Delegation in functional encryption

Jérémy Chotard

Thèse de doctorat
de l'Université de Limoges

Université de Limoges

École doctorale 610 - Spécialité Informatique

Delegation in functional encryption

Préparée à
l'Université de Limoges et
l'École normale supérieure

Soutenue le 02/12/2019 par

Jérémy CHOTARD

Encadrée par

Duong Hieu PHAN
Université de Limoges

David POINTCHEVAL
École normale supérieure

Rapporteurs

CATALANO Dario
Université de Catane

LIBERT Benoit
École normale supérieure
de Lyon

Examinateurs

ABDALLA Michel
École normale supérieure

CANARD Sébastien
Orange Labs

FONTAINE Caroline
École normale supérieure
de Paris-Saclay

xlim INSTITUT DE RECHERCHE

ENS | PSL

Delegation in functional encryption

Jérémy Chotard
Supervised by Duong Hieu Phan and David Pointcheval

Abstract

Functional encryption is a recent paradigm that generalizes the classical public key encryption. This formalization aims to finely manage both the access control to the encrypted data, and the information revealed by the decryption. This thesis studies possibilities of delegation through these two sides.

First, we deal with a multi-client context: several users provide each one an encryption of personal data, and an entity wishes to extract information from the aggregate of those inputs. Our contribution in this environment consists to provide these clients the possibility to give, or refuse, their consent for such an extraction. To this aim, we describe constructions of multi-client functional encryption. We then formalize several levels of security and provide methods to reach them. Eventually, we decentralize the construction of the functional decryption key, so that one needs the agreement of all clients to get a functional decryption key. All this, in a practical way.

Second, we consider a more specific case where a video content provider wishes to delegate the distribution of his creation, but without revealing it. Our solution is a key encapsulation mechanism, derived from attribute-based encryption, with a particular property. The provider uses it to encapsulate the key of the encrypted stream under several attributes, and provides the encapsulations to the distributor. This "content manager" can then use the property to combine the encapsulations and make a new one under the access policy of his choice.

Résumé

Le chiffrement fonctionnel est un paradigme récent qui généralise le chiffrement à clef publique classique. Cette formalisation a pour objectif de réguler plus finement le contrôle d'accès aux données chiffrées, ainsi que l'information dévoilée par le déchiffrement. Cette thèse étudie des possibilités de délégation au travers de ces deux aspects.

Dans un premier temps, nous travaillons dans un contexte multi-client: plusieurs utilisateurs fournissent chacun une donnée personnelle chiffrée, et une entité souhaite extraire de l'information de ces données. Notre contribution consiste à permettre à ces utilisateurs de donner, ou refuser, leur accord à cette extraction. Pour ce faire, nous décrivons des constructions de chiffrement fonctionnel multi-utilisateur, puis nous definissons plusieurs niveaux de sécurité et fournissons des méthodes pour les atteindre. Enfin, principal objectif de ces travaux, nous décentralisons la fabrication de la clef de déchiffrement, pour qu'une personne souhaitant une clef de déchiffrement ait besoin de l'accord de tous pour l'obtenir. Toutes les instantiations proposées dans ces travaux sont utilisables en pratique.

Dans second temps, nous considérons une autre problématique dans laquelle un producteur de contenu vidéo cherche à déléguer la distribution de sa création, sans la révéler. Notre solution est un mécanisme d'encapsulation de clef, dérivé du chiffrement par attributs, avec une propriété particulière. Ce producteur l'utilise pour encapsuler la clef du flux vidéo sous plusieurs attributs, et fournit les encapsulations au distributeur. Celui-ci peut alors utiliser la propriété pour combiner les encapsulations et en définir les conditions d'accès à sa guise.

Remerciements

Nombre de personnes autour de moi ont dû fournir des efforts conséquents pour me supporter ces dernières années. A chacune d'entre elles, j'aimerais prendre le temps d'exprimer mes sincères remerciements pour avoir fait ce qu'ont été ces années de doctorat. Cependant, je me rends compte en écrivant ses lignes du nombre de personnes que cela représente, ma thèse s'étant effectuée à la fois à Paris et Limoges. Aussi, si vous êtes l'une d'entre elles mais que votre nom ne figure pas dans ces pages, je vous prie de m'en excuser.

Mes premiers remerciements vont naturellement à mes deux encadrants, Duong Hieu Phan et David Pointcheval, pour les opportunitées qu'ils m'ont offert au travers de cette thèse, comme celle de travailler sur un thème de recherche que j'ai beaucoup apprécié, l'occasion de rencontrer des gens très intéressants (même si je ne suis pas connu pour ma langue déliée), ou encore d'assister à des conférences à l'étranger. Je les remercie aussi pour leur disponibilité malgré toutes leurs responsabilités, pour leur patience et surtout pour toute l'aide qu'ils m'ont fourni sur bien des plans.

Je remercie Dario Catalano et Benoit Libert pour la relecture de ce mémoire. J'espère que le sujet vous a intéressé, et que la lecture du mémoire n'a pas été trop fastidieuse. Je remercie également Sébastien Canard, Caroline Fontaine et Michel Abdalla d'avoir accepté le rôle d'examinateur pour la soutenance. Et merci aussi Michel pour les remarques de dernière minute, mon anglais reste encore très approximatif malgré ces trois ans de thèse.

Je tiens à remercier toute l'équipe du département informatique de l'ENS, qui m'a très bien accueilli pendant mon stage, et au sein de laquelle j'ai passé quelques moments mémorables durant toute la durée de mon doctorat. Au cours de ces années, cette équipe a beaucoup évolué, mais heureusement, la moitié du temps de travail d'un doctorant consistant à boire du café, la cuisine a très bien rempli le rôle de catalyseur social. Ainsi, je vous remercie pour ces instants caféinés, même si je ne bois personnellement jamais de café. Merci à Rafaël et Florian pour leur humour et leur bonne humeur, Alain, Fabrice et Geoffroy pour leurs explications sur des sujets que j'étais sensé maîtriser, Romain et Edouard pour tout le travail accompli en tant que co-auteur. Je remercie aussi Pierre-Alain, Adrian, Michele et Michele, Celine, Razvan, Anca pour son originalité en matière de couleurs et de décoration, Louiza, Damuhn, Damien, Houda, Pierrick pour avoir animé le laboratoire de façons très variées, Georg, Pooya, Chloé, Aurélien et Melissa pour les séjours à l'étranger, Huy, Léo, Brice, Antoine pour avoir organisé des soirées jeux auxquelles je n'ai jamais pu participer, Azam, Balthazar, Théo et Hugo.

De la même manière, je remercie les membres des équipes de XLIM qui ont

dessiné mon environnement à Limoges. Merci à Xavier, Joëlle, Paul, Adrien, Florent pour m'avoir aidé lorsque je suis arrivé à Limoges, à Neals, Nicolas, Mathieu pour m'avoir décrit une partie de l'état de l'art sur les stratégies gagnantes à Hearthstone. A ces derniers, ainsi qu'à Laura et Maxime, je souhaite bon courage pour la suite de vos thèses. Merci aussi à Léo, Don, Emmanuel et Olivier pour leurs réponses à mes questions en tout genre, Thanh, Sayantan (good luck for your post-doc), et je salue les doctorants de synthèse d'images, Arthur, François, Théo et Jérémy. Enfin, je remercie Philippe, Tristan, Julien et Damien pour m'avoir offert la possibilité d'encadrer des TD/TP intéressants, et puis l'IO.

J'aimerais aussi remercier tout le personnel de l'administration, à la fois à Limoges et Paris, qui s'est souvent efforcé de faire avancer les procédures qui me concernent, ceci malgré ma phobie des formalités administrative. Parmi ces personnes, je tiens à remercier plus particulièrement Lise-Marie Bivard, Sophie Jaudon et Joelle Isnard pour leur patience, mais surtout la sympathie dont elles ont fait preuve au cours de ces trois années.

Enfin, si pendant trois ans certains ont dû accepter ma façon désorganisée de faire les choses ou mon manque de communication, d'autres ont dû apprendre à vivre avec depuis bien longtemps, par choix ou par contrainte. Aussi je remercie mes amis Tourangeaux, avec qui j'aime me changer les idées autour d'une bière, d'un jeu de plateau ou autre, et je les remercie plus généralement pour les bons souvenirs qu'ils m'ont laissé, ces trois dernières années comme auparavant. Enfin, je remercie du fond du coeur ma famille, qui m'a modelé, mais pas choisi, pour tout le soutien et la chaleur qu'elle m'a apporté tout au long de ma vie. Mes propos à leur sujet resteront concis, ces personnes savent déjà à quel point elles comptent pour moi.

Contents

1 Introduction

A brief crypto story

Information is the key behind any decision, and it is primordial for conflicting entities to hide it from each other. From the Antiquity, cryptography was used to this aim, mainly to allow several beings, knowing each other, to communicate safely. Thus, most of the techniques used then relate to what we call today symmetric cryptography, a protocol where two trusted parties must share a same secret to communicate each other.

In the nineteenth-twentieth century, with the rise of long distance communication devices, and on the strength of new mathematical tools, military research interest in this science grew up. Among the scientific projections, August Kerchkoffs enunciated the famous principle, which states that the secrecy of an encryption scheme must not depend on the knowledge of the algorithm. The one-time pad was formalized, rotor machines were used in the first world war, their cryptanalysis followed... However, the main goal was still to build complex systems to hide a stream between two entities sharing a secret. Furthermore, the reliability of these systems was not dependent of complex mathematical problems, leaving open the possibility of a potential cryptanalysis.

Our recent era saw the development of an international network that allows real-time communication between the smallest cells of any human organization, the human himself. But in a context where even adversaries are directly connected, how can one really trust someone else? The establishment of this situation created new needs in numerical security, like proven authentication, the possibility to start a secure communication with a totally unknown individual, or the insurance that transmitted messages are not modified. This is where the modern cryptography started to develop, principally through the formalization of public key cryptography.

Public key cryptography regroups protocols that require to publish a key to encrypt/prove/share something. Few examples are the public key encryption, signature and key exchange. Usually, public key encryption is illustrated by a user that provides a second one an open lock while he keeps the key. This lock acts has a public key: the second user utilizes it to hide a message in a box that can be open by the first user only. A signature is a trace of a message that cannot be falsified, but can be verified publicly. And the key exchange allows two individuals to share a common key after several public interactions.

