s,1}, C_{s,2})$, two ciphertexts of the same message m under pk_s and PK_s respectively. One can then note that $C_{s,1}/c_{s,1} = \mathsf{PK}_s^R/\mathsf{pk}_s^r = c_{s,2}^{\mathsf{sk}_s}/C_{s,2}^{\mathsf{SK}_s}$.

The Re-Encryption in $\mathbb{G}_s$, for $s \in \{1, 2\}$.

Game $\mathbf{G}_0$: In the first game, the simulator just receives $c_s = (c_{s,1}, c_{s,2})$, and plays the real protocol using the t-out-of-n distributed keys $(\mathsf{sk}_{s,i})_i$ to provide the keys to the corrupted users and to generate the values $\alpha_i = c_{s,2}^{\mathsf{sk}_{s,i}} \cdot \mathsf{PK}_s^{r'_i}$ and $\beta_i = g_s^{r'_i}$, on behalf of the non-corrupted players. We assume that among t players, ℓ are honest and $t - \ell$ are corrupted. The latter are assumed to receive the secret keys $\mathsf{sk}_{s,i}$ and to generate their own outputs (α_i, β_i). The view of the attacker consists of the set of all the honest (α_i, β_i).

Game $\mathbf{G}_1$: The simulator is now given $c_s = (c_{s,1}, c_{s,2})$ and $C_s = (C_{s,1}, C_{s,2})$ that encrypt the same message. We want, for the appropriate Lagrange coefficients λ_i

$$C_{s,1} = c_{s,1} \cdot \prod \alpha_i^{\lambda_i} \qquad C_{s,2} = \prod \beta_i^{\lambda_i}.$$

Hence, the simulator can take, for all the honest players except the last one, $r'_i \xleftarrow{\$} \mathbb{Z}_p$ to compute $\alpha_i = c_{s,2}^{\mathsf{sk}_{s,i}} \cdot \mathsf{PK}_s^{r'_i}$ and $\beta_i = g_s^{r'_i}$. For the last honest player, from all the honest-user shares and corrupted-user shares, one sets

$$\alpha_\ell = (C_{s,1}/c_{s,1} \cdot \prod_{i \neq \ell} \alpha_i^{-\lambda_i})^{1/\lambda_\ell} \quad \beta_\ell = (C_{s,2} \cdot \prod_{i \neq \ell} \beta_i^{-\lambda_i})^{1/\lambda_\ell}.$$

Then, for the t players: $\prod \alpha_i^{\lambda_i} = c_{s,2}^{\mathsf{sk}_s} \cdot \mathsf{PK}_s^{r'}$ and $\prod \beta_i^{\lambda_i} = g_s^{r'}$, for $r' = \sum \lambda_i r'_i$ and with the implicit $r'_\ell = (R - \sum_{i \neq \ell} \lambda_i r'_i)/\lambda_\ell$. So $r' = R$. The view of the attacker remains exactly the same.

Game $\mathbf{G}_2$: In this game, the simulator also takes as input a Diffie-Hellman tuple $(A = g^r, B = \mathsf{PK}_s^r)$ with (g_s, PK_s): it first derives enough independent pairs $(A_i, B_i) = (g_s^{x_i} \cdot A^{y_i}, \mathsf{PK}_s^{x_i} \cdot B^{y_i})$, for random x_i, y_i, for all the non-corrupted players (excepted the last one), and computes $\alpha_i = c_{s,2}^{\mathsf{sk}_{s,i}} \cdot B_i, \beta_i = A_i$. Since $(g_s, \mathsf{PK}_s, A, B)$ is a Diffie-Hellman tuple, the view is perfectly indistinguishable from the previous one.

Game $\mathbf{G}_3$: In this game, the simulator now receives a random tuple (A, B), which makes all the (A_i, B_i) independent random pairs, the rest is unchanged: under the DDH assumption in $\mathbb{G}_s$, the view is computationally indistinguishable.

Game $\mathbf{G}_4$: This is the final simulation, where all the honest shares (α_i, β_i) are chosen at random, except the last ones $(\alpha_\ell, \beta_\ell)$ that are still computed as above to complete the values using c_s and C_s: the view is perfectly indistinguishable from the previous one and does not leak information.

As a consequence, we have proven that there is a simulator (defined in the last game) that produces a view indistinguishable from the real view, with just the input-output pairs. This proves that nothing else leaks. □

The Re-Encryption in $\mathbb{G}_T$.

The proof follows the same path as in the previous proof: one is given two ciphertexts $c_T = \mathsf{Encrypt}_T(m, (\mathsf{pk}_1, \mathsf{pk}_2); r_{11}, r_{12}, r_{21}, r_{22})$ and $C_T = \mathsf{Encrypt}_T(m, (\mathsf{PK}_1, \mathsf{PK}_2); R_{11}, R_{12}, R_{21}, R_{22})$ of the same message m under pk_T and PK_T respectively. One needs to simulate all the $\alpha_{1,i}, \alpha_{2,i}, \alpha_{3,i}, \alpha_{4,i}, \beta_i, \gamma_i, \delta_i$ for all the non-corrupted players. Since c_T and C_T encrypt the same message, and we want

$$C_{T,1} = c_{T,1} \cdot \prod \alpha_{1,i}^{\lambda_i} \cdot \alpha_{2,i}^{\lambda_i} \cdot \alpha_{4,i}^{\lambda_i} \qquad C_{T,2} = \prod \beta_i^{\lambda_i} \qquad C_{T,3} = \prod \gamma_i^{\lambda_i} \qquad C_{T,4} = \prod \delta_i^{\lambda_i}$$

the simulator can take, for all the honest players except the last one, $r'_{11,i}, r'_{12,i}, r'_{21,i}, r'_{22,i}, u_i, v_i \xleftarrow{\$} \mathbb{Z}_p$ to compute, in the first round:

$$\alpha_{3,i} = c_{T,4}^{\mathsf{sk}_{1,i}} \cdot e(g_1, g_2)^{-u_i} \qquad \alpha_{3,\ell} \xleftarrow{\$} \mathbb{G}_T \qquad \alpha_3 = \prod \alpha_{3,i}^{\lambda_i}$$

and in the second round, for all but the last honest player

$$\begin{aligned}
\alpha_{1,i} &= c_{T,2}^{\mathsf{sk}_{2,i}} \cdot e(\mathsf{PK}_1, g_2)^{r'_{21,i}} & \beta_i &= e(g_1, g_2)^{r'_{11,i}+u_i} \cdot e(\mathsf{PK}_1, g_2)^{r'_{22,i}} \\
\alpha_{2,i} &= c_{T,3}^{\mathsf{sk}_{1,i}} \cdot e(g_1, \mathsf{PK}_2)^{r'_{11,i}} & \gamma_i &= e(g_1, \mathsf{PK}_2)^{r'_{12,i}} \cdot e(g_1, g_2)^{r'_{21,i}+v_i} \\
\alpha_{4,i} &= \alpha_3^{\mathsf{sk}_{2,i}} \cdot e(\mathsf{PK}_1, g_2)^{v_i} & \delta_i &= e(g_1, g_2)^{r'_{12,i}+r'_{22,i}}
\end{aligned}$$

and for the last honest player:

$$\begin{aligned}
\alpha_{2,\ell} &\xleftarrow{\$} \mathbb{G}_T & \beta_\ell &= (C_{T,2} \times \prod_{i \neq \ell} \beta_i^{-\lambda_i})^{1/\lambda_\ell} \\
\alpha_{4,\ell} &\xleftarrow{\$} \mathbb{G}_T & \gamma_\ell &= (C_{T,3} \times \prod_{i \neq \ell} \gamma_i^{-\lambda_i})^{1/\lambda_\ell} \\
& & \delta_\ell &= (C_{T,4} \times \prod_{i \neq \ell} \delta_i^{-\lambda_i})^{1/\lambda_\ell}
\end{aligned}$$

which implies implicit values for $r'_{11,\ell}, r'_{12,\ell}, r'_{21,\ell}, r'_{22,\ell}, u_\ell, v_\ell$ because the above system is invertible, where X, Y, and Z are the constant values introduced by $c_{T,i}^{\mathsf{sk}_j}$, for some i, j:

$$\lambda_\ell \times \begin{pmatrix} \log \beta_\ell \\ \log \gamma_\ell \\ \log \delta_\ell \\ \log \alpha_{4,\ell} \\ \log \alpha_{3,\ell} \\ \log \alpha_{2,\ell} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & -\mathsf{sk}_1 & 1 & 0 \\ 0 & \mathsf{sk}_2 & 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathsf{sk}_{2,\ell} & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} r'_{11,\ell} \\ r'_{12,\ell} \\ r'_{21,\ell} \\ r'_{22,\ell} \\ u_\ell \\ v_\ell \end{pmatrix}$$

Then it is possible to set: $\alpha_{1,\ell} = (C_{T,1}/(c_{T,1} \cdot \alpha_2\alpha_4) \times \prod_{i \neq \ell} \alpha_{1,i}^{-\lambda_i})^{1/\lambda_\ell}$.

First, this is clear that the $\alpha_{3,i}$'s do not leak anything as they contain random masks $e(g_1, g_2)^{-u_i}$. Then, to prove that all the $\alpha_{1,i}, \alpha_{2,i}, \alpha_{4,i}, \beta_i, \gamma_i, \delta_i$ do not leak information, one can perform a similar proof as above for $\mathbb{G}_s$, by using the DDH assumption in both $\mathbb{G}_1$ and $\mathbb{G}_2$. Indeed, each element is masked using a pair either (g_2^r, PK_2^r) or (g_1^r, PK_1^r), for some random r. If one wants to have an indistinguishability under the SXDH assumption (and thus only one DDH assumption in one group), one could add more masks. But this does not make sense to have one key compromised and not the other one, for the same user. Hence, we tried to make the re-encryption as efficient as possible. □

We stress that for the re-encryption in $\mathbb{G}_1$ or $\mathbb{G}_2$, one just needs the DDH assumption in this group $\mathbb{G}_s$. But for the re-encryption in $\mathbb{G}_T$, one needs the DDH assumption in both $\mathbb{G}_1$ and $\mathbb{G}_2$ (the so-called SXDH assumption). We could rely on only one of the two, by adding masking factors, but this does not really make sense for a user to have his private key sk_1 being compromised without sk_2 (or the opposite).

In addition, zero-knowledge proofs can be provided to guarantee the re-encryption is honestly applied: they just consist in proofs of representations, when $g_s^{\mathsf{sk}_{s,i}}$ are all made public, for $s \in \{1, 2\}$ and all indices i.

3.2.4 Efficiency

In the concrete case where we have n servers able to perform a distributed protocol as described above, each of them has two scalars corresponding to a secret key for the encryption in $\mathbb{G}_1$ and a secret key for the encryption in $\mathbb{G}_2$. We recall that a ciphertext, in $\mathbb{G}_1$ or $\mathbb{G}_2$, is composed of two group elements, and a ciphertext in $\mathbb{G}_T$ is composed of four group elements. A recipient, that wants the result of either a decryption or a re-encryption with the help of t servers, has to perform a few exponentiations. The table below details the number of exponentiations for each player involved in the distributed protocols.

		per server	recipient
distributed decryption	in $\mathbb{G}_1/\mathbb{G}_2$	1	t
	in $\mathbb{G}_T$	4	$4t$
distributed re-encryption	in $\mathbb{G}_1/\mathbb{G}_2$	3	t
	in $\mathbb{G}_T$	13	$7t$

3.3 Applications

Boneh, Goh, and Nissim proposed two main applications to secure evaluation of quadratic polynomials: private information retrieval schemes (PIR) and electronic voting protocols. However, the use of our decentralized scheme for electronic voting is much more preferable than the BGN

scheme, as there is no way to trust any dealer in such a use-case. We propose two more applications that are related to the group testing and the consistency model in machine learning. Our applications are particularly useful in practice in a decentralized setting, as they deal with sensitive data. Interestingly, the use of distributed evaluation for quadratic polynomials in these applications is highly non-trivial and will be explained in the last section.

3.3.1 Encryption for Boolean Formulae

In this part, we detail the specific case of the evaluation of 2-DNF.

First, as explained in [BGN05], a way to guarantee the ciphertexts are encryption of inputs in $\{0,1\}$, the verification can be done with our scheme (or the one of BGN or Freeman) with the additional term $\mathsf{Add}_j(\mathsf{Multiply}(C_{x_j}, \mathsf{Add}(C_{x_j}, C_{-1}))$, multiplied by a random constant, so that it adds zero if inputs are correct, or it adds a random value otherwise. This introduces a quadratic term, just for the verification. This is at no extra cost if the Boolean formula is already quadratic, which will be the case of our applications.

Every Boolean formula can be expressed as a disjunction of conjunctive clauses (an OR of ANDs). This form is called disjunctive normal form (DNF) and, more precisely, k-DNF when each clause contains at most k literals. Thus, a 2-DNF formula over the variables $x_1, \ldots, x_n \in \{0,1\}$ is of the form

$$\bigvee_{i=1}^{m} (\ell_{i,1} \wedge \ell_{i,2}) \text{ with } \ell_{i,1}, \ell_{i,2} \in \{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}.$$

The conversion of 2-DNF formulae into multivariate polynomials of total degree 2 is simple: given $\Phi(x_1, \ldots, x_n) = \bigvee_{i=1}^{m}(\ell_{i,1} \wedge \ell_{i,2})$ a 2-DNF formula, define $\phi(x_1, \ldots, x_n) = \sum_{i=1}^{m}(y_{i,1} \times y_{i,2})$ where $y_{i,j} = \ell_{i,j}$ if $\ell_{i,j} \in \{x_1, \ldots, x_n\}$ or $y_{i,j} = (1 - \ell_{i,j})$ otherwise. In this conversion, a true literal is replaced by 1, and a false literal by 0. Then, an OR is converted into an addition, and an AND is converted into a multiplication. A NOT is just $(1 - x)$ when $x \in \{0,1\}$. $\phi(x_1, \ldots, x_n)$ is the multivariate polynomial of degree 2 corresponding to $\Phi(x_1, \ldots, x_n)$. As just said, this conversion works if for the inputs, we consider $1 \in \mathbb{Z}_p$ as true and $0 \in \mathbb{Z}_p$ as false, but for the output, $0 \in \mathbb{Z}_p$ is still considered as false whereas any other non-zero value is considered as true.

To evaluate the 2-DNF in an encrypted manner, we propose to encrypt the data and to calculate the quadratic polynomial corresponding to the 2-DNF as seen above by performing Adds and $\mathsf{Multiply}$s. Because the result of the 2-DNF is a Boolean, when a decryption is performed, if the result is equal to 0, one can consider it corresponds to the 0-bit (false) and else, it corresponds to the 1-bit (true).

Hence, when encrypting bits, we propose two different encodings before encryption, depending on the situation: either the 0-bit (false) is encoded by $0 \in \mathbb{Z}_p$ and the 1-bit (true) is encoded by any non-zero integer of $\mathbb{Z}_p^*$; or the 0-bit (false) is encoded by $0 \in \mathbb{Z}_p$ and the 1-bit (true) is encoded by $1 \in \mathbb{Z}_p$. With this second solution, it offers the possibility to perform one NOT on the data before Adds and $\mathsf{Multiply}$s by the operation $1 - x$. However, one has to be aware of making $\mathsf{Randomize}$ before decryption to mask the operations but also the input data in some situations: for example, if an Add is performed between three 1s, the result 3 leaks information and needs to be randomized.

Because one just wants to know whether the result is equal to 0 or the result is different from 0, we do not need to compute the logarithm: we can decrypt by just checking whether $c_{s,1} \cdot c_{s,2}^{\mathsf{sk}_s} = 1_s$ or not (for $s = T$, if $c_{T,1} \cdot c_{T,2}^{\mathsf{sk}_2} \cdot c_{T,3}^{\mathsf{sk}_1} \cdot c_{T,4}^{\mathsf{sk}_1 \cdot \mathsf{sk}_2} = 1_T$).

3.3.2 Group Testing on Encrypted Data

In this application we assume that a hospital collects some blood samples and wants to check which samples are positive or negative to a specific test. Group testing [Dor43] is an efficient

technique to detect positive samples with fewer tests in the case the proportion of positive cases is small. The technique consists in mixing some samples, and to perform tests on fewer mixes. More precisely, we denote $\mathbf{X} = (x_{ij})$ the matrix of the mixes: $x_{ij} = 1$ if the i-th sample is in the j-th mix, otherwise $x_{ij} = 0$. The hospital then sends the (blood) mixes to a laboratory for testing them: we denote y_j the result of the test on the j-th mix.

If a patient (its sample) is in a mix with a negative result, he is negative (not infected). If a patient (its sample) is in a mix with a positive result, we cannot say anything. However, for well-chosen parameters, if a patient is not declared negative, he is likely positive. Thus, for a patient i, the formula that we want to evaluate is $\neg\mathsf{F}_i(\mathbf{X}, \mathbf{y})$, which means the patient's test is positive (infected) or not, for $\mathsf{F}_i(\mathbf{X}, \mathbf{y}) = \bigvee_j (x_{ij} \wedge \neg y_j)$. The latter is indeed true if there is a mix containing a i-th sample for which the test is negative, and this should claim patient i negative (false). The matrix $\mathbf{X}$ of the samples needs to be encrypted since the patient does not want the laboratory to know his result. Because of the sensitiveness of the data, the result of the tests needs to be encrypted too. But the patient will need access to his own result.

In this scenario, the hospital computes for all i, j, $C_{x_{ij}} \in \mathbb{G}_1^2$, the encryption of x_{ij}, and the laboratory computes for all j, $C_{\overline{y_j}} \in \mathbb{G}_2^2$, the encryption of $\overline{y_j}$. Then, they both send the ciphertexts to an external database. With our homomorphic encryption scheme, to compute $\neg\mathsf{F}_i$, we can publicly evaluate the following formula: $C_i = \mathsf{Randomize}(\mathsf{Add}_j(\mathsf{Multiply}(C_{x_{ij}}, C_{\overline{y_j}}))$. Anybody can publicly verify the computations and if it is correct, a pool of controllers perform a distributed re-encryption of the result of patient i under his key PK_i. In this way, the patient cannot decrypt the database or the result of the tests directly, but only with the help of a pool of controller. The goal of the controllers is to limit access to the specific users only. Under an assumption about the collusions among the controllers, nobody excepted the users will have access to their own results.

3.3.3 Consistency Model on Encrypted Data

Another famous application is machine learning, where we have some trainers that fill a database and users who want to know a function of their inputs and the database. For privacy reasons, trainers do not want the users to learn the training set, and users do not want the trainers to learn their inputs. As in the previous case, we will involve a pool of distributed controllers to limit decryptions, but the controllers should not learn anything either.

Suppose a very large network of nodes in which some combinations should be avoided as they would result to failures. When a failure happens, the combination is stored in a database. And before applying a given combination, one can check whether it will likely lead to a failure, and then change. For example, the network can be a group of people where each of them can receive data. But, for some specific reasons, if a subgroup A of people is knowing a file a, the subgroup B must not have the knowledge of a file b. This case of application can be viewed as a consistency model [Sch14] which can be formally described as: the input is a vector of states (each being either true or false), and if in the database all the j-th states are true a new input needs to have its j-th state to be true; if all the j-th states in the database are false, the new input needs to have its j-th state to be false; otherwise the j-th state can be either true or false. As a consequence, if we denote the i-th element of the database as a vector $\mathbf{x}_i = (x_{ij})_j$ and the user's vector by $\mathbf{y} = (y_j)$, that vector $\mathbf{y}$ is said consistent with the database if the following predicate is true:

$$\bigwedge_j \Big((\wedge_i x_{ij} \wedge y_j) \vee (\wedge_i \overline{x_{ij}} \wedge \overline{y}_j) \vee (\vee_i x_{ij} \wedge \vee_i \overline{x_{ij}}) \Big) .$$

Let $X_j = \wedge_i x_{ij}$, $Y_j = \wedge_i \overline{x_{ij}}$, and $Z_j = \vee_i x_{ij} \wedge \vee_i \overline{x_{ij}}$. We define $\mathsf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{y})$ the formula we want to compute on the encrypted inputs:

$$\mathsf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{y}) = \bigwedge_j \Big((X_j \wedge y_j) \vee (Y_j \wedge \overline{y}_j) \vee Z_j \Big).$$

By definition, X_j, Y_j, and Z_j are exclusive, as X_j means the literals are all true, Y_j means the literals are all false, and Z_j means there are both true and false literals. So we have: $X_j \vee Z_j = \overline{Y_j}$ and $Y_j \vee Z_j = \overline{X_j}$. Thus, we have

$$\neg \mathsf{F}(\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{y}) = \bigvee_j \left((Y_j \vee \overline{y_j}) \wedge (X_j \vee y_j) \right).$$

Now, we see how the encryption and the decryption is performed to obtain the result of an evaluation. First, we explain how the trainers can update the database, when adding a vector $\mathbf{x}_m$. The values X_j are updated into X'_j as

$$X'_j = \bigwedge_{i=1}^{m} x_{ij} = \bigwedge_{i=1}^{m-1} x_{ij} \wedge x_{mj} = \begin{cases} X_j = \bigwedge_{i=1}^{m-1} x_{ij} & \text{if } x_{mj} = \mathsf{true} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

which is easy to compute for the trainer, since it knows $\mathbf{x}_m$ in clear, even if X_j is encrypted: the trainer can dynamically compute C_{X_j} the encryption of X_j, when adding a new line in the database, by just making a $\mathsf{Randomize}$ if x_{mj} is true (to keep the value X_j unchanged), or by replacing the value by a fresh encryption of 0 otherwise. Similarly, the trainer can update C_{Y_j}, the encryption of Y_j. On the user-side, he can compute C_{y_j} and $C_{\overline{y_j}}$ the encryptions of his inputs y_j and $\overline{y_j}$ respectively. Then, everyone and thus the controllers can compute:

$$C_j = \mathsf{Randomize}\left(\mathsf{Add}_j\left(\mathsf{Multiply}(\mathsf{Add}(C_{Y_j}, C_{\overline{y_j}}), \mathsf{Add}(C_{X_j}, C_{y_j}))\right)\right).$$

Because of the $\mathsf{Multiply}$, C_{Y_j} and $C_{\overline{y_j}}$ must be ciphertexts in $\mathbb{G}_1$, while C_{X_j} and C_{y_j} must be ciphertexts in $\mathbb{G}_2$. To allow a control of the final decryption, a pool of controllers re-encrypt for the user in a distributed way.

Chapter

4 Linearly-Homomorphic Signatures

This chapter introduces the building blocks of the two following ones: the Linearly-Homomorphic Signatures. For readability, it presents results coming from the two papers [HPP20, HP20].

Chapter content

The notion of homomorphic signatures dates back to [JMSW02], with notions in [ABC+12], but the linearly-homomorphic signatures, that allow to sign vector sub-spaces, were introduced in [BFKW09], with several follow-up by Boneh and Freeman [BF11b, BF11a] and formal security definitions in [Fre12].

We begin in a first section with the formal definition of linearly-homomorphic signature scheme, then, we will introduce a new property for linearly-homomorphic signature scheme: the tag randomizability. It will be the key element in our use-cases. Finally, we provide the security definition, so-called *unforgeability* in case of signatures. In fact in the Preliminaries we presented a weaker version of LH-Sign scheme without any tag called One-Time. The first construction proposed in Section 4.2 is a One-Time linearly-homomorphic signature while the two other constructions in Section 4.3 and in Section 4.4 are *full fledged* (not one-time). In the last section 4.5, we provide two generic conversions from a One-Time scheme to a full fledged one.

4.1 Definition, Properties and Security

Our definition of linearly-homomorphic signature scheme is inspired by the formal definition from Libert *et al.* [LPJY13], but with a possible private key associated to a tag:

Definition 25 — Linearly-Homomorphic Signature Scheme (LH-Sign)

A linearly-homomorphic signature scheme with messages in $\mathcal{M} \in \mathbb{G}^n$, for a cyclic group $(\mathbb{G}, \times)$ of prime order p, some $n \in \mathsf{poly}(\kappa)$, and some tag set $\mathcal{T}$, consists of the seven algorithms $(\mathsf{Setup}, \mathsf{SKeygen}, \mathsf{NewTag}, \mathsf{VerifTag}, \mathsf{Sign}, \mathsf{MultiplySign}, \mathsf{VerifSign})$:

$\mathsf{Setup}(1^\kappa)$**:** Given a security parameter κ, it outputs the global parameter param, which includes the tag space $\mathcal{T}$;

$\mathsf{SKeygen}(\mathsf{param}, n)$**:** Given a public parameter param and an integer n, it outputs a key

pair $(\mathsf{sk}, \mathsf{vk})$. We will assume that vk implicitly contains param and sk implicitly contains vk;

$\mathsf{NewTag}(\mathsf{sk})$**:** Given a signing key sk, it outputs a tag τ and its associated secret key $\tilde{\tau}$;

$\mathsf{VerifTag}(\mathsf{vk}, \tau)$**:** Given a verification key vk and a tag τ, it outputs 1 if the tag is valid and 0 otherwise;

$\mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M})$**:** Given a signing key, a secret key tag $\tilde{\tau}$ and a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}^n$, it outputs the signature σ under the tag τ;

$\mathsf{MultiplySign}(\mathsf{vk}, \tau, (\omega_i, \boldsymbol{M}_i, \sigma_i)_{i=1}^{\ell})$**:** Given a public key vk, a tag τ and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signed messages $\boldsymbol{M}_i$ in σ_i, it outputs a signature σ on the vector $\boldsymbol{M} = \prod_{i=1}^{\ell} \boldsymbol{M}_i^{\omega_i}$ under the tag τ;

$\mathsf{VerifSign}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$**:** Given a verification key vk, a tag τ, a vector-message $\boldsymbol{M}$ and a signature σ, it outputs 1 if $\mathsf{VerifTag}(\mathsf{vk}, \tau) = 1$ and σ is also valid relative to vk and τ, and 0 otherwise.

Note that we talk about *linearly homomorphic* signature with combinations component-wise in the exponents as we will consider a multiplicative group $(\mathbb{G}, \times)$ and messages directly in $\mathbb{G}$ instead of $\mathbb{Z}_p$. Moreover, this definition is more related to the notion of linearly homomorphic structure preserving signatures as the evaluated function is not provided in input of the verification algorithm unlike definitions as in [GVW15, CFN18] where the evaluated function matters.

The tag in $\mathsf{MultiplySign}$ allows linear combinations of signatures under the same tag but excludes any operation between signatures under different tags. The latter exclusion will be formalized by the unforgeability. However, the former property is the correctness: for any keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Keygen}(\mathsf{param}, n)$, for any tags $(\tau, \tilde{\tau}) \leftarrow \mathsf{NewTag}(\mathsf{sk})$, if for $i = 1, \ldots, \ell$, $\sigma_i = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M}_i)$ are valid signatures and $\sigma = \mathsf{MultiplySign}(\mathsf{vk}, \tau, \{\omega_i, \boldsymbol{M}_i, \sigma_i\}_{i=1}^{\ell})$ from some scalars ω_i, then both

$$\mathsf{VerifTag}(\mathsf{vk}, \tau) = 1 \qquad \mathsf{VerifSign}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma) = 1.$$

Our definition includes, but is more relaxed than, [LPJY13] as we allow a secret key associated to the tag, hence the NewTag algorithm: in such a case, the signer can only sign a message on a tag he generated himself. When there is no secret associated to the tag, one can actually consider that $\tilde{\tau} = \tau$ is enough to generate the signature (in addition to sk). Whereas the $\mathsf{MultiplySign}$ algorithm generates a signature under the same tag, we do not require to keep the same tag in the unforgeability notion below, this will allow our tag randomizability. However, we expect only signatures on linear combinations of messages already signed under a same tag, as we will formalize in the security notion.

Homomorphic Properties

From the definition of linearly homomorphic signature, we have the following property:

Property 26 (Message Homomorphism)**.** *Given several vector-messages with their signatures,* $\mathsf{MultiplySign}$ *generates the signature of any linear combination of the vector-messages.*

In addition, one can introduce a new algorithm $\mathsf{RandTag}$ associated to a new property for linearly-homomorphic signature schemes:

RandTag($\mathsf{vk}, \tau, \boldsymbol{M}, \sigma$): Given a verification key vk, a tag τ and a signature σ on a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}^n$, it outputs a new tag τ' and σ' a new signature on the new tag τ' of $\boldsymbol{M}$ still valid under the verification key vk.

Property 27 (Tag Randomizability)**.** *For any vector-message* $\boldsymbol{M} = (M_i)_i \in \mathbb{G}^n$*, key pair* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SKeygen}(\mathsf{param}, n)$*, tag* $(\tilde{\tau}, \tau) \leftarrow \mathsf{NewTag}(\mathsf{sk})$*, valid signature* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M})$*, and* $(\tau', \sigma') \leftarrow \mathsf{RandTag}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$ *the two following distributions are indistinguishable:*

$$\mathcal{D}_0 = \{(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)\} \qquad \mathcal{D}_1 = \{(\mathsf{vk}, \tau', \boldsymbol{M}, \sigma')\}.$$

Libert *et al.* [LPJY13] proposed a LH-Sign construction which security relies on the Simultaneous Double Pairing assumption, which is implied by the linear assumption in the symmetric case. In our use cases, the tag will be linked to the identity of a user. Hence, the tags need to be randomizable to provide privacy. However, we do not know how to build it in the standard model. Thus, we choose to focus on constructions secure in the generic bilinear group model [Sho97, BBG05, Boy08].

Notations and Constraints

Since we will mainly work on sub-vector spaces of dimension 2 (in a larger vector space), we will denote $\sigma = \mathsf{Sign}(\mathsf{sk}, (\boldsymbol{M}, \boldsymbol{M'}))$, with the verification check $\mathsf{VerifSign}(\mathsf{vk}, \sigma, (\boldsymbol{M}, \boldsymbol{M'})) = 1$, a signature that allows to derive a valid σ' for any linear combinations of $\boldsymbol{M}$ and $\boldsymbol{M'}$. In general, σ can be the concatenation of $\sigma_1 = \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M})$ and $\sigma_2 = \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M'})$, but some joint random coins may be needed, and some common elements can be merged (the tag).

We will also be interested in signing affine spaces: given a signature on $\boldsymbol{M}$ and $\boldsymbol{N}$, one wants to limit signatures on $\boldsymbol{M} \times \boldsymbol{N}^\alpha$ and $1 \times \boldsymbol{N}^\beta$. This is possible by expanding the messages with one more component: for $\overline{\boldsymbol{M}} = (g, \boldsymbol{M})$ and $\overline{\boldsymbol{N}} = (1, \boldsymbol{N})$, linear combinations are of the form $(g^\alpha, \boldsymbol{M}^\alpha \boldsymbol{N}^\beta)$. By imposing the first component to be g, one limits to $\alpha = 1$, and thus to $(g, \boldsymbol{M}\boldsymbol{N}^\beta) = \overline{\boldsymbol{M}} \times \overline{\boldsymbol{N}}^\beta$, while by imposing the first component to be 1, one limits to $\alpha = 0$, and thus to $(1, \boldsymbol{N}^\beta) = \overline{\boldsymbol{N}}^\beta$.

Unforgeability

Whereas linear combinations are possible under the same tag, other combinations (non-linear or under different tags) should not be possible. This is the unforgeability notion.

Definition 28 — Unforgeability for LH-Sign

For a LH-Sign scheme with messages in $\mathbb{G}^n$, for any adversary $\mathcal{A}$ that, given tags and signatures on messages $(\boldsymbol{M}_i)_i$ under tags $(\tau_i)_i$ both of its choice (for Chosen-Message Attacks), outputs a valid tuple $(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$ with $\tau \in \mathcal{T}$, there must exist $(\omega_i)_{i \in I_{\tau'}}$, where $I_{\tau'}$ is the set of messages already signed under some tag $\tau' \in \{\tau_i\}_i$, such that $\boldsymbol{M} = \prod_{i \in I_{\tau'}} \boldsymbol{M}_i^{\omega_i}$ with overwhelming probability.

Again, because our version is relaxed compared to [LPJY13], we do not exclude the adversary to be able to generate valid signatures under new tags. The linear-homomorphism for signatures, also known as signatures on vector-spaces, requires that the adversary cannot generate a valid signature on a message outside the vector spaces spanned by the already signed messages. Tags are just a way to keep together vectors that define vector spaces. The adversary can rename a vector space with another tag, this is not a security issue. On the opposite, we will exploit this feature for unlinkability with the additional randomizability property on tags.

4.2 Our One-Time LH-Sign Scheme

As in [LPJY13] and presented in the Example 12, we can consider a weaker notion of linearly-homomorphic signature: a one-time linearly-homomorphic signature (OT-LH-Sign), where the set of tags is a singleton $\mathcal{T} = \{\epsilon\}$. In this case, the algorithms NewTag and VerifTag can be dropped, as well as the τ and $\tilde{\tau}$.

We will consider a simplified variant of one-time linearly-homomorphic signature of Libert *et al.* [LPJY13] that can only be proven in the generic bilinear group model even if their scheme was originally built in the standard model.

Our One-Time linearly-homomorphic signature scheme with messages in $\mathcal{M} \in \mathbb{G}_1^n$, for some $n \in \mathsf{poly}(\kappa)$ consists of the five algorithms (Setup, SKeygen, Sign, MultiplySign, VerifSign):

Our One-Time LH-Sign Scheme

Setup(1^κ)**:** Given a security parameter κ, let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ be an asymmetric bilinear setting, where g and $\mathfrak{g}$ are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We set $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$;

SKeygen(param, n)**:** Given the public parameters param, one randomly chooses $\mathsf{sk}_i = s_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 1, \ldots, n$, which defines the signing key $\mathsf{sk} = (\mathsf{sk}_i)_{i=1}^n$, and the verification key $\mathsf{vk} = (\mathfrak{g}_i)_{i=0}^n$ for $\mathfrak{g}_i = \mathfrak{g}^{s_i}$ and $\mathfrak{g}_0 = \mathfrak{g}$;

Sign$(\mathsf{sk}, \boldsymbol{M} = (M_i)_i)$**:** Given a signing key $\mathsf{sk} = (s_i)_i$ and a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}_1^n$, one sets $\sigma = \prod_{i=1}^n M_i^{s_i} \in \mathbb{G}_1$;

MultiplySign$(\mathsf{vk}, (\omega_i, \boldsymbol{M}_i, \sigma_i)_{i=1}^\ell)$**:** Given a verification key and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signed messages $\boldsymbol{M}_i$ in σ_i, it outputs $\sigma = \prod \sigma_i^{\omega_i}$;

VerifSign$(\mathsf{vk}, \boldsymbol{M} = (M_i)_i, \sigma)$**:** Given a verification key vk, a vector-message $\boldsymbol{M}$, and a signature σ, one checks whether the equality $e(\sigma, \mathfrak{g}_0) = \prod_{i=1}^n e(M_i, \mathfrak{g}_i)$ holds or not.

If a message-signature is valid for a verification key vk, then it is also valid for the verification key $\mathsf{vk}' = \mathsf{vk}^\alpha$, for any α, as $e(\sigma, \mathfrak{g}_0) = \prod_{i=1}^n e(M_i, \mathfrak{g}_i)$ implies $e(\sigma, \mathfrak{g}_0^\alpha) = \prod_{i=1}^n e(M_i, \mathfrak{g}_i^\alpha)$.

However, for two different verification keys vk and vk′, and signatures σ and σ' of $\boldsymbol{M}$: $\prod_{i=1}^n e(M_i, \mathfrak{g}_i^\alpha \cdot \mathfrak{g}_i'^\beta) = \prod_{i=1}^n e(M_i, \mathfrak{g}_i)^\alpha \cdot e(M_i, \mathfrak{g}_i')^\beta = e(\sigma, \mathfrak{g}_0^\alpha) \cdot e(\sigma', \mathfrak{g}_0'^\beta)$, so $\sigma'' = \sigma^\alpha \sigma'^\beta$ is a valid signature of $\boldsymbol{M}$ under $\mathsf{vk}'' = \mathsf{vk}^\alpha \mathsf{vk}'^\beta$ if $\mathfrak{g}_0' = \mathfrak{g}_0$.

Hence, one can ask for a similar property on the keys than on the messages:

Property 29 (Key Homomorphism)**.** *Given a vector-message with signatures under several keys, it is possible to generate the signature of this vector-message under any linear combination of the keys.*

MultiplyKey$(\boldsymbol{M}, (\omega_i, \mathsf{vk}_i, \sigma_i)_{i=1}^\ell)$**:** Given a message $\boldsymbol{M}$ and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signatures σ_i of $\boldsymbol{M}$ under vk_i, it outputs a signature σ of $\boldsymbol{M}$ under the verification key $\mathsf{vk} = \prod_{i=1}^\ell \mathsf{vk}_i^{\omega_i}$.

Our scheme only supports the relaxed version:

Property 30 (Weak Key Homomorphism)**.** *Given a vector-message with signatures under several keys (with a specific restriction, as a common $\mathfrak{g}_0$ in our case), it is possible to generate the signature of this vector-message under any linear combination of the keys.*

Eventually, one needs to prove the unforgeability of our scheme:

Theorem 31 (Unforgeability)**.** *Let us consider an adversary $\mathcal{A}$ in the generic bilinear group model. Given valid pairs $(\boldsymbol{M}_j, \sigma_j)_j$ under a verification key vk ($\boldsymbol{M}_i$'s possibly of adversary's choice, for Chosen-Message Attacks), when $\mathcal{A}$ produces a new valid pair $(\boldsymbol{M}, \sigma)$ under the same verification key vk, there exist $(\alpha_j)_j$ such that $\boldsymbol{M} = \prod_j \boldsymbol{M}_j^{\alpha_j}$.*

Proof. The adversary $\mathcal{A}$ is given $(\boldsymbol{M}_j = (M_{j,i})_i, \sigma_j)_j$ which contains group elements in $\mathbb{G}_1$, as well as the verification key $\mathsf{vk} = (\mathfrak{g}_k)_k$ in $\mathbb{G}_2$. Note that in the generic bilinear group model, programmability of the encoding allows to simulate the signatures for chosen messages, which provides the security against Chosen-Message Attacks.