The first practical schemes for these protocols were described after the mid-

seventies: the first key exchange in [DH76], then signature and public key encryption in [RSA78]. They relied on mathematical problems from number theory the authors believed to be hard to break. However, there was no proof that one needs to solve those mathematical problems to violate the security supposedly given by the scheme. The way for a new methodology of proven security was opened by Shafi Goldwasser and Silvio Micalli in [GM84], which proposed a new public key encryption scheme associated to an original proof system for a formalized security model. More generally, it is now a standard that new cryptosystems come with an associated proof, stating that breaking the system implies solving an allegedly hard mathematical problem.

Our context

The Internet is now an exchange surface that allows a free circulation of information. The direct access to different services helps enterprises and institutions to better structure and delegate their tasks and needs. Consequently, new services related to informatics have emerged: some enterprises now rent computational power and storage space. This leads to a situation where clients willingly provide their information to untrusted organisms. This business model was democratized, and the use of external servers for storage or computational work is now referred as Cloud computing. The problem here is, when a source provides personal data, he technically loses control of it, so this "cloudy" context seemingly leaves no place for the user's privacy. The recent history has proven this suspicion was justified. We now know that private and governmental organizations have recorded, shared, sold data without the source's consent for statistics, advertising, spying or political purposes.

These circumstances call for new cryptographic solutions to protect an user unwilling to divulge his data, but also to permit the extraction of information if he allows it. A recent and promising candidate that partially solve this problematic is the fully homomorphic encryption. This primitive is an encryption protocol that allows two operations on the ciphertexts, which acts homomorphically on the corresponding plaintexts; a first instantiation was described in [Gen09]. It fits with our problematic in the sense that the server can work on the ciphertext without learning anything on the content. However, the original formalism of fully homomorphic encryption does not permit the server to extract information from the ciphertext after the computation. Also, while it aims to apply generic functions on the input, it is currently not efficient in most of the cases. In this thesis, we focus on an alternative primitive that reformulates the public key encryption: the functional encryption.

Functional encryption

Functional encryption is comparable to public key encryption, except that the key is associated to a function f: for any function f from a class $\mathcal{F}$, a functional decryption key dk_f can be computed such that, given any ciphertext c with underlying plaintext x, using dk_f, a user can efficiently compute $f(x)$, but does not get any additional information about x.

Mentioned for the first time in [AL10, LOS+10, OT10], functional encryption was originally aimed to generalize the description of access control in public key encryption, gathering together protocols like Identity-Based Encryption, Inner Product Encryption or Broadcast Encryption. At that time, the definition focused on

"all-or-nothing" protocols, where the decryption returns $f(x) = x$ when the user is authorized to decrypt and $\perp$ otherwise, and the first constructions considered were attribute-based encryption schemes.

Attribute-based encryption

Attribute-based encryption (ABE) was introduced by Sahai and Waters [SW05], it is a generalization of some advanced primitives such as identity-based encryption [Sha84, BF01] and broadcast encryption [FN94]. It gives a flexible way to define the target group of people who can receive the message: encryption and decryption can be based on the user's attributes. This primitive was further developed by Goyal *et al.* [GPSW06] who introduced two categories of ABE: "ciphertext-policy" attribute-based encryption (CP-ABE) and "key-policy" attribute-based encryption (KP-ABE). In a CP-ABE scheme, the secret key is associated with a set of attributes and the ciphertext is associated with an access policy over the universe of attributes: a user can decrypt a given ciphertext if he holds the attributes that satisfy the access policy underlying the ciphertext. KP-ABE is the dual to CP-ABE in the sense that an access policy is encoded into the users secret key, and a ciphertext is computed with respect to a set of attributes: the ciphertext is decryptable by a user only if the attributes in the ciphertext satisfy the user's access policy.

Over the last few years, there has been a lot of progress in constructing secure and efficient ABE schemes from different assumptions and for different settings [SW05, GPSW06, OSW07, GJPS08, HLR10, ALdP11, OT12, RW13, CCL+13, GVW13, YAHK14, BGG+14, KW19], to name a few.

Other classes of function

In parallel, through the articles [O'N10, BSW11], the authors supposed other classes of functions than "all-or-nothing", and generic constructions came with [GVW12, GKP+13b, GKP+13a, Wat15, ABSV15, GGG+14, BGJS16, BKS16]. Unfortunately, they all rely on non standard cryptographic assumptions (indistinguishability obfuscation, multi-linear maps), or offer black-box contributions where the ciphertext's size somehow depends of the number of queried keys. It is more important in practice, and it is an interesting challenge, to build functional encryption for restricted (but concrete) classes of functions, satisfying standard security definitions, under well-understood assumptions.

This was achieved in 2015, with the inner product function family. Abdalla, Bourse, De Caro, and Pointcheval [ABDP15] considered the question of building functional encryption for inner product functions. In their paper, they show that inner product functional encryption can be efficiently realized under standard assumptions like the decisional Diffie-Hellman and learning-with-errors assumptions [Reg05], but in a weak security model, named "selective security". Later, Agrawal, Libert and Stehlé [ALS16] considered "adaptive security" for inner product functional encryption and proposed constructions whose security is based on decisional Diffie-Hellman, learning-with-errors or Paillier's decisional composite residuosity [Pai99] assumptions. Finally, [Gay16, BCFG17] recently achieved single input functional encryption for quadratic functions. Their work proposes several schemes based either on matrix

decisional Diffie Hellman group assumption or on an idealized model called the generic group model.

Multi-user context

Note however that all the above works consider functional encryption schemes on an input from a single source. Whereas the input can be large, like a high-dimensional vector, the basic definition of functional encryption implies that the input data comes from only one party: all the coordinates of the vector are provided by one party, and all are encrypted at the same time. In many practical applications, the data are an aggregation of information coming from different parties that may not trust each other. Especially, in the cloud context, a server stores data from several independent sources (Fig. 1.1).

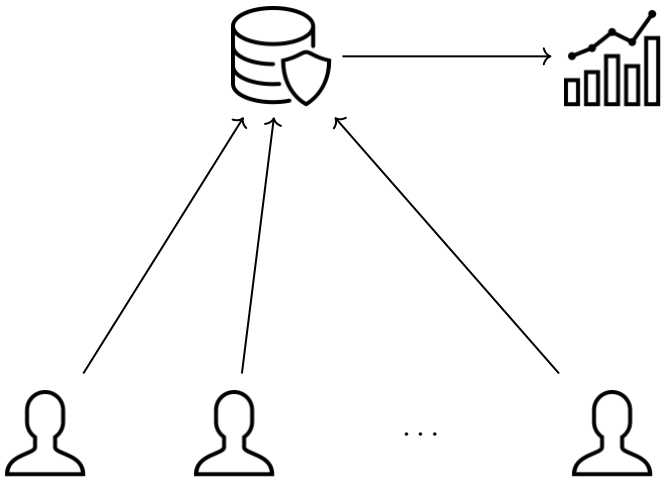

Figure 1.1: Clients trustfully leaving their data on a server which can perform analysis. This situation is possible through functional encryption. In this case, the clients provided encrypted data, and the server uses a key that leaks a precise function of the data, and no more.

To handle this, Gordon, Goldwasser *et al.* [GKL+13, GGG+14] introduced both the notions of multi-input and multi-client functional encryption which break down a single input x into an input vector $(x_1, \ldots, x_n)$ where the components are independent. In those primitives, anyone owning a functional decryption key dk_f, for an n-ary function f and multiple ciphertexts $c_1 = \mathsf{Encrypt}(\mathsf{ek}_1, x_1), \ldots, c_n = \mathsf{Encrypt}(\mathsf{ek}_n, x_n)$ ($\mathsf{Encrypt}(\mathsf{ek}_i, x_i, i, \ell)$ in the multi-client case), can compute $f(x_1, \ldots, x_n)$ but nothing else about the individual x_i's. Multi-input and multi-client functional encryption are similar except that the former presents no notion of ciphertext index or label: an user i can enter x_i and encrypt it as $c_i = \mathsf{Encrypt}(\mathsf{ek}_i, x_i)$. This means that in the latter, combination of ciphertexts generated for different labels does not give a valid global ciphertext and the adversary learns nothing from it. This little difference has important consequences in the control of the possible evaluations. To give an example, we consider a multi-client protocol where an user possesses one decryption key dk_f, and is given two encrypted vectors $\vec{c}_1, \vec{c}_2$ hiding the values x_1, x_2 under

the labels $1, 2$. This user can recover two values $f(x_1), f(x_2)$ and no more. If we repeat this situation in a multi-input protocol (without labels involved), the user can compute $f(x)$ for any mix x of any coordinate from x_1, x_2, respecting the users indexes. We can thus count up to 2^n possible pairs $(x, f(x))$, n being the size of the vectors. Such an amount of informations can totally reveal the plaintext hidden in $\vec{c}_1, \vec{c}_2$, making this simple example a potential and trivial attack. Hence, the adversary is strongly limited on the encryption and key queries it is allowed to ask in the case of multi-input functional encryption.

Numerous applications of multi-input functional encryption have been given in detail in [GGG+14]. Like the single-input version, multi-input and multi-client schemes first relied on indistinguishability obfuscation or multilinear maps, which we currently do not know how to instantiate under standard cryptographic assumptions. Extending inner product functional encryption to the multi-input setting has proved technically challenging. [AGRW17] builds the first multi-input inner product functional encryption, that is, each input slot encrypts a vector $\vec{x}_i \in \mathbb{Z}_p^m$ for some dimension m, each functional decryption key is associated with a vector $\vec{y}$, and decryption recovers $\langle \vec{x}, \vec{y} \rangle$ where $\vec{x} := (\vec{x}_i \| \cdots \| \vec{x}_n)$, $\vec{y} \in \mathbb{Z}_p^{n \cdot m}$, and n denotes the number of slots, which can be set up arbitrarily. They prove their construction secure under standard assumptions in bilinear groups. Concurrently, [LL16] build a two-input (*i.e.* $n = 2$) functional encryption using similar assumptions in bilinear groups. Very recently, [DOT18, ACF+18] gave a "function-hiding" multi-input functional encryption for inner products, where the functional decryption keys do not reveal their underlying functions. [ACF+18] also gives a generic transformation from single to multi-input for inner product functional encryption, which gives the first multi-input constructions whose security rely on decisional Diffie-Hellman, learning-with-errors or decisional composite residuosity.