For any combination query, the simulator will consider the input elements as independent variables $X_{j,i}$, V_j, and $\mathfrak{S}_k$ to formally represent the discrete logarithms of $M_{j,i}$ and σ_i in basis g, and $\mathfrak{g}_k$ in basis $\mathfrak{g}_0 = \mathfrak{g}$. As usual, any new element can be seen as a multivariate polynomial in these variables, of degree maximal 2 (when there is a mix between $\mathbb{G}_1$ and $\mathbb{G}_2$ group elements). If two elements correspond to the same polynomial, they are definitely equal, and the simulator will provide the same representation. If two elements correspond to different polynomials, the simulator will provide random independent representations. The view of the adversary remains unchanged unless the actual instantiations would make the representations equal: they would be equal with probability at most $2/p$, when the variables are set to random values. After N combination queries, we have at most $N^2/2$ pairs of different polynomials that might lead to a collision for a random setting with probability less than N^2/p. Excluding such collisions, we can thus consider the polynomial representations only, denoted $\sim$. Then, for the output $(\boldsymbol{M} = (M_k)_k, \sigma)$, one knows $\alpha_{k,j,i}, \beta_{k,j}, \gamma_{i,j}, \delta_j$, such that:

$$M_k \sim \sum_{j,i} \alpha_{k,j,i} X_{j,i} + \sum_j \beta_{k,j} V_j \qquad\qquad \sigma \sim \sum_{j,i} \gamma_{j,i} X_{j,i} + \sum_j \delta_j V_j.$$

As $((M_{j,i})_i, \sigma_j)_j$ and $((M_k)_k, \sigma)$, are valid input and output pairs, we have the following relations between polynomials:

$$V_j = \sum_i X_{j,i} \mathfrak{S}_i \qquad \sum_{j,i} \gamma_{j,i} X_{j,i} + \sum_j \delta_j V_j = \sum_k \left(\sum_{j,i} \alpha_{k,j,i} X_{j,i} + \sum_j \beta_{k,j} V_j \right) \mathfrak{S}_k$$
$$= \sum_{k,j,i} \alpha_{k,j,i} X_{j,i} \mathfrak{S}_k + \sum_{k,j} \beta_{k,j} V_j \mathfrak{S}_k$$

Hence, the two polynomials are equal:

$$\sum_{j,i} \gamma_{j,i} X_{j,i} + \sum_{j,i} (\delta_j - \alpha_{i,j,i}) X_{j,i} \mathfrak{S}_i = \sum_{k \neq i,j,i} \alpha_{k,j,i} X_{j,i} \mathfrak{S}_k + \sum_{k,j} \beta_{k,j} V_j \mathfrak{S}_k$$

which leads, for all i, j, to $\gamma_{j,i} = 0$ and $\delta_j = \alpha_{i,j,i}$, and for $k \neq i$, $\alpha_{k,j,i} = 0$ and $\beta_{k,j} = 0$. Hence, $M_k \sim \sum_j \delta_j X_{j,k}$ and $\sigma \sim \sum_j \delta_j V_j$, which means that we have $(\delta_j)_j$ such that $M_k = \prod_j M_{j,k}^{\delta_j}$ and $\sigma = \prod_j \sigma_j^{\delta_j}$. □

4.3 FSH LH-Sign Scheme

In [LPJY13], the authors proposed a *full-fledged* LH-Sign by adding a public tag during the signature. In our constructions, tags will be related to identities of users, and so, some kind of randomizability will be required for anonymity, which is not possible with their scheme. Instead, we will consider the scheme proposed in [FHS19], which is a full-fledged LH-Sign version of our previous scheme. We can describe it as follows, using our notations:

FSH LH-Sign Scheme

Setup(1^κ): Given a security parameter κ, let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ be an asymmetric bilinear setting, where g and $\mathfrak{g}$ are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. The set of tags is $\mathcal{T} = \mathbb{G}_1 \times \mathbb{G}_2$. We then define $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e; \mathcal{T})$;

SKeygen(param, n): Given the public parameters param, one randomly chooses $\mathsf{sk}_i = s_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 1, \ldots, n$, which defines the signing key $\mathsf{sk} = (\mathsf{sk}_i)_i$, and the verification key $\mathsf{vk} = (\mathfrak{g}_i)_{i=0}^n$ for $\mathfrak{g}_i = \mathfrak{g}^{s_i}$ and $\mathfrak{g}_0 = \mathfrak{g}$;

NewTag(sk): It chooses a random scalar $R \xleftarrow{\$} \mathbb{Z}_p$ and sets $\tau = (\tau_1 = g^{1/R}, \tau_2 = \mathfrak{g}_0^{1/R})$ and $\tilde{\tau} = R$;

VerifTag(vk, τ): Given a verification key $\mathsf{vk} = (\mathfrak{g}_i)_{i=0}^n$ and a tag $\tau = (\tau_1, \tau_2)$, it checks whether $e(\tau_1, \mathfrak{g}_0) = e(g, \tau_2)$ holds or not;

Sign(sk, $\tilde{\tau}$, $\boldsymbol{M} = (M_i)_i$): Given a signing key $\mathsf{sk} = (s_i)_i$ and a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}_1^n$, together with some secret tag $\tilde{\tau}$, one sets $\sigma = (\prod_i M_i^{s_i})^{\tilde{\tau}}$;

MultiplySign(vk, τ, $(\omega_i, \boldsymbol{M}_i, \sigma_i)_{i=1}^\ell$): Given a verification key vk, a tag τ and ℓ tuples of weights $\omega_i \in \mathbb{Z}_p$ and signed messages $\boldsymbol{M}_i$ in σ_i, it outputs $\sigma = \prod \sigma_i^{\omega_i}$;

VerifSign(vk, τ, $\boldsymbol{M} = (M_i)_i, \sigma$): Given a verification key $\mathsf{vk} = (\mathfrak{g}_i)_i$, a vector-message $\boldsymbol{M} = (M_i)_i$, and a signature σ under the tag $\tau = (\tau_1, \tau_2)$, one checks if the equalities $e(\sigma, \tau_2) = \prod_{i=1}^n e(M_i, \mathfrak{g}_i)$ and $e(\tau_1, \mathfrak{g}_0) = e(g, \tau_2)$ hold or not.

When the secret keys for tags are all privately and randomly chosen, independently for each signature, unforgeability has been proven in [FHS19], under Chosen-Message Attacks, in the generic bilinear group model. The intuition is the following: first, under the Knowledge of Exponent Assumption [Dam92, HT98, Gro10], from a new pair (τ_1, τ_2), on the input of either $(g, \mathfrak{g})$ or any other honestly generated pair $(g, \mathfrak{g}_0)$, one can extract the common exponent $1/R$ in the two components. Then, one can see σ as the signature with the secret key $(Rs_i)_i$, with the generator $\mathfrak{g}_0^{1/R}$, instead of $\mathfrak{g}_0$ in the previous construction.

However, if one knows two signatures σ and σ' on $\boldsymbol{M}$ and $\boldsymbol{M}'$ respectively, under the same tag $\tau = (\tau_1, \tau_2)$ with private key $\tilde{\tau}$, and the same key vk, then $\sigma^\alpha \sigma'^\beta$ is a valid signature of $\boldsymbol{M}^\alpha \boldsymbol{M}'^\beta$, still under the same tag τ and the same key vk: this is thus a LH-Sign, where one can control the families of messages that can be combined.

Our LH-Sign has the tag randomizability property, with the algorithm RandTag defined by:

RandTag(vk, τ, $\boldsymbol{M}$, σ): Given a verification key vk, a tag $\tau = (\tau_1, \tau_2)$ and a signature σ on a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}_1^n$, it chooses $\mu \in \mathbb{Z}_p^*$ and outputs $\tau' = (\tau_1^{1/\mu}, \tau_2^{1/\mu})$ and adapts $\sigma' = \sigma^\mu$.

Indeed, from a signature σ on $\boldsymbol{M}$ under the tag $\tau = (\tau_1, \tau_2)$ for the key vk, $\sigma' = \sigma^\mu$ is a new signature on $\boldsymbol{M}$ for the same key vk under the tag $\tau' = (\tau_1^{1/\mu}, \tau_2^{1/\mu})$, perfectly unlinkable to τ, as this is a new random Diffie-Hellman tuple in basis $(g, \mathfrak{g}_0)$ with $\tilde{\tau}' = \mu\tilde{\tau}$, for $\mathfrak{g}_0$ in vk.

As already explained above, we will essentially work on sub-vector spaces of dimension 2: we will thus denote $\sigma = (\sigma_1, \sigma_2) = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, (\boldsymbol{M}, \boldsymbol{M}'))$, under the tag $\tau = (\tau_1, \tau_2)$, where $\sigma_1 = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M})$ and $\sigma_2 = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M}')$, for a common private key $R = \tilde{\tau}$ which led to $\tau = (\tau_1, \tau_2)$.

4.4 Square Diffie-Hellman

This section is a preamble to the next one in which we will propose a new construction of LH-Sign scheme. We define here the useful building blocks: the Square Diffie-Hellman tuples, an extractability assumption that holds in the generic bilinear group model and two important theorems using Square Diffie-Hellman tuples.

Assumptions

We first begin be presenting the needed assumptions:

Definition 32 — Square Discrete Logarithm (SDL) Assumption

In a group $\mathbb{G}$ of prime order p, it states that for any generator g, given $y = g^x$ and $z = g^{x^2}$, it is computationally hard to recover x.

Definition 33 — Decisional Square Diffie-Hellman (DSqDH) Assumption

In a group $\mathbb{G}$ of prime order p, it states that for any generator g, the two following distributions are computationally indistinguishable:

$$\mathcal{D}_{\mathsf{sqdh}}(g) = \{(g, g^x, g^{x^2}), x \xleftarrow{\$} \mathbb{Z}_p\} \qquad \mathcal{D}^3_{\$}(g) = \{(g, g^x, g^y), x, y \xleftarrow{\$} \mathbb{Z}_p\}.$$

It is worth noticing that the DSqDH Assumption implies the SDL Assumption: if one can break SDL, from g, g^x, g^{x^2}, one can compute x and thus break DSqDH.

A fortiori, this implies indistinguishability between the two distributions

$$\mathcal{D}_{\mathsf{sqdh}}(\mathbb{G}) = \{(g, g^x, g^{x^2}), g \xleftarrow{\$} \mathbb{G}, x \xleftarrow{\$} \mathbb{Z}_p\} \qquad \mathcal{D}^3_{\$}(\mathbb{G}) = \{(g_1, g_2, g_3) \xleftarrow{\$} \mathbb{G}^3\}.$$

Below, for proofs, we will need to explicitly extract linear combinations, hence the additional assumption that holds in the generic bilinear group model:

Definition 34 — Extractability Assumption

The extractability assumption states that given n vectors $(\boldsymbol{M}_j = (M_{j,i})_i)_j$, for any adversary that produces a new vector $\boldsymbol{M} = (M_i)_i$ such that $\boldsymbol{M} = \prod_j \boldsymbol{M}_j^{\alpha_j}$, there exists an extractor that outputs $(\alpha_j)_j$.

Proof for Square Diffie-Hellman Tuples

As an SqDH-tuple $(\tau_1 = h, \tau_2 = h^{\tilde{\tau}}, \tau_3 = h^{\tilde{\tau}^2}) \in \mathbb{G}_1^3$ is a Diffie-Hellman tuple $(\tau_1, \tau_2, \tau_2, \tau_3)$, one can use a Schnorr-like proof:

- The prover chooses a random scalar $r \xleftarrow{\$} \mathbb{Z}_p$, and sets and sends $U \leftarrow \tau_1^r$, $V \leftarrow \tau_2^r$;
- The verifier chooses a random challenge $e \xleftarrow{\$} \{0,1\}^\kappa$;
- The prover sends back the response $s = e\tilde{\tau} + r \bmod p$;
- The verifier checks whether both $\tau_1^s = \tau_2^e \times U$ and $\tau_2^s = \tau_3^e \times V$.

This provides an interactive zero-knowledge proof of knowledge of the witness $\tilde{\tau}$ that (τ_1, τ_2, τ_3) is an SqDH-tuple.

Groth-Sahai Proof for Square Diffie-Hellman Tuples. If you just need a proof of validity of the tuple, this is possible, using the Groth-Sahai methodology [GS08], to provide a non-interactive proof of Square Diffie-Hellman tuple: in the asymmetric pairing setting, one sets a reference string $(\mathfrak{v}_{1,1}, \mathfrak{v}_{1,2}, \mathfrak{v}_{2,1}, \mathfrak{v}_{2,2}) \in \mathbb{G}_2^4$, such that $(\mathfrak{v}_{1,1}, \mathfrak{v}_{1,2}, \mathfrak{v}_{2,1}, \mathfrak{v}_{2,2})$ is a Diffie-Hellman tuple.

Given a Square Diffie-Hellman tuple $(\tau_1 = h, \tau_2 = h^{\tilde{\tau}}, \tau_3 = h^{\tilde{\tau}^2}) \in \mathbb{G}_1^3$, one first commits $\tilde{\tau}$: $\mathsf{Com} = (\mathfrak{c} = \mathfrak{v}_{2,1}^{\tilde{\tau}}\mathfrak{v}_{1,1}^{\mu}, \mathfrak{d} = \mathfrak{v}_{2,2}^{\tilde{\tau}}\mathfrak{g}^{\tilde{\tau}}\mathfrak{v}_{1,2}^{\mu})$, for a random $\mu \xleftarrow{\$} \mathbb{Z}_p$, and one sets $\pi_1 = \tau_1^{\mu}$ and $\pi_2 = \tau_2^{\mu}$, which satisfy

$$e(\tau_1, \mathfrak{c}) = e(\tau_2, \mathfrak{v}_{2,1}) \cdot e(\pi_1, \mathfrak{v}_{1,1}) \qquad e(\tau_1, \mathfrak{d}) = e(\tau_2, \mathfrak{v}_{2,2} \cdot \mathfrak{g}) \cdot e(\pi_1, \mathfrak{v}_{1,2})$$
$$e(\tau_2, \mathfrak{c}) = e(\tau_3, \mathfrak{v}_{2,1}) \cdot e(\pi_2, \mathfrak{v}_{1,1}) \qquad e(\tau_2, \mathfrak{d}) = e(\tau_3, \mathfrak{v}_{2,2} \cdot \mathfrak{g}) \cdot e(\pi_2, \mathfrak{v}_{1,2})$$

The proof $\mathsf{proof} = (\mathfrak{c}, \mathfrak{d}, \pi_1, \pi_2)$, when it satisfies the above relations, guarantees that (τ_1, τ_2, τ_3) is a Square Diffie-Hellman tuple. This proof is furthermore zero-knowledge, under the DDH assumption in $\mathbb{G}_2$: by switching $(\mathfrak{v}_{1,1}, \mathfrak{v}_{1,2}, \mathfrak{v}_{2,1}, \mathfrak{g} \times \mathfrak{v}_{2,2})$ into a Diffie-Hellman tuple, one can simulate the proof using the trapdoor in the reference string.

Moreover, one can apply a batch verification [BFI+10], and pack them in a unique one with random scalars $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \xleftarrow{\$} \mathbb{Z}_p$:

$$e(\tau_1^{x_{2,1}}\tau_2^{x_{2,2}}, \mathfrak{c}^{x_{1,1}}\mathfrak{d}^{x_{1,2}}) = e(\tau_2^{x_{2,1}}\tau_3^{x_{2,2}}, \mathfrak{v}_{2,1}^{x_{1,1}}\mathfrak{v}_{2,2}^{x_{1,2}}\mathfrak{g}^{x_{1,2}}) \times e(\pi_1^{x_{2,1}}\pi_2^{x_{2,2}}, \mathfrak{v}_{1,1}^{x_{1,1}}\mathfrak{v}_{1,2}^{x_{1,2}})$$

One thus just has to compute 13 exponentiations and 3 pairing evaluations for the verification, instead of 12 pairing evaluations.

In addition, the proof can be updated using $\rho_{\tau\to\tau'}$ such that $\tau' = \tau^{\rho_{\tau\to\tau'}}$ and randomized using $\mu' \xleftarrow{\$} \mathbb{Z}_p$: $\mathsf{Com}' = (\mathfrak{c}' = \mathfrak{c} \cdot \mathfrak{v}_{1,1}^{\mu'}, \mathfrak{d}' = \mathfrak{d} \cdot \mathfrak{v}_{1,2}^{\mu'})$, in addition $\pi_1' = \pi_1^{\rho_{\tau\to\tau'}} \cdot \tau_1'^{\mu'}$ and $\pi_2' = \pi_2^{\rho_{\tau\to\tau'}} \cdot \tau_2'^{\mu'}$.

Restricted Combinations of Vectors

When one wants to avoid any combination, and just allow to convert a signature of $\boldsymbol{M}$ into a signature of $\boldsymbol{M}^{\alpha}$, while they are all of the same format, one can use expanded vectors (as in Section 4.1), by concatenating a vector that satisfies this restriction: from multiple distinct (non-trivial) Square Diffie-Hellman tuples $(g_i, g_i^{w_i}, g_i^{w_i^2})$, a linear combination that is also a Square Diffie-Hellman tuple cannot use more than one input tuple. We prove it in two different cases: with random and independent bases g_i, but possibly public w_i's, or with a common basis $g_i = g$, but secret w_i's.

We stress that in the first theorem, the w_i's are random and public (assumed distinct), but the bases g_i's are truly randomly and independently generated.

Theorem 35. *Given n valid Square Diffie-Hellman tuples $(g_i, a_i = g_i^{w_i}, b_i = a_i^{w_i})$, with w_i, for random $g_i \xleftarrow{\$} \mathbb{G}^*$ and $w_i \xleftarrow{\$} \mathbb{Z}_p^*$, outputting $(\alpha_i)_{i=1,\ldots,n}$ such that $(G = \prod g_i^{\alpha_i}, A = \prod a_i^{\alpha_i}, B = \prod b_i^{\alpha_i})$ is a valid Square Diffie-Hellman, with at least two non-zero coefficients α_i, is computationally hard under the* DL *assumption.*

Intuitively, from Square Diffie-Hellman tuples where the exponents are known but random and the bases are also known and random, it is impossible to construct a new Square Diffie-Hellman tuple melting the exponents.

Proof. Up to a guess, which is correct with probability greater than $1/n^2$, we can assume that $\alpha_1, \alpha_2 \neq 0$. We are given a discrete logarithm challenge Z, in basis g. We will embed it in either g_1 or g_2, by randomly choosing a bit b:

- if $b = 0$: set $X = Z$, and randomly choose $v \xleftarrow{\$} \mathbb{Z}_p$ and set $Y = g^v$
- if $b = 1$: set $Y = Z$, and randomly choose $u \xleftarrow{\$} \mathbb{Z}_p$ and set $X = g^u$

We set $g_1 \leftarrow X(= g^u)$, $g_2 \leftarrow Y(= g^v)$, with either u or v unknown, and randomly choose $\beta_i \in \mathbb{Z}_p$, for $i = 3, \ldots, n$ to set $g_i \leftarrow g^{\beta_i}$. Eventually, we randomly choose w_i, for $i = 1, \ldots, n$ and output $(g_i, a_i = g_i^{w_i}, b_i = a_i^{w_i})$ together with w_i, to the adversary which outputs $(\alpha_i)_{i=1,\ldots,n}$ such that $(G = \prod g_i^{\alpha_i}, A = \prod a_i^{\alpha_i} = G^w, B = \prod b_i^{\alpha_i} = A^w)$ for some unknown w. We thus have the following relations:

$$\left(\alpha_1 u + \alpha_2 v + \sum_{i=3}^{n} \alpha_i \beta_i\right) \cdot x = \alpha_1 u w_1 + \alpha_2 v w_2 + \sum_{i=3}^{n} \alpha_i \beta_i w_i$$
$$\left(\alpha_1 u w_1 + \alpha_2 v w_2 + \sum_{i=3}^{n} \alpha_i \beta_i w_i\right) \cdot x = \alpha_1 u w_1^2 + \alpha_2 v w_2^2 + \sum_{i=3}^{n} \alpha_i \beta_i w_i^2$$

If we denote $T = \sum_{i=3}^{n} \alpha_i \beta_i$, $U = \sum_{i=3}^{n} \alpha_i \beta_i w_i$, and $V = \sum_{i=3}^{n} \alpha_i \beta_i w_i^2$, that can be computed, we deduce that:

$$(\alpha_1 u w_1 + \alpha_2 v w_2 + U)^2 = (\alpha_1 u + \alpha_2 v + T)(\alpha_1 u w_1^2 + \alpha_2 v w_2^2 + V)$$

which leads to

$$\alpha_1 \alpha_2 (w_1^2 - w_2^2) uv + \alpha_1 (V - 2Uw_1 + Tw_1^2) u + \alpha_2 (V - 2Uw_2 + Tw_2^2) v + (TV - U^2) = 0$$

We consider two cases:

1. $K = \alpha_2 (w_1^2 - w_2^2) v + V - 2Uw_1 + Tw_1^2 = 0 \bmod p$;
2. $K = \alpha_2 (w_1^2 - w_2^2) v + V - 2Uw_1 + Tw_1^2 \neq 0 \bmod p$;

which can be determined by checking whether the equality below holds or not:

$$g^{-(V - 2Uw_1 + Tw_1^2)/(\alpha_2 (w_1^2 - w_2^2))} = Y.$$

One can note that case (1) and case (2) are independent of the bit b.

- If the case (1) happens, but $b = 0$, one aborts. If $b = 1$ (which holds with probability $1/2$ independently of the case) then we can compute $v = -(V - 2Uw_1 + Tw_1^2)/(\alpha_2 (w_1^2 - w_2^2)) \bmod p$ which is the discrete logarithm of Z in the basis g.
- Otherwise, the case (2) appears. If $b = 1$ one aborts. If $b = 0$ (which holds with probability $1/2$ independently of the case), v is known and we have $\alpha_1 K u + \alpha_2 (V - 2Uw_2 + Tw_2^2) v + (TV - U^2) = 0 \bmod p$, which means that the discrete logarithm of Z in the basis g is $u = -(\alpha_2 (V - 2Uw_2 + Tw_2^2) v + (TV - U^2))/(\alpha_1 K) \bmod p$.

□

In the second scenario, the basis is common (for all i, $g_i = g$), but the w_i's are secret, still random and thus assumed distinct.

Theorem 36. *Given n valid Square Diffie-Hellman tuples $(g, a_i = g^{w_i}, b_i = a_i^{w_i})$ for any $g \in \mathbb{G}^*$ and random $w_i \xleftarrow{\$} \mathbb{Z}_p^*$, outputting $(\alpha_i)_{i=1,\ldots,n}$ such that $(G = \prod g^{\alpha_i}, A = \prod a_i^{\alpha_i}, B = \prod b_i^{\alpha_i})$ is a valid Square Diffie-Hellman, with at least two non-zero coefficients α_i, is computationally hard under the* SDL *assumption.*

Lemma 37. *Given any fixed value $\alpha \in \mathbb{Z}_p$ and n valid Square Diffie-Hellman tuples $(g, a_i = g^{w_i}, b_i = a_i^{w_i})$, for any $g \in \mathbb{G}$ and random $w_i \in \mathbb{Z}_p$, outputting $(\alpha_i)_{i=1,\ldots,n}$ such that $\alpha = \sum_{i=1}^{n} \alpha_i w_i$, with at least one non-zero coefficient α_i, is computationally hard under the* SDL *assumption.*

Proof of the lemma. Up to a guess, which is correct with probability greater than $1/n$, we can assume that $\alpha_1 \neq 0$. We are given a square discrete logarithm challenge $(g, Z_1 = g^z, Z_2 = g^{z^2})$, in basis g. We set $a_1 \leftarrow Z_1$, $b_1 \leftarrow Z_2$, and randomly choose $w_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 2, \ldots, n$ to set $(a_i \leftarrow g^{w_i}, b_i \leftarrow a_i^{w_i})$. We then output (g, a_i, b_i), $i = 1, \ldots, n$, to the adversary which outputs $(\alpha_i)_{i=1,\ldots,n}$ and α such that $\alpha_1 z + \sum_{i=2}^n \alpha_i w_i = \alpha$. At this stage, we solve the square discrete logarithm problem by returning $z = (\alpha - \sum_{i=2}^n \alpha_i w_i)/\alpha_1 \bmod p$. □

Proof of the theorem. Again, up to a guess, which is correct with probability greater than $1/n$, we can assume that $\alpha_1 \neq 0$. We are given a square discrete logarithm challenge $(g, Z_1 = g^z, Z_2 = g^{z^2})$, in basis g. We set $a_1 \leftarrow Z_1$, $a_2 \leftarrow Z_2$, and randomly choose $w_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 2, \ldots, n$ to set $(a_i \leftarrow g^{w_i}, b_i = a_i^{w_i})$. We then output (g, a_i, b_i), $i = 2, \ldots, n$, to the adversary that outputs $(\alpha_i)_{i=1,\ldots,n}$ such that $(G = \prod g^{\alpha_i}, A = \prod a_i^{\alpha_i} = G^w, B = \prod b_i^{\alpha_i} = A^w)$ for some unknown w. We thus have the following relations:

$$\left(\sum_{i=1}^n \alpha_i\right) \cdot w = \alpha_1 z + \sum_{i=2}^n \alpha_i w_i \qquad \left(\sum_{i=1}^n \alpha_i\right) \cdot w^2 = \alpha_1 z^2 + \sum_{i=2}^n \alpha_i w_i^2$$

which leads to

$$\left(\alpha_1 z + \sum_{i=2}^n \alpha_i w_i\right)^2 = \left(\alpha_1 + \sum_{i=2}^n \alpha_i\right) \times \left(\alpha_1 z^2 + \sum_{i=2}^n \alpha_i w_i^2\right).$$

If we denote $T = \sum_{i=2}^n \alpha_i w_i$, $U = \sum_{i=2}^n \alpha_i$, and $V = \sum_{i=2}^n \alpha_i w_i^2$, that can be computed from above scalars, we have $(\alpha_1 z + T)^2 = (\alpha_1 + U) \cdot (\alpha_1 z^2 + V)$, and thus

$$U\alpha_1 z^2 - 2T\alpha_1 z + (\alpha_1 + U)V - T^2 = 0 \bmod p.$$

Using Lemma 37 on the $n-1$ tuples (g, a_i, b_i), for $i = 2, \ldots, n$, the probability that $T = \sum_{i=2}^n \alpha_i w_i = 0$ is negligible, unless one can break the SDL Assumption. So we have $T \neq 0$, with two cases:

1. If $U \neq 0$ then, because computing square roots in $\mathbb{Z}_p$ is easy, one can solve the above quadratic equation for z that admits solutions, and obtain two solutions for z. By testing which one satisfies $g^z = Z_1$, one can find out the correct z and thus solve the SDL problem.
2. If $U = 0$, one can compute $z = (\alpha_1 V - T^2)/(2T\alpha_1) \bmod p$ and thus solve the SDL problem.

□

Randomizable Tags

As in the Definition 27, we can randomize the tags together with the messages: but just in a computational way, and not in a statistical way. Indeed, from a message-signature $(\boldsymbol{M}, \sigma)$ for a tag $\tau = (\tau_1, \tau_2, \tau_3, \pi)$, one can derive the signature for the message $\boldsymbol{M}' = \boldsymbol{M}^\alpha$ for the tag $\tau' = (\tau_1^\alpha, \tau_2^\alpha, \tau_3^\alpha, \pi')$, where π' can be adapted from π and α. The triple $(\tau_1^\alpha, \tau_2^\alpha, \tau_3^\alpha)$ in the tag is not uniformly random, as w has not changed, but it is computationally unlinkable to (τ_1, τ_2, τ_3) under the DDH assumption.

4.5 SqDH LH-Sign Scheme

In this section, we provide a generic transformation to convert a OT-LH-Sign to LH-Sign, using Square Diffie-Hellman tuples $(g, g^{w_i}, g^{w_i^2})$ for the tags. We thus obtain a new construction of linearly homomorphic signature with randomizable tags.

4.5.1 A First Generic Conversion

First Generic Conversion from OT-LH-Sign to LH-Sign

Let $\Sigma = (\mathsf{Setup}, \mathsf{SKeygen}, \mathsf{Sign}, \mathsf{MultiplySign}, \mathsf{VerifSign})$ be a OT-LH-Sign, we complete it into $\Sigma' = (\mathsf{Setup}', \mathsf{SKeygen}', \mathsf{NewTag}', \mathsf{VerifTag}', \mathsf{Sign}', \mathsf{MultiplySign}', \mathsf{VerifSign}')$ as follows:

$\mathsf{Setup}'(1^\kappa)$**:** It runs $\mathsf{Setup}(1^\kappa)$ to obtain param and adds the tag space $\mathsf{param}' = (\mathsf{param}, \mathbb{Z}_p^* \times \mathbb{G}^*)$;

$\mathsf{SKeygen}'(\mathsf{param}', n)$**:** It runs $\mathsf{SKeygen}(\mathsf{param}, n+3)$;

$\mathsf{NewTag}'(\mathsf{sk})$**:** It chooses a random scalar $w \xleftarrow{\$} \mathbb{Z}_p^*$ and a random group element $h \xleftarrow{\$} \mathbb{G}$, and sets $\tilde{\tau} = \tau = (w, h)$;

$\mathsf{VerifTag}'(\mathsf{vk}, \tau)$**:** It checks whether $\tau = (w, h) \in \mathbb{Z}_p^* \times \mathbb{G}$ or not;

$\mathsf{Sign}'(\mathsf{sk}, \tilde{\tau} = (w, h), \boldsymbol{M})$**:** It extends $\boldsymbol{M}$ into $\boldsymbol{M}'$ with the three additional components (h, h^w, h^{w^2}) and signs it as $\sigma = \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}')$;

$\mathsf{MultiplySign}'(\mathsf{vk}, \tau, \{\omega_i, \boldsymbol{M}_i, \sigma_i\}_{i=1}^{\ell})$**:** It simply computes $\sigma = \prod_i \sigma_i^{\omega_i}$ and $\tau' = (w, h' = \prod_i h^{\omega_i})$;

$\mathsf{VerifSign}'(\mathsf{vk}, \tau = (w, h), \boldsymbol{M}, \sigma)$**:** It first extends $\boldsymbol{M}$ into $\boldsymbol{M}'$ with the Square Diffie-Hellman tuple (h, h^w, h^{w^2}) and checks whether $\mathsf{VerifSign}(\mathsf{vk}, \boldsymbol{M}', \sigma) = 1$ or not.

One can note that the $\mathsf{MultiplySign}'$ provides a signature under a new tag τ', but this is still consistent with the definition of the LH-Sign. However, randomizability of the tag is not possible.

Theorem 38. *If Σ is OT-LH-Sign then Σ' is LH-Sign under the* DL *assumption and the extractability assumption (Definition 34).*

Proof. Since the tags are fully public, any NewTag'-query is answered by a random pair (w_i, g_i), and a Sign'-query is answered by simply forwarding a Sign-query to the Σ security game. Receiving the forgery $(\mathsf{vk}, \tau = (w, G), \boldsymbol{M}, \sigma)$, one first generates $\boldsymbol{M}'$ from $\boldsymbol{M}$ and τ and checks the validity, which means, according to the unforgeability of Σ, that there exist $(\alpha_i)_i$ such that $\boldsymbol{M}' = \prod \boldsymbol{M}_i'^{\alpha_i}$. The above extractability assumption provides these coefficients $(\alpha_i)_i$. If we just keep the 3 last components of each extended messages and the tags, we have square Diffie-Hellman triples $(g_i, a_i = g_i^{w_i}, b_i = a_i^{w_i})_i$, for random g_i and w_i (but possibly equal when the same tag is used several times), and the triple $(G = \prod g_i^{\alpha_i}, A = G^w = \prod a_i^{\alpha_i}, B = A^w = \prod b_i^{\alpha_i})$ extracted from the forgery. By combining the identical tags together, and so by summing in β_j the α_i's that correspond to the same triples (g_i, a_i, b_i), we have $(G = \prod g_j^{\beta_j}, A = G^w = \prod a_j^{\beta_j}, B = A^w = \prod b_j^{\beta_j})$, for random and distinct triples $(g_j, a_j, b_j)_j$. From Theorem 35, under the DL assumption, at most one coefficient is non-zero: none or β_J, and so at most one tag is represented: none or (g_J, a_J, b_J). Hence $\boldsymbol{M}$ is either $(1, \ldots, 1)$ or $\prod \boldsymbol{M}_i^{\alpha_i}$ for i such that $(g_i, a_i, b_i) = (g_J, a_J, b_J)$. □

4.5.2 A Second Generic Conversion

Second Generic Conversion from OT-LH-Sign to LH-Sign

Let $\Sigma = (\mathsf{Setup}, \mathsf{SKeygen}, \mathsf{Sign}, \mathsf{MultiplySign}, \mathsf{VerifSign})$ be a OT-LH-Sign, we complete it into $\Sigma' = (\mathsf{Setup}', \mathsf{SKeygen}', \mathsf{NewTag}', \mathsf{VerifTag}', \mathsf{Sign}', \mathsf{MultiplySign}', \mathsf{VerifSign}')$ as follows:

$\mathsf{Setup}'(1^\kappa)$**:** It runs $\mathsf{Setup}(1^\kappa)$ to obtain param and adds the tag space $\mathsf{param}' = (\mathsf{param}, \mathbb{G}^3 \times \Pi)$. Note that we need the group $\mathbb{G}$ to be extended to a bilinear setting $(\mathbb{G}, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ for the proofs;

$\mathsf{SKeygen}'(\mathsf{param}', n)$**:** It runs $\mathsf{SKeygen}(\mathsf{param}, n+3)$;

$\mathsf{NewTag}'(\mathsf{sk})$**:** It chooses a random scalar $w \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\tilde{\tau} = w$ and $\tau = (g, g^w, g^{w^2}, \pi)$, where π is a zero-knowledge proof of valid square Diffie-Hellman tuple for (g, g^w, g^{w^2});

$\mathsf{VerifTag}'(\mathsf{vk}, \tau)$**:** It checks the proof π on (g, g^w, g^{w^2});

$\mathsf{Sign}'(\mathsf{sk}, \tilde{\tau} = w, \boldsymbol{M})$**:** It extends $\boldsymbol{M}$ into $\boldsymbol{M}'$ with the three additional components (g, g^w, g^{w^2}), and signs it as $\sigma = \mathsf{Sign}(\mathsf{sk}, \boldsymbol{M}')$;

$\mathsf{MultiplySign}'(\mathsf{vk}, \tau = (\tau_1, \tau_2, \tau_3, \pi), \{\omega_i, \boldsymbol{M}_i, \sigma_i\}_{i=1}^{\ell})$**:** It computes $\sigma = \prod_i \sigma_i^{\omega_i}$, $\omega = \sum_i \omega_i$ and $\tau' = (\tau_1, \tau_2^{\omega}, \tau_3^{\omega}, \pi')$ with π' the updated proof of valid square Diffie-Hellman tuple;

$\mathsf{VerifSign}'(\mathsf{vk}, \tau = (\tau_1, \tau_2, \tau_3, \pi), \boldsymbol{M}, \sigma)$**:** It first checks whether $\mathsf{VerifTag}'(\mathsf{vk}, \tau) = 1$ or not, if the tag is valid, it extends $\boldsymbol{M}$ into $\boldsymbol{M}'$ with τ and checks whether $\mathsf{VerifSign}(\mathsf{vk}, \boldsymbol{M}', \sigma) = 1$ or not.

Note that for the $\mathsf{MultiplySign}'$ to be possible, one needs an homomorphic zero-knowledge proof of valid square Diffie-Hellman tuple, as the Groth-Sahai techniques [GS08] allow in a bilinear setting: let $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2}) \in \mathbb{G}_2^4$ be a Diffie-Hellman tuple, for a Square Diffie-Hellman tuple $(g, A = g^w, B = A^w) \in \mathbb{G}^3$ one can generate a commitment of w, $\mathsf{Com} = (c = v_{2,1}^w v_{1,1}^{\mu}, d = v_{2,2}^w v_{1,2}^{\mu} g^w) \in \mathbb{G}_2^2$, and the proofs $\mathsf{proof} = (\Theta = g^{\mu}, \Psi = A^{\mu}) \in \mathbb{G}^2$. The proof π thus consists of the pair $(\mathsf{Com}, \mathsf{proof})$, and is homomorphic. It is well-known to be perfectly-sound, and for the zero-knowledge property, one just has to switch from the Diffie-Hellman tuple $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2})$ to a random tuple $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2})$ because they are computationally indistinguishable under the DDH assumption in $\mathbb{G}_2$, or statistically indistinguishable in the generic group model. The latter assumption will be required for the security analysis below.

Theorem 39. *If Σ is OT-LH-Sign then Σ' is LH-Sign, in the generic group model.*

Proof. Let us consider an adversary that asks several tags $(\tau_i)_i$ and signatures $(\sigma_i)_i$ on messages $(\boldsymbol{M}_i)_i$ and tags of its choice, and eventually produces a forgery $(\tau, \boldsymbol{M}, \sigma)$ with probability ε. A forgery means that

- the tag is valid, and so the proof π is accepted;
- the signature is valid;
- $\boldsymbol{M}$ is not in the spans of the messages signed under the same tag.

First, as the signature Σ' is based on the $\mathsf{OT\text{-}LH\text{-}Sign}$ Σ thanks to the concatenation of the message and (τ_1, τ_2, τ_3) in the tags, we know that necessarily $\boldsymbol{M}'$ (the completion of $\boldsymbol{M}$ with the triple in the tag) is a linear combination of the extended messages involved in the signing queries, unless one has broken the unforgeability of Σ, which can just happen with negligible probability.