These works allow extraction of information from the aggregation of data from independent sources. However, they all suppose an entity centralizing all the secrets, at least to make functional decryption keys, leaving users powerless regarding the distribution of such keys, and defenseless against the corruption of this entity. On an other side, they potentially require interactions between the users to set-up the protocol, which is hardly possible in a practical situation involving a large number of users unaware of each other. Also, while it was considered in the original work [GGG+14], no one seems to handle with the user corruption. Finally, only multi-input functional encryption schemes were proposed to this point, leaving the open challenge to include a time-stamp system present in the multi-client version. All these points were studied in our article [CDG+18a].

First contribution

For the reason previously detailed about the difference between multi-input and multi-client functional encryption, our work [CDG+18a] focuses on the multi-client protocol. It introduces the notion of decentralized multi-client functional encryption, in which we decentralize the generation of functional decryption keys among the same clients as the ones that generate the ciphertexts.

A naive way to distribute the ciphertext generation could be to take a functional encryption scheme and to have a trusted party handling the setup and the key generation phases, while the encryption procedure would be left to many clients to execute by secure multi-party computation. This construction has two obvious

weaknesses. First, generating any ciphertext potentially requires heavy interactions, with everybody simultaneously online, and the full ciphertext has to be generated at once, with all the components being known at the same time, which is impossible if an user does not answer at the moment. Secondly, the trusted third party reserves the power to recover every client's private data and the choice of the decryption keys to provide, which completely goes against our main goal.

Another idea, which is the one behind our method, is the use of a multi-input/client functional encryption (multi-client in our case) scheme with multi-party computation to build the decryption key. Again, while multi-party computation often requires several interactions, our target is a non-interactive generation of the functional decryption keys (in the Inner-Product case). We avoid this problem, in [CDG+18a], by using an other multi-client scheme to assemble the key. The encryption of this second scheme then corresponds to the key construction of the first, and necessitates no interaction between the users.

The resulting schemes are the first practical instantiations of multi-client functional encryption scheme, with a label system allowing a finer control of the possible evaluations than the multi-input setting. They are highly practical as their efficiency is comparable to that of the decisional Diffie Hellman based inner product functional encryption scheme from [ALS16]: a value x_i is encrypted as one or two group elements C_i, depending of the security level needed. The setup phase, key generation and decryption all take time linear in the number of participants, and encryption takes time linear in the size of its input. Regarding security, two of the three schemes (Section 4.2.3, Section 5.2.1) are adaptive under classical assumptions either in prime order groups or pairing groups, and the last one (Section 4.2.1) is selective. In addition, we successfully address corruptions of clients, even adaptive ones in the centralized multi-client setting, exploring what Goldwasser *et al.* [GGG+14] highlighted as an "interesting direction". Finally, concerning the interactivity, the decentralized multi-client functional encryption scheme we present in Section 5.2 has a key generation protocol that does not require communications between the users.

Scheme	Non Interactive SetUp	Non Interactive Encrypt	Non Interactive KeyGen	Decentralized Keys
MIFE [AGRW17]	✓	✓	✓	✗
FE [ABDP15] + MPC in Encrypt	✓	✗	✓	✗
MCFE Section 4.2.3	✓	✓	✓	✗
MCFE Section 4.2.3 + MPC in KeyGen	✓	✓	✗	✓
DMCFE Section 5.2	✗	✓	✓	✓
DMCFE + DSum Section 5.1.6	✓	✓	✓	✓

Figure 1.2: Comparison of different cryptographic solutions to the problem of linearly aggregating private multi-client data.

However, those constructions present some restrictions that we briefly describe here. First, when we succeed in decentralizing the decryption key generation, the

set-up lose this property, which clearly goes against our objective. Also, the decentralization needs to introduce pairings because of a discrete logarithm computation. A alternative multi-party computation scheme could potentially remove this restriction. Another point is that our schemes do not capture the multi-input case, while the original definition described the multi-client functional encryption as a "multi-input with labels". Last cited, but not the least, the security level is at most an $\texttt{IND}^*$ derivative. This means that the schemes are not secure against the event where not all the users provide a encrypted value. Actually there is even a proof that our schemes are vulnerable in such a situation. Since this event is highly probable in a real-life context, we considered this limitation as a major problem.

Second contribution

Our recent work [CDG+18b] solves most of the previously evoked points.

We first deal with the limitation in the security model from [CDG+18a], that requires complete ciphertexts. Our solution is generic, as this is an additional layer, that is applied to the ciphertexts so that, unless the ciphertext is complete (with all the encrypted components), no information leaks about the individual ciphertexts, and thus on each components. This technique relies on a linear secret sharing scheme, hence the name "secret sharing encapsulation" (Section 6.1). It can also be seen as a decentralized version of "all-or-nothing transforms" from [Riv97, Boy99, CDH+00]. We propose a concrete instantiation in pairing-friendly groups, under the decisional bilinear Diffie-Hellman problem, in the random oracle model. We stress that this conversion transforms "any" $\texttt{IND}^*$-secure multi-client functional encryption (not necessarily for inner products) into an $\texttt{IND}$ equivalent at constant cost, meaning that it now manages the case where ciphertexts are missing. Our conversion just adds two group elements to individual ciphertexts.

A second part of this contribution deals with the number of encryptions under a same label. In the inner product multi-client functional encryption from [CDG+18a], the definition only allows to send a unique scalar (vector if we extend the key). Yet, when vectors are input, it makes sense to allow mix-and-match between the inputs. In addition, requiring a unique component per label for each client, while under his responsibility, is a strong limitation. What happens when the client makes a mistake? The Section 6.2 provides a solution specifically for the group based schemes from [CDG+18a]. Briefly explained, each user adds an inner product functional encryption layer on his encrypted vector. This permits to combine the vectors from different users on one label with respect to the coordinates, and it finally achieves the multi-input mix-and-match functionality on this label.

In Section 5.1, we also propose an efficient decentralized algorithm to generate a sum of private inputs, hence called distributed sum, which can convert an inner product multi-client functional encryption into a decentralized one: this technique is inspired from [KDK11], and only applies to the functional decryption key generation algorithm, so it is compatible with the other conversions (Fig. 1.3). Namely, this improves on the decentralization from [CDG+18a] since it does not require pairings, and because no interactions are needed in its set-up, achieving thus a fully decentralized multi-client functional encryption (Fig. 1.2).

Eventually, to maximize the security level, we show how to prove security against chosen ciphertext attacks (noted $\texttt{CCA}$) generically, simply by adding signatures to ciphertexts for each slot (Section 6.3). This transformation applies to any multi-client

scheme that reaches the IND-security level (the ideal one), and is not limited to the inner product functionality. Even though security against chosen ciphertext attacks is the de facto security notion for encryption, the only way to obtain a multi-client functional encryption secure against chosen ciphertext attacks prior to our work consists of applying the generic Naor Yung paradigm [NY90], that requires extra assumptions (simulation-sound non interactive zero knowledge arguments), and doubles the ciphertext size. This is true even assuming the random oracle model. In particular, the seminal Fujisaki Okamoto transform [FO99] is of no help for functional encryption, where decryption only recovers partial information of the plaintext.

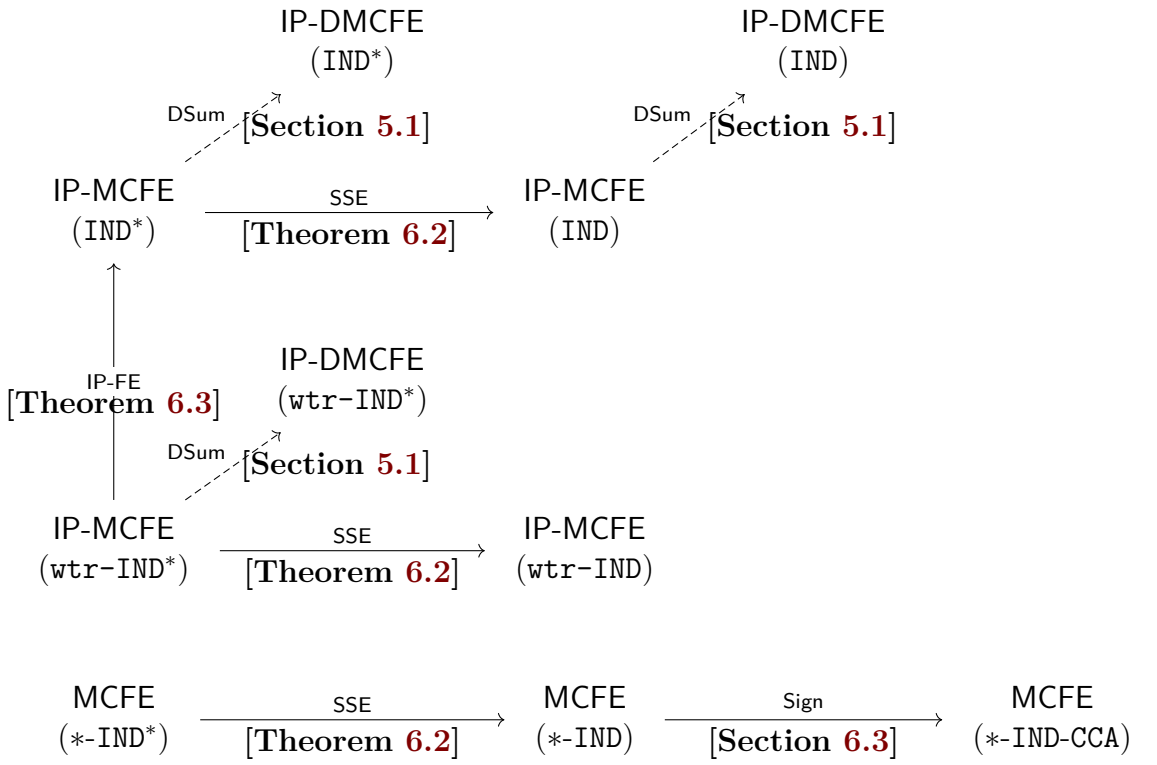

Figure 1.3: Contributions and theorems. Here, wtr stands for: "without repetitions", CCA for: "chosen ciphertexts attacks" and IND* is a security level that does not manage the case where an user does not provide a ciphertext, IND being the natural one.

All the above conversions preserve the efficiency of the underlying multi-client functional encryption. While the secret sharing encapsulation techniques introduce pairings, the two others do not: they only rely on the decisional or even computational Diffie Hellman assumptions, in the standard or random oracle models. But we also stress that our secret sharing encapsulation techniques is constant-size. A concurrent and independent work [ABKW19] also proposes a compiler from IND*-security to IND-security, without pairings, but ciphertext size becomes linear in the number of clients.

To sum up those two works, we develop a practical solution to give someone a right of refusal regarding the exploitation of his data, in a context of mass storage on external platform. These results adequately answer the problematic described in Fig. 1.1 if a server aims to compute weighted means over an aggregated database.

Third contribution

The last work presented in this thesis considers a different question: is it possible to encrypt data such that someone else can choose the final target without seeing the contents? Our article [CPP17] answers this problem positively using the attribute-based encryption.

Before explaining our scenario, we recall that attribute-based encryption can be qualified either ciphertext-policy or key-policy (we will note ABE, CP-ABE and KP-ABE), and both consider different situations. In KP-ABE, the encryptor has no control over who has access to the data he encrypts. This is the key-issuer who generates and controls the appropriate keys to grant or deny access to the users. In contrast, in CP-ABE, the encryptor is able to decide who should or should not have access to the data that he encrypts. In the applications we target such as pay-TV, this would mean that the access control is either dynamically managed by the encryptor (with a CP-ABE) or statically managed by the key-issuer (with a KP-ABE), while in real-life a third-party could be in charge of a dynamic policy.

In KP-ABE, the access policy is controlled at the key generation phase, while in CP-ABE, the access policy is controlled at the message encryption phase. We go a step further in this consideration by postponing the management of the access policy to a later phase and show how one can manage the access policies without knowing any secret nor the content of message.

Previous works on CP-ABE consider classical encryption: the encryptor, taking as input an access policy and a message, produces a corresponding ciphertext. The encryptor thus manages both the access policy and the encryption of the original message. This scenario is unavoidable when limiting the access policy as a single atomic attribute characterizing a user's identity (*e.g.,* identity-based encryption) or a target group of users (*e.g.,* identity-based broadcast encryption) because the encryptor needs to know the message to encrypt with the single attribute. However, in the general case, where the access policy is composed from sub-policies via AND and OR operators, the encryption of a message for the whole access policy can be computed from the ciphertexts of the sub-policies, without the knowledge of the original message.

Aiming to this scenario, where a combiner should manage the access policy without knowledge of the original message, we need an additional property in ABE: the homomorphic-policy. This property weakens the security of an ABE when the combiner colludes with legitimate users. However, in our practical application, there is no incentive for the combiner to break the scheme. The combiner is indeed involved in the protocol to improve on the flexibility of the access control, and even if it is corrupted, there is no harm for the system, comparing to the scenario where there is no combiner and everything is managed by a unique authority.

Considering Pay-TV, we can now separate the roles of the content provider and of the manager of the access policies (see Fig. 1.4): the content provider (C) encapsulates the same session key K under each attribute, encrypts the content under this session key K, and provides the encapsulation together with the encrypted content to the manager of the access policies (M). The latter broadcasts the encrypted content, but according to the access policy, it combines the appropriate encapsulations to produce a unique encapsulation, to be broadcast to the users (the recipients (R)). Each authorized user can decrypt this encapsulation (by owning attributes satisfying the access policy) and get the session key to decrypt the content. All secrets and keys are provided by a security manager (S), a trusted, incorruptible entity. His role

is not considered in our work, or is implicitly played by the content provider (C).

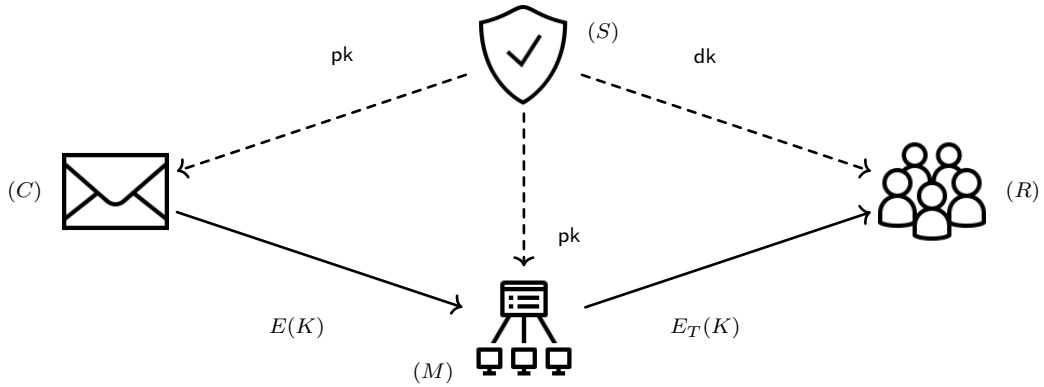

Figure 1.4: Separating the roles, the content provider (C) encapsulates the key K of his stream with the public key pk, and provides this encapsulation $E(K)$ to the content manager (M). This one chooses the target T and fixes the policy with the public key, and distributes the encapsulation $E_T(K)$ to the final receiver (R). (C) and (M) can be totally independents, or not. Also, (C) can play the role of the security manager (S), or not.

We can also envisage another case where the entities (C) and (M) are totally independent. To illustrate this, let us assume the manager (M) is a service of video conferencing (again referring to Fig. 1.4), and the content provider (C) is a client that asks (A) to organize a meeting with the participants (R). The authorized participants are identified by several attributes. At the moment of the meeting, (C) secretly gives (M) the encapsulations of the session key K, under the various attributes, so that it can publicly distribute it according to the appropriate policy to the participants. Only the authorized participants get access to the session K and can participate to the meeting. The manager (M) does not learn any secret information, and cannot eavesdrop the meeting.

As explained in the above context, the homomorphic-policy property is compatible for key encapsulation rather than for encryption. Technically, we thus need to define attribute-based key encapsulation mechanisms (ABKEM) which encapsulate a session key for an access policy. Then, the combination of two encapsulations of the same session key under two sub-policies into an encapsulation for the composed access policy is completed via the homomorphic-policy property: if we have encapsulations of a session key K under two policies p_1 and p_2, we will be able to produce an encapsulation of the same session key K for the policies $p_1 \vee p_2$ and $p_1 \wedge p_2$. The achievement of an homomorphic-policy ABKEM is the main contribution of this chapter. But of course, this is important to keep all the initial properties of an ABE scheme, and namely the collusion-resistance of the final encapsulation.

As explained above, our main contribution is the definition and construction of homomorphic-policy attribute-based key encapsulation mechanisms (HP-ABKEM). To this aim, we define properly the homomorphic policy property and ABKEM. We exploit special properties of LSSS for AND and OR operations and transforms them in an efficient way of combining the corresponding encapsulations. We also propose an efficient randomization method for making any ciphertext (possibly obtained

from the above combinations) statistically indistinguishable from a fresh ciphertext targeting the same policy, this part is important for the security of the system. Finally, we describe a construction of ABKEM that relies on the Lewko-Waters ABE scheme [LW11], which security holds in the random-oracle model. Putting altogether, our final result gives an HP-ABKEM which is as efficient as the Lewko-Waters ABE system. It is interesting that we get the homomorphic-policy property without paying an extra cost. Actually, the final encapsulation after several combinations turns out to be the same as the one the Lewko-Waters sender would have produced, hence the same security level, and namely the collusion-resistance (in the random-oracle model).

Organisation of this thesis

In addition to this introduction, this work is organized in six technical chapters plus the conclusion.

In Chapter 2 we specify some notations used all along this thesis, and we provide classical definitions and mathematical hypothesis needed for the proofs.

Chapter 3 chapter describes precisely the definitions of (decentralized) multi-client functional encryption, and the associated security notions.

Chapter 4 provides several constructions for the previously defined primitive, one theoretical and two others practical. The two last are parts of the results presented in [CDG+18a].

Chapter 5 describes two constructions of decentralized multi-client functional encryption. The first, generic, is taken from [CDG+18b], and the second from [CDG+18a].

In Chapter 6 we explain how to enhance, in several ways, the security of the schemes described in the previous chapters. All the material of this part are results from the article [CDG+18b]

Chapter 7 is the last technical chapter. It defines and provides a construction of homomorphic-policy attribute-based key encapsulation mechanism. This part gather the contributions of [CPP17].

Finally, the conclusion sum up some open questions, principally about the multi-client functional encryption.

2 Preliminaries

In this chapter we introduce, in this order, notations, standard cryptographic notions, and the mathematical assumptions needed for our work.

Contents

2.1 Notations and mathematical notions

Basics

We define the usual sets $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Z}_N$ as the natural numbers, integers and the integers modulo N. We denote the vector notation $\vec{x} = (x_i)_i$. We denote $\Pr[X = x]$ the probability for a random variable X to be equal to x, and U_n the uniform distribution over $\{0,1\}^n$. For any set S, we denote $s \xleftarrow{\$} S$ a random element uniformly picked in S. The abbreviation **PPT** stands for Probabilistic Polynomial Time. A PPT algorithm is running in polynomial time with respect to the input, meaning that it is asymptotically efficient. An **oracle** is an ideal algorithm which answers any given query, in a black-box way. A random oracle associates and returns uniform randomness to any input. An **adversary** Adv is an entity that tries to extract informations he is not allowed to get. In security proofs he is formalized as an algorithm, usually PPT. A **security parameter**, designated by λ in this work, is a value used to adapt the parameters of a cryptographic protocol to the supposed computational power of an adversary.

Prime order groups

We use a prime-order group generator GGen, a probabilistic polynomial time (PPT) algorithm that on input the security parameter 1^λ returns a description $\mathcal{G} = (\mathbb{G}, p, P)$ of an additive cyclic group $\mathbb{G}$ of order p for a 2λ-bit prime p, whose generator is P.

We use implicit representation of group elements as introduced in [EHK+13]. For $a \in \mathbb{Z}_p$, define $[a] = aP \in \mathbb{G}$ as the *implicit representation* of a in $\mathbb{G}$. More generally, for a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{Z}_p^{n\times m}$ we define $[\mathbf{A}]$ as the implicit representation of $\mathbf{A}$ in $\mathbb{G}$:

$$[\mathbf{A}] := \begin{pmatrix} a_{11}P & \dots & a_{1m}P \\ & & \\ a_{n1}P & \dots & a_{nm}P \end{pmatrix} \in \mathbb{G}^{n\times m}$$

We will always use this implicit notation of elements in $\mathbb{G}$, i.e., we let $[a] \in \mathbb{G}$ be an element in $\mathbb{G}$. Note that from a random $[a] \in \mathbb{G}$, it is generally hard to compute the value a (discrete logarithm problem in $\mathbb{G}$). Obviously, given $[a], [b] \in \mathbb{G}$ and a scalar $x \in \mathbb{Z}_p$, one can efficiently compute $[ax] \in \mathbb{G}$ and $[a + b] = [a] + [b] \in \mathbb{G}$.