As a consequence, the triple (τ_1, τ_2, τ_3) in the tag of the forgery is a linear combination of the Square Diffie-Hellman triples in the signing queries, with probability $\varepsilon' = \varepsilon - \mathsf{negl}()$:

- either (τ_1, τ_2, τ_3) is not a Square Diffie-Hellman tuple;
- or (τ_1, τ_2, τ_3) is a Square Diffie-Hellman tuple.

In the former case, where (τ_1, τ_2, τ_3) is not a Square Diffie-Hellman tuple, then we break the perfect soundness of Groth-Sahai proofs, as all the proofs for the honest tags have been generated

honestly. Hence, the latter case should happen with probability greater than $\varepsilon'' = \varepsilon' - \mathsf{negl}()$: (τ_1, τ_2, τ_3) is both a linear combination of the input triples but still a Square Diffie-Hellman tuple, with probability greater than ε''. Then, the Theorem 36 shows that (τ_1, τ_2, τ_3) is one of the input triples to a power α (or possibly $(1, 1, 1)$). However, to apply this theorem, we are given random Square Diffie-Hellman tuples as input and we should be able to generate the proofs of validity. To this aim, we switch the Groth-Sahai proofs in perfectly hiding mode: we replace a Non Diffie-Hellman tuple by a Diffie-Hellman tuple in the CRS, which is statistically indistinguishable to a generic adversary, as its probability to make the difference is N/p^2, where N is the number of group operations. So after this switch, from a list of Square Diffie-Hellman tuples, we simulate the proofs, and the adversary outputs a tuple (τ_1, τ_2, τ_3) that is both a linear combination of the input triples but still a Square Diffie-Hellman tuple, with probability greater than $\varepsilon'' - N/p^2$. As we are considering a forgery, several tags should be involved, which is excluded by the Theorem 36: ε'' is negligible, and so ε is negligible too. □

Universal Tag

Whereas only messages signed under the same tag can be combined, a message signed under the tag $\tau_0 = (1, 1, 1, \pi)$, where $\pi = (1, 1, 1)$ is a proof for $w = 0$ with $\mu = 0$ in the commitment Com, can be combined with any message. Such a tag $(1, 1, 1, \pi)$, which was not in $\mathcal{T}$, is a *universal tag*. Indeed, multiplied to any Square Diffie-Hellman tuple, this is still a Square Diffie-Hellman tuple. This does not contradict the Theorems 35 and 36, as they only deal with non-trivial Square Diffie-Hellman triples.

Chapter

5 MixNet

This chapter is based on the paper [HPP20] published in the proceedings of the International Conference on Practice and Theory of Public-Key Cryptography, PKC 2020.

Chapter content

In the two main techniques of Mix-Networks, Furukawa and Sako [FS01] make proofs of permutation matrices and Neff [Nef01] considers polynomials which remain identical with a permutation of the roots. The former approach with proof of permutation matrices is more classical, with many candidates. Groth and Lu [GL07] proposed the first non-interactive zero-knowledge (NIZK) proof of shuffle without random oracles, using Groth-Sahai proofs with pairings [GS08] but under an hypothesis proven in the generic bilinear group model. Even with that, computations are still very expensive because the overhead proof is linear in Nn, where n is the number of ciphertexts and N the number of mixing rounds. In addition, they needed a Common Reference String (CRS) linear in n. More recently, Fauzi *et al.* [FLSZ17] proposed a new pairing-based NIZK shuffle argument to improve the computation for both the prover and the verifier, and improved the soundness of the protocol. However they still had a CRS linear in the number of ciphertexts, and the soundness holds in the generic bilinear group model.

In this chapter we apply the construction of OT-LH-Sign and LH-Sign seen in the previous chapter to build a new mix-network. In our shuffle, each ciphertext C_i (encrypted vote in the ballot, in the context of electronic voting) is signed by its sender and the mix-server randomizes the ciphertexts $\{C_i\}$ and permutes them into the set $\{C'_i\}$ in a provable way. The goal of the proof is to show the existence of a permutation Π from $\{C_i\}$ to $\{C'_i\}$ such that for every i, $C'_{\Pi(i)}$ is a randomization of C_i. Then, the output ciphertexts can be mixed again by another mix-server. The unforgeability of the signature schemes will essentially provide the soundness of the proof of correct mixing: only permutations of ballots will be possible.

In a first step, we provide a high-level description of our construction to give the intuitions of our new method. However, this high-level presentation suffers several issues which are detailed. Then, the second section 5.2 provides the solutions with the full scheme. At this point, the global proof of mixing, after several mix-servers, is linear (and verification thus has a linear

cost) in the number of mix-servers. Therefore in a third step 5.3, we explain how to obtain a constant-time overhead for the proof to publish, and thus for the verification. We detail the security analysis in Section 5.4 with the three parts: soundness, unlinkability and correctness. Eventually, we conclude with some applications in Section 5.5.

5.1 Our Scheme: General Description

We first provide a high-level description of our mix-net in Figure 5.1. As said above, the goal of this presentation is just for the intuition: there are still many problems, that will be highlighted and addressed in the next sections. We need two signature schemes:

- any OT-LH-Sign scheme (Setup,Keygen,Sign,MultiplySign,VerifSign), with additional MultiplyKey, that will be used to sign ElGamal ciphertexts in $\mathbb{G}_1$: the ciphertexts C_i and the signatures σ_i belong to $\mathbb{G}_1$ and are verified with the user' verification keys $\mathsf{vk}_i = (\mathfrak{g}_k)_k$ in $\mathbb{G}_2$;
- and any LH-Sign with randomizable tag scheme (Setup^*, Keygen^*, NewTag^*, $\mathsf{RandTag}^*$, $\mathsf{VerifTag}^*$, Sign^*, $\mathsf{MultiplySign}^*$, $\mathsf{VerifSign}^*$) that will be used to sign users' verification keys vk_i in $\mathbb{G}_2$: the signatures Σ_i also belong to $\mathbb{G}_2$ and are verified with Certification Authority's verification key $\mathsf{VK} = (g_k)_k$ in $\mathbb{G}_1$.

CA = Certificate Authority, $\mathcal{U}_i$ = User$_i$, $\mathcal{S}_j$ = Mix-Server$_j$

Keys

CA's keys: $\begin{cases} (\mathsf{SK},\mathsf{VK}) \leftarrow \mathsf{Keygen}^*() & \text{Authority LH-Sign signing key} \\ (\mathsf{EK},\mathsf{DK}) \leftarrow \mathsf{EKeygen}() & \text{Authority homomorphic encryption key} \end{cases}$

$\mathcal{U}_i$'s keys: $(\mathsf{sk}_i,\mathsf{vk}_i) \leftarrow \mathsf{Keygen}()$ User OT-LH-Sign signing key

CA signs vk_i: $(\tilde{\tau}_i,\tau_i) \leftarrow \mathsf{NewTag}^*(\mathsf{SK})$ $\Sigma_i \leftarrow \mathsf{Sign}^*(\mathsf{SK},\tilde{\tau}_i,\mathsf{vk}_i)$

Ciphertext for randomization: $C_0 \leftarrow \mathsf{Encrypt}(\mathsf{EK},1)$

Initial ballots (for $i = 1,\ldots,n$)

$\mathcal{U}_i$ generates: $\begin{cases} C_i \leftarrow \mathsf{Encrypt}(\mathsf{EK},M_i) & \text{User's ballot encryption} \\ \sigma_{i,0} \leftarrow \mathsf{Sign}(\mathsf{sk}_i,C_0) & \text{User's signature on randomization} \\ \sigma_{i,1} \leftarrow \mathsf{Sign}(\mathsf{sk}_i,C_i) & \text{User's ballot signature} \end{cases}$

$\mathcal{BB}\mathsf{ox}^{(0)} = (C_i,\sigma_{i,0},\sigma_{i,1},\mathsf{vk}_i,\Sigma_i,\tau_i)_i$

Mix (j-th mix-server, for $i = 1,\ldots,n$)

From $\mathcal{BB}\mathsf{ox}^{(j-1)} = (C_i,\sigma_{i,0},\sigma_{i,1},\mathsf{vk}_i,\Sigma_i,\tau_i)_i$, $\mathcal{S}_j$ makes, for all i:

Randomization of the ballot:

$$C'_i = C_i \cdot C_0^{\gamma_{j,i}} \qquad \sigma^*_{i,1} = \mathsf{MultiplySign}(\mathsf{vk}_i,\{(\gamma_{j,i},C_0,\sigma_{i,0}),(1,C_i,\sigma_{i,1})\})$$

Randomization of the keys:

$$\begin{cases} \mathsf{vk}'_i = (\mathsf{vk}_i)^{\alpha_j} & \Sigma^*_i = \mathsf{MultiplySign}^*(\mathsf{VK},\tau_i,(\alpha_j,\mathsf{vk}_i,\Sigma_i)) \\ (\mathsf{VK},\tau'_i,\mathsf{vk}_i,\Sigma'_i) & = \mathsf{RandTag}^*(\mathsf{VK},\tau_i,\mathsf{vk}_i,\Sigma^*_i) \end{cases}$$

Adaptation of the signatures:

$$\sigma'_{i,0} = \mathsf{MultiplyKey}(C_0,(\alpha_j,\mathsf{vk}_i,\sigma_{i,0}))$$
$$\sigma'_{i,1} = \mathsf{MultiplyKey}(C'_1,(\alpha_j,\mathsf{vk}_i,\sigma^*_{i,1}))$$

$\mathcal{BB}\mathsf{ox}^{(j)} = (C'_{\Pi(i)},\sigma'_{\Pi(i),0},\sigma'_{\Pi(i),1},\mathsf{vk}'_{\Pi(i)},\Sigma'_{\Pi(i)},\tau'_{\Pi(i)})_i$

Figure 5.1: High-Level Description (Insecure Scheme)

Each user $\mathcal{U}_i$ generates a pair $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Keygen}()$ to sign vectors in $\mathbb{G}_1$. $\mathcal{U}_i$ first encrypts his message M_i under an ElGamal encryption scheme, with encryption key EK and signs it to obtain the signed-encrypted ballot $(C_i, \sigma_{i,1})$ under vk_i. Obviously, some guarantees are needed.

In order to be sure that a ballot is legitimate, all the verification keys must be certified by the system (certification authority CA) that signs vk_i under SK, where $(\mathsf{SK}, \mathsf{VK}) \leftarrow \mathsf{Keygen}^*()$, into Σ_i. Then, anyone can verify the certified keys $(\mathsf{vk}_i, \Sigma_i)_i$ are valid under the system verification key VK. Since we want to avoid combinations between verification keys, we use $\mathsf{LH\text{-}Sign}$ with randomizable tags to sign the verification keys with a tag τ_i per user $\mathcal{U}_i$.

Because of encryption, M_i is protected, but this is not enough as it will be decrypted in the end. One also needs to guarantee unlinkability between the input and output ballots to guarantee anonymity of users. As the ballot boxes contain the ciphertexts, as well as the verification keys, the ballots must be transformed in an unlinkable way, then they can be output in a permuted way.

To have C'_i unlinkable to C_i, C'_i must be a randomization of C_i. With an ElGamal encryption, it is possible to randomize a ciphertext by multiplying by an encryption of 1. Thus, anyone can compute an encryption C_0 of 1, and as we use an $\mathsf{OT\text{-}LH\text{-}Sign}$ scheme, from a signature $\sigma_{i,0}$ of C_0 under the user's key, one can adapt $\sigma_{i,1}$ by using the message homomorphism (Property 26) with $\mathsf{MultiplySign}$ to obtain $\sigma^*_{i,1}$. In the same way, vk'_i and τ'_i must be randomizations of respectively vk_i and τ_i. If $\mathsf{vk}'_i = \mathsf{vk}_i^\alpha$, its signature must be derived from Σ_i with $\mathsf{MultiplySign}^*$ and τ'_i is obtained with the randomizable tag (Property 27) with $\mathsf{RandTag}^*$. Eventually, as we change the verification key, $\sigma'_{i,0}$ and $\sigma'_{i,1}$ must be adapted, which is possible thanks to the weak key homomorphism (Property 30) with $\mathsf{MultiplyKey}$.

Then one generates a random permutation Π to output a new ballot-box with permuted randomized ballots $(\mathsf{vk}'_{\Pi(i)}, \Sigma'_{\Pi(i)}, C'_{\Pi(i)}, \sigma'_{\Pi(i),0}, \sigma'_{\Pi(i),1})_i$.

Difficulties

The above high-level scheme gives intuitions of our main approach. However, to get the required security, we still face a few issues that will be explained below and which motivate the full scheme described in the next section.

Expanded Vectors. From the signatures $\sigma_{i,0}$ and $\sigma_{i,1}$ with an $\mathsf{OT\text{-}LH\text{-}Sign}$ scheme, anyone can compute $\sigma = \mathsf{MultiplySign}(\mathsf{vk}_i, \{(\alpha, C_0, \sigma_{i,0}), (\beta, C_i, \sigma_{i,1})\})$ for any α, β. As explained in Section 4.1, we can impose $\beta = 1$ and the right format of C'_i.

Non-Trivial Transformation. The weak key homomorphism allows to randomize vk_i into $\mathsf{vk}'_i = \mathsf{vk}_i^\alpha$ but, with our scheme, $\mathsf{VerifSign}(\mathsf{vk}_i^\alpha, C_i, \sigma_{i,1})$ is valid for any $\alpha \neq 0$ if and only if $\mathsf{VerifSign}(\mathsf{vk}_i, C_i, \sigma_{i,1})$ is valid. This provides a link between vk'_i and vk_i. To solve this issue, we introduce a randomizer vk_0, as for the ciphertext. This is a special vector also signed by CA to randomize vk_i in a non-trivial way: $\mathsf{vk}'_i = (\mathsf{vk}_i \cdot \mathsf{vk}_0^{\delta_i})^\alpha$. We will thus also have the signature $\Sigma_{i,0}$ of vk_0 and the signature $\Sigma_{i,1}$ (instead of Σ_i) of vk_i, both under the same tag τ_i to allow combinations.

Legitimate Ballots. Whereas all the ballots must be signed, nothing prevents a mix-server from deleting a ballot or from adding a ballot signed by a legitimate user (that owns a valid key vk_i). If one first checks that the number of ballots is kept unchanged, it is still possible that a ballot is replaced by a new legitimate ballot. Since we will consider honest and corrupted users (and so honest and corrupted ballots), four cases are possible: one replaces an honest or corrupted ballot by another honest or corrupted one. Our scheme will not provide guarantees against the replacement of a corrupted ballot by another corrupted ballot. Nonetheless, by

adding a zero-knowledge proof of Diffie-Hellman tuple between the products of the verification keys before and after the mix, we can avoid all the other cases involving honest users.

Multiple Servers. After the last round, one gets a proof that the output ballot-box contains a permutation of randomized ciphertexts from the input ballot-box. However, the last mix-server could start from the initial ballot-box instead of the previous one, and then know the permutation. This would break anonymity, as soon as the last mix-server is dishonest. We will ask the mix-servers to sign their contributions to prove the multiple and independent permutations: each mix-server j generates the Diffie-Hellman proofs from $\mathcal{BB}\mathsf{ox}^{(j-1)}$ to $\mathcal{BB}\mathsf{ox}^{(j)}$, and signs them. We will then detail this solution in the next section, which will provide a proof linear in the number of ballots and in the number of mix-servers (because of the multiple signature). Thereafter, with specific multi-signature, one can become independent of the number of mix-servers.

5.2 Our Scheme: Full Description

With all the previous remarks and explanations, we can now provide the full description of our scheme which is given in Figure 5.2.

Keys. As we will sign expanded ciphertexts of dimension 4 (see below), each user needs a secret-verification key pair $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Keygen}(\mathsf{param}, 4)$ in $\mathbb{Z}_p^4 \times \mathbb{G}_2^5$. With our OT-LH-Sign, the first element of vk_i is common for all the users and initialized to $\mathfrak{g}_0 = \mathfrak{g}$. Then, one also needs a signature $\Sigma_i = (\Sigma_{i,0}, \Sigma_{i,1})$ with our LH-Sign from the certification authority of the pair $(\mathsf{vk}_0, \mathsf{vk}_i)$ where $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0, 1, 1)$ is used to make the non-trivial transformation on vk_i during the mixes. This signature is signed by the authority possessing $(\mathsf{SK}, \mathsf{VK}) \leftarrow \mathsf{Keygen}^*(\mathsf{param}', 5)$ in $\mathbb{Z}_p^5 \times \mathbb{G}_1^6$ with a specific tag τ_i per user. Eventually, each mix-server has a pair of (standard) signature scheme $(\mathsf{SK}_j, \mathsf{VK}_j) \leftarrow \mathsf{SKeygen}()$ just to sign with SSign its mixing contribution. The keys VK and $(\mathsf{VK}_j)_j$, as well as $\mathsf{EK} = h = g^d \in \mathbb{G}_1$ and the random $\ell \xleftarrow{\$} \mathbb{G}_1$, are assumed to be known to everybody.

As we are using El Gamal ciphertexts, the ciphertext for randomization is $C_0 = (g, h)$, the trivial encryption of $1 = g^0$, with random coin equal to 1.

Initial ballots. Each user encrypts his message M_i under EK to obtain $C_i = (a_i, b_i)$. With the remarks we already made, one needs to expand C_i into $\overline{C}_i = (g, \ell_i, a_i, b_i)$ and C_0 into $\overline{C}_0 = (1, \ell, g, h)$. The addition of the first element is due to the affine space we want in the signature σ_i (see Section 4.1) and the second element is because we randomize the third position of vk_i with $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0, 1, 1)$ and because the first position of vk_i is used for the verification but not to sign (the last four elements of vk_i are used to sign). Finally, $\sigma_i = (\sigma_{i,0}, \sigma_{i,1})$ is simply the OT-LH-Sign of $(\overline{C}_0, \overline{C}_i)$ under the signing key sk_i.

Mix. To make a mix, the j-th mix-server computes the randomized verification keys $\mathsf{vk}'_i = (\mathsf{vk}_i \cdot \mathsf{vk}_0^{\delta_i})^{\alpha}$, the randomized ciphertexts $\overline{C}'_i = \overline{C}_i \cdot \overline{C}_0^{\gamma_i}$ and the randomized tags $\tau'_i = \tau_i^{1/\mu_i}$, and updates the signatures σ'_i and Σ'_i, thanks to the properties of the signatures. The random scalar α is common to all the ballots, but $\gamma_i, \delta_i, \mu_i$ are independent random scalars for each ballot. Then, the mix-server chooses a permutation Π and sets the j-th ballot-box $\mathcal{BB}\mathsf{ox}^{(j)}$ with all the randomized and permuted ballots $(C'_{\Pi(i)}, \ell'_{\Pi(i)}, \sigma'_{\Pi(i)}, \mathsf{vk}'_{\Pi(i)}, \Sigma'_{\Pi(i)}, \tau'_{\Pi(i)})_i$. As already explained, the mix-server also needs to make a proof $\mathsf{proof}^{(j)}$ from $\mathcal{BB}\mathsf{ox}^{(j-1)}$ to $\mathcal{BB}\mathsf{ox}^{(j)}$, to guarantee the proper relations between the products of the verification keys and the products of the messages, and signs it in $\mathsf{sig}^{(j)}$. Finally, the output of the mix contains $\mathcal{BB}\mathsf{ox}^{(j)}$ and $(\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{j}$ the set of proofs and mix-server signatures of the previous mixes until the j-th mix.

CA = Certificate Authority, $\mathcal{U}_i$ = User$_i$, $\mathcal{S}_j$ = Mix-Server$_j$

MixSetup(1^κ):

Let $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e) \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{param}' = \{\mathsf{param}, \mathcal{T} = \mathbb{G}_2 \times \mathbb{G}_1\}$;
Let $\mathsf{NIZK_{DH}\text{-}param} \leftarrow \mathsf{NIZK_{DH}\text{-}Setup}(1^\kappa)$ and $\mathsf{Sparam} \leftarrow \mathsf{SSetup}(1^\kappa)$;
Let $(\mathsf{DK} = d, \mathsf{EK} = h = g^d) \leftarrow \mathsf{EKeygen}(1^\kappa)$ and $\overline{C}_0 = (1, \ell, g, h)$ for $\ell \xleftarrow{\$} \mathbb{G}_1$; It outputs $\mathsf{Mix\text{-}param} = (\mathsf{param}', \mathsf{NIZK_{DH}\text{-}param}, \mathsf{Sparam}, \mathsf{EK}, \ell)$.

MixKeygen(Mix-param):

$$\mathsf{CA}: \begin{cases} \mathsf{SK} = (S_1, S_2, S_3, S_4, S_5) \xleftarrow{\$} \mathbb{Z}_p^5,\ \mathsf{VK} = (g, g^{S_1}, g^{S_2}, g^{S_3}, g^{S_4}, g^{S_5}) \\ \text{and for each user } \mathcal{U}_i,\ \tilde{\tau}_i = R_i \xleftarrow{\$} \mathbb{Z}_p,\ \tau_i = (\tau_{i,1} = \mathfrak{g}^{1/R_i}, \tau_{i,2} = g^{1/R_i}) \end{cases}$$

$\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0 = \mathfrak{g}, 1, 1)$

$$\mathcal{S}_j: \{\ \ (\mathsf{SK}_j, \mathsf{VK}_j) \leftarrow \mathsf{SKeygen}()$$

$$\mathcal{U}_i: \begin{cases} \mathsf{sk}_i = (u_i, v_i, x_i, y_i) \xleftarrow{\$} \mathbb{Z}_p^4,\ \mathsf{vk}_i = (\mathfrak{g}_0 = \mathfrak{g}, \mathfrak{f}_i = \mathfrak{g}_0^{u_i}, \mathfrak{l}_i = \mathfrak{g}_0^{v_i}, \mathfrak{g}_i = \mathfrak{g}_0^{x_i}, \mathfrak{h}_i = \mathfrak{g}_0^{y_i}) \\ \Sigma_i = \left(\Sigma_{i,0} = \mathfrak{g}^{S_3\tilde{\tau}_i},\ \Sigma_{i,1} = (\mathfrak{g}_0^{S_1}\mathfrak{f}_i^{S_2}\mathfrak{l}_i^{S_3}\mathfrak{g}_i^{S_4}\mathfrak{h}_i^{S_5})^{\tilde{\tau}_i}\right) \end{cases}$$

MixInit($\mathsf{sk}_i, M_i, \mathsf{vk}_i, \Sigma_i, \tau_i$):

$\mathcal{U}_i$ chooses $r_i \xleftarrow{\$} \mathbb{Z}_p$ and $\ell_i \xleftarrow{\$} \mathbb{G}_1$ and computes

$$C_i = (a_i = g^{r_i}, b_i = h^{r_i} M_i) \qquad \overline{C}_i = (g, \ell_i, a_i, b_i)$$

$$\sigma_i = (\sigma_{i,0} = \ell^{v_i} g^{x_i} h^{y_i}, \sigma_{i,1} = g^{u_i} \ell_i^{v_i} a_i^{x_i} b_i^{y_i})$$

It outputs $\mathcal{B}_i = (C_i, \ell_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i)$.

$$\mathcal{BB}\mathsf{ox}^{(0)} = (\mathcal{B}_i)_{i=1}^N$$

Mix($\mathsf{SK}_j, \mathcal{BB}\mathsf{ox}^{(j-1)}, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{j-1}, \Pi_j$):

From $\mathcal{BB}\mathsf{ox}^{(j-1)} = (C_i, \ell_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i)_i, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{j-1}$,
$\mathcal{S}_j$ chooses $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and for each ballot i, $\gamma_i, \delta_i, \mu_i \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$a'_i = a_i \cdot g^{\gamma_i} \quad b'_i = b_i \cdot h^{\gamma_i} \quad \ell'_i = \ell_i \cdot \ell^{\gamma_i} \quad \sigma'_{i,1} = \sigma_{i,1} \cdot \sigma_{i,0}^{\gamma_i} \cdot \ell_i'^{\delta_i} \quad \sigma'_{i,0} = \sigma_{i,0} \cdot \ell^{\delta_i}$$

$$\mathfrak{g}'_0 = \mathfrak{g}_0^\alpha \qquad \mathfrak{f}'_i = \mathfrak{f}_i^\alpha \qquad \mathfrak{l}'_i = (\mathfrak{l}_i \cdot \mathfrak{g}_0^{\delta_i})^\alpha \qquad \mathfrak{g}'_i = \mathfrak{g}_i^\alpha \qquad \mathfrak{h}'_i = \mathfrak{h}_i^\alpha$$

$$\Sigma'_{i,1} = (\Sigma_{i,1} \cdot \Sigma_{i,0}^{\delta_i})^{\alpha\mu_i} \qquad \Sigma'_{i,0} = \Sigma_{i,0}^{\alpha\mu_i} \qquad \tau'_{i,1} = \tau_{i,1}^{1/\mu_i} \qquad \tau'_{i,2} = \tau_{i,2}^{1/\mu_i}$$

$$\begin{cases} \mathsf{proof}^{(j)} &= \mathsf{NIZK_{DH}\text{-}Proof}((\mathfrak{g}_0, \mathfrak{g}'_0, \prod \mathfrak{f}_i, \prod \mathfrak{f}'_i) \wedge (g, h, \prod a'_i / \prod a_i, \prod b'_i / \prod b_i)) \\ \mathsf{sig}^{(j)} &= \mathsf{SSign}(\mathsf{SK}_j, \mathsf{proof}^{(j)}) \end{cases} \mathcal{S}_j$$

outputs $\mathcal{BB}\mathsf{ox}^{(j)} = (C'_{\Pi_j(i)}, \ell'_{\Pi_j(i)}, \sigma'_{\Pi_j(i)}, \mathsf{vk}'_{\Pi_j(i)}, \Sigma'_{\Pi_j(i)}, \tau'_{\Pi_j(i)})_i, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{j}$

Figure 5.2: Detailed Shuffling of ElGamal Ciphertexts

$\mathsf{MixVerif}(\mathcal{BB}\mathsf{ox}^{(0)}, \mathcal{BB}\mathsf{ox}^{(N)}, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{N})$:

After N mixes, the input of the verifier is:

$$\mathcal{BB}\mathsf{ox}^{(0)} = (\overline{C}_i, \sigma_{i,1}, \mathsf{vk}_i, \Sigma_{i,1}, \tau_{i,1})_{i=1}^{n}$$

$$\mathcal{BB}\mathsf{ox}^{(N)} = (\overline{C}'_i, \sigma'_{i,1}, \mathsf{vk}'_i, \Sigma'_{i,1}, \tau'_{i,1})_{i=1}^{n'}, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})_{k=1}^{N}$$

It outputs 1 if: $n = n'$, the $(\mathsf{vk}_i)_i$ are all distinct
$\forall k$,

$$\begin{aligned} \mathsf{NIZK}_{\mathsf{DH}}\text{-}\mathsf{Verif}(\mathsf{proof}^{(k)}) &= 1 \\ \mathsf{SVerif}(\mathsf{VK}_k, \mathsf{proof}^{(k)}, \mathsf{sig}^{(k)}) &= 1 \end{aligned}$$

and $\forall i$,

$$\begin{aligned} \mathsf{VerifSign}(\mathsf{vk}_i, \overline{C}_i, \sigma_{i,1}) &= 1 = \mathsf{VerifSign}^*(\mathsf{VK}, \tau_i, \mathsf{vk}_i, \Sigma_{i,1}) \\ \mathsf{VerifSign}(\mathsf{vk}'_i, \overline{C}'_i, \sigma'_{i,1}) &= 1 = \mathsf{VerifSign}^*(\mathsf{VK}, \tau'_i, \mathsf{vk}'_i, \Sigma'_{i,1}) \end{aligned}$$

Figure 5.3: Detailed Verification of Shuffling

Proofs. Let us denote $\mathfrak{F} = \prod \mathfrak{f}_i = \mathfrak{g}_0^{\sum u_i}$ and $\mathfrak{F}' = \prod \mathfrak{f}'_i = \mathfrak{g}'_0{}^{\sum u_i}$ the product of the second element of the user's verification key on all the input ballots and output ballots. If the input and output ballot-boxes contain the same ballots (with the same secret u_i), then $\mathfrak{F}' = \mathfrak{F}^{\alpha}$, with $\mathfrak{g}'_0 = \mathfrak{g}_0^{\alpha}$. Hence one adds a proof of Diffie-Hellman tuple for $(\mathfrak{g}_0, \mathfrak{g}'_0, \mathfrak{F}, \mathfrak{F}')$ as described in Preliminaries 2.4.3. Together with the verification that there is the same number of ballots in the input and output of the mix, we will show that the same (honest) users are represented in the two ballot-boxes. Since we cannot allow multiple ballots from the same user, we have the guarantee that the same messages from all the honest users are represented in the two ballot-boxes.

The additional proof of Diffie-Hellman tuple for $(g, h, \prod a'_i / \prod a_i, \prod b'_i / \prod b_i)$ will limit the exchange of ballots for corrupted users, as the products of the plaintexts must remain the same: $\prod M'_i = \prod M_i$. Since we already know these products will be the same for honest users, this products must be the same from corrupted users. This will limit the impact of the attack of Cortier-Smyth [CS13].

With these two Diffie-Hellman proofs, the output ballots are a permutation of the input ones. We could use any non-interactive zero-knowledge proofs of Diffie-Hellman tuples $(\mathsf{NIZK}_{\mathsf{DH}}\text{-}\mathsf{Setup}, \mathsf{NIZK}_{\mathsf{DH}}\text{-}\mathsf{Proof}, \mathsf{NIZK}_{\mathsf{DH}}\text{-}\mathsf{Verif})$ and any signature $(\mathsf{SSetup}, \mathsf{SSign}, \mathsf{SVerif})$ to sign the proofs but the next section will provide interesting choices, from the length point of view.

Verification. The complete verification process, after N mix-servers, is presented in Figure 5.3. After all the mixes are done, it just requires the input ballot-box $\mathcal{BB}\mathsf{ox}^{(0)}$, the output ballot-box $\mathcal{BB}\mathsf{ox}^{(N)}$, and the signed proofs $(\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})$, for $k = 1, \ldots, N$ without the elements that were useful for randomization only. The verifier checks the number of input ballots is the same as the number of output ballots, the verification keys (the $\mathfrak{f}_i$'s) in input ballots are all distinct, the signatures $\sigma_{i,1}, \sigma'_{i,1}, \Sigma_{i,1}$ and $\Sigma'_{i,1}$ are valid on individual input and output tuples (equations recalled in 5.4) and all the proofs $\mathsf{proof}^{(k)}$ with the signatures $\mathsf{sig}^{(k)}$ are valid with $\mathsf{NIZK}_{\mathsf{DH}}\text{-}\mathsf{Verif}$ and SVerif respectively. For that, we suppose that the statement is included in each zero-knowledge proof. Thus, even if the intermediate ballot-boxes are not given to the verifier, it is still possible to perform the verification.

5.3 Scalability

5.3.1 Constant-Size Proof

From Figure 5.3, one can note that our mix-net provides a quite compact proof, as it just requires $\mathcal{BB}\mathsf{ox}^{(0)}$ and $\mathcal{BB}\mathsf{ox}^{(N)}$, and the signed proofs $(\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)})$, for $k = 1, \ldots, N$. The size is thus linear in n and N. This is the same for the verification complexity.

Whereas the linear complexity in n cannot be avoided, as the ballot-box must be transferred, the part linear in N could be avoided. Indeed, each proof $\mathsf{proof}^{(j)}$ ensures the relations from the $j-1$-th ballot-box to the j-th ballot-box. The global chain of proofs ensures the relations from the initial ballot-box to the last ballot-box. From the soundness point on view, a compact global proof would be enough. But for privacy, one wants to be sure that multiple mix-servers contributed, to get unlinkability as soon as one server is honest.

To avoid the dependence in N, one can use Groth-Sahai proofs [GS08] (see 2.4.3 for details) to combine together the proofs into a unique one as already used in Chase *et al.* [CKLM12]. However, to be sure that all the mix-servers contributed: each mix-server does as above, but also receives a partial proof $\mathsf{proof}'^{(j-1)}$ from the initial ballot-box to the $j-1$-th ballot-box and, thanks to the homomorphic properties of the Groth-Sahai proof, updates it into $\mathsf{proof}'^{(j)}$, to prove the relation from the initial ballot-box and the j-th ballot-box, as shown in 2.4.3 for the Diffie-Hellman proof between the products of the keys (the proof is similar for the product of the ciphertexts but with $\mathbb{G}_1$ and $\mathbb{G}_2$ swapped). At the end of the mixing steps, one has the same elements as above, plus the global proof $\mathsf{proof}'^{(N)}$. All the mix-servers can now verify the proofs and the contributions of all the servers. Only this global proof can be kept, but signed by all the servers: using the multi-signature of Boneh-Drijvers-Neven [BDN18], that is recalled in 2.4.2, the size of the signature msig keeps constant, whatever the number of mix-servers. Hence, after multiple mixing steps, the size of the mixing proof (with the input and output ballot-boxes) remains constant.

5.3.2 Efficiency

We consider VK and $(\mathsf{VK}_j)_j$ are long-term keys known to everybody, as well as EK and ℓ. However, for fair comparison, we do not consider vk_i as long-term keys, and consider them as part of the input of the verifier. But we insist that the $\mathfrak{f}_i$'s in the input ballot-box must be all distinct.

Size of Verifier's Input: The verifier receives:

$$(\overline{C}_i, \sigma_{i,1}, \mathsf{vk}_i, \Sigma_{i,1}, \tau_i)_{i=1}^n \quad (\overline{C}'_i, \sigma'_{i,1}, \mathsf{vk}'_i, \Sigma'_{i,1}, \tau'_i)_{i=1}^n \quad (\mathsf{proof}'^{(N)}, \mathsf{msig}'^{(N)})$$

As the first element $\mathfrak{g}_0$ of vk_i is common to all the users (as well as $\mathfrak{g}'_0$ of vk'_i), the set of all the users' verification keys is represented by $4 \times n + 1$ elements of $\mathbb{G}_2$. Then, all input or output ballots contains $2 \times 5n$ elements from $\mathbb{G}_1$ and $2 \times (6n + 1)$ elements from $\mathbb{G}_2$.

The global proof $\mathsf{proof}'^{(N)}$ is just 4 elements of $\mathbb{G}_1$ and 4 elements of $\mathbb{G}_2$ and msig one element in $\mathbb{G}_2$. Hence, the full verifier's input contains: $10n + 4$ elements of $\mathbb{G}_1$, $12n + 6$ elements of $\mathbb{G}_2$, whatever the number of mix-servers.

Verifier's Computation. Using batch verification [CHP07, BFI+10, HHK+17], the verifier only needs to make $8n + 7$ pairing evaluations to verify together all the signatures $\sigma_{i,1}$, $\sigma'_{i,1}$, $\Sigma_{i,1}$, $\Sigma'_{i,1}$, τ_i, τ'_i, 6 pairing evaluations to verify $\mathsf{proof}'^{(N)}$ and 2 pairing evaluations to verify msig.

With some specific choices of the bases for the batch verification, as presented in Figure 5.4, one can improve to $8n + 14$ pairing evaluations for the global verification. This has to be compared to the $4n + 1$ pairing evaluations that have anyway to be performed to verify the signatures in the initial ballot-box.

$\sigma_{i,0}$	1		$= e(\sigma_{i,0}^{-1}, \mathfrak{g}_0)$	$e(g, \mathfrak{g}_i)$	$e(h, \mathfrak{h}_i)$	$e(\ell, \mathfrak{l}_i)$	
$\sigma'_{i,0}$	1		$= e(\sigma'^{-1}_{i,0}, \mathfrak{g}'_0)$	$e(g, \mathfrak{g}'_i)$	$e(h, \mathfrak{h}'_i)$	$e(\ell, \mathfrak{l}'_i)$	
$\Sigma_{i,0}$	$e(\tau_{i,2}, \Sigma_{i,0})$		$= e(g_3, \mathfrak{g}_0)$				
$\Sigma'_{i,0}$	$e(\tau'_{i,2}, \Sigma'_{i,0})$		$= e(g_3, \mathfrak{g}'_0)$				
$\sigma_{i,1}$	$e(\sigma_{i,1}, \mathfrak{g}_0)$		$= e(g, \mathfrak{f}_i)$	$e(a_i, \mathfrak{g}_i)$	$e(b_i, \mathfrak{h}_i)$	$e(\ell_i, \mathfrak{l}_i)$	$3n$ $+2$
$\sigma'_{i,1}$		$e(\sigma'_{i,1}, \mathfrak{g}'_0)$	$= e(g, \mathfrak{f}'_i)$	$e(a'_i, \mathfrak{g}'_i)$	$e(b'_i, \mathfrak{h}'_i)$	$e(\ell'_i, \mathfrak{l}'_i)$	$+3n$ $+1$
τ_i	$e(\tau_{i,2}, \mathfrak{g})$		$= e(g, \tau_{i,1})$				$+0$
τ'_i	$e(\tau'_{i,2}, \mathfrak{g})$		$= e(g, \tau'_{i,1})$				$+0$
$\Sigma_{i,1}$	$e(g_1, \mathfrak{g}_0)$	$e(\tau_{i,2}, \Sigma_{i,1})$	$=$	$e(g_2, \mathfrak{f}_i)$	$e(g_3, \mathfrak{l}_i)$	$e(g_4, \mathfrak{g}_i)\ e(g_5, \mathfrak{h}_i)$	$+n$ $+4$
$\Sigma'_{i,1}$	$e(\tau'_{i,2}, \Sigma'_{i,1})$	$e(g_1^{-1}, \mathfrak{g}'_0)$	$=$	$e(g_2, \mathfrak{f}'_i)$	$e(g_3, \mathfrak{l}'_i)$	$e(g_4, \mathfrak{g}'_i)\ e(g_5, \mathfrak{h}'_i)$	$+n$
msig	$e(\mathcal{H}_0(\mathsf{proof}^{(N)}), \mathsf{avk})$		$= e(g, \mathsf{msig})$				$+1$
$\mathsf{proof}^{(N)}$ with $\mathfrak{F} = \prod_i \mathfrak{f}_i$,$\mathfrak{F}' = \prod_i \mathfrak{f}'_i$, $A = \prod_i a'_i / \prod a_i$ and $B = \prod_i b'_i / \prod b_i$:							
$e(c^{x_{1,1}} d^{x_{1,2}}, \mathfrak{g}^{x_{2,1}} \mathfrak{F}^{x_{2,2}}) = e(v_{2,1}^{x_{1,1}} (v_{2,2} \cdot g)^{x_{1,2}}, \mathfrak{g}'^{x_{2,1}} \mathfrak{F}'^{x_{2,2}}) e(v_{1,1}^{x_{1,1}} v_{1,2}^{x_{1,2}}, \Theta^{x_{2,1}} \Psi^{x_{2,2}})$							$+3$
$e(g^{x'_{2,1}} A^{x'_{2,2}}, \mathfrak{c}^{x'_{1,1}} \mathfrak{d}^{x'_{1,2}}) = e(g'^{x'_{2,1}} B^{x'_{2,2}}, \mathfrak{v}_{2,1}^{x'_{1,1}} (\mathfrak{v}_{2,2} \cdot \mathfrak{g})^{x'_{1,2}}) e(\theta^{x'_{2,1}} \psi^{x'_{2,2}}, \mathfrak{v}_{1,1}^{x'_{1,1}} \mathfrak{v}_{1,2}^{x'_{1,2}})$							$+3$
							$= 8n$ $+14$

Figure 5.4: Verification Cost. One can remark that several pairings have common bases g, g_1, g_2, g_3, g_4, g_5 and $\mathfrak{g}_0 = \mathfrak{g}$, which can be combined together in order to decrease the number of pairings to be computed for the verification.

5.4 Security Analysis

Let us now formally prove the two security properties: the *soundness* means the output ballot-box contains a permutation of randomizations of the input ballot-box and *privacy* means one cannot link an input ciphertext to an output ciphertext, as soon as one mix-server is honest.

We stress that we are in a particular case where users have private signing keys, and ballots are signed. Unfortunately these keys allow to trace the ballots: with $\mathsf{sk}_i = (u_i, v_i, x_i, y_i)$ and $\mathfrak{g}'_0$, one can recover vk'_i, which contradicts privacy for this ballot. They might also allow to exchange some ballots, which contradicts soundness for these ballots. As a consequence, we do not provide any guarantee to corrupted users, whose keys have been given to the adversary (or even possibly generated by the adversary), but we expect honest users to be protected:

- *soundness for honest users* means that all the plaintexts of the honest users in the input ballot-box are in the output ballot-box;
- *privacy for honest users* means that ballots of honest users are unlinkable from the input ballot-box to the output ballot-box.

5.4.1 Proof of Soundness

As just explained, we first study the soundness of our protocol, but for honest users only, in the certified key setting, where all the users must prove the knowledge of their private keys before getting their verification keys vk_i certified by the Certification Authority in Σ_i.

Definition 40 — Soundness for Honest Users

A mix-net M is said *sound for honest users* in the certified key setting, if any PPT adversary $\mathcal{A}$ has a negligible success probability in the following security game:

1. The challenger generates the certification keys $(\mathsf{SK}, \mathsf{VK})$ and the encryption keys $(\mathsf{DK}, \mathsf{EK})$;
2. The adversary $\mathcal{A}$ then
   - decides on the corrupted users $\mathcal{I}^*$ and generates itself their keys $(\mathsf{vk}_i)_{i\in\mathcal{I}^*}$;
   - proves its knowledge of the secrete keys to get the certifications Σ_i on vk_i, for $i \in \mathcal{I}^*$;
   - decides on the set $\mathcal{I}$ of the (honest and corrupted) users that will generate a ballot;
   - generates the ballots $(\mathcal{B}_i)_{i\in\mathcal{I}^*}$ for the corrupted users but provides the messages $(M_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ for the honest users;
3. The challenger generates the keys of the honest users $(\mathsf{sk}_i, \mathsf{vk}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ and their ballots $(\mathcal{B}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$. The initial ballot-box is thus defined by $\mathcal{BB}\mathsf{ox} = (\mathcal{B}_i)_{i\in\mathcal{I}}$;
4. The adversary mixes $\mathcal{BB}\mathsf{ox}$ in a provable way into $(\mathcal{BB}\mathsf{ox}', \mathsf{proof})$.

The adversary wins if $\mathsf{MixVerif}(\mathcal{BB}\mathsf{ox}, \mathcal{BB}\mathsf{ox}', \mathsf{proof}) = 1$ but $\{\mathsf{Decrypt}^*(\mathcal{BB}\mathsf{ox})\} \neq \{\mathsf{Decrypt}^*(\mathcal{BB}\mathsf{ox}')\}$, where $\mathsf{Decrypt}^*$ extracts the plaintexts (using the decryption key DK), but ignores ballots of non-honest users (using the private keys of honest users) and sets of plaintexts can have repetitions.

One can note that this security game does not depend on the mixing steps, but just considers the global mixing, from the input ballot-box $\mathcal{BB}\mathsf{ox}$ to the output ballot-box $\mathcal{BB}\mathsf{ox}'$. The proof proof contains all the elements for proving the honest behavior. In our case, this is just the two Diffie-Hellman proofs.

Theorem 41 (Soundness for Honest Users of Our Mix-Net)**.** *Our mix-net protocol is sound for honest users, in the certified key setting, assuming the unforgeability against Chosen-Message Attacks of the* LH-Sign *and* OT-LH-Sign *signature schemes and the* SEDL *assumption.*

Proof. For proving this theorem, we will assume the verification is successful ($\mathsf{MixVerif}(\mathcal{BB}\mathsf{ox}, \mathcal{BB}\mathsf{ox}', \mathsf{proof}) = 1$) and show that for all the honest ballots, in the input and output ballot-boxes, there is a permutation from the input ones to the outputs ones. And we do it in two steps: first, honest keys vk'_i in the output ballot-box are permuted randomizations of the honest keys vk_i in the input ballot-box; then we prove it for the plaintexts.

Permutation of Honest Keys. We first modify the security game by using the unforgeability against Chosen-Message Attacks of the $\mathsf{LH\text{-}Sign}$ signature scheme: we are given VK, and ask the Tag-oracle and the Signing-oracle to obtain Σ_i on all the verification keys vk_i and vk_0. The rest remains unchanged. Note that because of the proof of knowledge of the private keys sk_i before getting vk_i certified, one can also extract them. Actually, one just needs to extract u_i for all the corrupted users. Then one knows all the legitimate u_i's (for honest and corrupted users).

Under the unforgeability of the signature scheme (Setup^*, Keygen^*, NewTag^*, $\mathsf{RandTag}^*$, $\mathsf{VerifTag}^*$, Sign^*, $\mathsf{MultiplySign}^*$, $\mathsf{VerifSign}^*$), for any output ballot with verification key vk'_j there exists a related legitimate verification key vk_i such that $\mathsf{vk}'_j = \mathsf{vk}_i^{\alpha_i} \times \mathsf{vk}_0^{z_i}$, for some scalars z_i, and α_i.

Since in our construction $\mathsf{vk}_i = (\mathfrak{g}_0, \mathfrak{f}_i, \mathfrak{l}_i, \mathfrak{g}_i, \mathfrak{h}_i)$ and $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0, 1, 1)$, and $\mathsf{vk}'_j = (\mathfrak{g}'_0, \mathfrak{f}'_j, \mathfrak{l}'_j, \mathfrak{g}'_j, \mathfrak{h}'_j)$ and $\mathsf{vk}'_0 = (1, 1, \mathfrak{g}'_0, 1, 1)$ with a common $\mathfrak{g}'_0$ for all the keys, α_i is a common scalar α: $\mathsf{vk}'_j = (\mathsf{vk}_i \times \mathsf{vk}_0^{\delta_i})^{\alpha}$ and $\mathsf{vk}'_0 = \mathsf{vk}_0^{\alpha}$. As a consequence, all the keys in the output ballot-box are derived in a similar way from legitimate keys (signed by the Certification Authority): $u'_j = u_i$ remains unchanged. However this does not means they were all in the input ballot-box: the adversary could insert a ballot with a legitimate verification key vk_i, which was not in the initial ballot-box.

The verification process also includes a Diffie-Hellman proof for the tuple $(\mathfrak{g}_0, \mathfrak{g}'_0, \prod_i \mathfrak{f}_i, \prod_j \mathfrak{f}'_j)$. This means that $\sum_i u_i$ are the same on the input ballots and the output ballots. As one additionally checks the numbers of input ballots and output ballots are the same, the adversary can just replace an input ballot by a new one: if $\mathcal{N}$ is the set of new ballots and $\mathcal{D}$ the set of deleted ballots, the sums must compensate: $\sum_{\mathcal{D}} u_i = \sum_{\mathcal{N}} u_i$.

The second game uses the SEDL assumption and the simulation-soundness of the proof of knowledge of sk_i (in the certified key setting): Let us be given a tuple $(\mathfrak{g}, \mathfrak{f} = \mathfrak{g}^u, g, f = g^u)$, as input of a SEDL challenge in $\mathbb{G}_2$ and $\mathbb{G}_1$: the simulator will guess an honest user i^* that will be deleted, and implicitly sets $u_{i^*} = u$, with $\mathfrak{f}_{i^*}$, which allows it to use $f = g^{u_{i^*}}$ in the signature of $\overline{C}_{i^*}$ on the first component g, while all the other scalars are chosen by the simulator $(v_{i^*}, x_{i^*}, y_{i^*})$, as well as all the other honest user' keys, the authority signing keys, and, for all the corrupted users, the secret element u_i can be extracted at the certification time (using the extractor from the zero-knowledge proof of knowledge) while the zero-knowledge simulator is used for i^*, thanks to the simulation-soundness.

If some honest user is deleted in the output ballot-box, with probability greater than $1/n$, this is i^*: as shown above, $\sum_{\mathcal{D}} u_i = \sum_{\mathcal{N}} u_i$, so $u_{i^*} = \sum_{\mathcal{N}} u_i - \sum_{\mathcal{D}\setminus\{i^*\}} u_i$, which breaks the symmetric external discrete logarithm assumption.

Permutation of Honest Ballots. The last game uses the unforgeability of the OT-LH-Sign signature scheme under Chosen-Message Attacks: the simulator receives one verification key vk, that will be assigned at a random honest user i^*, whereas all the other keys are honestly generated. The simulator also generates $(\mathsf{SK}, \mathsf{VK})$ and $(\mathsf{DK}, \mathsf{EK})$, as well as all signatures Σ_i and the honest ballots (with a signing query for σ_{i^*}). Then, the adversary outputs a proven mix of the ballot-box. We have just proven that there exists a bijection Π from $\mathcal{I}$ into $\mathcal{J}$ such that $\mathsf{vk}'_{\Pi(i)} = (\mathsf{vk}_i \times \mathsf{vk}_0^{\delta_i})^{\alpha}$ for some scalar δ_i, for all the honest users i among the input users in $\mathcal{I}$.

From the signature verification on the output tuples, $C'_{\Pi(i)}$ is signed under $\mathsf{vk}'_{\Pi(i)}$ in $\sigma'_{\Pi(i),1}$, for every i: $e(\sigma'_{\Pi(i),1}, \mathfrak{g}'_0) = e(g, \mathfrak{f}_i^{\alpha}) \cdot e(\ell'_{\Pi(i)}, \mathfrak{l}_i^{\alpha} \mathfrak{g}_0^{\alpha\delta_i}) \cdot e(a'_{\Pi(i)}, \mathfrak{g}_i^{\alpha}) \cdot e(b'_{\Pi(i)}, \mathfrak{h}_i^{\alpha})$, and since the same α appears in $\mathfrak{g}'_0 = \mathfrak{g}_0^{\alpha}$, then for every i, we have

$$\begin{aligned} e(\sigma'_{\Pi(i)}, \mathfrak{g}_0) &= e(g, \mathfrak{f}_i) \cdot e(\ell'_{\Pi(i)}, \mathfrak{l}_i \mathfrak{g}_0^{\delta_i}) \cdot e(a'_{\Pi(i)}, \mathfrak{g}_i) \cdot e(b'_{\Pi(i)}, \mathfrak{h}_i) \\ &= e(g, \mathfrak{f}_i) \cdot e(\ell'_{\Pi(i)}, \mathfrak{l}_i) \cdot e(a'_{\Pi(i)}, \mathfrak{g}_i) \cdot e(b'_{\Pi(i)}, \mathfrak{h}_i) \cdot e(\ell'^{\delta_i}_{\Pi(i)}, \mathfrak{g}_0) \end{aligned}$$

and so $\sigma'_{\Pi(i)} / \ell'^{\delta_i}_{\Pi(i)}$ is a signature of $\overline{C}'_{\Pi(i)} = (g, \ell'_{\Pi(i)}, a'_{\Pi(i)}, b'_{\Pi(i)})$ under vk_i: under the unforgeability assumption of the signature scheme, $C'_{\Pi(i^*)}$ is necessarily a linear combination of the already signed vectors under vk_{i^*}, which are C_{i^*} and C_0, with some coefficients u, v: $a'_{\Pi(i^*)} = a_{i^*}^u g^v$, $b'_{\Pi(i^*)} = b_{i^*}^u h^v$, and $g = g^u 1^v$. Hence, $u = 1$, which means that $C'_{\Pi(i^*)}$ is a randomization of C_{i^*}.

We stress that for this property to hold, each key vk_i must appear at most once in the ballots, otherwise some combinations would be possible. Hence the test that all the $\mathfrak{f}_i$'s are distinct in the input ballot-box. □

We stress that this proposition only guarantees permutation of ciphertexts for honest users. There is indeed no formal guarantee for corrupted users whose signing keys are under the control of a mix-server. The latter could indeed replace the ciphertexts of some corrupted users, by some

other ciphertexts under the same identity or even under the identity of another corrupted user. One can note that replacing ciphertexts (and plaintexts) even for corrupted users is not that easy because of the additional Diffie-Hellman proof on the ciphertexts, which implies $\prod M_i = \prod M'_i$ where the first product is over all the messages M_i in $\mathcal{BB}\mathsf{ox}$ and the second product is over all the messages M'_i in $\mathcal{BB}\mathsf{ox}'$. However, this property is more for the privacy, as we will see below. As a consequence, our result that guarantees a permutation on the honest ballots is optimal. We cannot guarantee anything for the users that share their keys with the mix-servers.

5.4.2 Proof of Privacy: Unlinkability

After proving the soundness, we have to prove the anonymity (a.k.a. unlinkability), which can also be seen as zero-knowledge property. More precisely, as for the soundness, privacy will only be guaranteed for honest users.

Definition 42 — Privacy for Honest Users

A mix-net M is said to provide *privacy for honest users* in the certified key setting, if any PPT adversary $\mathcal{A}$ has a negligible advantage in guessing b in the following security game:

1. The challenger generates the certification keys $(\mathsf{SK}, \mathsf{VK})$ and the encryption keys $(\mathsf{DK}, \mathsf{EK})$;
2. The adversary $\mathcal{A}$ then
   - decides on the corrupted users $\mathcal{I}^*$ and generates itself their keys $(\mathsf{vk}_i)_{i\in\mathcal{I}^*}$;
   - proves its knowledge of the secret keys to get the certifications Σ_i on vk_i, for $i \in \mathcal{I}^*$;
   - decides on the corrupted mix-servers $\mathcal{J}^*$ and generates itself their keys $(\mathsf{VK}_j)_{j\in\mathcal{J}^*}$;
   - decides on the set $\mathcal{J}$ of the (honest and corrupted) mix-servers that will make mixes;
   - decides on the set $\mathcal{I}$ of the (honest and corrupted) users that will generate a ballot;
   - generates the ballots $(\mathcal{B}_i)_{i\in\mathcal{I}^*}$ for the corrupted users but provides the messages $(M_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ for the honest users;
3. The challenger generates the keys of the honest mix-servers $(\mathsf{SK}_j,\mathsf{VK}_j)_{j\in\mathcal{J}\setminus\mathcal{J}^*}$ the keys of the honest users $(\mathsf{sk}_i, \mathsf{vk}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$ and their ballots $(\mathcal{B}_i)_{i\in\mathcal{I}\setminus\mathcal{I}^*}$.

The initial ballot-box is thus defined by $\mathcal{BB}\mathsf{ox} = (\mathcal{B}_i)_{i\in\mathcal{I}}$. The challenger randomly chooses a bit $b \xleftarrow{\$} \{0,1\}$ and then enters into a loop for $j \in \mathcal{J}$ with the attacker:

- let $\mathcal{I}^*_{j-1}$ be the set of indices of the ballots of the corrupted users in the input ballot-box $\mathcal{BB}\mathsf{ox}^{(j-1)}$;
- if $j \in \mathcal{J}^*$, $\mathcal{A}$ builds itself the new ballot-box $\mathcal{BB}\mathsf{ox}^{(j)}$ with the proof $\mathsf{proof}^{(j)}$;
- if $j \notin \mathcal{J}^*$, $\mathcal{A}$ provides two permutations $\Pi_{j,0}$ and $\Pi_{j,1}$ of its choice, with the restriction they must be identical on $\mathcal{I}^*_{j-1}$, then the challenger runs the mixing with $\Pi_{j,b}$, and provides the output $(\mathcal{BB}\mathsf{ox}^{(j)}, \mathsf{proof}^{(j)})$;

In the end, the adversary outputs its guess b' for b. The experiment outputs 1 if $b' = b$ and 0 otherwise.

Contrarily to the soundness security game, the adversary can see the outputs of all the mixing steps to make its decision, hence the index j for the mix-servers. In addition, some can be honest, some can be corrupted. We will assume at least one is honest.

Moreover, the privacy proof of our Mix-Net protocol will depend on a new assumption:

First, let us define some kind of credential as follows for a scalar u and a basis $g \in \mathbb{G}_1$, with $\mathfrak{g} \in \mathbb{G}_2$, $r, t \in \mathbb{Z}_p$:

$$\mathsf{Cred}(u, g; \mathfrak{g}, r, t) = \left(g, g^t, g^r, g^{tr+u}, \mathfrak{g}, \mathfrak{g}^t, \mathfrak{g}^u\right)$$

Definition 43 — Unlinkability Assumption

In groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p, for any $g \in \mathbb{G}_1$ and $\mathfrak{g} \in \mathbb{G}_2$, it states that the distributions $\mathcal{D}_{g,\mathfrak{g}}(u, u)$ and $\mathcal{D}_{g,\mathfrak{g}}(u, v)$ are computationally indistinguishable, for any $u, v \in \mathbb{Z}_p$:

$$\mathcal{D}_{g,\mathfrak{g}}(u, v) = \left\{ (\mathsf{Cred}(u, g; \mathfrak{g}, r, t), \mathsf{Cred}(v, g; \mathfrak{g}', r', t')); \begin{array}{l} \mathfrak{g}' \xleftarrow{\$} \mathbb{G}_2, \\ r, t, r', t' \xleftarrow{\$} \mathbb{Z}_p \end{array} \right\}$$

Intuitively, as we can write the credential as, where $\times$ stands for the element-wise product,

$$\mathsf{Cred}(u, g; \mathfrak{g}, r, t) = \left(\begin{pmatrix} g \\ \mathfrak{g} \end{pmatrix}, \begin{pmatrix} g \\ \mathfrak{g} \end{pmatrix}^t, \begin{pmatrix} g \\ g^t \end{pmatrix}^r \times \begin{pmatrix} 1 \\ g^u \end{pmatrix}, \mathfrak{g}^u \right)$$

the third component is an ElGamal ciphertext of the g^u, which hides it, and makes indistinguishable another encryption g^u from an encryption of g^v while, given $(\mathfrak{g}, \mathfrak{g}^u)$ and $(\mathfrak{g}', \mathfrak{g}'^v)$, one cannot guess whether $u = v$, under the DDH assumption in $\mathbb{G}_2$. However the pairing relation allows to check consistency:

$$e(g^{rt+u}, \mathfrak{g}) = e(g^r, \mathfrak{g}^t) \cdot e(g, \mathfrak{g}^u) = e(g^r, \mathfrak{g}^t) \cdot e(g, \mathfrak{g})^u$$
$$e(g^{r't'+v}, \mathfrak{g}') = e(g^{r'}, \mathfrak{g}'^{t'}) \cdot e(g, \mathfrak{g}'^v) = e(g^{r'}, \mathfrak{g}'^{t'}) \cdot e(g, \mathfrak{g}')^v$$

Because of the independent group elements $\mathfrak{g}$ and $\mathfrak{g}' = \mathfrak{g}^s$ in the two credentials, this assumption clearly holds in the generic bilinear group model, as one would either need to compare $u = v$ or equivalently $rt = r't'$, whereas combinations only lead to $e(g, \mathfrak{g})$ to the relevant powers rt, $sr't'$, as well as u and sv, for an unknown s.

Theorem 44. *Our Mix-Net protocol provides privacy for honest users, in the certified key setting, if (at least) one mix-server is honest, under our unlinkability assumption (see Definition 5.4.2), and the* DDH *assumptions in both* $\mathbb{G}_1$ *and* $\mathbb{G}_2$.

Proof. This proof will follow a series of games $(\mathbf{G}_i)_i$, where we study the advantage Adv_i of the adversary in guessing b. We start from the real security game and conclude with a game where all the ballots are random, independently from the permutations. Hence, the advantage will be trivially 0.

Game $\mathbf{G}_0$: This is the real game, where the challenger (our simulator) generates SK and VK for the certification authority signature, and randomly chooses $d \xleftarrow{\$} \mathbb{Z}_p$ to generate the encryption public key $\mathsf{EK} = h = g^d$. One also sets $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0 = \mathfrak{g}^A, 1, 1)$ and $C_0 = \mathsf{Encrypt}_{\mathsf{EK}}(1) = (g, h)$ expanded into $\overline{C}_0 = (1, \ell, C_0)$ with the noise parameter $\ell \xleftarrow{\$} \mathbb{G}_1$. Actually, $A = 1$ in the initial step, when the user encrypts his message M_i, but since the shuffling may happens after several other shuffling iterations, we have the successive exponentiations to multiple α (in A) for vk_0. The attacker $\mathcal{A}$ chooses the set of the initial indices of the corrupted users $\mathcal{I}^*$ and the set of the initial indices of the corrupted

mix-servers $\mathcal{J}^*$, provides their verification keys $((\mathsf{vk}_i)_{i\in\mathcal{I}^*}, (\mathsf{VK}_j)_{j\in\mathcal{J}^*})$ together with an extractable zero-knowledge proof of knowledge of sk_i.

From $\mathcal{I}$ and $\mathcal{J}$, one generates the signing keys for the honest mix-servers $j \in \mathcal{J}\backslash\mathcal{J}^*$, and set J to the index of the last honest mix-server. For each $i \in \mathcal{I}$, one chooses $\tau_i = R_i \xleftarrow{\$} \mathbb{Z}_p$ and sets $\tau_i = (\tau_{i,1} = \mathfrak{g}^{1/R_i}, \tau_{i,2} = g^{1/R_i})$. For each honest user $i \in \mathcal{I}\backslash\mathcal{I}^*$, one randomly chooses $u_i, v_i, x_i, y_i, r_i, \rho_i \xleftarrow{\$} \mathbb{Z}_p$ to generate $\mathsf{vk}_i = (\mathfrak{g}_0 = \mathfrak{g}, \mathfrak{f}_i = \mathfrak{g}_0^{u_i}, \mathfrak{l}_i = \mathfrak{g}_0^{v_i}, \mathfrak{g}_i = \mathfrak{g}_0^{x_i}, \mathfrak{h}_i = \mathfrak{g}_0^{y_i})$, and eventually generates all the signatures Σ_i of $(\mathsf{vk}_i, \mathsf{vk}_0)$ under SK with respect to the tag τ_i (using SK and $(\tilde{\tau}_i)_i$).

For the corrupted users, the simulator directly receives the set of ballots $(\mathcal{B}_i = (\overline{C}_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i))_{i\in\mathcal{I}^*}$ while for the honest users, it receives $(M_i)_{i\in\mathcal{I}\backslash\mathcal{I}^*}$ and computes

$$C_i = \mathsf{Encrypt}_{\mathsf{EK}}(M_i) = (a_i = g^{r_i}, b_i = h^{r_i} M_i) \qquad \overline{C}_i = (g, \ell_i = \ell^{\rho_i}, C_i)$$

and the signature σ_i of $(\overline{C}_i, \overline{C}_0)$ under sk_i. The input ballot-box is then $\mathcal{BB}\mathsf{ox}^{(0)} = \{(\mathcal{B}_i)_{i\in\mathcal{I}}\}$ including the ballots of the honest and corrupted users. Let $\mathcal{I}_0^* = \mathcal{I}^*$ be the set of the initial indices of the corrupted users.

The simulator randomly chooses $b \xleftarrow{\$} \{0,1\}$ and now begins the loop of the mixes: depending if the mix-server j is corrupted or not, the simulator directly receives $(\mathcal{BB}\mathsf{ox}^{(j)}, \mathsf{proof}^{(j)})$ from the adversary or receives $(\Pi_{j,0}, \Pi_{j,1})$. In the latter case, one first checks if $\Pi_{j,0}|_{\mathcal{I}_{j-1}^*} = \Pi_{j,1}|_{\mathcal{I}_{j-1}^*}$ using the honest secret keys to determine $\mathcal{I}_{j-1}^*$. Then, the simulator randomly chooses global $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and individual $\gamma_i, \delta_i, \mu_i \xleftarrow{\$} \mathbb{Z}_p$ for all the users, as an honest mix-server would do, to compute

$$\begin{aligned}
\mathsf{vk}_i' &= (\mathfrak{g}_0' = \mathfrak{g}_0^\alpha, \mathfrak{f}_i' = \mathfrak{f}_i^\alpha, \mathfrak{l}_i' = (\mathfrak{l}_i \cdot \mathfrak{g}_0^{\delta_i})^\alpha, \mathfrak{g}_i' = \mathfrak{g}_i^\alpha, \mathfrak{h}_i' = \mathfrak{h}_i^\alpha) = (\mathsf{vk}_i \cdot \mathsf{vk}_0^{\delta_i})^\alpha \\
\mathsf{vk}_0' &= (1, 1, \mathfrak{g}_0', 1, 1) = \mathsf{vk}_0^\alpha \\
\overline{C}_i' &= (g, \ell_i' = \ell_i \cdot \ell_0^{\gamma_i}, a_i' = a_i \cdot g_0^{\gamma_i}, b_i' = b_i \cdot h_0^{\gamma_i}) = \overline{C}_i \cdot \overline{C}_0{}^{\gamma_i} \\
\sigma_i' &= (\sigma_{i,0}' = \sigma_{i,0} \cdot \ell_0'^{\delta_i}, \sigma_{i,1}' = \sigma_{i,1} \cdot \sigma_{i,0}^{\gamma_i} \cdot \ell_i'^{\delta_i}) \\
\Sigma_i' &= (\Sigma_{i,0}' = \Sigma_{i,0}^{\alpha\mu_i}, \Sigma_{i,1}' = (\Sigma_{i,1} \cdot \Sigma_{i,0}^{\delta_i})^{\alpha\mu_i}) \\
\tau_i' &= (\tau_{i,1}' = \tau_{i,1}^{1/\mu_i}, \tau_{i,2}' = \tau_{i,2}^{1/\mu_i})
\end{aligned}$$

and sets $\mathcal{BB}\mathsf{ox}^{(j)} = (\mathcal{B}'_{\Pi_{j,b}(i)})_i$. Eventually, the simulator computes the proof $\mathsf{proof}^{(j)}$ for $(\mathfrak{g}_0, \mathfrak{g}_0', \prod \mathfrak{f}_i, \prod \mathfrak{f}_i')$ and $(g, h, \prod a_i'/\prod a_i, \prod b_i'/\prod b_i)$, and signs it using SK_j.

After the full loop on all the mix-servers, the adversary outputs its guess b': $\mathsf{Adv}_{\mathbf{G}_0} = \Pr_{\mathbf{G}_0}[b' = b]$. One important remark is that under the previous soundness result, which has exactly the same setup, the input ballot-box for the last honest mix-server necessarily contains a randomization of the initial honest ballots (the adversary against the soundness is the above adversary together with the honest simulator up to its last honest round, that does not need any secret). Only the behavior of this last honest mix-server will be modified below.

Game $\mathbf{G}_1$: We first switch the Diffie-Hellman proofs for $(\mathfrak{g}_0, \mathfrak{g}_0', \prod \mathfrak{f}_i, \prod \mathfrak{f}_i')$ to the zero-knowledge setting: if the input ballot-box for the last honest mix-server is not a randomization of the initial honest ballots, that can be tested using the decryption key, one has built a distinguisher between the settings of the zero-knowledge proofs. In this new setting, one can use the zero-knowledge simulator that does not use α. Under the zero-knowledge property, $\mathsf{Adv}_{\mathbf{G}_0} < \mathsf{Adv}_{\mathbf{G}_1} + \mathsf{negl}()$.

Game $\mathbf{G}_2$: We also switch the proofs for $(g, h, \prod a_i'/\prod a_i, \prod b_i'/\prod b_i)$ to the zero-knowledge setting: as above, the distance remains negligible. In this new setting, one can use the

zero-knowledge simulator that does not use $\sum_i \gamma_i$. Under the zero-knowledge property, $\mathsf{Adv}_{\mathbf{G}_1} < \mathsf{Adv}_{\mathbf{G}_2} + \mathsf{negl}()$.

Game $\mathbf{G}_3$: In this game, we do not know anymore the decryption key, and use the indistinguishability of the encryption scheme (which relies on the Decisional Diffie-Hellman assumption): in an hybrid way, we replace the ciphertexts C_i of the honest users by an encryption of 1: $C_i = \mathsf{Encrypt}_{\mathsf{EK}}(1)$. Under the DDH assumption in $\mathbb{G}_1$, $\mathsf{Adv}_{\mathbf{G}_2} < \mathsf{Adv}_{\mathbf{G}_3} + \mathsf{negl}()$.

Game $\mathbf{G}_4$: This corresponds to $C_i = (a_i = g^{r_i}, b_i = h^{r_i})$. But now we can know d, but ℓ is random: under the DDH assumption, we can replace the random value $\ell_i = \ell^{\rho_i}$ by $\ell_i = \ell^{r_i}$. Ultimately, we set $\overline{C}_i = (g, \ell_i = \ell^{r_i}, a_i = g^{r_i}, b_i = h^{r_i})$ for $r_i \xleftarrow{\$} \mathbb{Z}_p$, for all the honest users, in the initial ballot-box. Under the DDH assumption in $\mathbb{G}_1$, $\mathsf{Adv}_{\mathbf{G}_3} < \mathsf{Adv}_{\mathbf{G}_4} + \mathsf{negl}()$.

Game $\mathbf{G}_5$: In this game, one can first extract the keys of the corrupted users during the certification phase. Then, all the honest mix-servers generate random signing keys sk'_i, random tags τ'_i, and random encryptions C'_i of 1, for all the honest users (the one who do not correspond to the extracted keys), and generate the signatures using the signing keys SK and sk'_i, but still behave honestly for the ballots of the corrupted users. Then, they apply the permutations $\Pi_{j,b}$ on the randomized ballots.

Lemma 45 (Random Ballots for Honest Users)**.** *Under the Unlinkability Assumption (see Definition 5.4.2) and* DDH *assumption in $\mathbb{G}_2$, the view is computationally indistinguishable:* $\mathsf{Adv}_{\boldsymbol{G}_4} < \mathsf{Adv}_{\boldsymbol{G}_5} + \mathsf{negl}()$.

In this last game, the i-th honest user is simulated with initial and output (after each honest mix-server) ciphertexts that are random encryptions of 1, and initial and output signing keys (and thus verification keys vk_i and vk'_i) independently random. As a consequence, permutations $\Pi_{j,b}$ are applied on random ballots, which is perfectly indistinguishable from applying $\Pi_{j,1-b}$ (as we have restricted the two permutations to be identical on ballots of corrupted users): $\mathsf{Adv}_{\mathbf{G}_5} = 0$. Which leads to $\mathsf{Adv}_0 \leq \mathsf{negl}()$. □

Proof of Lemma 45. In the above sequences of games, from $\mathbf{G}_0$ to $\mathbf{G}_4$, we could have checked whether the honest vk_i's in the successive ballot-boxes are permutations of randomized honest initial keys, just using the secret keys of the honest users. So, we can assume in the next hybrid games, from $\mathbf{G}_0(j)$ to $\mathbf{G}_8(j)$, for $j = N, \ldots, 1$ that the input ballots in $\mathcal{BB}\mathsf{ox}^{(j-1)}$ contain proper permutations of randomized honest initial keys, as nothing is modified before the generation of this ballot-box. In the following series of hybrid games, for index j, the honest mix-servers up to the $j-1$-th round play as in $\mathbf{G}_4$ and from the $j+1$-th round, they play as in $\mathbf{G}_5$. Only the behavior of the j-th mix-server is modified: starting from an honest behavior. Hence, $\mathbf{G}_0(N) = \mathbf{G}_4$.

Game $\mathbf{G}_0(j)$: In this hybrid game, we assume that the initial ballot-box has been correctly generated (with $\overline{C}_i = (g, \ell_i = \ell^{r_i}, a_i = g^{r_i}, b_i = h^{r_i})$ for $r_i \xleftarrow{\$} \mathbb{Z}_p$, for all the honest users), and mixing steps up to $\mathcal{BB}\mathsf{ox}^{(j)}$ have been honestly generated (excepted the zero-knowledge proofs that have been simulated). The next rounds are generated at random by honest mix-servers: random signing keys sk'_i and random ciphertexts $\overline{C}'_i = (g, \ell'_i = \ell^{r'_i}, a'_i = g^{r'_i}, b'_i = h^{r'_i})$, with random r'_i, and then correct signatures, using SK and sk'_i. The following sequence of games will modify the randomization of $\mathcal{BB}\mathsf{ox}^{(j-1)}$ into $\mathcal{BB}\mathsf{ox}^{(j)}$ if the j-th mix-server is honest.

Game $\mathbf{G}_1(j)$: We now start modifying the randomization of the ballots by the j-th mix-server, for the corrupted users. As we assumed the signatures Σ_i provided by the certification

authority from a proof of knowledge of sk_i, our simulator has access to $\mathsf{sk}_i = (u_i, v_i, x_i, z_i)$ for all the corrupted users. The mixing step consists in updating the ciphertexts, the keys and the signatures, and we show how to do it without using α such that $\mathfrak{g}'_0 = \mathfrak{g}_0^{\alpha}$ but, instead, just $\mathfrak{g}'_0$, sk_i, $\overline{C}_0 = (1, \ell, g, h)$ and the individual random coins γ_i, δ_i: from $\mathcal{B}_i$ a received ballot of a corrupted user, one can compute $\mathsf{vk}'_i = (\mathfrak{g}'_0, {\mathfrak{g}'_0}^{u_i}, {\mathfrak{g}'_0}^{v_i+\delta_i}, {\mathfrak{g}'_0}^{x_i}, {\mathfrak{g}'_0}^{y_i})$ and $\overline{C}'_i = \overline{C}_i \cdot \overline{C}_0^{\gamma_i}$, and then the signatures σ'_i and Σ'_i using the signing keys, and choosing $\tilde{\tau}'_i \xleftarrow{\$} \mathbb{Z}_p$. This simulation is perfect for the corrupted users: $\mathsf{Adv}_{\mathbf{G}_1(j)} = \mathsf{Adv}_{\mathbf{G}_0(j)}$.

Game $\mathbf{G}_2(j)$: We now modify the simulation of the honest ballots. In this game, we choose random $d, e \xleftarrow{\$} \mathbb{Z}_p$ for $h = g^d$ and $\ell = g^e$. Then we have simulated $\overline{C}_i = (g, \ell_i = \ell^{r_i}, a_i = g^{r_i}, b_i = h^{r_i})$ the ciphertext in $\mathcal{BB}\mathsf{ox}^{(0)}$ and we can set $\overline{C}'_i = (g, \ell'_i = \ell^{r'_i}, a'_i = g^{r'_i}, b'_i = h^{r'_i})$ the ciphertext in $\mathcal{BB}\mathsf{ox}^{(j)}$ for known random scalars $r_i, r'_i \xleftarrow{\$} \mathbb{Z}_p$, where r'_i is actually $r_i + \gamma_i$: γ_i is the accumulation of all the noises. All the signatures are still simulated using the signing keys (and $\tilde{\tau}'_i = R'_i \xleftarrow{\$} \mathbb{Z}_p$), with $\mathfrak{g}'_0 = \mathfrak{g}_0^{\alpha}$ for a random scalar α. This simulation is perfectly the same as above: $\mathsf{Adv}_{\mathbf{G}_2(j)} = \mathsf{Adv}_{\mathbf{G}_1(j)}$.