Pairing group

We also use a pairing-friendly group generator PGGen, a PPT algorithm that on input 1^λ returns a description $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, P_1, P_2, e)$ of asymmetric pairing groups where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are additive cyclic groups of order p for a 2λ-bit prime p, P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map. Define $P_T := e(P_1, P_2)$, which is a generator of $\mathbb{G}_T$. We again use implicit representation of group elements. For $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$, define $[a]_s = aP_s \in \mathbb{G}_s$ as the implicit representation of a in G_s . Given $[a]_1$, $[a]_2$, one can efficiently compute $[ab]_T$ using the pairing e. For two matrices $\mathbf{A}$, $\mathbf{B}$ with matching dimensions define $e([\mathbf{A}]_1, [\mathbf{B}]_2) := [\mathbf{AB}]_T \in \mathbb{G}_T$.

2.2 Generic cryptography notions

2.2.1 About randomness

Randomness is a crucial element in cryptography for keys, nonces, passwords... However, true randomness is not possible to produce with deterministic algorithms, making it a valuable resource.

Pseudo-random function

This justifies the use of pseudo-randomness, a source indistinguishable from true uniform randomness. In this work (Section 4.1 and Section 5.1.4) we use a pseudo-random function [GGM86], indistinguishable from a randomly chosen function, because of its reproducibility.

Definition 2.1: Pseudo-Random Function family

Let be $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ a family of efficiently computable functions used in the following security game:

- $\mathsf{SetUp}(\lambda)$: takes as input a security parameter λ. The challenger $\mathcal{C}$ picks a random bit $b \xleftarrow{\$} \{0,1\}$, a random key $k \xleftarrow{\$} \mathcal{K}$, and defines $f : x \to F(k,x)$ if $b = 0$ or $f \xleftarrow{\$} \mathcal{F}_{\mathcal{X}\to\mathcal{Y}}$ if $b = 1$.
- $\mathsf{Query}(x)$: the adversary $\mathcal{A}$ submits a request x to $\mathcal{C}$, who computes and returns the value $f(x)$. $\mathcal{A}$ can use this algorithm any polynomial number of time.
- Guess: $\mathcal{A}$ concludes this security game by sending $\mathcal{C}$ his guess b' for the bit b.

A such function family is called Pseudo-Random Function (PRF) family if the following quantity is negligible:

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{PRF}} = P(b' = 0|b = 0) - P(b' = 0|b = 1)$$

Among the different possible properties of a pseudo-random function stands the key homomorphism. This property, defined in [BLMR13], describes the conservation of a structure between the key and output spaces. This property is the main mechanism behind the scheme from Section 4.1.

Definition 2.2: Key-Homomorphic Pseudo-Random Function family

We call Key-Homomorphic Pseudo-Random Function ($\mathsf{KH\text{-}PRF}$) family a set of functions $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, where $(\mathcal{K}, \oplus)$ and $(\mathcal{Y}, \otimes)$ are two groups, with the two following properties:

- F is a PRF.
- $F(k_1, x) \otimes F(k_2, x) = F(k_1 \oplus k_2, x)$ for all keys $k_1, k_2 \in \mathcal{K}$ and values $x \in \mathcal{X}$.

Since the Key-Homomorphism only appears as an additional property, the security definition remains the same as the original PRF.

Leftover hash lemma

Another way to produce randomness is through randomness extractors. Those algorithms, given a "low quality" randomness, output a something "statically close" to the uniform distribution. Here is the formal definition of those notions:

- behind the named quality is the **min-entropy**, defined as the value $\mathsf{ME}(X) := -\log(\max_x \Pr(X = x))$, where is X a random variable.
- the **statistical distance** between two random variables is, given two random variables X, Y, the value $\mathsf{SD}(X, Y) := \frac{1}{2}\sum_x |\Pr[X = x] - \Pr[Y = x]|$

Definition 2.3: Randomness Extractor

A function $\mathsf{Ext} : \mathcal{X} \times \{0,1\}^n \to \{0,1\}^v$ is an (m, ϵ)-extractor (for space $\mathcal{X}$), if for all (X, Z) such that X is distributed over $\mathcal{X}$ and $\mathsf{ME}(X|Z) \geq m$, we get $\mathsf{SD}(\mathsf{Ext}(X, S), U_v|(S, Z)) \leq \epsilon$ where $S \sim U_n$ denotes the seed of Ext.

The main difference with the usual pseudo-randomness described above is the proximity to real randomness, the statistical distance offered by extractors being stronger than the computational indistinguishability of pseudo-random generators.

The proof of the scheme Section 5.1.5 requires a practical randomness extractor, and mentions the use of the leftover hash lemma. This result from [ILL89, HILL99] mainly states that specific hash functions, the universal hash functions, can be used as randomness extractors:

Definition 2.4: ρ-Universal Hashing

A family $\mathcal{H}$ of (deterministic) functions $h : \mathcal{X} \to \{0,1\}^n$ is called a ρ-universal hash family (on $\mathcal{X}$), if for any $x_1 \neq x_2 \in \mathcal{X}$ we have $\Pr_{h \xleftarrow{\$} \mathcal{H}}[h(x_1) = h(x_2)] \leq \rho$.

And finally, we recall the Leftover Hash Lemma, following the description from [BDK+11]:

Theorem 2.1: Leftover Hash Lemma

Assume that the family $\mathcal{H}$ of functions $h : \mathcal{X} \to \{0,1\}^n$ is a $\frac{1+\gamma}{2^n}$-universal hash family. Then a function $\mathsf{Ext}(x, h) := h(x)$, where $h \xleftarrow{\$} \mathcal{H}$, is an (m, ϵ)-extractor, where $\epsilon = \frac{1}{2}\sqrt{\gamma + \frac{1}{2^{m-n}}}$.

2.2.2 Basic primitives

Symmetric key encryption

Symmetric encryption is one of the most basic cryptographic primitive. It uses the same key for the encryption and the decryption. Since symmetric encryption

is a pseudo-random permutation, without an included key derivation algorithm the encryption key can be used only once.

Definition 2.5: Symmetric Key Encryption scheme

A Symmetric Key Encryption (SKE) $(\mathsf{SEnc}, \mathsf{SDec})$ with key space $\mathcal{K}$ is defined as:

- $\mathsf{SEnc}(K, m)$: given a key K and a message m, outputs a ciphertext ct;
- $\mathsf{SDec}(K, \mathsf{ct})$: given a key K and a ciphertext ct, output a plaintext.

The following must hold:

Correctness. For all m in the message space, $\Pr[\mathsf{SDec}(K, \mathsf{SEnc}(K, m)) = m] = 1$, where the probability is taken over $K \xleftarrow{\$} \mathcal{K}$.

One time security is a formalization of what offers the classical one time pad. It consists of an unique message sent between two parties sharing a same key.

Definition 2.6: One Time Security

For any PPT adversary $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}^{\mathsf{OT}}_{\mathsf{SKE}}(\mathcal{A}) = \left| 2 \times \Pr\left[b' = b : \begin{array}{l} (m_0, m_1) \leftarrow \mathcal{A}(1^\lambda) \\ K \xleftarrow{\$} \mathcal{K}, b \xleftarrow{\$} \{0,1\}, \mathsf{ct} = \mathsf{SEnc}(K, m_b) \\ b' \leftarrow \mathcal{A}(\mathsf{ct}) \end{array} \right] - 1 \right|.$$

If the key space is larger than the message space, on can simply use the one-time pad to build a one-time secure symmetric encryption. Otherwise, a pseudo-random generator can stretch the key to the right length.

Signature

A signature scheme ensures authenticity and integrity of a message. The signer makes a trace of a message using a secret, which can be publicly verified. Signature schemes may offer several properties (homomorphism, group signature...) and security levels. In this work (more precisely in Section 6.1.4), we use a standard signature scheme, but with a quite strong security: strong unforgeability. Here we provide the definitions of both signature scheme and strong unforgeability:

Definition 2.7: Signature Scheme

A signature scheme on $\mathcal{M}$ is defined by four algorithms:

- $\mathsf{SetUp}(\lambda)$: takes as input the security parameter λ, and outputs the public parameters pp;
- $\mathsf{KeyGen}(\mathsf{pp})$: takes as input the public parameters and outputs a pair of signing and verification keys $(\mathsf{sk}, \mathsf{vk})$;

- $\mathsf{Sign}(\mathsf{sk}, m)$: takes as input a signing key sk and a message $m \in \mathcal{M}$, and outputs a signature s;
- $\mathsf{Verif}(\mathsf{vk}, m, s)$: takes as input the verification key vk, a message m and a signature s, and returns either 1 or 0, whether the signature is valid or not.

Strong unforgeability ensures an adversary cannot create a new pair (message, signature) from a set of pairs of his choice. The difference with the weaker existential notion is that it prevents the adversary to make a new signature from a previously queried message, when it originally was considered as a trivial attack.

Definition 2.8: Strong Unforgeability

Let us consider a signature scheme, no adversary $\mathcal{A}$ should be able to win the following security game against a challenger $\mathcal{C}$:

- Initialize: the challenger $\mathcal{C}$ runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{SetUp}(\lambda)$ as well as the key generation algorithm $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and provides vk to the adversary $\mathcal{A}$;
- Signature queries $\mathsf{QSign}(m)$: $\mathcal{A}$ has unlimited and adaptive access to the signing oracle, and receives the signature $s \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$. One appends (m, s) to the list $\mathsf{Queries}$;
- Finalize: $\mathcal{A}$ provides a pair (m, s). $\mathcal{A}$ wins if $\mathsf{Verif}(\mathsf{vk}, m, s)$ when $(m, s) \notin \mathsf{Queries}$.

We say Sign is strongly unforgeable if for any adversary $\mathcal{A}$, the following value is negligible

$$\mathsf{Adv}^{\mathsf{SUF}}_{\mathsf{Sign}}(\mathcal{A}) = \Pr\left[\begin{array}{c}(m, s) \leftarrow \mathcal{A}^{\mathsf{QSign}}, \\ (m, s) \notin \mathsf{Queries}, \\ \mathsf{Verif}(\mathsf{pp}, m, s) = 1\end{array}\right]$$

Key encapsulation mechanism

Key encapsulation mechanism, as the name presumes, is used to transmit a cryptographic key. It is generally used in the combination of symmetric/asymmetric cryptography to share a key for a symmetric protocol. It is also our use-case in the last chapter (Chapter 7), where we aim to delegate the access control over an encrypted stream.

Definition 2.9: Key Encapsulation Mechanism

An Key Encapsulation Mechanism (KEM) is defined by the three following algorithms:

- $\mathsf{SetUp}(\lambda)$: takes as input the security parameter, and outputs the master secret key sk and the public key pk;
- $\mathsf{Encaps}(\mathsf{pk})$: takes as input the public key pk to output a key K and an encapsulation E of this key;

- $\mathsf{Decaps}(\mathsf{sk}, E)$: takes as input a decryption key and an encapsulation E, to output the encapsulated key K or $\perp$.

Correctness. For any $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$ and $(K, E) \leftarrow \mathsf{Encaps}(\mathsf{pk}, p)$, $\mathsf{Decaps}(\mathsf{sk}, E) = K$.

Indistinguishability. Security notion very similar to the indisguishability of public key encryption, but here the adversary cannot distinguish between two encapsulated keys instead of messages.

Definition 2.10: `IND`-security game for KEM

Let us consider an KEM. No adversary $\mathcal{A}$ should be able to break the following security game against a challenger:

- Initialization: the challenger runs the setup algorithm to get $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$, and provides pk to the adversary $\mathcal{A}$;
- Challenge: the challenger runs $(K, E) \leftarrow \mathsf{Encaps}(\mathsf{pk}, p)$, and sets $K_b \leftarrow K$ and K_{1-b} as a random key, for a random bit b. It provides (E, K_0, K_1) to the adversary;
- Finalize: the adversary $\mathcal{A}$ outputs its guess b' on the bit b.

We then define $\mathsf{Adv}^{\mathtt{ind}}(\mathcal{A}) = |2 \times \Pr[b' = b] - 1|$, and say that an KEM is (t, ε)-adaptively secure if no adversary $\mathcal{A}$ running within time t can get $\mathsf{Adv}^{\mathtt{ind}}(\mathcal{A}) \geq \varepsilon$.

Secret sharing schemes

For any application with limited access, one needs to define the *access structure*, which precises which combinations of conditions grant access to the data or to the system.

Definition 2.11: Access Structure

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ be a set of parties (human players or attributes). An access structure in $\mathcal{P}$ is a collection $\mathbb{A} \subseteq 2^{\mathcal{P}} \backslash \{\varnothing\}$. The sets in $\mathbb{A}$ are called the authorized sets, while the others are called unauthorized sets.

When some minimal sets of parties are required to access the system (but any superset is good too), only monotone access structures make sense, since one can always ignore any supplementary party.

Definition 2.12: Monotone Access Structure

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ be a set of parties and $\mathbb{A}$ an access structure in $\mathcal{P}$. $\mathbb{A}$ is said monotone if, for any subsets $B, C \subseteq \mathcal{P}$, if $B \subseteq C$, when $B \in \mathbb{A}$ then $C \in \mathbb{A}$.

In order to control access rights according to a monotone access structure, the use of a secret sharing scheme that spreads the secret key among several players is a classical technique. One must use a secret sharing scheme that just allows authorized sets to reconstruct the secret key. This is even better if the secret key is never fully reconstructed, but just in a virtual way to run the restricted process (such as signature or decryption).

Definition 2.13: Secret Sharing Scheme

A secret sharing scheme over a set of parties $\mathcal{P}$, for an access structure $\mathbb{A}$ over $\mathcal{P}$, allows to share a secret s among the players, with shares $\nu_1, \ldots, \nu_m$ such that:

- any set of parties in $\mathbb{A}$ can efficiently reconstruct the secret s from their shares;
- any set of parties not in $\mathbb{A}$ has no information about the secret s from their shares.

A linear secret sharing scheme is quite appropriate to share a secret key in order to run the restricted process in a distributed way, since many cryptographic primitives have such linear properties.

Definition 2.14: Linear Secret Sharing Scheme

A Linear Secret Sharing Scheme (LSSS) over a field $\mathbb{K}$ and a set of parties $\mathcal{P}$ is defined by a share-generating matrix $\mathbf{A} \in \mathbb{K}^{m \times n}$ and a labeling map $\rho : \{1, \ldots, m\} \to \mathcal{P}$ according to the access policy $\mathbb{A}$: for any $I \subset \{1, \ldots, m\}$, anyone can efficiently find a vector $\vec{c} \in \mathbb{K}^m$ with support I such that $\vec{c}^t \cdot \mathbf{A} = (1, 0, \ldots, 0)$ if and only if $\rho(I) \in \mathbb{A}$.

2.2.3 Functional encryption

Functional encryption is the core of this work. In this section we provide its original definition, which we need to understand the scheme from [ABDP15] cited in the introduction, and used in Section 6.2. We then give the multi-input version to understand the differences with the multi-client Definition 3.2, also cited in the introduction. And finally we describe the attribute-based encryption, needed for Chapter 7.

The original functional encryption

As mentioned in the introduction, functional encryption originally aims to generalize the public key encryption, covering encryption schemes with different access control, or computation over encrypted data. It works like public key encryption, but returns a function of the key and the message. Public key encryption is thus a specific instantiation of functional encryption with a functionality that returns $\perp$ or the message depending of the validity of the key.

Definition 2.15: Functional Encryption

A private-key, single input Functional Encryption (FE) for a family $\mathcal{F}$ consists of the following PPT algorithms:

- $\mathsf{SetUp}(\lambda)$: on input a security parameter, it outputs a master secret key msk and a public key mpk. The latter is implicitly input of all other algorithms.
- $\mathsf{Encrypt}(\mathsf{msk}, x)$: on input the master secret key and a message x, it outputs a ciphertext ct.
- $\mathsf{DKeyGen}(\mathsf{msk}, f)$: on input the master secret key and a function $f \in \mathcal{F}$, it outputs a decryption key dk_f.
- $\mathsf{Dec}(\mathsf{ct}, \mathsf{dk}_f)$: deterministic algorithm that returns $f(x)$, if ct is a valid encryption of x, or $\perp$ otherwise.

Correctness and security, as defined below, must hold:

Correctness. For any message x, and any function f in the family $\mathcal{F}$, we have: $\Pr[\mathsf{Dec}(\mathsf{ct}, \mathsf{dk}_f) = f(x)] = 1$, where the probability is taken over $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{SetUp}(\lambda)$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{msk}, x)$, and $\mathsf{dk}_f \leftarrow \mathsf{DKeyGen}(\mathsf{msk}, f)$.

Indistinguishability. The security notion is defined by an indistinguishability game. Like the classical public key encryption, the goal is to distinguish the bit hidden in a ciphertext. However, the adversary now is allowed to compute a functionality with functional decryption keys, so this functionality must not trivially reveal this bit:

Definition 2.16: IND-Security Game for FE

Let FE be a functional encryption scheme. No adversary $\mathcal{A}$ should be able to win the following security game:

- Initialize: runs $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{SetUp}(\lambda)$, choose a random bit $b \xleftarrow{\$} \{0, 1\}$ and returns mpk to $\mathcal{A}$.
- $\mathsf{QLeftRight}(x_0, x_1)$: on input two messages (x_0, x_1), returns $\mathsf{Enc}(\mathsf{msk}, x_b)$.
- $\mathsf{QDKeyGen}(f)$: on input a function $f \in \mathcal{F}$, returns $\mathsf{DKeyGen}(\mathsf{msk}, f)$.
- Finalize: it outputs the guess b' of $\mathcal{A}$ on the bit b, unless some f was queried to QDKeyGen and (x_0, x_1) was queried to QLeftRight such that $f(x_0) \neq f(x_1)$, in which case it outputs a uniformly random bit, independent of $\mathcal{A}$'s guess.

The adversary $\mathcal{A}$ has unlimited and adaptive access to the left-right encryption oracle QLeftRight, and to the key generation oracle QDKeyGen. We say FE is IND-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathtt{IND}}_{\mathsf{FE}}(\mathcal{A}) = |\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]|$ is negligible.

Multi-input functional encryption

Multi-input functional encryption is the first attempt to extend the functional encryption setting to multi-party computation. This protocol, like the original functional encryption, allows the computation of a function over an encrypted input, the difference is that this ciphertext is now an aggregation from several independent sources. In that respect, multi-input functional encryption and multi-client functional encryption (Section 3.1) are very close. Roughly speaking, the latter can be considered as an extension of the former that allows re-usability through labels.

Definition 2.17: Multi-Input Functional Encryption

A private-key, Multi-Input Functional Encryption (MIFE) for a family $\mathcal{F}$ over n users consists of the following PPT algorithms:

- $\mathsf{SetUp}(\lambda)$: on input a security parameter, it outputs a master secret key msk, encryption keys $(\mathsf{ek}_i)_i$ and a public key mpk. The latter is implicitly input of all other algorithms.
- $\mathsf{Encrypt}(\mathsf{ek}_i, x_i)$: on input an encryption key ek_i and a message x_i, it outputs a ciphertext ct.
- $\mathsf{DKeyGen}(\mathsf{msk}, f)$: on input the master secret key and a function $f \in \mathcal{F}$, it outputs a decryption key dk_f.
- $\mathsf{Dec}((\mathsf{ct}_i)_{i\in[n]}, \mathsf{dk}_f)$: deterministic algorithm that returns $f(\vec{x})$, if $(\mathsf{ct}_i)_{i\in[n]}$ is a valid encryption of $\vec{x} = (x_i)_i \in \mathcal{M}^n$, or $\perp$ otherwise.

Correctness and security, as defined below, must hold:

Correctness. For any vector of messages $\vec{x}$, and any function f in the family $\mathcal{F}$, we have: $\Pr[\mathsf{Dec}((\mathsf{ct}_i)_{i\in[n]}, \mathsf{dk}_f) = f(\vec{x})] = 1$, where the probability is taken over $(\mathsf{msk}, (\mathsf{ek}_i)_i, \mathsf{mpk}) \leftarrow \mathsf{SetUp}(\lambda)$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x_i)$, and $\mathsf{dk}_f \leftarrow \mathsf{DKeyGen}(\mathsf{msk}, f)$.

Indistinguishability. The security notion is quite the same as functional encryption. But since the ciphertext now comes from several sources, the finalize only considers function of full vectors, and thus incomplete ciphertext must not reveal anything to the adversary. This point makes this definition comparable to our `IND`-security, defined in Section 6.1.