Before continuing, we study the format of the initial and randomized ballots: by denoting σ_i the initial signature in $\mathcal{BB}\mathsf{ox}^{(0)}$ and σ'_i the signature to generate in $\mathcal{BB}\mathsf{ox}^{(j)}$, we have the following relations:

$$e(\sigma_{i,0}, \mathfrak{g}_0) = e(g, \mathfrak{g}_i \mathfrak{h}_i{}^d \mathfrak{l}_i{}^e) \qquad e(\sigma_{i,1}, \mathfrak{g}_0) = e(g, \mathfrak{f}_i(\mathfrak{g}_i \mathfrak{h}_i{}^d \mathfrak{l}_i{}^e)^{r_i})$$
$$e(\sigma'_{i,0}, \mathfrak{g}'_0) = e(g, \mathfrak{g}'_i \mathfrak{h}'_i{}^d \mathfrak{l}'_i{}^e) \qquad e(\sigma'_{i,1}, \mathfrak{g}'_0) = e(g, \mathfrak{f}'_i(\mathfrak{g}'_i \mathfrak{h}'_i{}^d \mathfrak{l}'_i{}^e)^{r'_i})$$

If we formally denote $\sigma_{i,0} = g^{t_i}$ and $\sigma_{i,1} = g^{s_i}$, then we have

$$\mathfrak{g}_0{}^{t_i} = \mathfrak{g}_i \mathfrak{h}_i{}^d \mathfrak{l}_i{}^e \text{ and } \mathfrak{g}_0{}^{s_i} = \mathfrak{f}_i(\mathfrak{g}_i \mathfrak{h}_i{}^d \mathfrak{l}_i{}^e)^{r_i} = \mathfrak{f}_i \mathfrak{g}_0{}^{t_i r_i}$$

which implies $s_i = u_i + t_i r_i$. Similarly, if we formally denote $\sigma'_{i,0} = g'^{t'_i}$ and $\sigma'_{i,1} = g^{s'_i}$, and set α as the product of all the α's and δ_i as aggregation of all the δ_i's (with α's) in the previous rounds plus this round, from

$$\mathfrak{g}_0{}^{\alpha t'_i} = {\mathfrak{g}'_0}^{t'_i} = \mathfrak{g}'_i \mathfrak{h}'_i{}^d \mathfrak{l}'_i{}^e = \mathfrak{g}_i{}^{\alpha} \mathfrak{h}_i{}^{\alpha d} (\mathfrak{l}_i \mathfrak{g}_0^{\delta_i})^{\alpha e}$$
$$\mathfrak{g}_0{}^{\alpha s'_i} = {\mathfrak{g}'_0}^{s'_i} = \mathfrak{f}'_i(\mathfrak{g}'_i \mathfrak{h}'_i{}^d \mathfrak{l}'_i{}^e)^{r'_i} = \mathfrak{f}_i^{\alpha}(\mathfrak{g}_i^{\alpha} \mathfrak{h}_i^{\alpha d} (\mathfrak{l}_i \mathfrak{g}_0^{\delta_i})^{\alpha e})^{r'_i}$$

we also have $\mathfrak{g}_0{}^{t'_i} = (\mathfrak{g}_i \mathfrak{h}_i{}^d \mathfrak{l}_i^e) \mathfrak{g}_0^{\delta_i e}$ and $\mathfrak{g}_0{}^{s'_i} = \mathfrak{f}_i(\mathfrak{g}_i \mathfrak{h}_i^d \mathfrak{l}_i^e)^{r'_i} \mathfrak{g}_0^{e\delta_i r'_i}$ which implies $s'_i = u_i + t'_i r'_i$. As consequence:

$$\sigma_{i,1} = g^{u_i} \cdot (g^{r_i})^{t_i} = g^{u_i} \cdot a_i{}^{t_i} \text{ and } \sigma'_{i,1} = g^{u_i} \cdot (g^{r'_i})^{t'_i} = g^{u_i} \cdot a'_i{}^{t'_i}$$

Game $\mathbf{G}_3(j)$: Let us randomly choose scalars $u_i, r_i, r'_i, t_i, t'_i$ and α, then, from $(g, \mathfrak{g}_0)$, we can set $\mathfrak{g}'_0 \leftarrow \mathfrak{g}_0^{\alpha}$, $a_i \leftarrow g^{r_i}$, $\sigma_{i,1} \leftarrow a_i^{t_i} g^{u_i}$, $\mathfrak{f}_i \leftarrow \mathfrak{g}_0^{u_i}$, as well as $a'_i \leftarrow g^{r'_i}$, $\sigma'_{i,1} \leftarrow a'_i{}^{t'_i} g^{u_i}$, $\mathfrak{f}'_i \leftarrow {\mathfrak{g}'_0}^{u_i}$.

Then, one additionally chooses $x_i, y_i \xleftarrow{\$} \mathbb{Z}_p$ and sets

$$\mathfrak{g}_i \leftarrow \mathfrak{g}_0^{x_i} \quad \mathfrak{h}_i \leftarrow \mathfrak{g}_0^{y_i} \qquad \mathfrak{l}_i \leftarrow (\mathfrak{g}_0^{t_i}/(\mathfrak{g}_i \mathfrak{h}_i^d))^{1/e} \qquad \overline{C}_i \leftarrow (g, a_i^e, a_i, a_i^d)$$
$$\mathfrak{g}'_i \leftarrow {\mathfrak{g}'_0}^{x_i} \quad \mathfrak{h}'_i \leftarrow {\mathfrak{g}'_0}^{y_i} \qquad \mathfrak{l}'_i \leftarrow ({\mathfrak{g}'_0}^{t'_i}/(\mathfrak{g}'_i \mathfrak{h}'_i{}^d))^{1/e} \qquad \overline{C}'_i \leftarrow (g, a'_i{}^e, a'_i, a'_i{}^d)$$

By construction: $\mathfrak{g}_0^{t_i} = \mathfrak{g}_i \mathfrak{h}_i^d \mathfrak{l}_i^e$, ${\mathfrak{g}'_0}^{t'_i} = \mathfrak{g}'_i \mathfrak{h}'_i{}^d \mathfrak{l}'_i{}^e$, and

$$\sigma_{i,1} = a_i^{t_i} g^{u_i} = g^{t_i r_i} \times g^{u_i} \qquad \sigma'_{i,1} = a'_i{}^{t'_i} g^{u_i} = g^{t'_i r'_i} \times g^{u_i}$$

With $\sigma_{i,0} \leftarrow g^{t_i}$ and $\sigma'_{i,0} \leftarrow g^{t'_i}$, σ_i and σ'_i are valid signatures of $(\overline{C}_i, \overline{C}_0)$ and $(\overline{C}'_i, \overline{C}_0)$ respectively. Then, the verification keys $\mathsf{vk}_i = (\mathfrak{g}_0, \mathfrak{f}_i, \mathfrak{l}_i, \mathfrak{g}_i, \mathfrak{h}_i)$ and $\mathsf{vk}'_i = (\mathfrak{g}'_0, \mathfrak{f}'_i, \mathfrak{l}'_i, \mathfrak{g}'_i, \mathfrak{h}'_i)$ are

correctly related for the secret keys (u_i, v_i, x_i, y_i). From $\mathfrak{l}_i = (\mathfrak{g}_0^{t_i}/(\mathfrak{g}_i\mathfrak{h}_i^d))^{1/e} = \mathfrak{g}_0^{(t_i - x_i - dy_i)/e}$: we have $v_i = (t_i - x_i - dy_i)/e$. From $\mathfrak{l}'_i = (\mathfrak{g}_0'^{t'_i}/(\mathfrak{g}'_i\mathfrak{h}_i'^d))^{1/e} = \mathfrak{g}_0'^{(t'_i - x_i - dy_i)/e}$: we have $v'_i = (t'_i - x_i - dy_i)/e = (t'_i - t_i)/e + v_i$, which means that $\delta_i = (t'_i - t_i)/e$.

Using the signing key SK, we can complete and sign vk_i (with random R_i) and vk'_i (with random R'_i, which implicitly defines μ_i). As shown above, this perfectly simulates the view of the adversary for the honest ballots in the initial ballot-box $\mathcal{BB}\mathsf{ox}^{(0)}$, with $\mathcal{B}_i = (\overline{C}_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i)$ and a randomized version in the updated ballot-box $\mathcal{BB}\mathsf{ox}^{(j)}$, with $\mathcal{B}'_i = (\overline{C}'_i, \sigma'_i, \mathsf{vk}'_i, \Sigma'_i, \tau'_i)$: $\mathsf{Adv}_{\mathbf{G}_3(j)} = \mathsf{Adv}_{\mathbf{G}_2(j)}$.

Game $\mathbf{G}_4(j)$: Let us be given $\mathsf{Cred}(u_i, g; \mathfrak{g}_0, r_i, t_i)$ and $\mathsf{Cred}(u_i, g; \mathfrak{g}'_0, r'_i, t'_i)$, for random $u_i \xleftarrow{\$} \mathbb{Z}_p$, which provide all the required inputs from the first part of the simulation in the previous game (before choosing x_i, y_i). They all follow the distribution $\mathcal{D}_{g,\mathfrak{g}_0}(u_i, u_i)$. As we do not need to know α to randomize ballots for corrupted users, we can thus continue the simulation as above, in a perfectly indistinguishable way: $\mathsf{Adv}_{\mathbf{G}_4(j)} = \mathsf{Adv}_{\mathbf{G}_3(j)}$.

Game $\mathbf{G}_5(j)$: Let us be given two credentials of u_i and u'_i, $\mathsf{Cred}(u_i, g; \mathfrak{g}_0, r_i, t_i)$ and $\mathsf{Cred}(u'_i, g; \mathfrak{g}'_0, r'_i, t'_i)$, for random $u_i, u'_i \xleftarrow{\$} \mathbb{Z}_p$. Inputs follow the distribution $\mathcal{D}_{g,\mathfrak{g}_0}(u_i, u'_i)$ and we do as above. Under the Unlinkability Assumption (see Definition 5.4.2) the view is computationally indistinguishable: $\mathsf{Adv}_{\mathbf{G}_4(j)} < \mathsf{Adv}_{\mathbf{G}_5(j)} + \mathsf{negl}()$.

Game $\mathbf{G}_6(j)$: We receive a Multi Diffie-Hellman tuple $(\mathfrak{g}_0, \mathfrak{g}_i, \mathfrak{h}_i, \mathfrak{g}'_0, \mathfrak{g}'_i, \mathfrak{h}'_i) \xleftarrow{\$} \mathcal{D}^6_{\mathsf{mdh}}(\mathfrak{g}_0)$. So we know all the scalars, except x_i, y_i and α, which are implicitly defined by the input challenge. Then, by choosing $t_i, t'_i \xleftarrow{\$} \mathbb{Z}_p$, we can define $\mathfrak{l}_i, \mathfrak{l}'_i$ as in the previous game, and the ciphertexts and signatures are generated honestly with random scalars $r_i, r'_i \xleftarrow{\$} \mathbb{Z}_p$: $\mathsf{Adv}_{\mathbf{G}_6(j)} = \mathsf{Adv}_{\mathbf{G}_5(j)}$.

Game $\mathbf{G}_7(j)$: We now receive $(\mathfrak{g}_0, \mathfrak{g}_i, \mathfrak{h}_i, \mathfrak{g}'_0, \mathfrak{g}'_i, \mathfrak{h}'_i) \xleftarrow{\$} \mathcal{D}^6_{\$}(\mathfrak{g}_0)$. We do the simulation as above. The view of the adversary is indistinguishable under the DDH assumption in $\mathbb{G}_2$: $\mathsf{Adv}_{\mathbf{G}_6(j)} < \mathsf{Adv}_{\mathbf{G}_7(j)} + \mathsf{negl}()$.

In this game, $\mathsf{vk}'_i = (\mathfrak{g}'_0, \mathfrak{f}_i = \mathfrak{g}_0'^{u'_i}, \mathfrak{l}_i = \mathfrak{g}_0'^{v'_i}, \mathfrak{g}_i = \mathfrak{g}_0'^{x'_i}, \mathfrak{h}_i = \mathfrak{g}_0'^{y'_i})$, with $x'_i, y'_i \xleftarrow{\$} \mathbb{Z}_p$ because of the random tuple, $v'_i = v_i + (t'_i - t_i)/e$, for random t'_i and t_i, it is thus also random, and u'_i is chosen at random.

Game $\mathbf{G}_8(j)$: We now choose at random the signing keys $\mathsf{sk}_i = (u_i, v_i, x_i, y_i)$ and $\mathsf{sk}'_i = (u'_i, v'_i, x'_i, y'_i)$ in order to sign the ciphertexts: $\mathsf{Adv}_{\mathbf{G}_8(j)} = \mathsf{Adv}_{\mathbf{G}_7(j)}$.

With this last game, one can see that $\mathbf{G}_8(1) = \mathbf{G}_5$. Furthermore, for each round $j = N, \ldots, 1$, we have $\mathsf{Adv}_{\mathbf{G}_0(j)} \leq \mathsf{Adv}_{\mathbf{G}_8(j)} + \mathsf{negl}()$, while $\mathbf{G}_0(j-1) = \mathbf{G}_8(j)$: $\mathsf{Adv}_{\mathbf{G}_4} = \mathsf{Adv}_{\mathbf{G}_0}(N) \leq \mathsf{Adv}_{\mathbf{G}_8}(1) + \mathsf{negl}() = \mathsf{Adv}_{\mathbf{G}_5} + \mathsf{negl}()$.

5.4.3 Proof of Correctness

For the reader convenience, we also show the correctness of our mix-net: if the input ballot box is correct and the mix-servers follow the protocol then the verifier outputs 1.

If the initial ballot-box $\mathcal{BB}\mathsf{ox}^{(0)} = (\overline{C}_i, \sigma_{i,1}, \mathsf{vk}_{i,1}, \Sigma_{i,1}, \tau_{i,1})_{i=1}^n$ is correct, then

$$\mathsf{VerifSign}(\mathsf{vk}_i, \overline{C}_i, \sigma_{i,1}) = 1 \quad \text{and} \quad \mathsf{VerifSign}^*(\mathsf{VK}, \tau_i, \mathsf{vk}_i, \Sigma_{i,1}) = 1.$$

The final ballot-box is $\mathcal{BB}\mathsf{ox}^{(N)} = (\overline{C}'_i, \sigma'_{i,1}, \mathsf{vk}'_{i,1}, \Sigma'_{i,1}, \tau'_{i,1})_{i=1}^{n'}$ and the proof of each mix-servers are $(\mathsf{proof}^{(k)}, \sigma^{(k)})_{k=1}^N$. If all the mix-servers follow the protocol then $n = n'$ and for all k, $\mathsf{NIZK}_{\mathsf{DH}}\text{-}\mathsf{Verif}(\mathsf{proof}^{(k)}) = 1$ and $\mathsf{SVerif}(\mathsf{VK}_j, \mathsf{proof}^{(k)}, \sigma^{(k)}) = 1$. Let α_k be the witness of the proof $\mathsf{proof}^{(k)}$ and $\alpha = \prod_{k=1}^N \alpha_k$. One needs to verify if $\mathsf{VerifSign}(\mathsf{vk}'_i, \overline{C}'_i, \sigma'_{i,1}) = 1$ and

$\mathsf{VerifSign}^*(\mathsf{VK}, \tau'_i, \mathsf{vk}'_i, \Sigma'_{i,1}) = 1$:
First one can remark that $\mathsf{VerifSign}(\mathsf{vk}'_i, \overline{C}'_0, \sigma'_{i,0}) = 1$ because

$$\begin{aligned}
e(\sigma'_{i,0}, \mathfrak{g}'_0) &= e(\sigma_{i,0} \cdot \ell^{\delta_i}, \mathfrak{g}_0)^{\alpha} = e(\sigma_{i,0}, \mathfrak{g}_0)^{\alpha} \cdot e(\ell^{\delta_i}, \mathfrak{g}_0)^{\alpha} \\
&= e(1, \mathfrak{f}_i)^{\alpha} e(\ell, \mathfrak{l}_i)^{\alpha} e(g, \mathfrak{g}_i)^{\alpha} e(h, \mathfrak{h}_i)^{\alpha} \cdot e(\ell^{\delta_i}, \mathfrak{g}_0)^{\alpha} \\
&= e(1, \mathfrak{f}'_i) e(g, \mathfrak{g}'_i) e(h, \mathfrak{h}'_i) \cdot e(\ell, \mathfrak{l}_i)^{\alpha} e(\ell, \mathfrak{g}_0^{\delta_i})^{\alpha} \\
&= e(1, \mathfrak{f}'_i) e(g, \mathfrak{g}'_i) e(h, \mathfrak{h}'_i) \cdot e(\ell, \mathfrak{l}_i^{\alpha} \cdot \mathfrak{g}_0^{\delta_i \alpha}) \\
&= e(1, \mathfrak{f}'_i) e(g, \mathfrak{g}'_i) e(h, \mathfrak{h}'_i) e(\ell, \mathfrak{l}'_i)
\end{aligned}$$

Now, one can check $\mathsf{VerifSign}(\mathsf{vk}'_i, \overline{C}'_i, \sigma'_{i,1}) = 1$ with the help of the previous computation:

$$\begin{aligned}
e(\sigma'_{i,1}, \mathfrak{g}'_0) &= e(\sigma'_{i,1}, \mathfrak{g}_0^{\alpha}) = e(\sigma_{i,1} \cdot \sigma_{i,0}^{\gamma_i} \cdot {\ell'_i}^{\delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(\sigma_{i,1}, \mathfrak{g}_0)^{\alpha} \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \cdot e({\ell'_i}^{\delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}_i)^{\alpha} e(\ell_i, \mathfrak{l}_i)^{\alpha} e(a_i, \mathfrak{g}_i)^{\alpha} e(b_i, \mathfrak{h}_i)^{\alpha} \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \cdot e({\ell'_i}^{\delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}_i^{\alpha}) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \cdot e({\ell'_i}^{\delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \cdot e(\ell_i, \mathfrak{l}_i^{\alpha}) e(\ell_i^{\delta_i} \cdot \ell^{\gamma_i \delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \cdot e(\ell_i, \mathfrak{l}_i^{\alpha}) e(\ell_i^{\delta_i}, \mathfrak{g}_0^{\alpha}) e(\ell^{\gamma_i \delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \cdot e(\ell_i, \mathfrak{l}_i^{\alpha} \cdot \mathfrak{g}_0^{\alpha \delta_i}) e(\ell_i \cdot \ell^{\gamma_i \delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \cdot e(\ell_i, \mathfrak{l}'_i) e(\ell^{\gamma_i \delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma_{i,0}^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \cdot e(\ell^{\gamma_i \delta_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e((\sigma_{i,0} \ell^{\delta_i})^{\gamma_i}, \mathfrak{g}_0^{\alpha}) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(\sigma'_{i,0}, \mathfrak{g}_0^{\alpha})^{\gamma_i} \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(1, \mathfrak{f}'_i)^{\gamma_i} e(\ell, \mathfrak{l}'_i)^{\gamma_i} e(g, \mathfrak{g}'_i)^{\gamma_i} e(h, \mathfrak{h}'_i)^{\gamma_i} \\
&= e(g, \mathfrak{f}'_i) e(\ell_i, \mathfrak{l}'_i) e(a_i, \mathfrak{g}'_i) e(b_i, \mathfrak{h}'_i) \cdot e(1^{\gamma_i}, \mathfrak{f}'_i) e(\ell^{\gamma_i}, \mathfrak{l}'_i) e(g^{\gamma_i}, \mathfrak{g}'_i) e(h^{\gamma_i}, \mathfrak{h}'_i) \\
&= e(g, \mathfrak{f}'_i) e(\ell_i \cdot \ell^{\gamma_i}, \mathfrak{l}'_i) e(a_i \cdot g^{\gamma_i}, \mathfrak{g}'_i) e(b_i \cdot h^{\gamma_i}, \mathfrak{h}'_i) \\
&= e(g, \mathfrak{f}'_i) e(\ell'_i, \mathfrak{l}'_i) e(a'_i, \mathfrak{g}'_i) e(b'_i, \mathfrak{h}'_i)
\end{aligned}$$

About the tags, one can see

$$e(g, \tau'_{i,1}) = e(g, \tau_{i,1}^{1/\mu_i}) = e(g, \tau_{i,1})^{1/\mu_i} = e(\tau_{i,2}, \mathfrak{g})^{1/\mu_i} = e(\tau_{i,2}^{1/\mu_i}, \mathfrak{g}) = e(\tau'_{i,2}, \mathfrak{g}).$$

For the certification, setting $\mathsf{VK} = (g_i)_i^6$ and $\tilde{\tau}_i = R_i$, one has $\tilde{\tau}'_i = R_i \mu_i$ and:

$$\begin{aligned}
e(\tau'_{i,2}, \Sigma'_{i,1}) &= e(g^{1/(R_i \mu_i)}, (\Sigma_{i,1} \cdot {\Sigma_{i,0}}^{\delta_i})^{\alpha \mu_i}) \\
&= e(g, \Sigma_{i,1}^{1/R_i})^{\alpha} \cdot e(g, \Sigma_{i,0}^{1/R_i})^{\delta_i \alpha} \\
&= e(g_1, \mathfrak{g}_0)^{\alpha} e(g_2, \mathfrak{f}_i)^{\alpha} e(g_3, \mathfrak{l}_i)^{\alpha} e(g_4, \mathfrak{g}_i)^{\alpha} e(g_5, \mathfrak{h}_i)^{\alpha} \cdot e(g, \Sigma_{i,0}^{1/R_i})^{\delta_i \alpha} \\
&= e(g_1, \mathfrak{g}'_0) e(g_2, \mathfrak{f}'_i) e(g_4, \mathfrak{g}'_i) e(g_5, \mathfrak{h}'_i) \cdot e(g_3, \mathfrak{l}_i^{\alpha}) e(g, \Sigma_{i,0}^{1/R_i})^{\delta_i \alpha} \\
&= e(g_1, \mathfrak{g}'_0) e(g_2, \mathfrak{f}'_i) e(g_4, \mathfrak{g}'_i) e(g_5, \mathfrak{h}'_i) \cdot e(g_3, \mathfrak{l}_i^{\alpha}) e(g_3, \mathfrak{g}_0^{\delta_i \alpha}) \\
&= e(g_1, \mathfrak{g}'_0) e(g_2, \mathfrak{f}'_i) e(g_4, \mathfrak{g}'_i) e(g_5, \mathfrak{h}'_i) \cdot e(g_3, \mathfrak{l}_i^{\alpha} \mathfrak{g}_0^{\delta_i \alpha}) \\
&= e(g_1, \mathfrak{g}'_0) e(g_2, \mathfrak{f}'_i) e(g_3, \mathfrak{l}'_i) e(g_4, \mathfrak{g}'_i) e(g_5, \mathfrak{h}'_i)
\end{aligned}$$

Randomization

In this part, we prove that a randomized ballot is a correct randomization of a ballot. A ballot $\mathcal{B}_i$ for a user $\mathcal{U}_i$ can be parametrized by (sk_i, M_i) the secret key of the user and his message: $\mathcal{B}_i(\mathsf{sk}_i, M_i) = (\overline{C}_i, \sigma_i, \mathsf{vk}_i, \Sigma_i, \tau_i)$. The three elements $(\overline{C}_i, \mathsf{vk}_i, \tau_i)$ need to be proved indistinguishable from fresh ones but (σ_i, Σ_i) are two deterministic signatures depending on the three previous elements.

A fresh ciphertext $\overline{C}_i$ is of the form $(g, \ell_i, g^{r_i}, h^{r_i} M_i)$ with ℓ_i random in $\mathbb{G}_1$ and r_i random in $\mathbb{Z}_p$. The outputted ciphertext is $\overline{C}'_i = \overline{C}_i \cdot \overline{C}_0^{\gamma_{j,i}} = (g, \ell'_i, g^{r'_i}, h^{r'_i} M_i)$ with $\ell'_i = \ell_i \cdot \ell^{\gamma_{j,i}}$ and $r'_i = r_i + \gamma_{j,i}$. Thus, $\overline{C}'_i$ is perfectly indistinguishable from a fresh ciphertext.

For the tag, $\tau_i = (\mathfrak{g}^{1/R_i}, g^{1/R_i})$ and $\tau'_i = \tau_i^{1/\mu_i}$ thus we have $\tau'_i = (\mathfrak{g}^{1/R'_i}, g^{1/R'_i})$ with $R'_i = R_i \mu_i$ which also corresponds to a fresh tag in a perfectly indistinguishable way.

About the verification key, we have $\mathsf{vk}_0 = (1, 1, \mathfrak{g}_0, 1, 1)$, $\mathsf{vk}_i = (\mathfrak{g}_0, \mathfrak{g}_0^{u_i}, \mathfrak{g}_0^{v_i}, \mathfrak{g}_0^{x_i}, \mathfrak{g}_0^{y_i})$ for $\mathsf{sk}_i = (u_i, v_i, x_i, y_i)$, and $\mathsf{vk}'_i = (\mathsf{vk}_i \cdot \mathsf{vk}_0^{\delta_i})^{\alpha}$ thus, $\mathsf{vk}'_i = (\mathfrak{g}'_0, {\mathfrak{g}'_0}^{u_i}, {\mathfrak{g}'_0}^{v'_i}, {\mathfrak{g}'_0}^{x_i}, {\mathfrak{g}'_0}^{y_i})$ with $\mathfrak{g}'_0 = \mathfrak{g}_0^{\alpha}$ and $v'_i = v_i + \delta_i$ which is indistinguishable from a fresh verification key under the DDH in $\mathbb{G}_2$.

5.5 Applications

We now discuss use-cases of mix-nets: electronic voting and anonymous routing. In both cases, a mix-server can, on the fly, perform individual verifications and randomization of ballots, as well as the product of the $\mathfrak{f}_i$'s and the ciphertexts adaptively until the ballots are all sent. Eventually, at the closing time for a vote or at the end of a time lapse for routing, one just has to do and sign global proof of Diffie-Hellman tuples, and then output the ballots in a permuted order.

5.5.1 Electronic Voting

Our mix-net fits well the case of e-voting because after the multiple mixing steps, all the mix-servers can perform a second round to sign in a compact way the constant-size proof, certifying each of their contributions. The input size as well as the computation cost of the verifier are both independent on the number of mixing steps. To our knowledge it is the first scheme with this very nice property.

About security, as explained, soundness and privacy are guaranteed for the honest users only: honest users are sure that their votes are randomized in the output ballot-box, and their input-output ballots are unlinkable. This is of course the most important requirements. However, since the u_i's are used to guarantee that no ballots are deleted or inserted, this is important those values to be unknown to the mix-server.

In the Section 4.4, we proposed a construction that uses Square Diffie-Hellman tuples $(\mathfrak{g}_r, \mathfrak{A}_i = \mathfrak{g}_r^{w_i}, \mathfrak{B}_i = \mathfrak{A}_i^{w_i})$ as tags to add in any one-time linearly homomorphic signature to obtain a linearly homomorphic signature with randomizable tags. Then, one can use $\prod \mathfrak{A}'_j = (\prod \mathfrak{A}_i)^{\alpha}$ instead of $\prod \mathfrak{f}'_j$ and $(\prod \mathfrak{f}_i)^{\alpha}$, in the Diffie-Hellman tuple, to guarantee the permutation of the verification keys. Only the privacy of the w_i's is required to guarantee the soundness. Even if the tag can be randomized just in a computational way and not in a statistical way, this is enough for our mix-net application. We can also exploit the universal tag to optimize our construction of mix-net. Indeed, instead of having $\Sigma_{i,0}$ the LH-Sign of vk_0 for each user, it is possible to have $\Sigma_0 = \mathsf{Sign}^*(\mathsf{SK}, w, \mathsf{vk}_0)$ and still be able to randomize vk_i and adapt its signature $\Sigma_{i,1}$ keeping the tag τ_i per user.

The proof that $\prod M_i = \prod M'_i$ is actually never used in the previous security proofs, as it counts for privacy in e-voting only. Indeed, in our privacy security game we let the adversary choose the messages of the honest users. In a voting scheme, the adversary could not choose them and would like to learn the vote of a target voter. The first mix-server could take the vote (ciphertext) of this voter and ask several corrupted voters to duplicate this vote. The bias in the

tally would reveal the vote of the target voter: the proof on the products of the plaintexts avoids this modification during the mixing. This does not exclude the attack of Cortier-Smyth [CS13] if the votes are publicly sent, as the corrupted voters could simply use the ciphertext for their own ballots.

5.5.2 Message Routing

Another important use case of mix-nets is in routing protocols where the mix-servers are proxy servers guaranteeing that no one can trace a request of a message. In this scenario, it is not possible to perform a second round on the mix-servers to obtain the multi-signature and the efficiency is thus linear in the number of mixing steps. It is still an open problem to avoid the second round while maintaining the independence in the number of mix-servers.

Chapter

6 Anonymous Credentials

This chapter is based on the paper [HP20] under submission.

Chapter content

The most recent papers on attribute-based anonymous credential schemes are [FHS19, San20]. The former proposes the first constant-size credential to prove k-of-N attributes, with computational complexity in $O(N-k)$ for the prover and in $O(k)$ for the verifier. However, it only works for one credential issuer ($K=1$). The latter one improves this result by enabling multiple showings of relations (r) of attributes. All the other known constructions allow, at best, selective (s) disclosures of attributes.

In [CL11], Canard and Lescuyer use aggregate signatures to construct an ABC system. It is thus the closest to our approach. Instead of having *tags*, their signatures take *indices* as input. We follow a similar path but we completely formalize this notion of tag/index with an EphemerId scheme. To our knowledge, aggregate signatures are the only way to deal with multiple credential issuers but still allowing to show a unique compact credential for the proof of possession of attributes coming from different credential issuers. However, the time-complexity of a prover during a verification depends on the number k of shown attributes. We solve this issue at the cost of a larger key for the credential issuers (but still in the same order as [FHS19, San20]) and a significantly better showing cost for the prover (also better than [FHS19, San20]).

After precising some notations and reviewing classical definitions in Section 2.1, we informally describe, in Section 6.1, the two important primitives that we will use in our construction of anonymous credentials: the EphemerId and ART-Sign schemes. In Section 6.2, we provide their full definitions and a concrete instantiation. From there, we will be able to define and construct, in Section 6.3, our ABC scheme from EphemerId and ART-Sign schemes. The full instantiation is given in Section 6.4. Traceability is defined and instantiated in Section 6.5. Finally, related work is discussed in Section 6.6.

6.1 Overview of our New Primitives

The usual way to perform authentication is by presenting a certified public key and proving ownership of the associated private key with a zero-knowledge proof of knowledge. The certified public key is essentially the signature by a Certification Authority (CA) on a public key and an identity pair, with a *standard* signature scheme. In case of attribute-based authentication, the attribute is signed together with the public key in the certificate. The latter thus signs two objects, with different purpose, the public key associated to a private key, and the identity or an attribute.

In the same vein as labelled encryption schemes, we define tag-based signatures to dissociate the user-key which will be a provable tag and $\mathcal{A}\mathsf{ttr}$ which will be the signed message (attribute or identity). This flexibility will allow randomizability of one without affecting the other, leading to anonymous credentials.

Notations. In this chapter, vectors will be denoted between brackets $[\ldots]$ and unions will be concatenations: $[a, b] \cup [a, c] = [a, b, a, c]$, keeping the ordering. On the other hand, sets will be denoted between braces $\{\ldots\}$, with possible repetitions: $\{a, b\} \cup \{a, c\} = \{a, a, b, c\}$ as in [San20], but without ordering.

6.1.1 Tag-based Signatures

For a pair $(\tilde{\tau}, \tau)$, where τ is a tag and $\tilde{\tau}$ corresponds to the secret part of the tag, one can define a new primitive called *tag-based signature*, where we assume all the used tags τ to be *valid* (either because they are all valid, or their validity can be checked):

Tag-based Signatures

A tag-based signatures consists of the algorithms:

$\mathsf{Setup}(1^\kappa)$**:** Given a security parameter κ, it outputs the global parameter param, which includes the message space $\mathcal{M}$ and the tag space $\mathcal{T}$;

$\mathsf{Keygen}(\mathsf{param})$**:** Given a public parameter param, it outputs a key pair $(\mathsf{sk}, \mathsf{vk})$;

$\mathsf{GenTag}(\mathsf{param})$**:** Given a public parameter param, it generates a witness-word pair $(\tilde{\tau}, \tau)$;

$\mathsf{Sign}(\mathsf{sk}, \tau, m)$**:** Given a signing key sk, a tag τ, and a message m, it outputs the signature σ under the tag τ;

$\mathsf{VerifSign}(\mathsf{vk}, \tau, m, \sigma)$**:** Given a verification key vk, a tag τ, a message m and a signature σ, it outputs 1 if σ is valid relative to vk and τ, and 0 otherwise.

The security notion would expect no adversary able to forge, for any honest pair $(\mathsf{sk}, \mathsf{vk})$, a new signature for a pair (τ, m), for a valid tag τ, if the signature has not been generated using sk and the tag τ on the message m. Generically, $\tilde{\tau}$ can be sk and τ can be vk, then this is just

a classical signature of m. Another case is when $\tilde{\tau} = \tau$, and then this can just be a classical signature of the message-pair (τ, m).

However more subtil situations can be handled: in our use-cases, τ will be a word for some language $\mathcal{L}$ representing the authorized users and $\tilde{\tau}$ a witness (for $\tau \in \mathcal{L}$). According to the language $\mathcal{L}$, which can be a strict subset of the whole set $\mathcal{T}$, one may have to prove the actual membership $\tau \in \mathcal{L}$ (the validity of the tag) for the validity of the signature. It might be important in the unforgeability security notion. On the other hand, one may also have to prove the knowledge of the witness $\tilde{\tau}$, in an interactive and zero-knowledge way for authentication.

The latter can be performed, using the interactive protocol $(\mathsf{ProveKTag}(\tilde{\tau}), \mathsf{VerifKTag}(\tau))$. This will be useful for the freshness in the authentication process. The former can also be proven using an interactive protocol $(\mathsf{ProveVTag}(\tilde{\tau}), \mathsf{VerifVTag}(\tau))$. However this verification can also be non-interactive or even public, without needing any private witness. The only requirement is that this proof or verification of membership should not reveal the private witness involved in the proof of knowledge, as the witness will be used for authentication.

Now the tag and the message are two distinct elements in the signature, we will introduce new properties for each of them:

- randomizable tags: if τ can be randomized, but still with an appropriate zero-knowledge proof of knowledge of $\tilde{\tau}$, one can get anonymous credentials, where τ is a randomizable public key and an attribute is signed;
- aggregate signatures: one can aggregate signatures generated for different messages (attributes), even different keys (multi-authority) but all on the same tag τ.

By combining both properties, we will provide a compact scheme of attribute-based anonymous credentials. When a trapdoor allows to link randomized tags, one gets traceability.

6.1.2 Signatures with Randomizable Tags

As tags are seen as words in some language $\mathcal{L}$, randomizable tags will make sense for random-self reducible languages [TW87]: the word τ defined by a witness $\tilde{\tau}$ and some additional randomness r can be derived into another word $\tilde{\tau}'$ associated to $\tilde{\tau}'$ and r' (either r' only or both $\tilde{\tau}'$ and r' are uniformly random). When randomizing τ into τ', one must be able to keep track of the change from to update $\tilde{\tau}$ to $\tilde{\tau}'$ and the signatures. Formally, we will require to have the three algorithms:

$\mathsf{RandTag}(\tau)$**:** Given a tag τ as input, it outputs a new tag τ' and the randomization link $\rho_{\tau\to\tau'}$;

$\mathsf{DerivWitness}(\tilde{\tau}, \rho_{\tau\to\tau'})$**:** Given a witness $\tilde{\tau}$ (associated to the tag τ) and a randomization link between τ and a tag τ' as input, it outputs a witness $\tilde{\tau}'$ for the tag τ';

$\mathsf{DerivSign}(\mathsf{vk}, \tau, m, \sigma, \rho_{\tau\to\tau'})$**:** Given a valid signature σ on tag τ and message m, and $\rho_{\tau\to\tau'}$ the randomization link between τ and another tag τ', it outputs a new signature σ' on the message m and the new tag τ'. Both signatures are under the same key vk.

From a valid witness-word pair $(\tilde{\tau}, \tau) \leftarrow \mathsf{GenTag}(\mathsf{param})$, if $(\tau', \rho) \leftarrow \mathsf{RandTag}(\tau)$ and $\tilde{\tau}' \leftarrow \mathsf{DerivWitness}(\tilde{\tau}, \rho)$ then $(\tilde{\tau}', \tau')$ should also be a valid witness-word pair.

In addition, for compatibility with the tag and correctness of the signature scheme, we require that for all honestly generated keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Keygen}(\mathsf{param})$, all messages m, and all tags $(\tilde{\tau}, \tau) \leftarrow \mathsf{GenTag}(\mathsf{param})$, if $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau, m)$, $(\tau', \rho) \leftarrow \mathsf{RandTag}(\tau)$ and $\sigma' \leftarrow \mathsf{DerivSign}(\mathsf{vk}, \tau, m, \sigma, \rho)$, then the algorithm $\mathsf{VerifSign}(\mathsf{vk}, \tau', m, \sigma')$ should output 1.

For privacy reasons, in case of probabilistic signatures, it will not be enough to just randomize the tag, but the random coins too:

RandSign$(\mathsf{vk}, \tau, m, \sigma)$**:** Given a valid signature σ on tag τ and message m, it outputs a new signature σ' on the same message m and tag τ.