Definition 2.18: `IND`-security game for MIFE

Let MIFE be a multi-input functional encryption scheme. No adversary $\mathcal{A}$ should be able to win the following security game:

- Initialize: runs $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{SetUp}(\lambda)$, choose a random bit $b \xleftarrow{\$} \{0,1\}$ and returns mpk to $\mathcal{A}$.
- $\mathsf{QLeftRight}(x_i^0, x_i^1)$: on input two messages (x_i^0, x_i^1), returns $\mathsf{Enc}(\mathsf{ek}_i, x_i^b)$.

- $\mathsf{QDKeyGen}(f)$: on input a function $f \in \mathcal{F}$, returns $\mathsf{DKeyGen}(\mathsf{msk}, f)$.
- Finalize: it outputs the guess b' of $\mathcal{A}$ on the bit b, unless some f was queried to $\mathsf{QDKeyGen}$ and $(\vec{x}_0, \vec{x}_1)$ was queried to $\mathsf{QLeftRight}$ such that $f(\vec{x}_0) \neq f(\vec{x}_1)$, in which case it outputs a uniformly random bit, independent of $\mathcal{A}$'s guess.

The adversary $\mathcal{A}$ has unlimited and adaptive access to the left-right encryption oracle $\mathsf{QLeftRight}$, and to the key generation oracle $\mathsf{QDKeyGen}$. We say FE is $\mathtt{IND}$-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathtt{IND}}_{\mathsf{FE}}(\mathcal{A}) = |\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]|$ is negligible.

Attribute-based encryption

Finally, the last part of our work focuses on a specific use of attribute-based encryption. Like the original public key encryption, it is an all-or-nothing functionality, meaning that the decryption returns either $\perp$ or the original message. The specificity here is in the access control management; a predicate P, corresponding to the access policy, and attributes $\mathsf{a} \in \mathfrak{A}$ are encoded somewhere in the ciphertext or in the key, and the decryption returns the original message if and only if $P(\mathsf{a}_1, \cdots, \mathsf{a}_k) = 1$. When the predicate is encoded in the ciphertext and the attributes in the keys, it is called Ciphertext Policy attribute-based encryption. In the opposite case, it is called Key Policy attribute-based encryption, but our work only considers the first case:

Definition 2.19: Ciphertext-Policy Attribute-Based Encryption

An Ciphertext-Policy Attribute-Based Encryption ($\mathsf{CP\text{-}ABE}$) scheme over an attribute space $\mathfrak{A}$ is defined by four algorithms:

- $\mathsf{SetUp}(\lambda)$: takes as input the security parameter, and outputs the master secret key msk and the public key pk;
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{A})$: takes as input the master secret key msk and a set of attributes A, to output the private decryption key dk_{A};
- $\mathsf{Encrypt}(\mathsf{pk}, p, m)$: takes as input the public key, a policy p, and a message m, to output the encryption $C_p = (\mathsf{ct}, p)$;
- $\mathsf{Decrypt}(\{\mathsf{dk}, C)$: takes as input a decryption key dk and an encryption C, to output a message m or $\perp$.

Correctness. For any $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$, $\mathsf{dk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{A})$, and $C_p \leftarrow \mathsf{Encrypt}(\mathsf{pk}, p, m)$, $\mathsf{Decrypt}(\mathsf{dk}, C_p) = m$ if A satisfies the policy p.

Indistinguishability. The main security property is the usual indistinguishability ($\mathtt{IND}$), similar to the usual public key encryption. It should prevent leaks from adaptively chosen decryption keys, thus modeling potential collusions:

Definition 2.20: IND-security game for CP-ABE

Let us consider an CP-ABE over an attribute space $\mathfrak{A}$. No adversary $\mathcal{A}$ should be able to break the following security game against a challenger:

- Initialization: the challenger runs the setup algorithm to get the values $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$, and provides pk to the adversary $\mathcal{A}$;
- Key Queries QKeyGen: the adversary $\mathcal{A}$ can ask KeyGen-queries, for any attribute set A of its choice to get dk_{A};
- Challenge Challenge: the adversary $\mathcal{A}$ provides a policy p and two messages m_0, m_1 to the challenger that picks a random bit b and runs $C_p \leftarrow \mathsf{Encrypt}(\mathsf{pk}, p, m_b)$. It provides C_p to the adversary;
- Key Queries QKeyGen: the adversary $\mathcal{A}$ can again ask KeyGen-queries of its choice;
- Finalize: the adversary $\mathcal{A}$ outputs its guess b' on the bit b.

We also define the event Cheat, which means that a user owns a set of attributes A that satisfies p: in such a case, the adversary can trivially guess b. Hence, we only allow adversaries such that $\Pr[\mathsf{Cheat}] = 0$. We then define $\mathsf{Adv}^{\mathtt{ind}}(\mathcal{A}) = |2 \times \Pr[b' = b] - 1|$, and say that an CP-ABE is (t, ε)-adaptively secure if no adversary $\mathcal{A}$ running within time t can get $\mathsf{Adv}^{\mathtt{ind}}(\mathcal{A}) \geq \varepsilon$.

2.3 Assumptions

In this part we define the mathematical assumptions used in this work. They are all related to groups, and count among the most classical assumptions in group-based cryptography.

2.3.1 Prime order group assumptions

This first one states that given two random elements from the group, no one can compute their product, which is the basic idea behind every group assumptions.

Definition 2.21: Computational Diffie-Hellman assumption

The Computational Diffie-Hellman (CDH) assumption states that, in a prime-order group $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, no PPT adversary can compute $[xy]$, from $[x]$ and $[y]$ for $x, y \xleftarrow{\$} \mathbb{Z}_p$, with non-negligible success probability.

Equivalently, this assumption states it is hard to compute $[a^2]$ from $[a]$ for $a \xleftarrow{\$} \mathbb{Z}_p$. This comes from the fact that $4[xy] = [(x+y)^2] - [(x-y)^2]$.

The second is the decisional problem associated to the first one, given two random elements from the group, no one can distinguish between the product a third random element.

Definition 2.22: Decisional Diffie-Hellman assumption

The Decisional Diffie-Hellman (DDH) assumption states that, in a prime-order group $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, no PPT adversary can distinguish between the two following distributions with non-negligible advantage:

$$\{([a],[r],[ar]) \mid a, r \xleftarrow{\$} \mathbb{Z}_p\} \text{ and } \{([a],[r],[s]) \mid a, r, s \xleftarrow{\$} \mathbb{Z}_p\}.$$

Equivalently, this assumption states it is hard to distinguish, knowing $[a]$, a random element from the span of $[\vec{a}]$ for $\vec{a} = \binom{1}{a}$, from a random element in $\mathbb{G}^2$: $[\vec{a}] \cdot r = [\vec{a}r] = \binom{[r]}{[ar]} \approx \binom{[r]}{[s]}$.

We can extend this second assumption in a vector of random elements with the help of the following proposition:

Proposition 2.1

For any distinguisher $\mathcal{A}$ running within time t, the best advantage $\mathcal{A}$ can get in distinguishing

$$\begin{aligned} \mathcal{D}_m &= \{(X, (Y_j, Z_j = \mathsf{CDH}(X, Y_j))_j) \mid X, Y_j \xleftarrow{\$} \mathbb{G}, j = 1, \ldots, m\} \\ \mathcal{D}'_m &= \{(X, (Y_j, Z_j)_j) \mid X, Y_j, Z_j \xleftarrow{\$} \mathbb{G}, j = 1, \ldots, m\}. \end{aligned}$$

is bounded by $\mathsf{Adv}^{\mathsf{ddh}}(t + 4m \times t_{\mathbb{G}})$, where $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$.

Proof

One can first note that the best advantage one can get, within time t, between

$$\begin{aligned} \mathcal{D} &= \{(X, Y, Z = \mathsf{CDH}(X, Y)) \mid X, Y \xleftarrow{\$} \mathbb{G}\} \\ \mathcal{D}' &= \{(X, Y, Z) \mid X, Y, Z \xleftarrow{\$} \mathbb{G}\}. \end{aligned}$$

is bounded by $\mathsf{Adv}^{\mathsf{ddh}}(t)$. This is actually the DDH assumption. One can note that $\mathcal{D}_m$ and $\mathcal{D}'_m$ can be rewritten as

$$\begin{aligned} \mathcal{D}_m &= \{(X, (Y_j = g^{u_j} Y^{v_j}, Z_j = X^{u_j} \cdot \mathsf{CDH}(X, Y)^{v_j})_j) \mid X, Y \xleftarrow{\$} \mathbb{G}, u_j, v_j \xleftarrow{\$} \mathbb{Z}_p\} \\ \mathcal{D}'_m &= \{(X, (Y_j = g^{u_j} Y^{v_j}, Z_j = X^{u_j} \cdot Z^{v_j})_j) \mid X, Y, Z \xleftarrow{\$} \mathbb{G}, u_j, v_j \xleftarrow{\$} \mathbb{Z}_p\} \end{aligned}$$

Since, from (X, Y, Z), the m tuples require 4 additional exponentiations per index j, one get the expected bound.

2.3.2 Pairing group assumptions

These assumptions, also considered classical, concerns the kind of groups we refer to in Section 2.1. The main difference with prime order groups is that the group $\mathbb{G}_T$ now allows one multiplication.

This first assumption relies with the original group assumptions, presuming that DDH holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

Definition 2.23: Symmetric eXternal Diffie-Hellman assumption

The Symmetric eXternal Diffie-Hellman (SXDH) assumption states that, in a pairing group $\mathcal{PG} \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

The second really exploits the potential offered by the pairing. Three random elements from $\mathbb{G}_1$ or $\mathbb{G}_2$ are given, and the goal is to distinguish between the product of those three elements in $\mathbb{G}_T$ or a random element from $\mathbb{G}_T$.