Correctness extends the above one, where the algorithm $\mathsf{VerifSign}(\mathsf{vk}, \tau', m, \sigma'')$ should output 1 with $\sigma'' \leftarrow \mathsf{RandSign}(\mathsf{vk}, \tau', m, \sigma')$. One additionally expects unlinkability: the following distributions are (computationally) indistinguishable, for any vk and m (possibly chosen by the adversary), where for $i = 0, 1$, $(\tilde{\tau}_i, \tau_i) \leftarrow \mathsf{GenTag}(1^\kappa)$, $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, \tau_i, m)$, $(\tau_i', \rho_i) \leftarrow \mathsf{RandTag}(\tau_i)$, $\sigma_i' \leftarrow \mathsf{DerivSign}(\mathsf{vk}, \tau_i, m, \sigma_i, \rho_i)$ and $\sigma_i'' \leftarrow \mathsf{RandSign}(\mathsf{vk}, \tau_i', m, \sigma_i')$:

$$\mathcal{D}_0 = \{(m, \mathsf{vk}, \tau_0, \sigma_0, \tau_0', \sigma_0'', \tau_1, \sigma_1, \tau_1', \sigma_1'')\} \qquad \mathcal{D}_1 = \{(m, \mathsf{vk}, \tau_0, \sigma_0, \tau_1', \sigma_1'', \tau_1, \sigma_1, \tau_0', \sigma_0'')\}.$$

6.1.3 Aggregate Signatures

Boneh et al. [BGLS03] remarked it was possible to aggregate the BLS signature [BLS01], we will follow this path, but for tag-based signatures, with possible aggregation only between signatures with the same tag, in a similar way as the indexed aggregated signatures [CL11]. We will even consider aggregation of public keys, which can either be a simple concatenation or a more evolved combination as in [BDN18]. Hence, an aggregate (tag-based) signature scheme (Aggr-Sign) is a signature scheme with the algorithms:

Aggregate Signature Scheme

An aggregate (tag-based) signature scheme (Aggr-Sign) is a signature scheme with the algorithms:

AggrKey$(\{\mathsf{vk}_j\}_{j=1}^{\ell})$**:** Given ℓ verification keys vk_j, it outputs an aggregated verification key avk;

AggrSign$(\tau, (\mathsf{vk}_j, m_j, \sigma_j)_{j=1}^{\ell})$**:** Given ℓ signed messages m_j in σ_j under vk_j and the same tag τ, it outputs a signature σ on the message-set $\boldsymbol{M} = \{m_j\}_{j=1}^{\ell}$ under the tag τ and aggregated verification key avk.

We remark that keys can evolve (either in a simple concatenation or a more compact way) but messages also become sets. While we will still focus on signing algorithm of a single message with a single key, we have to consider verification algorithms on message-sets and for aggregated verification keys. In the next section, we combine aggregation with randomizable tags, and we will handle verification for message-sets.

Correctness of an aggregate (tag-based) signature scheme requires that for any valid tag-pair $(\tilde{\tau}, \tau)$ and honestly generated keys $(\mathsf{sk}_j, \mathsf{vk}_j) \leftarrow \mathsf{Keygen}(\mathsf{param})$, if $\sigma_j = \mathsf{Sign}(\mathsf{sk}_j, \tau, m_j)$ are valid signatures for $j = 1, \cdots, \ell$, then for both key $\mathsf{avk} \leftarrow \mathsf{AggrKey}(\{\mathsf{vk}_j\}_{j=1}^{\ell})$ and signature $\sigma = \mathsf{AggrSign}(\tau, (\mathsf{vk}_j, m_j, \sigma_j)_{j=1}^{\ell})$, the verification $\mathsf{VerifSign}(\mathsf{avk}, \tau, \{m_j\}_{j=1}^{\ell}, \sigma)$ should output 1.

6.2 Aggregate Signatures with Randomizable Tags

After the informal presentation of our new primitive, we describe the full definition of aggregate signature scheme with randomizable tags. We will then provide a concrete construction that we will extend to attribute-based anonymous credentials. While the compactness of the credentials will exploit the aggregation of signature, as in [CL11], privacy will rely on the randomizability of the tags. However, their specific format will allow more compact anonymous credentials.

6.2.1 Anonymous Ephemeral Identities

As our randomizable tags will be used as ephemeral identities (ephemeral key pairs), we denote them EphemerId:

Definition 46 — EphemerId Scheme

An EphemerId scheme consists of the algorithms:

$\mathsf{Setup}(1^\kappa)$**:** Given a security parameter κ, it outputs the global parameter param, which includes the tag space $\mathcal{T}$;

$\mathsf{GenTag}(\mathsf{param})$**:** Given a public parameter param, it outputs a tag τ and its secret part $\tilde{\tau}$;

$(\mathsf{ProveVTag}(\tilde{\tau}), \mathsf{VerifVTag}(\tau))$**:** This (possibly interactive) protocol corresponds to the verification of the tag τ. At the end of the protocol, the verifier outputs 1 if it accepts τ as a valid tag and 0 otherwise;

$\mathsf{RandTag}(\tau)$**:** Given a tag τ as input, it outputs a new tag τ' and the randomization link $\rho_{\tau\to\tau'}$ between τ and τ';

$\mathsf{DerivWitness}(\tilde{\tau}, \rho_{\tau\to\tau'})$**:** Given a witness $\tilde{\tau}$ (associated to the tag τ) and a link between the tags τ and τ' as input, it outputs a witness $\tilde{\tau}'$ for the tag τ';

$(\mathsf{ProveKTag}(\tilde{\tau}), \mathsf{VerifKTag}(\tau))$**:** This optional interactive protocol corresponds to the proof of knowledge of $\tilde{\tau}$. At the end of the protocol, the verifier outputs 1 if it accepts the proof and 0 otherwise.

The security notions are the usual properties of zero-knowledge proofs for the two protocols $(\mathsf{ProveKTag}(\tilde{\tau}), \mathsf{VerifKTag}(\tau))$ and $(\mathsf{ProveVTag}(\tilde{\tau}), \mathsf{VerifVTag}(\tau))$, with zero-knowledge and soundness. But the RandTag must also randomize the tag τ within an equivalence class, in an unlinkable way:

- Correctness: the language $\mathcal{L} \subset \mathcal{T}$ might be split in equivalence classes (denoted $\sim$, with possibly a unique huge class), then for any τ issued from GenTag and $\tau' \leftarrow \mathsf{RandTag}(\tau)$, we must have $\tau' \sim \tau$;
- Soundness: the verification process for the validity of the tag should not accept an invalid tag (not in the language);
- Knowledge Soundness: in case of the optional proof of knowledge, extraction of the witness should be possible when the verifier accepts the proof with non-negligible probability;
- Zero-knowledge: the proof of validity and the proof of knowledge should not reveal any information about the witness;
- Unlinkability: for any pair (τ_1, τ_2) issued from GenTag, the two distributions $\{(\tau_1, \tau_2, \tau_1', \tau_2')\}$ and $\{(\tau_1, \tau_2, \tau_2', \tau_1')\}$, where $\tau_1' \leftarrow \mathsf{RandTag}(\tau_1)$ and $\tau_2' \leftarrow \mathsf{RandTag}(\tau_2)$, must be (computationally) indistinguishable.

In the case of unique equivalence class for τ, one can expect perfect unlinkability. In case of multiple equivalence classes for τ, these classes should be computationally indistinguishable to provide unlinkability.

6.2.2 Aggregate Signatures with Randomizable Tags

We can now provide the formal definition of an aggregate signature scheme with randomizable tags, where some algorithms exploit compatibility between the EphemerId scheme and the signature scheme:

Definition 47 — Aggregate Signatures with randomizable tags (ART-Sign)

An ART-Sign scheme, associated to an EphemerId scheme $\mathcal{E}$ = (Setup, GenTag, (ProveVTag, VerifVTag), RandTag, DerivWitness) consists of the algorithms (Setup, Keygen, Sign, AggrKey, AggrSign, DerivSign, RandSign, VerifSign):

Setup(1^κ)**:** Given a security parameter κ, it runs $\mathcal{E}$.Setup and outputs the global parameter param, which includes $\mathcal{E}$.param with the tag space $\mathcal{T}$, and extends it with the message space $\mathcal{M}$;

Keygen(param)**:** Given a public parameter param, it outputs a key-pair (sk,vk);

Sign(sk, τ, m)**:** Given a signing key, a valid tag τ, and a message $m \in \mathcal{M}$, it outputs the signature σ;

AggrKey($\{\mathsf{vk}_j\}_{j=1}^{\ell}$)**:** Given ℓ verification keys vk_j, it outputs an aggregated verification key avk;

AggrSign($\tau, (\mathsf{vk}_j, m_j, \sigma_j)_{j=1}^{\ell}$)**:** Given ℓ signed messages m_j in σ_j under vk_j and the same valid tag τ, it outputs a signature σ on the message-set $\boldsymbol{M} = \{m_j\}_{j=1}^{\ell}$ under the tag τ and aggregated verification key avk;

VerifSign($\mathsf{avk}, \tau, \boldsymbol{M}, \sigma$)**:** Given a verification key avk, a valid tag τ, a message-set $\boldsymbol{M}$ and a signature σ, it outputs 1 if σ is valid relative to avk and τ, and 0 otherwise;

DerivSign($\mathsf{avk}, \tau, \boldsymbol{M}, \sigma, \rho_{\tau\to\tau'}$)**:** Given a signature σ on a message-set $\boldsymbol{M}$ under a valid tag τ and aggregated verification key avk, and the randomization link $\rho_{\tau\to\tau'}$ between τ and another tag τ', it outputs a signature σ' on the message-set $\boldsymbol{M}$ under the new tag τ' and the same key avk;

RandSign($\mathsf{avk}, \tau, \boldsymbol{M}, \sigma$)**:** Given a signature σ on a message-set $\boldsymbol{M}$ under a valid tag τ and aggregated verification key avk, it outputs a new signature σ' on the message-set $\boldsymbol{M}$ and the same tag τ.

We stress that all the tags must be valid. Moreover, using algorithms from $\mathcal{E}$, tags are randomizable at any time, and signatures adapted and randomized, even after an aggregation: avk and $\boldsymbol{M}$ can either be single key and message or aggregations of keys and messages. One can remark that only protocol (ProveVTag, VerifVTag) from $\mathcal{E}$ is involved in the ART-Sign scheme, as one just needs to check the validity of the tag, not the ownership. The latter will be useful in anonymous credentials with fresh proof of ownership.

Unforgeability.

In the Chosen-Message Unforgeability security game, the adversary has unlimited access to the following oracles, with lists KList and TList initially empty:

- $\mathcal{O}$GenTag() outputs the tag τ and keeps track of the associated witness $\tilde{\tau}$, with $(\tilde{\tau}, \tau)$ appended to TList;
- $\mathcal{O}$Keygen() outputs the verification key vk and keeps track of the associated signing key sk, with (sk, vk) appended to KList;
- $\mathcal{O}$Sign(τ, vk, m), for $(\tilde{\tau}, \tau) \in$ TList and $(\mathsf{sk}, \mathsf{vk}) \in$ KList, outputs Sign(sk, τ, m).

It should not be possible to generate a signature that falls outside the range of DerivSign, RandSign, or AggrSign:

Definition 48 (Unforgeability for ART-Sign)**.** An ART-Sign scheme is said unforgeable if, for any adversary $\mathcal{A}$ that, given signatures σ_i for tuples $(\tau_i,\mathsf{vk}_i,m_i)$ of its choice but for τ_i and vk_i issued from the GenTag and Keygen algorithms respectively (for Chosen-Message Attacks), outputs a tuple $(\mathsf{avk},\tau,\boldsymbol{M},\sigma)$ where both τ is a valid tag and σ is a valid signature w.r.t. $(\mathsf{avk},\tau,\boldsymbol{M})$, there exists a subset J of the signing queries with a common tag $\tau' \in \{\tau_i\}_i$ such that $\tau \sim \tau'$, $\forall j \in J, \tau_j = \tau'$, avk is an aggregated key of $\{\mathsf{vk}_j\}_{j\in J}$, and $\boldsymbol{M} = \{m_j\}_{j\in J}$, with overwhelming probability.

Since there are multiple secrets, we can consider corruptions of some of them:

- $\mathcal{O}\mathsf{CorruptTag}(\tau)$, for $(\tilde{\tau},\tau) \in \mathsf{TList}$, outputs $\tilde{\tau}$;
- $\mathcal{O}\mathsf{Corrupt}(\mathsf{vk})$, for $(\mathsf{sk},\mathsf{vk}) \in \mathsf{KList}$, outputs sk.

The forgery should not involve a corrupted key (but corrupted tags are allowed). Note again that all the tags are valid (either issued from GenTag or verified). In the unforgeability security notion, some limitations might be applied to the signing queries: one-time queries (for a given tag-key pair) or a bounded number of queries.

Unlinkability.

Randomizability of both the tag and the signature are expected to provide anonymity, with some unlinkability property:

Definition 49 (Unlinkability for ART-Sign)**.** An ART-Sign scheme is said unlinkable if, for any avk and $\boldsymbol{M}$, no adversary $\mathcal{A}$ can distinguish the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$, where for $i = 0, 1$, we have $(\tilde{\tau}_i,\tau_i) \leftarrow \mathsf{GenTag}(1^\kappa)$, $(\tau_i',\rho_i) \leftarrow \mathsf{RandTag}(\tau_i)$, σ_i is any valid signature of $\boldsymbol{M}$ under τ_i and vk, $\sigma_i' \leftarrow \mathsf{DerivSign}(\mathsf{avk},\tau_i,\boldsymbol{M},\sigma_i,\rho_i)$ and $\sigma_i'' \leftarrow \mathsf{RandSign}(\mathsf{avk},\tau_i',\boldsymbol{M},\sigma_i')$:

$$\mathcal{D}_0 = \{(\boldsymbol{M},\mathsf{avk},\tau_0,\sigma_0,\tau_0',\sigma_0'',\tau_1,\sigma_1,\tau_1',\sigma_1'')\} \quad \mathcal{D}_1 = \{(\boldsymbol{M},\mathsf{avk},\tau_0,\sigma_0,\tau_1',\sigma_1'',\tau_1,\sigma_1,\tau_0',\sigma_0'')\}.$$

6.2.3 One-Time ART-Sign Scheme with Square Diffie-Hellman Tags (SqDH)

Our construction will provide an aggregate signature with randomizable tags based on the second linearly homomorphic signature scheme of 4.5.2.

Description of the EphemerId Scheme.

With tags in $\mathcal{T} = \mathbb{G}_1^3$, in an asymmetric bilinear setting $(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,p,g,\mathfrak{g},e)$, and τ is a Square Diffie-Hellman tuple $(h,h^{\tilde{\tau}},h^{\tilde{\tau}^2})$, one can define the SqDH EphemerId scheme:

EphemerId Scheme

Setup(1^κ)**:** Given a security parameter κ, let $(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,p,g,\mathfrak{g},e)$ be an asymmetric bilinear setting, where g and $\mathfrak{g}$ are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. The set of tags is $\mathcal{T} = \mathbb{G}_1^3$. We then define $\mathsf{param} = (\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,p,g,\mathfrak{g},e;\mathcal{T})$;

GenTag(param)**:** Given a public parameter param, it randomly chooses a generator $h \xleftarrow{\$} \mathbb{G}_1^*$ and outputs $\tilde{\tau} \xleftarrow{\$} \mathbb{Z}_p^*$ and $\tau = (h,h^{\tilde{\tau}},h^{\tilde{\tau}^2}) \in \mathbb{G}_1^3$.

ProveVTag($\tilde{\tau}$), VerifVTag(τ)**:** The prover constructs the proof $\pi = \mathsf{proof}(\tilde{\tau} : \tau = (h,h^{\tilde{\tau}},h^{\tilde{\tau}^2}))$ (see 4.4 for the Groth-Sahai [GS08] proof). The verifier outputs 1 if it accepts the proof and 0 otherwise.

RandTag(τ)**:** Given a tag τ as input, it chooses $\rho_{\tau\to\tau'} \xleftarrow{\$} \mathbb{Z}_p$ and constructs $\tau' = \tau^{\rho_{\tau\to\tau'}}$ the derived tag. It outputs $(\tau',\rho_{\tau\to\tau'})$.

DerivWitness($\tilde{\tau}, \rho_{\tau\to\tau'}$): The derived witness remains unchanged: $\tilde{\tau}' = \tilde{\tau}$.

Valid tags are Square Diffie-Hellman pairs in $\mathbb{G}_1$:

$$\mathcal{L} = \{(h, h^x, h^{x^2}), h \in \mathbb{G}_1^*, x \in \mathbb{Z}_p^*\} = \cup_{x\in\mathbb{Z}_p^*}\mathcal{L}_x \qquad \mathcal{L}_x = \{(h, h^x, h^{x^2}), h \in \mathbb{G}_1^*\}$$

The randomization does not affect the exponents, hence there are $p-1$ different equivalence classes $\mathcal{L}_x$, for all the non-zero exponents $x \in \mathbb{Z}_p^*$, and correctness is clearly satisfied within equivalence classes. The validity check (see 4.4) is sound as the Groth-Sahai commitment is in the perfectly binding setting. Such tags also admit an interactive Schnorr-like zero-knowledge proof of knowledge of the exponent $\tilde{\tau}$ for $(\mathsf{ProveKTag}(\tilde{\tau}), \mathsf{VerifKTag}(\tau))$ which also provides extractability (knowledge soundness). Under the DSqDH and DL assumptions, given the tag τ, it is hard to recover the exponent $\tilde{\tau} = x$. The tags, after randomization, are uniformly distributed in the equivalence class, and under the DSqDH-assumption, each class is indistinguishable from $\mathbb{G}_1^3$, and thus one has unlinkability.

Description of the One-Time SqDH-based ART-Sign Scheme.

The above EphemerId scheme can be extended into an ART-Sign scheme where implicit vector messages are signed. As the aggregation can be made on signatures of messages under the same tag but from various signers, the description is given for signers indexed by j and one-component messages indexed by (j, i). However, the scheme needs to be state-full as there is the limitation for a signer j not to sign more than one message by index (j, i) for a given tag: a signer must use two different indices to sign two messages for one tag.

One-Time SqDH-based ART-Sign Scheme

Setup(1^κ): It extends the above setup with the set of messages $\mathcal{M} = \mathbb{Z}_p$;

Keygen(param): Given the public parameters param, it outputs the signing and verification keys

$$\begin{array}{lllll} \mathsf{sk}_{j,i} = (& \mathsf{SK}_j = [& t, \quad u, \quad v &], & \mathsf{SK}'_{j,i} = [\quad r_i, \quad s_i \quad] \quad) \xleftarrow{\$} \mathbb{Z}_p^5, \\ \mathsf{vk}_{j,i} = (& \mathsf{VK}_j = [& \mathfrak{g}^t, \quad \mathfrak{g}^u, \quad \mathfrak{g}^v &], & \mathsf{VK}'_{j,i} = [\quad \mathfrak{g}^{r_i}, \quad \mathfrak{g}^{s_i} \quad] \quad) \in \mathbb{G}_2^5. \end{array}$$

Note that one could dynamically add new $\mathsf{SK}'_{j,i}$ and $\mathsf{VK}'_{j,i}$ to sign implicit vector messages: $\mathsf{sk}_j = \mathsf{SK}_j \cup [\mathsf{SK}'_{j,i}]_i$, $\mathsf{vk}_j = \mathsf{VK}_j \cup [\mathsf{VK}'_{j,i}]_i$;

Sign($\mathsf{sk}_{j,i}, \tau, m$): Given a signing key $\mathsf{sk}_{j,i} = [t, u, v, r, s]$, a message $m \in \mathbb{Z}_p$ and a public tag $\tau = (\tau_1, \tau_2, \tau_3)$, it outputs the signature

$$\sigma = \tau_1^{t+r+ms} \times \tau_2^u \times \tau_3^v.$$

AggrKey($\{\mathsf{vk}_{j,i}\}_{j,i}$): Given verification keys $\mathsf{vk}_{j,i}$, it outputs the aggregated verification key $\mathsf{avk} = [\mathsf{avk}_j]_j$, with $\mathsf{avk}_j = \mathsf{VK}_j \cup [\mathsf{VK}'_{j,i}]_i$ for each j;

AggrSign($\tau, (\mathsf{vk}_{j,i}, m_{j,i}, \sigma_{j,i})_{j,i}$): Given tuples of verification key $\mathsf{vk}_{j,i}$, message $m_{j,i}$ and signature $\sigma_{j,i}$ all under the same tag τ, it outputs the signature $\sigma = \prod_{j,i} \sigma_{j,i}$ of the concatenation of the messages verifiable with $\mathsf{avk} \leftarrow \mathsf{AggrKey}(\{\mathsf{vk}_{j,i}\}_{j,i})$;

DerivSign($\mathsf{avk}, \tau, \boldsymbol{M}, \sigma, \rho_{\tau\to\tau'}$): Given a signature σ on tag τ and a message-set $\boldsymbol{M}$, and $\rho_{\tau\to\tau'}$ the randomization link between τ and another tag τ', it outputs $\sigma' = \sigma^{\rho_{\tau\to\tau'}}$;

RandSign($\mathsf{avk}, \tau, \boldsymbol{M}, \sigma$): The scheme being deterministic, it returns σ;

VerifSign($\mathsf{avk}, \tau, \boldsymbol{M}, \sigma$): Given a valid tag $\tau = (\tau_1, \tau_2, \tau_3)$, an aggregated verification key $\mathsf{avk} = [\mathsf{avk}_j]$ and a message-set $\boldsymbol{M} = [\mathsf{m}_j]$, with both for each j, $\mathsf{avk}_j = \mathsf{VK}_j \cup [\mathsf{VK}'_{j,i}]_i$ and $\mathsf{m}_j = [m_{j,i}]_i$, and a signature σ, one checks if the following equality holds or not, where $n_j = \#\{\mathsf{VK}'_{j,i}\}$:

$$e(\sigma, \mathfrak{g}) = e\left(\tau_1, \prod_j {\mathsf{VK}_{j,1}}^{n_j} \times \prod_i \mathsf{VK}'_{j,i,1} \cdot {\mathsf{VK}'_{j,i,2}}^{m_{j,i}}\right) \times e\left(\tau_2, \prod_j {\mathsf{VK}_{j,2}}^{n_j}\right) \times e\left(\tau_3, \prod_j {\mathsf{VK}_{j,3}}^{n_j}\right).$$

In case of similar public keys in the aggregation (a unique index j), $\mathsf{avk} = \mathsf{VK} \cup [\mathsf{VK}'_i]_i$ and verification becomes, where $n = \#\{\mathsf{VK}'_i\}$,

$$e(\sigma, \mathfrak{g}) = e\left(\tau_1, {\mathsf{VK}_1}^n \times \prod_{i=1}^{n} \mathsf{VK}'_{i,1} \cdot {\mathsf{VK}'_{i,2}}^{\boldsymbol{M}_i}\right) \times e\left(\tau_2, {\mathsf{VK}_2}^n\right) \times e\left(\tau_3, {\mathsf{VK}_3}^n\right).$$

Recall that the validity of the tag has to be verified, either with a proof of knowledge of the witness (as it will be the case in the ABC scheme, or with the proof $\pi = \mathsf{proof}(\tilde{\tau} : \tau = (h, h^{\tilde{\tau}}, h^{\tilde{\tau}^2}))$ (see 4.4 for the Groth-Sahai [GS08] proof).

Security of the One-Time SqDH-based ART-Sign Scheme.

As argued in [HPP20], the signature scheme defined above is unforgeable in the generic group model [Sho97], if signing queries are asked at most once per tag-index pair:

Theorem 50. *The One-Time SqDH-based ART-Sign is unforgeable with one signature only per index, for a given tag, even with adaptive corruptions of keys and tags, in the generic group model.*

Proof. As argued in [HPP20], when the bases of the tags are *random*, even if the exponents are known, the signature that would have signed messages $\boldsymbol{M} = (g, g^{m_1}, \ldots, g, g^{m_n})$ is an unforgeable linearly-homomorphic signature. This means it is only possible to linearly combine signatures with the same tag. As issued signatures are on pairs (g, g^{m_i}), under a different pair of keys for each such signed pair (whether they are from the same global signing key SK or not, as we exclude repetitions for an index), which can be seen as tuples $(1, 1, \ldots, g, g^{m_i}, \ldots, 1, 1)$, completed with 1's, the invariant generators g imply coefficients 0 and 1 in the linear combination: all the pairs (g, g^{m_i}) have been signed under the same tag. This proves unforgeability, even with corruptions of the tags, but without repetitions of tag-index. One can also consider corruptions of the signing keys, as they are all independent: one just needs to guess under which key will be generated the forgery. □

About unlinkability, it relies on the DSqDH assumption, but between credentials that contain the same messages at the same shown indices (the same message-vector $\boldsymbol{M}$):

Theorem 51. *The One-Time SqDH-based ART-Sign, with message-vectors, is unlinkable under the* DSqDH *assumption.*

Proof. As already noticed, the tags are randomizable among all the square Diffie-Hellman triples with the same exponent, which are indistinguishable from random triples in $\mathbb{G}_1^3$, so for any pair of tags $(\tilde{\tau}_i, \tau_i) \leftarrow \mathsf{GenTag}(1^\kappa)$, for $i = 0, 1$, when randomized into τ'_i respectively, the distributions $(\tau_0, \tau_1, \tau'_0, \tau'_1)$ and $(\tau_0, \tau_1, \tau'_1, \tau'_0)$ are indistinguishable under the DSqDH assumption. For any avk and $\boldsymbol{M}$, the signatures are deterministic and unique for a tag τ, so they are functions (even if

not efficiently computable) of $(\mathsf{avk}, \tau, \boldsymbol{M})$, so the distributions $(\boldsymbol{M}, \mathsf{avk}, \tau_0, \sigma_0, \tau_1, \sigma_1, \tau'_0, \sigma'_0, \tau'_1, \sigma_1)$ and $(\boldsymbol{M}, \mathsf{avk}, \tau_0, \sigma_0, \tau_1, \sigma_1, \tau'_1, \sigma'_1, \tau'_0, \sigma_0)$ are also indistinguishable under the DSqDH assumption. No need of randomization of the signatures. □

6.2.4 Bounded ART-Sign Scheme with Square Diffie-Hellman Tags (SqDH)

The above signature scheme limits to one-time signatures: only one signature can be generated for a given tag-index, otherwise signatures can be later forged on any message for this index, by linearity: the vector space spanned by (g, g^m) (in case of just one signature issued for one index) is just $(g^\alpha, g^{\alpha m})$ and the constraint of g for the first component implies $\alpha = 1$; on the other hand, the vector space spanned by (g, g^m) and $(g, g^{m'})$ (in case of two signatures issued for one index) is $\mathbb{G} \times \mathbb{G}$, and even the constraint of g for the first component does not limit anything for the second component.

This will be enough for our ABC application, as one usually has one attribute value for a specific kind of information (age, city, diploma, etc), but in practice this implies the signer to either keep track of all the indices already signed for one tag or to sign all the messages at once. We provide another kind of combinations, that could be applied on our SqDH signature scheme that will have interesting application to an ABC scheme.

Description of the Bounded SqDH-based ART-Sign Scheme.

We propose here an alternative where the limitation is on the total number n of messages signed for each tag by each signer:

Bounded SqDH-based ART-Sign Scheme

Setup(1^κ)**:** It extends the above Ephemerld-setup with the set of messages $\mathcal{M} = \mathbb{Z}_p$;

Keygen(param, n)**:** Given the public parameters param and a length n, it outputs the signing and verification keys

$$\begin{aligned} \mathsf{sk}_j &= [\ t,\ u,\ v,\ s_1,\ \ldots,\ s_{2n-1}\ ] \xleftarrow{\$} \mathbb{Z}_p^{2n+2}, \\ \mathsf{vk}_j = \mathfrak{g}^{\mathsf{sk}_j} &= [\ T,\ U,\ V,\ S_1,\ \ldots,\ S_{2n-1}\ ] \in \mathbb{G}_2^{2n+2}. \end{aligned}$$

Sign(sk_j, τ, m)**:** Given a signing key $\mathsf{sk}_j = [t, u, v, s_1, \ldots, s_{2n-1}]$, a message $m \in \mathbb{Z}_p$ and a public tag $\tau = (\tau_1, \tau_2, \tau_3)$, it outputs the signature

$$\sigma = \tau_1^{t+\sum_1^{2n-1} s_\ell m^\ell} \times \tau_2^u \times \tau_3^v.$$

AggrKey$(\{\mathsf{vk}_j\}_j)$**:** Given verification keys vk_j, it outputs the aggregated verification key $\mathsf{avk} = [\mathsf{vk}_j]_j$;

AggrSign$(\tau, (\mathsf{vk}_j, m_{j,i}, \sigma_{j,i})_{j,i})$**:** Given tuples of verification key vk_j, message $m_{j,i}$ and signature $\sigma_{j,i}$ all under the same tag τ, it outputs the signature $\sigma = \prod_{j,i} \sigma_{j,i}$ of the concatenation of the messages verifiable with $\mathsf{avk} \leftarrow \mathsf{AggrKey}(\{\mathsf{vk}_j\}_j)$;

DerivSign$(\mathsf{avk}, \tau, \boldsymbol{M}, \sigma, \rho_{\tau\to\tau'})$**:** Given a signature σ on tag τ and a message-set $\boldsymbol{M}$, and $\rho_{\tau\to\tau'}$ the randomization link between τ and another tag τ', it outputs $\sigma' = \sigma^{\rho_{\tau\to\tau'}}$;

RandSign$(\mathsf{avk}, \tau, \boldsymbol{M}, \sigma)$**:** The scheme being deterministic, it returns σ;

VerifSign$(\mathsf{avk}, \tau, \boldsymbol{M}, \sigma)$**:** Given a valid tag $\tau = (\tau_1, \tau_2, \tau_3)$, an aggregated verification key $\mathsf{avk} = [\mathsf{vk}_j]_j$ and a message-set $\boldsymbol{M} = [\mathsf{m}_j]_j$, with for each j, $\mathsf{m}_j = [m_{j,i}]_i$, and a

signature σ, one checks if the following equality holds or not, where $n_j = \#\{m_{j,i}\}$:

$$e(\sigma, \mathfrak{g}) = e\left(\tau_1, \prod_j T_j^{n_j} \times \prod_{\ell=1}^{2n-1} S_{j,\ell}^{\sum_i m_{j,i}^\ell}\right) \times e\left(\tau_2, \prod_j U_j^{n_j}\right) \times e\left(\tau_3, \prod_j V_j^{n_j}\right)$$

Recall that the validity of the tag has to be verified, as for the other version.

Security of the Bounded SqDH-based ART-Sign Scheme.

The linear homomorphism of the signature from [HPP20] still allows combinations. But when the number of signing queries is at most $2n$ per tag, the verification of the signature implies 0/1 coefficients only:

Theorem 52. *The bounded SqDH-based ART-Sign is unforgeable with a bounded number of signing queries per tag, even with adaptive corruptions of keys and tags, in both the generic group model and the random oracle model.*

Proof. As argued in [HPP20] and recalled in Theorem 35, when the bases of the tags are random, even if the exponents are known, the signature that would have signed messages $\boldsymbol{M} = (g^{m^1}, \ldots, g^{m^{2n-1}})$, for $m \in \mathbb{Z}_p$, is an unforgeable linearly-homomorphic signature. This means it is only possible to linearly combine signatures with the same tag. We fix the limit to n signatures σ_i queried on distinct messages m_i, for $i = 1, \ldots, n$ under vk_j: one can derive the signature $\sigma = \prod \sigma_i^{\alpha_i}$ on $\left(g^{\sum_i \alpha_i m_i^1}, \ldots, g^{\sum_i \alpha_i m_i^{2n-1}}\right)$. Whereas the forger claims this is a signature on $\left(g^{\sum_i a_i^1}, \ldots, g^{\sum_i \alpha_i a_i^n}\right)$, on $n_j \leq n$ values $a_1, \ldots, a_{n_j}$, as one cannot combine more than n attributes. Because of the constraint on τ_2, we additionally have $\sum \alpha_i = n_j \bmod p$:

$$\sum_{i=1}^{n} \alpha_i m_i^\ell = \sum_{i=1}^{n_j} a_i^\ell \bmod p \qquad \text{for } \ell = 0, \ldots, 2n-1$$

Let us first move on the left hand side the elements $a_k \in \{m_i\}$, with only $n' \leq n_j$ new elements, we assume to be the first ones, and we note $\beta_i = \alpha_i$ if $m_i \notin \{a_k\}$ and $\beta_i = \alpha_i - 1$ if $m_i \in \{a_k\}$:

$$\sum_{i=1}^{n} \beta_i m_i^\ell = \sum_{i=1}^{n'} a_i^\ell \bmod p \qquad \text{for } \ell = 0, \ldots, 2n-1$$

We thus have the system

$$\sum_{i=1}^{n} \beta_i m_i^\ell + \sum_{i=1}^{n'} \gamma_i a_i^\ell = 0 \bmod p \qquad \text{for } \ell = 0, \ldots, 2n-1, \text{ with } \gamma_i = -1$$

This is a system of $2n$ equations with at most $n + n' \leq 2n$ unknown values β_i's and γ_i's, and the Vandermonde matrix is invertible: $\beta_i = 0$ and $\gamma_i = 0$ for all index i. As a consequence, the vector $(\alpha_i)_i$ only contains 0 or 1 components.

This proves unforgeability, even with corruptions of the tags, but with a number of signed messages bounded by n. One can also consider corruptions of the signing keys, as they are all independent: one just needs to guess under which key will be generated the forgery. □

About unlinkability, it relies on the DSqDH assumption, with the same proof as the previous one-time scheme, except we can consider un-ordered message-sets $\boldsymbol{M}$:

Theorem 53. *The bounded SqDH-based ART-Sign, with message-sets, is unlinkable.*

A slightly more compact scheme is described in Appendix B.

6.3 Multi-Authority Anonymous Crendentials

In this section, we first define an anonymous attribute-based credential scheme, in the certified key setting (we assume a Certification Authority that first checks the knowledge of the secret keys before certifying public keys. The latter are then always checked before used by any players in the system). We assume that an identity id is associated (and included) to any vk, which is in turn included in sk. Then, we will show how to construct such a scheme based on EphemerId and ART-Sign schemes.

6.3.1 Definition

Our general definition supports multiple users $(\mathcal{U}_i)_i$ and multiple credential issuers $(\mathsf{CI}_j)_j$:

Definition 54 — Anonymous Credential

An anonymous credential system is defined by the following algorithms:

$\mathsf{Setup}(1^\kappa)$**:** It takes as input a security parameter and outputs the public parameters param;

$\mathsf{CIKeyGen}(\mathsf{ID})$**:** It generates the key pair $(\mathsf{sk}, \mathsf{vk})$ for the credential issuer with identity ID;

$\mathsf{UKeyGen}(\mathsf{id})$**:** It generates the key pair $(\mathsf{usk}, \mathsf{uvk})$ for the user with identity id;

$(\mathsf{CredObtain}(\mathsf{usk}, \mathsf{vk}, a), \mathsf{CredIssue}(\mathsf{uvk}, \mathsf{sk}, a))$**:** A user with identity id (associated to $(\mathsf{usk}, \mathsf{uvk})$) runs $\mathsf{CredObtain}$ to obtain a credential on the attribute a from the credential issuer ID (associated to $(\mathsf{sk}, \mathsf{vk})$) running $\mathsf{CredIssue}$. At the end of the protocol, the user receives a credential σ;

$\mathsf{CredAggr}(\mathsf{usk}, \{(\mathsf{vk}_j, a_j, \sigma_j)\}_j)$**:** It takes as input a secret key usk of a user and a list of credentials $(\mathsf{vk}_j, a_j, \sigma_j)$ and outputs a credential σ of the aggregation of the attributes;

$(\mathsf{CredShow}(\mathsf{usk}, \{(\mathsf{vk}_j, a_j)\}_j, \sigma), \mathsf{CredVerify}(\{(\mathsf{vk}_j, a_j)\}_j)$**:** In this two-party protocol, a user with identity id (associated to $(\mathsf{usk}, \mathsf{uvk})$) runs $\mathsf{CredShow}$ and interacts with a verifier running $\mathsf{CredVerify}$ to prove that he owns a valid credential σ on $\{a_j\}_j$ issued respectively by credential issuers ID_j (associated to $(\mathsf{sk}_j, \mathsf{vk}_j)$).

6.3.2 Security Model

The security model of anonymous credentials was already defined in various papers. We follow [FHS19, San20], with multi-show unlinkable credentials, but considering multiple credential issuers. Informally, the scheme needs to have the three properties:

- Correctness: the verifier must accept any credential obtained by an aggregation of honestly issued credentials on attributes;
- Unforgeability: the verifier should not accept a credential on a set of attributes for which the user did not obtain all the individual credentials for himself;
- Anonymity: credentials shown multiple times by a user should be unlinkable, even for the credential issuers. This furthermore implies that credentials cannot be linked to their owners.

For the two above security notions of unforgeability and anonymity, one can consider malicious adversaries able to corrupt some parties. We thus define the following lists: HU the list of honest user identities, CU the list of corrupted user identities, similarly we define HCI and CCI

for the honest/corrupted credential issuers. For a user identity id, we define $\mathsf{Att}[\mathsf{id}]$ the list of the attributes of id and $\mathsf{Cred}[\mathsf{id}]$ the list of his individual credentials obtained from the credential issuers. All these lists are initialized to the empty set. For both unforgeability and anonymity, the adversary has unlimited access to the oracles:

- $\mathcal{O}\mathsf{HCI}(\mathsf{ID})$ corresponds to the creation of an honest credential issuer with identity ID. If he already exists (i.e. $\mathsf{ID} \in \mathsf{HCI} \cup \mathsf{CCI}$), it outputs $\perp$. Otherwise, it adds $\mathsf{ID} \in \mathsf{HCI}$ and runs $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{CIKeyGen}(\mathsf{ID})$ and returns vk;
- $\mathcal{O}\mathsf{CCI}(\mathsf{ID}, \mathsf{vk})$ corresponds to the corruption of a credential issuer with identity ID and optionally public key vk. If he does not exist yet (i.e. $\mathsf{ID} \notin \mathsf{HCI} \cup \mathsf{CCI}$), it creates a new corrupted credential issuer with public key vk by adding ID to CCI. Otherwise, if $\mathsf{ID} \in \mathsf{HCI}$, it removes ID from HCI and adds it to CCI and outputs sk;
- $\mathcal{O}\mathsf{HU}(\mathsf{id})$ corresponds to the creation of an honest user with identity id. If the user already exists (i.e. $\mathsf{id} \in \mathsf{HU} \cup \mathsf{CU}$), it outputs $\perp$. Otherwise, it creates a new user by adding $\mathsf{id} \in \mathsf{HU}$ and running $(\mathsf{usk}, \mathsf{uvk}) \leftarrow \mathsf{UKeyGen}(\mathsf{id})$. It initializes $\mathsf{Att}[\mathsf{id}] = \{\}$ and $\mathsf{Cred}[\mathsf{id}] = \{\}$ and returns uvk;
- $\mathcal{O}\mathsf{CU}(\mathsf{id}, \mathsf{uvk})$ corresponds to the corruption of a user with identity id and optionally public key uvk. If the user does not exist yet (i.e. $\mathsf{id} \notin \mathsf{HU} \cup \mathsf{CU}$), it creates a new corrupted user with public key uvk by adding id to CU. Otherwise, if $\mathsf{id} \in \mathsf{HU}$, it removes id from HU and adds it to CU and outputs usk and all the associated credentials $\mathsf{Cred}[\mathsf{id}]$;
- $\mathcal{O}\mathsf{ObtIss}(\mathsf{id}, \mathsf{ID}, a)$ corresponds to the issuing of a credential from a credential issuer with identity ID (associated to $(\mathsf{sk}, \mathsf{vk})$) to a user with identity id (associated to $(\mathsf{usk}, \mathsf{uvk})$) on the attribute a. If $\mathsf{id} \notin \mathsf{HU}$ or $\mathsf{ID} \notin \mathsf{HCI}$, it outputs $\perp$. Otherwise, it runs $\sigma \leftarrow (\mathsf{CredObtain}(\mathsf{usk}, \mathsf{id}), \mathsf{CredIssue}(\mathsf{uvk}, \mathsf{sk}, a))$ and adds (ID, a) to $\mathsf{Att}[\mathsf{id}]$ and (ID, a, σ) to $\mathsf{Cred}[\mathsf{id}]$;
- $\mathcal{O}\mathsf{Obtain}(\mathsf{id}, \mathsf{ID}, a)$ corresponds to the issuing of a credential from the adversary playing the role of a malicious credential issuer with identity ID (associated to vk) to an honest user with identity id (associated to $(\mathsf{usk}, \mathsf{uvk})$) on the attribute a. If $\mathsf{id} \notin \mathsf{HU}$ or $\mathsf{ID} \notin \mathsf{CCI}$, it outputs $\perp$. Otherwise, it runs $\mathsf{CredObtain}(\mathsf{usk}, a)$ and adds (ID, a) to $\mathsf{Att}[\mathsf{id}]$ and (ID, a, σ) to $\mathsf{Cred}[\mathsf{id}]$;
- $\mathcal{O}\mathsf{Issue}(\mathsf{id}, \mathsf{ID}, a)$ corresponds to the issuing of a credential from an honest credential issuer with identity ID (associated to $(\mathsf{sk}, \mathsf{vk})$) to the adversary playing the role of a malicious user with identity id (associated to uvk) on the attribute a. If $\mathsf{id} \notin \mathsf{CU}$ or $\mathsf{ID} \notin \mathsf{HCI}$, it outputs $\perp$. Otherwise, it runs $\mathsf{CredIssue}(\mathsf{uvk}, \mathsf{sk}, a)$ and adds (ID, a) to $\mathsf{Att}[\mathsf{id}]$ and (ID, a, σ) to $\mathsf{Cred}[\mathsf{id}]$;
- $\mathcal{O}\mathsf{Show}(\mathsf{id}, \{(\mathsf{ID}_j, a_j)\}_j)$ corresponds to the showing by an honest user with identity id (associated to $(\mathsf{usk},\mathsf{uvk})$) of a credential on the set $\{(\mathsf{ID}_j, a_j)\}_j \subset \mathsf{Att}[\mathsf{id}]$. If $\mathsf{id} \notin \mathsf{HU}$, it outputs $\perp$. Otherwise, it runs $\mathsf{CredShow}(\mathsf{usk}, \{(\mathsf{vk}_j, a_j)\}_j, \sigma)$ with the adversary playing the role of a malicious verifier.

Definition 55 (Unforgeability)**.** An anonymous credential scheme is said unforgeable if, for any polynomial time adversary adversary $\mathcal{A}$ having access to $\mathcal{O} = \{\mathcal{O}\mathsf{HCI}, \mathcal{O}\mathsf{CCI}, \mathcal{O}\mathsf{HU}, \mathcal{O}\mathsf{CU}, \mathcal{O}\mathsf{ObtIss}, \mathcal{O}\mathsf{Issue}, \mathcal{O}\mathsf{Show}\}$, $\mathsf{Adv}^{\mathrm{unf}}(\mathcal{A}) = |\Pr[\mathsf{Exp}^{\mathrm{unf}}_{\mathcal{A}}(1^\kappa) = 1]|$ is negligible where

$\mathsf{Exp}^{\mathbf{unf}}_{\mathcal{A}}(1^\kappa)$**:**
$\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$
$\{(\mathsf{ID}_j, a_j)\}_j \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{param})$
$b \leftarrow (\mathcal{A}(), \mathsf{CredVerify}(\{(\mathsf{vk}_j, a_j)\}_j))$
If $\exists \mathsf{id} \in \mathsf{CU}$, $\forall j$, either $\mathsf{ID}_j \in \mathsf{CCI}$, or $\mathsf{ID}_j \in \mathsf{HCI}$ and $(\mathsf{ID}_j, a_j) \in \mathsf{Att}[\mathsf{id}]$,
  then return 0
Return b

Intuitively, the adversary wins the security game if it manages to prove its ownership of a credential, on behalf of a corrupted user $\mathsf{id} \in \mathsf{CU}$ whereas this user did not ask the attributes to the honest credential issuers. Note that attributes from the corrupted credential issuers can be generated by the adversary itself, using the secret keys.

Definition 56 (Anonymity)**.** An anonymous credential scheme is said anonymous if, for any polynomial time adversary $\mathcal{A}$ having access to $\mathcal{O} = \{\mathcal{O}\mathsf{HCI}, \mathcal{O}\mathsf{CCI}, \mathcal{O}\mathsf{HU}, \mathcal{O}\mathsf{CU}, \mathcal{O}\mathsf{Obtain}, \mathcal{O}\mathsf{Show}\}$, $\mathsf{Adv}^{\mathrm{ano}}(\mathcal{A}) = |\Pr[\mathsf{Exp}^{\mathrm{ano}-1}_{\mathcal{A}}(1^\kappa) = 1] - \Pr[\mathsf{Exp}^{\mathrm{ano}-0}_{\mathcal{A}}(1^\kappa) = 1]|$ is negligible where

$\mathsf{Exp}^{\mathbf{ano}-b}_{\mathcal{A}}(1^\kappa)$**:**
$\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$
$(\mathsf{id}_0, \mathsf{id}_1, \{(\mathsf{ID}_j, a_j)\}_j) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{param})$
If for some ID_j, $(\mathsf{ID}_j, a_j) \notin \mathsf{Att}[\mathsf{id}_0] \cap \mathsf{Att}[\mathsf{id}_1]$, then return 0
$(\mathsf{CredShow}(\mathsf{usk}_b, \{a_j\}_j, \sigma), \mathcal{A}())$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}}()$
If $\mathsf{id}_0 \in \mathsf{CU}$ or $\mathsf{id}_1 \in \mathsf{CU}$, then return 0
Return b^*

First, note that we do not hide the attributes nor the issuers during the showing, but just the user, as we want to prove their ownership by the anonymous user. Intuitively, the adversary wins the security game if it can distinguish showings from users id_0 and id_1 of its choice, on the same set of attributes $\{(\mathsf{ID}_j, a_j)\}_j$, even after having verified credentials from the two identities, as it has access to the oracle $\mathcal{O}\mathsf{Show}$. Note that contrarily to [San20], unless the attributes contain explicit ordering (as it will be the case with our first construction), we are dealing with unlinkability as soon as the sets of attributes are the same for the two players (with the second construction).

6.3.3 Anonymous Credential from EphemerId and ART-Sign Scheme

Let $\mathcal{E}$ be an EphemerId scheme and $\mathsf{S}^{\mathrm{art}}$ an ART-Sign scheme, one can construct an anonymous attribute-based credential scheme. The user's keys will be tag pairs and the credentials will be ART-Sign signatures on both the tags and the attributes. Since the signature is aggregatable and the tag is randomizable, the user can anonymously show any aggregation of credentials:

Anonymous Credential from EphemerId and ART-Sign Scheme

$\mathsf{Setup}(1^\kappa)$**:** Given a security parameter κ, it runs $\mathsf{S}^{\mathrm{art}}.\mathsf{Setup}$ and outputs the public parameters param which includes all the parameters;

$\mathsf{CIKeyGen}(\mathsf{ID})$**:** Credential issuer CI with identity ID, runs $\mathsf{S}^{\mathrm{art}}.\mathsf{Keygen}(\mathsf{param})$ to obtain his key pair $(\mathsf{sk}, \mathsf{vk})$;

$\mathsf{UKeyGen}(\mathsf{id})$**:** User $\mathcal{U}$ with identity id, runs $\mathcal{E}.\mathsf{GenTag}(\mathsf{param})$ to obtain his key pair $(\mathsf{usk}, \mathsf{uvk})$. In the case witnesses are required for the signatures, $(\mathsf{usk}, \mathsf{uvk})$ are provided to the credential issuers;

$(\mathsf{CredObtain}(\mathsf{usk}, a), \mathsf{CredIssue}(\mathsf{uvk}, \mathsf{sk}, a))$**:** User $\mathcal{U}$ with identity id and key-pair $(\mathsf{usk}, \mathsf{uvk})$ asks the credential issuer CI for a credential on attribute a: $\sigma = \mathsf{S}^{\mathrm{art}}.\mathsf{Sign}(\mathsf{sk}, \mathsf{uvk}, a)$;

$\mathsf{CredAggr}(\mathsf{usk}, \{(\mathsf{vk}_j, a_j, \sigma_j)\}_j)$**:** Given credentials σ_j on attributes (ID_j, a_j) under the same user key uvk, it outputs the signature $\sigma = \mathsf{S}^{\mathrm{art}}.\mathsf{AggrSign}(\mathsf{uvk}, \{(\mathsf{vk}_j, a_j, \sigma_j)\}_j)$ on the set of attributes $\{a_j\}_j$ under uvk and the aggregated verification key avk of all the vk_j;

$(\mathsf{CredShow}(\mathsf{usk}, \{(\mathsf{vk}_j, a_j)\}_j, \sigma), \mathsf{CredVerify}(\{(\mathsf{vk}_j, a_j)\}_j)$**:** User $\mathcal{U}$ randomizes his public key $(\mathsf{uvk}', \rho) = \mathcal{E}.\mathsf{RandTag}(\mathsf{uvk})$ and computes the aggregated key avk =

$\mathsf{S}^{\mathrm{art}}.\mathsf{AggrKey}(\{\mathsf{vk}_j\}_j)$. Then, it adapts the secret key $\mathsf{usk}' = \mathcal{E}.\mathsf{DerivWitness}(\mathsf{usk}, \rho)$ as well as the aggregated signature $\sigma' = \mathsf{S}^{\mathrm{art}}.\mathsf{DerivSign}(\mathsf{avk}, \mathsf{uvk}, \{a_j\}_j, \sigma, \rho)$ and randomizes it:
$\sigma'' = \mathsf{S}^{\mathrm{art}}.\mathsf{RandSign}(\mathsf{avk}, \mathsf{uvk}', \{a_j\}_j, \sigma')$. Finally, it sends to the verifier $\mathcal{V}$ the anonymous credential $(\mathsf{avk}, \{a_j\}_j, \mathsf{uvk}', \sigma'')$. The verifier first checks the freshness of the credential with a proof of ownership of uvk' using the interactive protocol $(\mathcal{E}.\mathsf{ProveKTag}(\mathsf{usk}'), \mathcal{E}.\mathsf{VerifKTag}(\mathsf{uvk}'))$ and then verifies the validity of the credential with
$\mathsf{S}^{\mathrm{art}}.\mathsf{VerifSign}(\mathsf{avk}, \mathsf{uvk}', \{a_j\}_j, \sigma'')$.

If one considers corruptions, when one corrupts a user, his secret key is provided, when one corrupts a credential issuer, his secret key is provided.

By replacing all the algorithms by their instantiations for the proposed constructions of EphemerId and ART-Sign schemes, we obtain our constructions of anonymous attribute-based credential schemes. The SqDH construction uses an aggregate signature with (public) randomizable tag, and unforgeability holds even if the witnesses are known. As a consequence, this construction allows corruption of the Credential Issuers and of the users.

Theorem 57. *Assuming EphemerId achieves knowledge soundness and ART-Sign is unforgeable, the generic construction is an unforgeable attribute-based credential scheme, in the certified key model.*

Proof. Let $\mathcal{A}$ be an adversary against the unforgeability of our anonymous credential scheme. We build an adversary $\mathcal{B}$ against the unforgeability of the ART-Sign. As we are in the certified key model, even for the corrupted players, the simulator knows the secret keys, as they can be extracted at the certification time. Our adversary $\mathcal{B}$ runs the unforgeability security game of the ART-Sign, and answers the oracle queries asked by $\mathcal{A}$ as follows:

- $\mathcal{O}\mathsf{HCI}(\mathsf{ID})$: If $\mathsf{ID} \in \mathsf{HCI} \cup \mathsf{CCI}$, $\mathcal{B}$ outputs $\perp$. Otherwise, it adds $\mathsf{ID} \in \mathsf{HCI}$, asks the query $\mathcal{O}\mathsf{Keygen}()$ and forwards the answer to $\mathcal{A}$;
- $\mathcal{O}\mathsf{CCI}(\mathsf{ID}, \mathsf{vk})$: If $\mathsf{ID} \notin \mathsf{HCI} \cup \mathsf{CCI}$, $\mathcal{B}$ adds $\mathsf{ID} \in \mathsf{CCI}$. Otherwise, if $\mathsf{ID} \in \mathsf{HCI}$ with keys $(\mathsf{sk}, \mathsf{vk})$, it moves ID from HCI to CCI. It then asks the query $\mathcal{O}\mathsf{Corrupt}(\mathsf{vk})$ and forwards the answer to $\mathcal{A}$;
- $\mathcal{O}\mathsf{HU}(\mathsf{id})$: If $\mathsf{id} \in \mathsf{HU} \cup \mathsf{CU}$, $\mathcal{B}$ outputs $\perp$. Otherwise, it adds $\mathsf{id} \in \mathsf{HU}$, asks the query $\mathcal{O}\mathsf{GenTag}()$ and forwards the answer to $\mathcal{A}$;
- $\mathcal{O}\mathsf{CU}(\mathsf{id}, \mathsf{uvk})$: If $\mathsf{id} \notin \mathsf{HU} \cup \mathsf{CU}$, $\mathcal{B}$ adds $\mathsf{id} \in \mathsf{CU}$. Otherwise, if $\mathsf{id} \in \mathsf{HU}$ with keys $(\mathsf{usk}, \mathsf{uvk})$, it moves id from HU to CU, asks the query $\mathcal{O}\mathsf{CorruptTag}(\mathsf{uvk})$ and forwards the answer to $\mathcal{A}$;
- $\mathcal{O}\mathsf{ObtIss}(\mathsf{id}, \mathsf{ID}, a)$: If $\mathsf{id} \notin \mathsf{HU}$ or $\mathsf{ID} \notin \mathsf{HCI}$, $\mathcal{B}$ outputs $\perp$. Otherwise, id is associated to $(\mathsf{usk}, \mathsf{uvk})$ and ID is associated to $(\mathsf{sk}, \mathsf{vk})$. Then $\mathcal{B}$ asks the query $\mathcal{O}\mathsf{Sign}(\mathsf{vk}, \mathsf{uvk}, a)$, adds (ID, a) to $\mathsf{Att}[\mathsf{id}]$ and (ID, a, σ) to $\mathsf{Cred}[\mathsf{id}]$ and outputs σ.
- $\mathcal{O}\mathsf{Obtain}(\mathsf{id}, \mathsf{ID}, a)$: If $\mathsf{id} \notin \mathsf{HU}$ or $\mathsf{ID} \notin \mathsf{CCI}$, $\mathcal{B}$ outputs $\perp$. Otherwise, id is associated to $(\mathsf{usk}, \mathsf{uvk})$ and ID is associated to $(\mathsf{sk}, \mathsf{vk})$. Then $\mathcal{B}$ runs $\sigma = \mathsf{Sign}(\mathsf{sk}, \mathsf{uvk}, a)$ and adds (ID, a) to $\mathsf{Att}[\mathsf{id}]$ and (ID, a, σ) to $\mathsf{Cred}[\mathsf{id}]$;
- $\mathcal{O}\mathsf{Issue}(\mathsf{id}, \mathsf{ID}, a)$: If $\mathsf{id} \notin \mathsf{CU}$ or $\mathsf{ID} \notin \mathsf{HCI}$, $\mathcal{B}$ outputs $\perp$. Otherwise, id is associated to $(\mathsf{usk}, \mathsf{uvk})$ and ID is associated to $(\mathsf{sk}, \mathsf{vk})$. Then $\mathcal{B}$ runs $\sigma = \mathsf{Sign}(\mathsf{sk}, \mathsf{uvk}, a)$ and adds (ID, a) to $\mathsf{Att}[\mathsf{id}]$ and (ID, a, σ) to $\mathsf{Cred}[\mathsf{id}]$;

- $\mathcal{O}\mathsf{Show}(\mathsf{id}, \{(\mathsf{ID}_j, a_j)\}_j)$: If $\mathsf{id} \notin \mathsf{HU}$ or $\{(\mathsf{ID}_j, a_j)\}_j) \not\subset \mathsf{Att}[\mathsf{id}]$, $\mathcal{B}$ outputs $\perp$. Otherwise, id is associated to $(\mathsf{usk}, \mathsf{uvk})$ and each ID_j is associated to $(\mathsf{sk}_j, \mathsf{vk}_j)$. Furthermore, for each (ID_j, a_j), there is σ_j such that $(\mathsf{ID}_j, a_j, \sigma_j) \in \mathsf{Cred}[\mathsf{id}]$. Then $\mathcal{B}$ first randomizes the key uvk with $(\mathsf{uvk}', \rho) = \mathcal{E}.\mathsf{RandTag}(\mathsf{uvk})$, computes the aggregated key $\mathsf{avk} = \mathsf{S}^{\mathrm{art}}.\mathsf{AggrKey}(\{\mathsf{vk}_j\}_j)$ and adapts the secret key $\mathsf{usk}' = \mathcal{E}.\mathsf{DerivWitness}(\mathsf{usk}, \rho)$. From the obtained credentials σ_j, it computes the aggregated signature $\sigma = \mathsf{S}^{\mathrm{art}}.\mathsf{AggrSign}(\mathsf{uvk}, \{(\mathsf{vk}_j, a_j, \sigma_j)\}_j)$, adapts it: $\sigma' = \mathsf{S}^{\mathrm{art}}.\mathsf{DerivSign}(\mathsf{avk}, \mathsf{uvk}, \{a_j\}_j, \sigma, \rho)$, and randomizes it: $\sigma'' = \mathsf{S}^{\mathrm{art}}.\mathsf{RandSign}(\mathsf{avk}, \mathsf{uvk}', \{a_j\}_j, \sigma')$. $\mathcal{B}$ outputs $(\mathsf{avk}, \{a_j\}_j, \mathsf{uvk}', \sigma'')$ and makes the $\mathcal{E}.\mathsf{ProveKTag}(\mathsf{usk}')$ part of the interactive proof of ownership.

Eventually, the adversary $\mathcal{A}$ runs a showing for $\{(\mathsf{vk}_j, a_j)\}_j$, with a credential $(\mathsf{avk}, \{a_j\}_j, \mathsf{uvk}^*, \sigma^*)$ and a proof of knowledge of usk^* associated to uvk^*: in case of success, $\mathcal{B}$ outputs the signature $(\mathsf{avk}, \{a_j\}_j, \mathsf{uvk}^*, \sigma^*)$.

In case of validity of the showing, except with negligible probability,

- from the knowledge soundness of the $\mathsf{EphemerId}$ scheme, this means there is $\mathsf{id} \in \mathsf{CU}$, associated to $(\mathsf{usk}, \mathsf{uvk})$, with $\mathsf{uvk} \sim \mathsf{uvk}^*$;

- from the unforgeability of the aggregate signature with randomizable tags, all the tags a_j's have been signed for uvk and vk. These individual credentials have thus been issued either by the adversary on behalf of a corrupted credential issuer $\mathsf{ID}_j \in \mathsf{CCI}$ or from an oracle query to ID_j for id.

This is thus a legitimate showing with overwhelming probability: $\mathcal{B}$ win with negligible probability. Hence, the adversary $\mathcal{A}$ can only win with negligible probability. □

As explained above, the security relies on both the soundness of the $\mathsf{EphemerId}$ scheme and the unforgeability of the aggregate signature with randomizable tags. In our construction, the witness is not needed for signing, and unforgeability of the $\mathsf{ART\text{-}Sign}$ holds even if the witnesses are all known to the adversary. Hence, corruption of users would just help to run the proof of knowledge of the witnesses, and corruption of credential issuers for the issuing of credentials, which would not help for forgeries (in the above security model). Of course, we also have to take care of the way keys are generated and the number of signatures that will be issued to guarantee the unforgeability.

Theorem 58. *Assuming $\mathsf{EphemerId}$ is zero-knowledge and $\mathsf{ART\text{-}Sign}$ is unlinkable, the generic construction is an anonymous attribute-based credential scheme, in the certified key model.*

Proof. From the unlinkability of the $\mathsf{ART\text{-}Sign}$, the tuple $(\mathsf{avk}, \boldsymbol{M}, \tau', \sigma'')$ does not leak any information about the initial tag τ. Hence, a credential does not leak any information about uvk_b. In addition, if the proof of knowledge of the witness is zero-knowledge, it does not leak any information about uvk_b either. □

6.4 SqDH-based Anonymous Credentials

Thanks to our aggregated signatures that tolerate corruptions of users and signers, we will be able to consider corruptions of users and credential issuers, and even possible collusions. In the first construction, we consider attributes where the index i determines the topic (age, city, diploma) and the exact value is encoded in $a_i \in \mathbb{Z}_p^*$ (possibly $\mathcal{H}(m) \in \mathbb{Z}_p^*$ if the value is a large bitstring), or 0 when empty. The second construction will not require any such ordering on the attributes. Free text will be possible.

6.4.1 The Basic SqDH-based Anonymous Credential Scheme

The basic construction directly follows the instantiation of the above construction with the SqDH-based ART-Sign:

Basic SqDH-based ART-Sign Scheme

Setup(1^κ): Given a security parameter κ, let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ be an asymmetric bilinear setting, where g and $\mathfrak{g}$ are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We then define $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e, \mathcal{H})$, where $\mathcal{H}$ is an hash function in $\mathbb{G}_1$;

CIKeyGen(ID): Credential issuer CI with identity ID, generates its keys for n kinds of attributes

$$\begin{array}{l} \mathsf{sk}_j = (\ \mathsf{SK}_j = [\ t,\ u,\ v\ ],\ \mathsf{SK}'_{j,i} = [\ r_i,\ s_i\ ]_i\ ) \xleftarrow{\$} \mathbb{Z}_p^{3+2n}, \\ \mathsf{vk}_j = (\ \mathsf{VK}_j = [\ \mathfrak{g}^t,\ \mathfrak{g}^u,\ \mathfrak{g}^v\ ],\ \mathsf{VK}'_{j,i} = [\ \mathfrak{g}^{r_i},\ \mathfrak{g}^{s_i}\ ]_i\ ) \in \mathbb{G}_2^{3+2n}. \end{array}$$

More keys for new attributes can be generated on-demand: by adding the pair $[r, s] \xleftarrow{\$} \mathbb{Z}_p^2$ to the secret key and $[\mathfrak{g}^r, \mathfrak{g}^s]$ to the verification key, the keys can works on $n+1$ kinds of attributes;

UKeyGen(id): User $\mathcal{U}$ with identity id, sets $h = \mathcal{H}(\mathsf{id}) \in \mathbb{G}_1^*$, generates its secret tag $\tilde{\tau} \xleftarrow{\$} \mathbb{Z}_p^*$ jointly with CA and computes $\tau = (h, h^{\tilde{\tau}}, h^{\tilde{\tau}^2}) \in \mathbb{G}_1^3$: $\mathsf{usk} = \tilde{\tau}$ and $\mathsf{uvk} = \tau = (h, h^{\tilde{\tau}}, h^{\tilde{\tau}^2})$;

(CredObtain(usk, a_i), CredIssue(uvk, sk, a_i)): User $\mathcal{U}$ with identity id and $\mathsf{uvk} = (\tau_1, \tau_2, \tau_3)$ asks to the credential issuer CI for a credential on the attribute a_i: $\sigma = \tau_1^{t+r_i+a_i s_i} \times \tau_2^u \times \tau_3^v$. The credential issuer uses the appropriate index i, making sure this is the first signature for this index;

CredAggr(usk, $\{(\mathsf{VK}_j, \mathsf{VK}'_{j,i}, a_{j,i}, \sigma_{j,i})\}_{j,i}$): Given credentials $\sigma_{j,i}$ on attributes $(\mathsf{ID}_j, a_{j,i})$ under the same user key uvk, it outputs the signature $\sigma = \prod_{j,i} \sigma_{j,i}$;

(CredShow(usk, $\{(\mathsf{VK}_j, \mathsf{VK}'_{j,i}, a_{j,i})\}_{j,i}, \sigma$), CredVerify($\{(\mathsf{VK}_j, \mathsf{VK}'_{j,i}, a_{j,i})\}_{j,i}$):

First, user $\mathcal{U}$ randomizes his public key with a random $\rho \xleftarrow{\$} \mathbb{Z}_p^*$ into $\mathsf{uvk}' = (\tau_1^\rho, \tau_2^\rho, \tau_3^\rho)$, concatenates the keys $\mathsf{avk} = \cup_j([\mathsf{VK}_j] \cup [\mathsf{VK}'_{j,i}]_i)$, and adapts the signature $\sigma' = \sigma^\rho$. Then it sends the anonymous credential $(\mathsf{avk}, \{a_{j,i}\}_{j,i}, \mathsf{uvk}', \sigma')$ to the verifier. The latter first checks the freshness of the credential with a proof of ownership and validity of uvk′ using a Schnorr-like interactive proof and then verifies the validity of the credential: with $n_j = \#\{\mathsf{VK}'_{j,i}\}$:

$$e(\sigma, \mathfrak{g}) = e\left(\tau_1, \prod_j \mathsf{VK}_{j,1}{}^{n_j} \times \prod_i \mathsf{VK}'_{j,i,1} \cdot \mathsf{VK}'_{j,i,2}{}^{a_{j,i}}\right) \times e\left(\tau_2, \prod_j \mathsf{VK}_{j,2}{}^{n_j}\right) \times e\left(\tau_3, \prod_j \mathsf{VK}_{j,3}{}^{n_j}\right).$$

We stress that for the unforgeability of the signature, generator h for each tag must be random, and so it is generated as $\mathcal{H}(\mathsf{id})$, with a hash function $\mathcal{H}$ in $\mathbb{G}_1$ that we model as a random oracle in the security proof. This way, the credential issuers will automatically know the basis for each user. There is no privacy issue as this basis is randomized when used in an anonymous credential. Moreover, the user needs his secret key $\tilde{\tau}$ to be random. Therefore, he jointly generates $\tilde{\tau}$ with the Certification Authority (see Appendix A). During the showing of a credential, the user has to prove the knowledge of the witness for the validity of the tag. This

is thus an interactive protocol. In this construction, we can consider a polynomial number n of attributes per credential issuer, where a_i is associated to key $\mathsf{vk}_{j,i}$ of the Credential Issuer CI_j. Again, to keep the unforgeability of the signature, the credential issuer should provide at most one attribute per key $\mathsf{vk}_{j,i}$ for a given tag. At the showing time, for proving the ownership of k attributes (possibly from K different credential issuers), the users has to perform $k-1$ multiplications in $\mathbb{G}_1$ to aggregate the credentials into one, and 4 exponentiations in $\mathbb{G}_1$ for randomization, but just one element from $\mathbb{G}_1$ is sent, as anonymous credential, plus an interactive Schnorr-like proof of SqDH-tuple with knowledge of usk (see 4.4: 2 exponentiations in $\mathbb{G}_1$, 2 group elements from $\mathbb{G}_1$, and a scalar in $\mathbb{Z}_p$); whereas the verifier first has to perform 4 exponentiations and 2 multiplications in $\mathbb{G}_1$ for the proof of validity/knowledge of usk, and less than $3k$ multiplications and k exponentiations in $\mathbb{G}_2$, and 3 pairings to check the credential. While this is already better than [CL11], we can get a better construction.

6.4.2 A Compact SqDH-based Anonymous Credential Scheme

Instead of having a specific key $\mathsf{VK}'_{j,i}$ for each family of attributes $a_{j,i}$, and thus limiting to one issuing per family of attributes for each user, we can use the bounded SqDH-based ART-Sign, with free-text attributes: we consider $2n-1$ keys, where n is the maximum number of attributes issued for one user by a credential issuer, whatever the attributes are:

Bounded SqDH-based ART-Sign Scheme

Setup(1^κ): Given a security parameter κ, let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ be an asymmetric bilinear setting, where g and $\mathfrak{g}$ are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We then define $\mathsf{param} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e, \mathcal{H})$, where $\mathcal{H}$ is an hash function in $\mathbb{G}_1$;

CIKeyGen(ID): Credential issuer CI with identity ID, generates its keys for n maximum attributes per user

$$\begin{array}{rcccccccc} \mathsf{sk}_j & = [& t, & u, & v, & s_1, & \ldots, & s_{2n-1} &] \xleftarrow{\$} \mathbb{Z}_p^{2n+2}, \\ \mathsf{vk}_j = \mathfrak{g}^{\mathsf{sk}_j} & = [& T, & U, & V, & S_1, & \ldots, & S_{2n-1} &] \in \mathbb{G}_2^{2n+2}. \end{array}$$

UKeyGen(id): User $\mathcal{U}$ with identity id, sets $h = \mathcal{H}(\mathsf{id}) \in \mathbb{G}_1^*$, generates its random secret tag $\tilde{\tau} \xleftarrow{\$} \mathbb{Z}_p^*$ jointly with CA and computes $\tau = (h, h^{\tilde{\tau}}, h^{\tilde{\tau}^2}) \in \mathbb{G}_1^3$: $\mathsf{usk} = \tilde{\tau}$ and $\mathsf{uvk} = \tau = (h, h^{\tilde{\tau}}, h^{\tilde{\tau}^2})$;

(CredObtain(usk, a), CredIssue(uvk, sk, a)): User $\mathcal{U}$ with identity id and $\mathsf{uvk} = (\tau_1, \tau_2, \tau_3)$ asks to the credential issuer CI for a credential on the attribute a: $\sigma = \tau_1^{t+\sum_{\ell=1}^{2n-1} s_\ell a^\ell} \times \tau_2^u \times \tau_3^v$. Note that $a \in \mathbb{Z}_p^*$, so it can be a hash value of the actual free-text attribute;

CredAggr(usk, $\{(\mathsf{vk}_j, a_{j,i}, \sigma_{j,i})\}_{j,i}$): Given credentials $\sigma_{j,i}$ on attributes $(\mathsf{ID}_j, a_{j,i})$ under the same user key uvk, it outputs the signature $\sigma = \prod_{j,i} \sigma_{j,i}$;

(CredShow(usk, $\{(\mathsf{vk}_j, a_{j,i})\}_{j,i}, \sigma$), CredVerify($\{(\mathsf{vk}_j, a_{j,i})\}_{j,i}$): First, a user $\mathcal{U}$ randomizes his public key with a random $\rho \xleftarrow{\$} \mathbb{Z}_p^*$, $\mathsf{uvk}' = (\tau_1^\rho, \tau_2^\rho, \tau_3^\rho)$, concatenates the keys $\mathsf{avk} = \cup_j[\mathsf{vk}_j]$, and adapts the signature $\sigma' = \sigma^\rho$. Then it sends the anonymous credential $(\mathsf{avk}, \{a_{j,i}\}_{j,i}, \mathsf{uvk}', \sigma')$ to the verifier. The latter first checks the freshness of the credential with a proof of ownership and validity of uvk' using a Schnorr-like interactive proof and then verifies the validity of the credential: with $n_j = \#\{a_{j,i}\}$:

$$e(\sigma, \mathfrak{g}) = e\left(\tau_1, \prod_j T_j^{n_j} \prod_{\ell=1}^{2n-1} S_{j,\ell}^{\sum_i a_{j,i}^\ell}\right) \times e\left(\tau_2, \prod_j U_j^{n_j}\right) \times e\left(\tau_3, \prod_j V_j^{n_j}\right)$$

Again, we stress that for the unforgeability of the signature, generator h for each tag and $\tilde{\tau}$ must be random. And the credential issuer should provide at most n attributes per user, even if in this construction, we can consider an exponential number N of attributes per credential issuer, as $a_{j,i}$ is any scalar in $\mathbb{Z}_p^*$. More concretely, $a_{j,i}$ can be given as the output of a hash function into $\mathbb{Z}_p$ from any bitstring. At the showing time, for proving the ownership of k attributes (possibly from K different credential issuers), the users has to perform $k-1$ multiplications in $\mathbb{G}_1$ to aggregate the credentials into one, and 4 exponentiations in $\mathbb{G}_1$ for randomization, but just one group element for $\mathbb{G}_1$ is sent, as anonymous credential, plus an interactive Schnorr-like proof of SqDH-tuple with knowledge of usk (see 4.4: 2 exponentiations in $\mathbb{G}_1$, 2 group elements from $\mathbb{G}_1$, and a scalar in $\mathbb{Z}_p$); whereas the verifier first has to perform 4 exponentiations and 2 multiplications in $\mathbb{G}_1$ for the proof of validity/knowledge of usk, and less than $2n \cdot (K + 3k)$ multiplications in $\mathbb{G}_2$, $2n \cdot k$ exponentiations in $\mathbb{G}_2$ and 3 pairings to check the credential.

In the particular case of just one credential issuer with verification key $\mathsf{vk} = (T, U, V, [S_i]_{i=1}^{2n-1})$, the verification of the credential σ on the k attributes $\{a_i\}$ just consists of

$$e(\sigma, \mathfrak{g}) = e\left(\tau_1, T^k \prod_{\ell=1}^{2n-1} S_\ell^{\sum_i a_i^\ell}\right) \times e\left(\tau_2, U^k\right) \times e\left(\tau_3, V^k\right).$$

The communication is of constant size (one group element in $\mathbb{G}_1$). We stress that n is just a limit of the maximal number of attributes issued by the credential issuer for one user but the universe of the possible attributes is exponentially large, and there is no distinction between the families of attributes.

6.5 Traceable Anonymous Credentials

As the SqDH-based ART-Sign schemes provide computational unlinkability only, it opens the door for possible traceability in case of abuse, with anonymous but traceable tags:

Definition 59 — Traceable EphemerId

This is an extension of an EphemerId scheme with a modified GenTag algorithm and an additional TraceId one:

GenTag(1^κ)**:** Given a security parameter 1^κ, it outputs the user-key pair (usk,uvk) and the tracing key utk;

TraceId(utk, uvk′)**:** Given the tracing key utk associated to uvk and a public key uvk′, it outputs a proof π of whether uvk $\sim$ uvk′ or not.

JudgeId(uvk, uvk′, π)**:** two public keys and a proof, the judge checks the proof π and outputs 1 if it is correct.

Providing the tracing keys to a tracing authority during the key generation for the users will allow traceability.

6.5.1 Traceable Anonymous Credentials

For traceability, we need an additional player: the *tracing authority*. During the user's key generation, this tracing authority will either be the certification authority, or a second authority, that also has to certify user's key uvk once it has received the tracing key utk.

In case of abuse of a credential σ under anonymous key uvk′, a tracing algorithm outputs the initial uvk and id, with a proof a correct tracing. A new security notion is quite important: *non-frameability*, which means that the tracing authority should not be able to declare guilty a wrong user: only correct proofs are accepted by the judge. We consider a non-interactive

proof of tracing, produced by the TraceId algorithm and verified by anybody using the JudgeId algorithm. This proof could be interactive.

6.5.2 Traceable SqDH-based Anonymous Credentials

With our Square Diffie-Hellman based EphemerId scheme where $\mathsf{uvk} = \tau = (h, h^{\tilde{\tau}}, h^{\tilde{\tau}^2})$ in an asymmetric bilinear setting $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, \mathfrak{g}, e)$ where g and $\mathfrak{g}$ are random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, $\mathsf{usk} = \tilde{\tau}$ and $\mathsf{utk} = \mathfrak{g}^{\tilde{\tau}}$. The latter tracing key indeed allows to check whether $\tau' \sim \tau$ or not: $e(\tau'_1, \mathsf{utk}) = e(\tau'_2, \mathfrak{g})$ and $e(\tau'_2, \mathsf{utk}) = e(\tau'_3, \mathfrak{g})$. If one already knows the tags are valid (SqDH tuples), this is enough to verify whether $e(\tau'_1, \mathsf{utk}) = e(\tau'_2, \mathfrak{g})$ holds or not. But we provide the complete proof, as it is already quite efficient: in order to prove it, the TraceId algorithm can use a Groth-Sahai proof as shown in 6.5.3 that proves, in a zero-knowledge way, the existence of utk such that

$$e(\tau_1, \mathsf{utk}) = e(\tau_2, \mathfrak{g}) \qquad e(\tau_2, \mathsf{utk}) = e(\tau_3, \mathfrak{g})$$
$$e(\tau'_1, \mathsf{utk}) = e(\tau'_2, \mathfrak{g}) \qquad e(\tau'_2, \mathsf{utk}) = e(\tau'_3, \mathfrak{g}).$$

The first line proves that utk is the good tracing key for $\mathsf{uvk} = \tau$, and the second line shows it applies to $\mathsf{uvk}' = \tau'$ too. These are the equations verified by JudgeId algorithm. This can also be a proof of innocence of id with key uvk if the first line is satisfied while the second one is not.

With such a proof, the tracing authority cannot frame a user. We thus have a secure traceable anonymous credential scheme. Note however that, since we let the user choose the secret key $\tilde{\tau}$ in GenTag, one user could decide to use the same as another user. Either the tracing authority first checks that, using the new tracing key on all the previous tags, and reject, or this is considered a collusion of users, and at the tracing time, both users will be accused.

6.5.3 Groth-Sahai Proof for Square Diffie-Hellman Tracing

For the proof of tracing, one wants to show $\tau' \sim \tau$, where τ is the reference tag for a user (certified at the registration time). With the tracing key $\mathsf{utk} = \mathfrak{g}^{\tilde{\tau}}$, one needs to show

$$e(\tau_1, \mathsf{utk}) = e(\tau_2, \mathfrak{g}) \qquad e(\tau_2, \mathsf{utk}) = e(\tau_3, \mathfrak{g})$$
$$e(\tau'_1, \mathsf{utk}) = e(\tau'_2, \mathfrak{g}) \qquad e(\tau'_2, \mathsf{utk}) = e(\tau'_3, \mathfrak{g})$$

but without revealing $\mathsf{utk} \in \mathbb{G}_2$. This is equivalent, for random $\alpha_1, \alpha_2, \alpha'_1, \alpha'_2 \xleftarrow{\$} \mathbb{Z}_p$, to have:

$$e(T_1, \mathsf{utk}) = e(T_2, \mathfrak{g}) \qquad \text{with} \qquad T_1 = {\tau_1}^{\alpha_1} \cdot {\tau_2}^{\alpha_2} \cdot {\tau'_1}^{\alpha'_1} \cdot {\tau'_2}^{\alpha'_2}$$
$$T_2 = {\tau_2}^{\alpha_1} \cdot {\tau_3}^{\alpha_2} \cdot {\tau'_2}^{\alpha'_1} \cdot {\tau'_2}^{\alpha'_2}$$

One can commit utk: as above, with the reference string $(\mathfrak{v}_{1,1}, \mathfrak{v}_{1,2}, \mathfrak{v}_{2,1}, \mathfrak{v}_{2,2}) \in \mathbb{G}_2^4$, such that $(\mathfrak{v}_{1,1}, \mathfrak{v}_{1,2}, \mathfrak{v}_{2,1}, \mathfrak{v}_{2,2})$ is a Diffie-Hellman tuple, one computes $\mathsf{Com} = (\mathfrak{c} = \mathfrak{v}_{2,1}^{\lambda}\mathfrak{v}_{1,1}^{\mu}, \mathfrak{d} = \mathfrak{v}_{2,2}^{\lambda}\mathfrak{v}_{1,2}^{\mu} \times \mathsf{utk})$, for random $\lambda, \mu \xleftarrow{\$} \mathbb{Z}_p$, and one sets $\pi_1 = T_1^{\lambda}$ and $\pi_2 = T_1^{\mu}$, which should satisfy

$$e(T_1, \mathfrak{c}) = e(\pi_1, \mathfrak{v}_{2,1}) \cdot e(\pi_2, \mathfrak{v}_{1,1}) \qquad e(T_1, \mathfrak{d}) = e(T_2, \mathfrak{g}) \cdot e(\pi_1, \mathfrak{v}_{2,2}) \cdot e(\pi_2, \mathfrak{v}_{1,2})$$

The random values $\alpha_1, \alpha_2, \alpha'_1, \alpha'_2$ can be either chosen by the verifier in case of interactive proof, or set from $H(\tau_1, \tau_2, \tau_3, \tau'_1, \tau'_2, \tau'_3)$.

6.6 Related Work

On Figure 6.1, we provide some comparisons with the most efficient ABC schemes, where the column "P" (for policy) indicates whether the scheme just allows selective disclosure of attributes

Scheme	P	T	k-of-N attributes from $K = 1$ credential issuer			
			|CI key| $\mathbb{G}_1, \mathbb{G}_2$	|Show| $\mathbb{G}_1, \mathbb{G}_2, (\mathbb{G}_T), \mathbb{Z}_p$	Prover exp., pairings	Verifier exp., pairings
[CL11]	s	✗	$\mathbf{1}, \mathbf{1}$	$16, 2, (4), 7$	$16\mathbb{G}_1 + 2\mathbb{G}_2 + 10\mathbb{G}_T$, $18 + k$	$12\mathbb{G}_1 + 20\mathbb{G}_T$, $18 + k$
[FHS19]	s	✗	$0, N$	$8, 1, 2$	$9\mathbb{G}_1 + 1\mathbb{G}_2, 0$	$4\mathbb{G}_1, k + 4$
[San20]	**r**	✗	$0, 2N + 1$	$2, 2, (1), 2$	$(2(N-k)+2)\mathbb{G}_1 + 2\mathbb{G}_2, 1$	$(k+1)\mathbb{G}_1 + 1\mathbb{G}_T$, 5
Sec. 6.4.1	s	✓	$0, 2k + 3$	$\mathbf{3}, \mathbf{0}, \mathbf{1}$	$\mathbf{6}\mathbb{G}_\mathbf{1}, \mathbf{0}$	$\mathbf{4}\mathbb{G}_\mathbf{1} + \mathbf{k}\mathbb{G}_\mathbf{2}, \mathbf{3}$
Sec. 6.4.2	s	✓	$0, 2N + 2$	$\mathbf{3}, \mathbf{0}, \mathbf{1}$	$\mathbf{6}\mathbb{G}_\mathbf{1}, \mathbf{0}$	$\mathbf{4}\mathbb{G}_\mathbf{1} + \mathbf{2N}\mathbb{G}_\mathbf{2}, \mathbf{3}$

Scheme	$k = 1$-of-N attribute from K credential issuers			
	|CI key| $\mathbb{G}_1, \mathbb{G}_2$	|Show| $\mathbb{G}_1, \mathbb{G}_2, (\mathbb{G}_T), \mathbb{Z}_p$	Prover exp., pairings	Verifier exp., pairings
[CL11]	$\mathbf{K} \times (\mathbf{1}, \mathbf{1})$	$16, 2, (4), 7$	$16\mathbb{G}_1 + 2\mathbb{G}_2 + 10\mathbb{G}_T$, $18 + k$	$12\mathbb{G}_1 + 20\mathbb{G}_T$, $18 + k$
[FHS19]	$K \times (0, N)$	$K \times (8, 1, 2)$	$K \times (9\mathbb{G}_1 + 1\mathbb{G}_2, 0)$	$K \times (4\mathbb{G}_1, k + 4)$
[San20]	$K \times (0, 2N + 1)$	$K \times (2, 2, (1), 2)$	$K \times ((2(N-k)+2)\mathbb{G}_1 + 2\mathbb{G}_2, 1)$	$K \times ((k+1)\mathbb{G}_1 + 1\mathbb{G}_T, 5)$
Sec. 6.4.1	$K \times (0, 2k + 3)$	$\mathbf{3}, \mathbf{0}, \mathbf{1}$	$\mathbf{6}\mathbb{G}_\mathbf{1}, \mathbf{0}$	$\mathbf{4}\mathbb{G}_\mathbf{1} + \mathbf{k}\mathbb{G}_\mathbf{2}, \mathbf{3}$
Sec. 6.4.2	$K \times (0, 2N + 2)$	$\mathbf{3}, \mathbf{0}, \mathbf{1}$	$\mathbf{6}\mathbb{G}_\mathbf{1}, \mathbf{0}$	$\mathbf{4}\mathbb{G}_\mathbf{1} + \mathbf{2KN}\mathbb{G}_\mathbf{2}, \mathbf{3}$

Figure 6.1: Comparison of different ABC systems.

(s) or relations between attributes (r). The column "T" (for traceability) checks whether traceability is possible or not. Then, "|CI key|" gives the size of the keys (public keys of the credential issuers) required to verify the credentials, "|Show|" is the communication bandwidth during a show, while "Prover" and "Verifier" are the computational cost during a show, for the prover and the verifier respectively. Bandwidths are in number of elements $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and $\mathbb{Z}_p$. Computations are in number of exponentiations in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, and of pairings. We ignore multiplications. We denote N the global number of attributes owned by a user, k the number of attributes he wants to show and K the number of credential issuers involved in the issuing of the credentials. In the first table, we focus on the particular case of proving a credential with k attributes, among N attributes issued from 1 credential issuer. Our first scheme, from Section 6.4.1, is already the most efficient, but this is even better for a larger K, as shown in the second table. But this is for a limited number of attributes. Our second scheme, from Section 6.4.2 has similar efficiency, but with less limitations on the attributes. Note that both schemes have a constant-size communication for the showing of any number of attributes, and the computation cost for the prover is almost constant too (as we ignore multiplications).

Canard-Lescuyer Scheme. In 2013, Canard and Lescuyer proposed a traceable attribute-based anonymous credential scheme [CL13], based on sanitizable signatures: "Protecting privacy by sanitizing personal data: a new approach to anonymous credentials".

The intuition consists in allowing the user to "sanitize" the global credentials issued by the credential issuer, in order to keep visible only the required attributes. Then for unlinkability, the signatures are encrypted under an ElGamal encryption scheme.

Unfortunately, we found an attack in their scheme. The public key contains $g \xleftarrow{\$} \mathbb{G}_1$ and $\mathfrak{g} \xleftarrow{\$} \mathbb{G}_2$, and the ElGamal secret key is $\alpha \xleftarrow{\$} \mathbb{Z}_p$, the tracing key. The public encryption key is $h = g^\alpha$, but they also need $\mathfrak{h} = \mathfrak{g}^\alpha$ to be published for some verifications.

With this value $\mathfrak{h}$, anybody can break the semantic security of the ElGamal encryption, and then break the privacy of the anonymous credential.

Chapter

7 Conclusion

This thesis proposed new protocols integrating *privacy by design* and made its contribution to decentralization, electronic voting and anonymous authentication.

We have first presented in chapter 3 a new decentralized encryption scheme. It makes possible for a company to study encrypted databases of users by authorizing evaluations of quadratic polynomials on them, while at the same time, maintaining a level of security by controlling the computations. Hence, the company is forced by the system to recover only the result of the evaluation and nothing more.

Then, we studied in chapter 4 the linearly homomorphic signatures with two particular schemes. They feature tag randomizability, a property of great interest in our use-cases. While the first scheme only has one class of equivalence for tags and thus, ensures perfect anonymity, we proposed a second one with multiple classes so that a user can be anonymous within it. Both are proven secure in the Generic Bilinear Group Model and building such a scheme remains an issue in the standard model. Without the tag randomizability property, linearly homomorphic signature constructions are already known but the tag is usually hashed which is not compatible with randomizability.

Thanks to the LH-Sign scheme with randomizable tags, we constructed two schemes:

- A new method to build mix-networks,
- A new traceable multi-authority anonymous credential protocol.

The mix-networks scheme presented in chapter 5 followed a totally new approach compared to the previously known solutions. We avoided the proof of an explicit permutation on all the ciphertexts (per mixing step) extensively using the linearly-homomorphic signature schemes with tag randomizability. That made the proof of correctness implicit.

The computational complexity for each mix-server is linear in the number of ballots. The final proof implies just a constant-size overhead and the verification is also linear in the number of ballots, but independent of the number of rounds of mixing. This led to a highly scalable technique.

Finally, the anonymous credential protocol presented in chapter 6 is multi-user - a user can anonymously show a constant-size credential coming from several credential issuers - and for the first time traceable: the anonymity may be revoked by a judge in case of abuse.

While we optimized the size of a credential for a showing, the credential issuers needs to send the attributes individually. Whereas with redactable signatures it is possible to send one signature on all the attributes and then, redact some of them. One open problem is then to find a signature scheme including both properties: aggregatable and redactable. This could lead to more efficient scheme by improving the memory cost of a user.

All of the studied scenarios required new and more complex security concepts than the already existing ones. We used the homomorphic property of encryptions a lot, signatures and even zero-knowledge proofs but each time the malleability needed to be controlled.

Appendix

A Joint Generation of Square Diffie-Hellman Tuples

As already explained, for the unlinkability property to hold in the anonymous credential protocol, we need the user secret key $\mathsf{usk} = \tilde{\tau}$ random. Of course, this could be done with generic two-party computation, between the user and the Certification Authority.

- The user chooses $\tilde{\tau}_1 \xleftarrow{\$} \mathbb{Z}_p$ and computes $(A_1 = h^{\tilde{\tau}_1}, B_1 = A_1^{\tilde{\tau}_1})$.
- On its side, the Certification Authority chooses $\tilde{\tau}_2 \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$A = A_1 \cdot h^{\tilde{\tau}_2} = h^{\tilde{\tau}_1+\tilde{\tau}_2} \qquad B = B_1 \cdot (A_1^2 \cdot h^{\tilde{\tau}_2})^{\tilde{\tau}_2} = A_1^{\tilde{\tau}_1} \cdot A_1^{2\tilde{\tau}_2} h^{\tilde{\tau}_2^2} = h^{(\tilde{\tau}_1+\tilde{\tau}_2)^2}.$$

  It then sends and certifies $\tau = (h, A, B)$ together with $\tilde{\tau}_2$ so that the user can compute $\tilde{\tau} = \tilde{\tau}_1 + \tilde{\tau}_2$.

Appendix

B Another Bounded SqDH-Based ART-Sign

We can slightly reduce the parameters of the bounded SqDH-based ART-Sign, but with some limitations on the number of attributed to be signed. It relies on a hash function, modelled as a random oracle in the security analysis.

Description of the Bounded SqDH-based ART-Sign Scheme 2.

We thus propose here a second version, still with the limitation on the total number of messages signed for each tag, but the public keys are twice smaller:

Bounded SqDH-based ART-Sign Scheme 2

Setup(1^κ): It extends the above EphemerId-setup with the set of messages $\mathcal{M} = \{0,1\}^*$, but also a hash function $\mathcal{H}$ into $\mathbb{Z}_p$;

Keygen(param, n): Given the public parameters param and a length n, it outputs the signing and verification keys

$$\begin{aligned} \mathsf{sk}_j &= [\ t,\ u,\ v,\ s_1,\ \ldots,\ s_n\ ] \xleftarrow{\$} \mathbb{Z}_p^{n+3}, \\ \mathsf{vk}_j = \mathfrak{g}^{\mathsf{sk}_j} &= [\ T,\ U,\ V,\ S_1,\ \ldots,\ S_n\ ] \in \mathbb{G}_2^{n+3}. \end{aligned}$$

Sign(sk_j, τ, m): Given a signing key $\mathsf{sk}_j = [t, u, v, s_1, \ldots, s_n]$, a message $m \in \mathbb{Z}_p$ and a public tag $\tau = (\tau_1, \tau_2, \tau_3)$, it outputs the signature

$$\sigma = \tau_1^{t+\sum_{\ell=1}^{n} s_\ell \mathcal{H}(m)^\ell} \times \tau_2^u \times \tau_3^v.$$

AggrKey($\{\mathsf{vk}_j\}_j$): Given verification keys vk_j, it outputs the aggregated verification key $\mathsf{avk} = [\mathsf{vk}_j]_j$;

AggrSign($\tau, (\mathsf{vk}_j, m_{j,i}, \sigma_{j,i})_{j,i}$): Given tuples of verification key vk_j, message $m_{j,i}$ and signature $\sigma_{j,i}$ all under the same tag τ, it outputs the signature $\sigma = \prod_{j,i} \sigma_{j,i}$ of the concatenation of the messages verifiable with $\mathsf{avk} \leftarrow \mathsf{AggrKey}(\{\mathsf{vk}_j\}_j)$;

DerivSign(avk, $\tau, \boldsymbol{M}, \sigma, \rho_{\tau\to\tau'}$): Given a signature σ on tag τ and a message-set $\boldsymbol{M}$, and $\rho_{\tau\to\tau'}$ the randomization link between τ and another tag τ', it outputs $\sigma' = \sigma^{\rho_{\tau\to\tau'}}$;

RandSign(avk, $\tau, \boldsymbol{M}, \sigma$): The scheme being deterministic, it returns σ;

VerifSign(avk, $\tau, \boldsymbol{M}, \sigma$): Given a valid tag $\tau = (\tau_1, \tau_2, \tau_3)$, an aggregated verification key $\mathsf{avk} = [\mathsf{vk}_j]_j$ and a message-set $\boldsymbol{M} = [\mathsf{m}_j]_j$, with for each j, $\mathsf{m}_j = [m_{j,i}]_i$, and a signature σ, one checks if the following equality holds or not, where $n_j = \#\{m_{j,i}\}$:

$$e(\sigma, \mathfrak{g}) = e\left(\tau_1, \prod_j T_j^{n_j} \times \prod_{\ell=1}^{n} S_{j,\ell}^{\sum_i \mathcal{H}(m_{j,i})^\ell}\right) \times e\left(\tau_2, \prod_j U_{j,2}^{n_j}\right) \times e\left(\tau_3, \prod_j V_{j,3}^{n_j}\right)$$

We also recall that the validity of the tag has to be verified, as before, for the signature to be considered valid.

Security of the Bounded SqDH-based ART-Sign Scheme 2.

The linear homomorphism of the signature from [HPP20] still allows combinations. But when the number of signing queries is at most n per tag, the verification of the signature implies 0/1 coefficients only, with overwhelming probability:

Theorem 60. *The bounded SqDH-based ART-Sign defined above is unforgeable with a bounded number of signing queries per tag, even with adaptive corruptions of keys and tags, in both the generic group model and the random oracle model, as soon as $q_{\mathcal{H}}^n \ll p$, where $q_{\mathcal{H}}$ is the number of hash queries and p the order of the group (the output of the hash function).*

Proof. As argued in [HPP20], when the bases of the tags are random, even if the exponents are known, the signature that would have signed messages $\boldsymbol{M} = (g^{m^1}, \ldots, g^{m^n})$, for $m \in \mathbb{Z}_p$, is an unforgeable linearly-homomorphic signature. This means it is only possible to linearly combine signatures with the same tag: from up to n signatures σ_i on distinct messages m_i, for $i = 1, \ldots, n$ under vk_j, one can derive the signature $\sigma = \prod \sigma_i^{\alpha_i}$ on $\left(g^{\sum_i \alpha_i m_i^1}, \ldots, g^{\sum_i \alpha_i m_i^n}\right)$. Whereas the forger claims this is a signature on $\left(g^{\sum_i a_i^1}, \ldots, g^{\sum_i \alpha_i a_i^n}\right)$, on n_j values $a_1, \ldots, a_{n_j}$. Because of the constraint on τ_2, we have $\sum \alpha_i = n_j \bmod p$:

$$\sum_{i=1}^{n} \alpha_i m_i^\ell = \sum_{i=1}^{n_c} a_i^\ell \bmod p \qquad \text{for } \ell = 0, \ldots, n$$

Let us first move on the left hand side the elements $a_k \in \{m_i\}$, with only $n' \leq n_j$ new elements, we assume to be the first ones, and we note $\beta_i = \alpha_i$ if $m_i \notin \{a_k\}$ and or $\beta_i = \alpha_i - 1$ if $m_i \in \{a_k\}$:

$$\sum_{i=1}^{n} \beta_i m_i^\ell = \sum_{i=1}^{n'} a_i^\ell \bmod p \qquad \text{for } \ell = 0, \ldots, n$$

Our goal is to prove that $n' = 0$ and the α_i's are only 0 or 1.

So, first, let us assume that $n' = 0$: there is no new element. The matrix $(m_i^\ell)_{i,\ell}$, for $i = 1, \ldots, n$ and $\ell = 0, \ldots, n-1$ is a Vandermonde matrix, that is invertible: hence the unique possible vector (β_i) is the zero-vector. As a consequence, the vector $(\alpha_i)_i$ only contains 0 or 1 components.

Now, we assume $n' = 1$: there is exactly one element $a_1 \notin \{m_i\}$. We can move it on the left side:

$$\beta_0 a_1^\ell + \sum_{i=1}^{n} \beta m_i^\ell = 0 \bmod p \qquad \text{for } \ell = 0, \ldots, n, \text{ with } \beta_0 = -1$$

Again, the matrix $(m_i^\ell)_{i,\ell}$, for $i = 0, \ldots, n$ where we denote $m_0 = a_1$, and $\ell = 0, \ldots, n$, is a Vandermonde matrix, that is invertible: hence the unique possible vector (β_i) is the zero-vector, which contradicts the fact that $\beta_0 = -1$.

Eventually, we assume $n' > 1$: there are at least two elements $a_k \notin \{m_i\}$. We can move a_1 on the left side:

$$\beta_0 a_1^\ell + \sum_{i=1}^{n} \beta m_i^\ell = \sum_{i=2}^{n'} a_i^\ell \bmod p \qquad \text{for } \ell = 0, \ldots, n, \text{ with } \beta_0 = -1$$

Again, because of the invertible matrix, for the $n' - 1$ elements on the right hand side, there is a unique possible vector (β_i), and the probability for $\beta_0 = -1$ is negligible, as the new elements

a_k are random (if they are issued from a hash value): probability $1/p$ for each possible choice on the $n' - 1 < n$ attributes on the right hand side. Hence, as soon as $q_{\mathcal{H}}^n \ll p$, the probability for a combination to allow $\beta_0 = -1$ is negligible.

As a conclusion, one can only combine initial messages with a weight 1 (or 0). This proves unforgeability, even with corruptions of the tags, but with a number of signed messages bounded by n, and random messages (issued from a hash function). One can also consider corruptions of the signing keys, as they are all independent: one just needs to guess under which key will be generated the forgery. □

Unlinkability remains unchanged.

Bibliography

[ABC+12] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat, and Brent Waters. Computing on authenticated data. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 1–20, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany. *Cited on pages 6 and 47.*

[ABKW19] Michel Abdalla, Fabrice Benhamouda, Markulf Kolhweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. Cryptology ePrint Archive, Report 2019/020, 2019. https://eprint.iacr.org/2019/020. *Cited on page 4.*

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 209–236, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. *Cited on page 20.*

[AHM+18] Nuttapong Attrapadung, Goichiro Hanaoka, Shigeo Mitsunari, Yusuke Sakai, Kana Shimizu, and Tadanori Teruya. Efficient two-level homomorphic encryption in prime-order bilinear groups and A fast implementation in WebAssembly. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *ASIACCS 18*, pages 685–697, Incheon, Republic of Korea, April 2–6, 2018. ACM Press. *Cited on page 26.*

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany. *Cited on page 49.*

[BBP19] Olivier Blazy, Laura Brouilhet, and Duong Hieu Phan. Anonymous identity based encryption with traceable identities. 2019. *Cited on page 5.*

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany. *Cited on page 32.*

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany. *Cited on page 22.*

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017,*

Part I, volume 10401 of *LNCS*, pages 67–98, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. *Cited on page 4.*

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. *Cited on pages 20, 21, 67, and 84.*

[BF11a] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany. *Cited on pages 6 and 47.*

[BF11b] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 1–16, Taormina, Italy, March 6–9, 2011. Springer, Heidelberg, Germany. *Cited on pages 6 and 47.*

[BFI+10] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch Groth-Sahai. In Jianying Zhou and Moti Yung, editors, *ACNS 10*, volume 6123 of *LNCS*, pages 218–235, Beijing, China, June 22–25, 2010. Springer, Heidelberg, Germany. *Cited on pages 22, 54, and 67.*

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87, Irvine, CA, USA, March 18–20, 2009. Springer, Heidelberg, Germany. *Cited on pages 6 and 47.*

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 403–422, Taormina, Italy, March 6–9, 2011. Springer, Heidelberg, Germany. *Cited on page 8.*

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany. *Cited on page 84.*

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341, Cambridge, MA, USA, February 10–12, 2005. Springer, Heidelberg, Germany. *Cited on pages 8, 17, 25, and 44.*

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany. *Cited on pages 20 and 84.*

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007*, volume 4596 of *LNCS*, pages 411–422, Wroclaw, Poland, July 9–13, 2007. Springer, Heidelberg, Germany. *Cited on page 20.*

[Boy08] Xavier Boyen. The uber-assumption family (invited talk). In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 39–56, Egham, UK, September 1–3, 2008. Springer, Heidelberg, Germany. *Cited on page 49.*

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. *Cited on page 13.*

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany. *Cited on page 4.*

[CC09] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM CCS 2009*, pages 121–130, Chicago, Illinois, USA, November 9–13, 2009. ACM Press. *Cited on page 4.*

[CDG+18a] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 703–732, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. *Cited on page 4.*

[CDG+18b] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021, 2018. https://eprint.iacr.org/2018/1021. *Cited on page 4.*

[CDHK15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 262–288, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany. *Cited on page 9.*

[CF15] Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1518–1529, Denver, CO, USA, October 12–16, 2015. ACM Press. *Cited on page 25.*

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 257–270, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany. *Cited on page 5.*

[CFN18] Dario Catalano, Dario Fiore, and Luca Nizzardo. On the security notions for homomorphic signatures. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 183–201, Leuven, Belgium, July 2–4, 2018. Springer, Heidelberg, Germany. *Cited on page 48.*

[CFNP00] Benny Chor, Amos Fiat, Moni Naor, and Benny Pinkas. Tracing traitors. *IEEE Transactions on Information Theory*, 46(3):893–910, 2000. *Cited on page 5.*

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981. *Cited on pages 4 and 5.*

[CHP07] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 246–263, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany. *Cited on page 67.*

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. *Cited on page 67.*

[CL11] Sébastien Canard and Roch Lescuyer. Anonymous credentials from (indexed) aggregate signatures. In Abhilasha Bhargav-Spantzel and Thomas Groß, editors, *DIM'11, Proceedings of the 2013 ACM Workshop on Digital Identity Management, Chicago, IL, USA - October 21, 2011*, pages 53–62. ACM, 2011. *Cited on pages 8, 81, 84, 98, and 101.*

[CL13] Sébastien Canard and Roch Lescuyer. Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng, editors, *ASIACCS 13*, pages 381–392, Hangzhou, China, May 8–10, 2013. ACM Press. *Cited on pages 9 and 101.*

[CPRT18] Chris Culnane, Olivier Pereira, Kim Ramchen, and Vanessa Teague. Universally verifiable MPC with applications to IRV ballot counting. Cryptology ePrint Archive, Report 2018/246, 2018. https://eprint.iacr.org/2018/246. *Cited on page 26.*

[CS13] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *J. Comput. Secur.*, 21(1):89–148, January 2013. *Cited on pages 66 and 79.*

[Dam92] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany. *Cited on page 52.*

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany. *Cited on page 14.*

[Des93] Yvo Desmedt. Computer security by redefining what a computer is. In *Proceedings on the 1992-1993 Workshop on New Security Paradigms*, NSPW '92-93, page 160–166, New York, NY, USA, 1993. Association for Computing Machinery. *Cited on page 19.*

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. *Cited on pages 3 and 18.*

[Dor43] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943. *Cited on page 44.*

[DPP20] Xuan Thanh Do, Duong Hieu Phan, and David Pointcheval. Traceable inner product functional encryption. In Stanislaw Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 564–585, San Francisco, CA, USA, February 24–28, 2020. Springer, Heidelberg, Germany. *Cited on page 5.*

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany. *Cited on pages 3 and 15.*

[FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019. *Cited on pages 9, 51, 52, 81, 92, and 101.*

[FLSZ17] Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. An efficient pairing-based shuffle argument. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 97–127, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. *Cited on page 61.*

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. *Cited on page 4.*

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 44–61, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. *Cited on pages 8, 17, and 25.*

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 697–714, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany. *Cited on page 47.*

[FS01] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 368–387, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany. *Cited on page 61.*

[Gay16] Romain Gay. Functional encryption for quadratic functions, and applications to predicate encryption. Cryptology ePrint Archive, Report 2016/1106, 2016. https://eprint.iacr.org/2016/1106. *Cited on page 4.*

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. *Cited on pages 4 and 6.*

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. *Cited on page 4.*

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press. *Cited on page 4.*

[GGJS13] Shafi Goldwasser, Vipul Goyal, Abhishek Jain, and Amit Sahai. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/727, 2013. https://eprint.iacr.org/2013/727. *Cited on page 4.*

[GKL+13] S. Dov Gordon, Jonathan Katz, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774, 2013. https://eprint.iacr.org/2013/774. *Cited on page 4.*

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. *Cited on page 4.*

[GL07] Jens Groth and Steve Lu. A non-interactive shuffle with pairing based verifiability. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 51–67, Kuching, Malaysia, December 2–6, 2007. Springer, Heidelberg, Germany. *Cited on page 61.*

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304, Providence, RI, USA, May 6–8, 1985. ACM Press. *Cited on pages 7 and 21.*

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988. *Cited on page 18.*

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 171–185, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. *Cited on page 7.*

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. *Cited on page 52.*

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany. *Cited on pages 7, 8, 22, 54, 58, 61, 67, 87, and 89.*

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477, Portland, OR, USA, June 14–17, 2015. ACM Press. *Cited on page 48.*

[HHK+17] Gottfried Herold, Max Hoffmann, Michael Klooß, Carla Ràfols, and Andy Rupp. New techniques for structural batch verification in bilinear groups with applications to groth-sahai proofs. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1547–1564, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. *Cited on page 67.*

[HP20] Chloé Hébant and David Pointcheval. Traceable constant-size multi-authority credentials. Cryptology ePrint Archive, Report 2020/657, 2020. https://eprint.iacr.org/2020/657. *Cited on pages 8, 9, 47, and 81.*

[HPP19] Chloé Hébant, Duong Hieu Phan, and David Pointcheval. Decentralized evaluation of quadratic polynomials on encrypted data. In Zhiqiang Lin, Charalampos Papamanthou, and Michalis Polychronakis, editors, *ISC 2019*, volume 11723 of *LNCS*, pages 87–106, New York City, NY, USA, September 16–18, 2019. Springer, Heidelberg, Germany. *Cited on pages 8, 9, and 25.*

[HPP20] Chloé Hébant, Duong Hieu Phan, and David Pointcheval. Linearly-homomorphic signatures and scalable mix-nets. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 597–627, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany. *Cited on pages 8, 9, 47, 61, 89, 91, and 108.*

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 408–423, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany. *Cited on page 52.*

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262, San Jose, CA, USA, February 18–22, 2002. Springer, Heidelberg, Germany. *Cited on pages 6 and 47.*

[Ker83] Auguste Kerckhoffs. La cryptographie militaire, ou les chiffres usités en temps de guerre, avec un nouveau procédé de déchiffrement applicable aux systèmes à double clef. *Paris: Librairie militaire de L. Baudoin, 1883. 64 pp. See also Journal des sciences militaires, Paris. N.S. Tome 9. 59.*, 1883. *Cited on page 2.*

[KMH+19] Yutaka Kawai, Takahiro Matsuda, Takato Hirano, Yoshihiro Koseki, and Goichiro Hanaoka. Proxy re-encryption that supports homomorphic operations for re-encrypted ciphertexts. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E102.A:81–98, 01 2019. *Cited on page 25.*

[LPJY13] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 289–307, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. *Cited on pages 19, 47, 48, 49, 50, and 51.*

[LW11] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 568–588, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany. *Cited on page 4.*

[LYJP14] Benoît Libert, Moti Yung, Marc Joye, and Thomas Peters. Traceable group encryption. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 592–610, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany. *Cited on page 5.*

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 116–125, Philadelphia, PA, USA, November 5–8, 2001. ACM Press. *Cited on page 61.*

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 41–62, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany. *Cited on page 4.*

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany. *Cited on page 6.*

[PPS12] Duong Hieu Phan, David Pointcheval, and Mario Strefler. Decentralized dynamic broadcast encryption. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 166–183, Amalfi, Italy, September 5–7, 2012. Springer, Heidelberg, Germany. *Cited on page 4.*

[Riv00] Ron Rivest. Two signature schemes, 2000. *Cited on page 6.*

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978. *Cited on page 2.*

[San20] Olivier Sanders. Efficient redactable signature and application to anonymous credentials. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 628–656, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany. *Cited on pages 81, 82, 92, 94, and 101.*

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. *Cited on page 18.*

[Sch14] Rob Schapire. Computer science 511 – theoretical machine learning, 2014. *Cited on page 45.*

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949. *Cited on page 3.*

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. *Cited on pages 6 and 37.*

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany. *Cited on pages 14, 49, and 89.*

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany. *Cited on page 4.*

[TW87] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *28th FOCS*, pages 472–482, Los Angeles, CA, USA, October 12–14, 1987. IEEE Computer Society Press. *Cited on page 83.*

[Ver17] Damien Vergnaud. Comment on 'Attribute-Based Signatures for Supporting Anonymous Certification' by N. Kaaniche and M. Laurent (ESORICS 2016). *The Computer Journal*, 60(12):1801–1808, 2017. *Cited on page 9.*

RÉSUMÉ

Avec l'utilisation massive du stockage dématérialisé, l'homomorphisme est devenu l'une des propriétés les plus largement employées en cryptologie. Dans cette thèse, nous allons étudier comment l'utiliser dans des protocoles multi-utilisateurs concrets qui nécessitent non seulement de la confidentialité, mais aussi de l'anonymat, de l'authentification ou encore de la vérifiabilité. Pour cela, nous utilisons des schémas homomorphes de chiffrement, de signature numérique et de preuves à divulgation nulle de connaissances, mais, à chaque fois, nous devrons limiter leurs capacités de malléabilité pour atteindre le niveau de sécurité préalablement défini.

Tout d'abord, l'aspect confidentiel est abordé au travers de l'étude de calculs sur des bases de données externalisées. Être capable d'appliquer des fonctions sur des données chiffrées sans avoir à les télécharger pour les déchiffrer entièrement est permet de profiter de la puissance de calcul du serveur qui est généralement supérieure à celle du client. Cela peut être également indispensable lorsqu'une société sans droit d'accès à une base de données de clients souhaite obtenir le résultat d'un calcul. La quantité d'information apprise ne doit pas être supérieure à celle contenue dans le résultat du calcul. Nous proposons pour cela un schéma de chiffrement décentralisé qui permet d'évaluer des fonctions quadratiques sur les données externalisées tout en ayant un contrôle des opérations grâce à un groupe d'inspecteurs.

Cependant, la confidentialité des données n'est pas toujours la propriété la plus recherchée pour un système car elle ne protège pas l'identité de l'expéditeur. Pour le vote électronique, chaque bulletin chiffré doit être associé à un électeur afin de vérifier que celui-ci était autorisé à voter, mais après la phase de vote, l'anonymat doit être assuré. Pour cela une solution est de mélanger plusieurs fois l'urne de sorte que, au moment du dépouillement, qui correspond au déchiffrement, aucun lien entre le vote et l'électeur ne puisse être fait. C'est le fonctionnement d'un réseau de serveurs-mélangeurs dont nous proposons une nouvelle construction basée sur des signatures linéairement homomorphes avec un coût de vérification de l'urne finale indépendant du nombre de mélanges. Ce protocole est donc d'autant plus efficace que le nombre de mélanges augmente et représente un progrès par rapport aux constructions déjà connues.

Dans certains cas, avoir un anonymat parfait permettrait l'utilisation malveillante d'un système et la cryptologie doit aussi tenir compte de ces abus potentiels. La troisième contribution de cette thèse consiste en la proposition du premier protocole d'accréditation anonyme multi-autorités traçable : un utilisateur demande une accréditation auprès d'une autorité émettrice et peut l'utiliser pour accéder à un système tout en restant anonyme. En cas d'abus, une autorité juge peut lever l'anonymat et retrouver un utilisateur malveillant grâce au traçage. De plus, ce protocole, tout en étant aussi efficace que les précédents pour une seule autorité émettrice, permet d'agréger des accréditations d'autorités émettrices distinctes pour avoir une accréditation de taille optimale .

MOTS CLÉS

Cryptographie à Clé Publique ⋆ Protocoles Homomorphes ⋆ Anonymat ⋆ Vote Electronique ⋆ Calculs Multipartites

ABSTRACT

With the massive use of dematerialized storage, homomorphism has become one of the most widely used properties in cryptology. In this thesis we will study how to use it in concrete multi-users protocols requiring not only confidentiality but also anonymity, authentication or verifiability. Homomorphic encryption schemes, homomorphic digital signatures and homomorphic zero-knowledge proofs will be used together, but each time restricted to achieve the desired level of security.

First, the confidential aspect is studied for computations on large outsourced databases. Being able to apply functions on encrypted data without having to download and decrypt it entirely may be essential and allows to take advantage of the computational power of the server. This can also be interesting when a third-party company without right-access to the database wants to obtain the result of a computation. However, some guarantees on the learned information need to be taken. To this end, we present a decentralized encryption scheme that allows controlled evaluation of quadratic functions on outsourced data thanks to a group of controllers.

However, sometimes confidentiality of the data is not the most desired property for a system as it does not protect the sender. For electronic voting, each encrypted ballot must be associated with its voter to verify that he is allowed to vote. After the voting phase, anonymity is achieved by shuffling so that, during the count, which corresponds to the decryption, no link between votes and voters can be made. We propose a new construction of mix-network based on linearly homomorphic signatures which allows for the first time a verification which is cost-independent of the number of mix-servers. This scalable mix-net improves the efficiency compared to already known constructions, especially with an increasing number of shuffles.

Nevertheless, with perfect anonymity comes the threat of malicious use of the system. Cryptology must consider these possible abuses and we propose the first multi-authority anonymous credential protocol with traceability property: a user asks a credential issuer for a credential and uses it to access a system while remaining anonymous. In case of abuse, an authority can revoke anonymity and trace a malicious user. The scheme is as efficient as the previously known credential schemes while achieving the multi-credential issuer functionality.

KEYWORDS

Public-Key Cryptography ⋆ Homomorphic Protocols ⋆ Anonymity ⋆ E-voting ⋆ Multi-party Computation