Definition 2.24: Decisional Bilinear Diffie Hellman assumption

The Decisional Bilinear Diffie Hellman (DBDH) assumption states that, in a pairing group $\mathcal{PG} \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, for any PPT adversary, the following advantage is negligible, where the probability distribution is over $a, b, c, s \xleftarrow{\$} \mathbb{Z}_p$:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{A}) = |&\Pr[1 \leftarrow \mathcal{A}(\mathcal{PG}, [a]_1, [b]_1, [b]_2, [c]_2, [abc]_T)] \\ &- \Pr[1 \leftarrow \mathcal{A}(\mathcal{PG}, [a]_1, [b]_1, [b]_2, [c]_2, [s]_T)]|.\end{aligned}$$

Once again it can be extended in a vector of elements in the Q-fold DBDH assumption:

Proposition 2.2: Q-fold DBDH

For any integer Q, the Q-fold DBDH assumption states for any PPT adversary, the following advantage is negligible, where the probability distribution is over $a, b, c_i, s_i \xleftarrow{\$} \mathbb{Z}_p$:

$$\begin{aligned}\mathsf{Adv}^{Q\text{-}\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{A}) = |&\Pr[1 \leftarrow \mathcal{A}(\mathcal{PG}, [a]_1, [b]_1, [b]_2, \{[c_i]_2, [abc_i]_T\}_{i\in[Q]})] \\ &- \Pr[1 \leftarrow \mathcal{A}(\mathcal{PG}, [a]_1, [b]_1, [b]_2, \{[c_i]_2, [s_i]_T\}_{i\in[Q]})]|.\end{aligned}$$

And this lemma prove that Q-fold DBDH assumption is equivalent to classical DBDH assumption:

Lemma 2.1: Random self reducibility of DBDH

For any adversary $\mathcal{A}$ against the Q-fold DBDH, running within time t, there exists an adversary $\mathcal{B}$ running within time $t + 2Q(t_{\mathbb{G}_T} + t_{\mathbb{G}_2})$, where $t_{\mathbb{G}_T}$ and $t_{\mathbb{G}_2}$ denote respectively the time for an exponentiation in $\mathbb{G}_T$ and $\mathbb{G}_2$ (we only take into account the time for exponentiations here), such that

$$\mathsf{Adv}^{Q\text{-}\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{B}).$$

Proof

Upon receiving a DBDH challenge $(\mathcal{PG}, [a]_1, [b]_1, [b]_2, [c]_2, [s]_T)$, $\mathcal{B}$ samples the values $\alpha_i, c'_i \xleftarrow{\$} \mathbb{Z}_p$ computes $[c_i]_2 := [\alpha_i \cdot c]_2 + [c'_i]_2$, $[s_i]_T := [\alpha_i \cdot s]_T + [c_i \cdot ab]_T$ for all $i \in [Q]$, and gives the challenge $(\mathcal{PG}, [a]_1, [b]_1, [b]_2, \{[c_i]_2, [s_i]_T\}_{i\in[Q]})$ to $\mathcal{A}$.

3 Multi-client functional encryption

This section is devoted to formalize multi-client functional encryption, decentralized multi-client functional encryption and their security models. We also discuss about the links and differences between several security level.

Contents

3.1 Multi-client functional encryption

We now define a private-key MCFE as in [CDG+18b], which is a variant of the definition from [GGG+14, GKL+13, CDG+18a]. The difference is, since we consider the symmetric setting, we allow users to decrypt using their own keys, to get back their component plaintexts at least when the ciphertexts were fully generated.

3.1.1 Definition

An MCFE scheme encrypts vectors of data from several senders i using personal encryption keys ek_i, and it allows the controlled computation of functions f on these heterogeneous data with a functional decryption key dk_f. In this work, we consider the symmetric-key setting, where the encryption key ek_i can also serve to decrypt individual ciphertext for slot $i \in [n]$, using the Decrypt algorithm. It works like the functional decryption algorithm FDecrypt, they both take as input a complete vector of individual ciphertexts, but the former uses ek_i to decrypt an individual ciphertext, while the latter uses dk_f to evaluate f on the global plaintext. The algorithm Decrypt will be used when proving CCA security in Section 6.3. We can thus define the MCFE definition as follows.

Definition 3.1: Multi-Client Functional Encryption

A multi-client functional encryption (MCFE) on $\mathcal{M}$ over a set of n senders is defined by five algorithms:

- $\mathsf{SetUp}(\lambda)$: takes as input the security parameter λ, and outputs the public parameters mpk, the master secret key msk and the n private encryption keys ek_i;
- $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$: takes as input a user encryption key ek_i, a value x_i to encrypt, and a label ℓ, and outputs the ciphertext $C_{\ell,i}$;
- $\mathsf{DKeyGen}(\mathsf{msk}, f)$: takes as input the master secret key msk and a function $f : \mathcal{M}^n \to \mathcal{R}$, and outputs a functional decryption key dk_f;
- $\mathsf{FDecrypt}(\mathsf{dk}_f, \ell, \vec{C})$: takes as input a functional decryption key dk_f, a label ℓ, and an n-vector ciphertext $\vec{C}$, and outputs $f(\vec{x})$, if $\vec{C}$ is a valid encryption of $\vec{x} = (x_i)_i \in \mathcal{M}^n$ for the label ℓ, or $\perp$ otherwise.
- $\mathsf{Decrypt}(\mathsf{ek}_i, \ell, \vec{C})$: takes as input a label ℓ, a user encryption key ek_i and an n-vector ciphertext $\vec{C}$, and outputs a value x_i, if $\vec{C}$ is a valid encryption of $\vec{x} = (x_i)_i \in \mathcal{M}^n$ for the label ℓ, or $\perp$ otherwise.

We will assume public parameters being implicitly included in the keys.

Correctness. Given $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$, for any label ℓ, any function $f : \mathcal{M}^n \to \mathcal{R}$, and any vector $\vec{x} = (x_i)_i \in \mathcal{M}^n$, if $C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$, for $i \in \{1, \ldots, n\}$, and $\mathsf{dk}_f \leftarrow \mathsf{DKeyGen}(\mathsf{msk}, f)$, then $\mathsf{FDecrypt}(\mathsf{dk}_f, \ell, \vec{C}_\ell = (C_{\ell,i})_i) = f(\vec{x} = (x_i)_i)$ and $\mathsf{Decrypt}(\mathsf{ek}_i, \ell, \vec{C}_\ell) = x_i$.

Relation MIFE-MCFE

One can notice this definition makes MCFE very close to the MIFE described in Definition 2.17. In MIFE, every ciphertext for every slot can be combined with any other ciphertext for any other slot, and used with functional decryption keys to decrypt an exponential number of values, as soon as there are more than one ciphertext per slot. This "mix-and-match" feature is crucial for some of the applications of MIFE, such as building indistinguishability obfuscation [GGG+14]. However, it also means the information leaked about the underlying plaintext is enormous, and in many applications, the security guarantees simply become void, especially when many functional decryption keys are queried. In the case of inner product, as soon as m well-chosen functional decryption keys are queried (*i.e.* for linearly independent vectors), the plaintexts are completely revealed. In the multi-client setting however, since only ciphertexts with the same label can be combined for decryption, information leakage of the plaintext is much controled.

Also we remark that, for both MIFE and MCFE, private-key is more relevant than their public-key counterparts. Essentially, in a public-key MCFE, an encryption of unknown plaintext x_i, for some label ℓ, can be used together with encryptions of arbitrarily chosen values x'_j for each slot $j \in [n]$ (for the same label ℓ) and a functional decryption key for some function f, to obtain the value $f(x'_1, \cdots, x'_{i_1}, x_i, x'_{i+1}, \cdots, x'_n)$. Since the values x'_j for $j \neq i$ are arbitrarily chosen, this reveals typically too much information on x_i for practical uses. In the case of inner product, that means that, from $\mathsf{Enc}(i, x_i, \ell)$, $\mathsf{dk}_{\vec{y}}$, and the public key, one can efficiently extract the values $x_i y_i + \sum_{j \neq i} x'_j y_j$ for chosen x'_j, which exactly reveals the partial inner product $x_i y_i$ (see [AGRW17] for more details on the limitations of public-key IP-FE in the multi-input setting). In order to prevent the adversary from a trivial win, one should make the restriction that the adversary is only allowed to ask functional decryption keys dk_f for functions f that satisfy $f(x^0_1, \cdot, \ldots, \cdot) = f(x^1_1, \cdot, \ldots, \cdot)$, $f(\cdot, x^0_2, \ldots, \cdot) = f(\cdot, x^1_2, \ldots, \cdot)$, $\ldots$, $f(\cdot, \cdot, \ldots, x^0_n) = f(\cdot, \cdot, \ldots, x^1_n)$. This would essentially exclude any function. A private-key encryption solves this issue, and is still well-suited for practical applications.

3.1.2 Security

The security model is the usual left-or-right indistinguishability [BDJR97], but where the adversary should not be able to get functional decryption keys that trivially help distinguish the encrypted vectors. Also, as explained in [GGG+14, GKL+13, CDG+18a], one has to consider corruptions, since the senders do not trust each other, and they can collude and give their secret keys to the adversary who will play on their behalf.

IND* and IND

The reader will notice this definition describes two different versions, a weaker called IND*, and a stronger, IND. The former is given in [CDG+18a] (more precisely wtr-IND* without QEncrypt, see Section 3.1.2 and Section 4.2.2). In this one there is the additional restriction on incomplete ciphertexts which formerly says that a challenge on a partial encryption of a vector over a label is considerate illegitimate. This constraint may seem artificial, but the construction given in [CDG+18a] was vulnerable to this kind of attack. The latter is an improvement, and appears in

[CDG+18b]. This article fixes the shortcomings of the security achieved in [CDG+18a], principally by allowing the adversary to query the left-or-right encryption oracle for some honest users, but not necessarily all of them, leading to incomplete ciphertexts.

We highlight the differences with the security definition from [CDG+18a] and the one from [CDG+18b]. Namely, the extra requirements in gray corresponds to the IND*-security and the framed to the IND one. Eventually, we note that the IND notion implicitly achieves CPA security. So we also consider CCA security, with additional functional decryption queries (dotted).

Definition 3.2: IND*, IND -CCA -Security Game for MCFE

Let us consider MCFE, a scheme over a set of n senders. No adversary $\mathcal{A}$ should be able to win the following security game against a challenger $\mathcal{C}$, with unlimited and adaptive access to the oracles $\mathsf{QEncrypt}$, $\mathsf{QLeftRight}$, $\mathsf{QFDecrypt}$, $\mathsf{QDKeyGen}$, and $\mathsf{QCorrupt}$ described below:

- Initialize: the challenger $\mathcal{C}$ runs the setup algorithm $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It provides mpk to the adversary $\mathcal{A}$;
- Encryption queries $\mathsf{QEncrypt}(i, x, \ell)$: outputs the ciphertext $C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x, \ell)$;
- Challenge queries $\mathsf{QLeftRight}(i, x^0, x^1, \ell)$: outputs the ciphertext $C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x^b, \ell)$;
- Functional decryption queries $\mathsf{QFDecrypt}(f, \ell, \vec{C})$: the oracle first asks for the functional decryption key dk_f, and then outputs $\mathsf{FDecrypt}(\mathsf{dk}_f, \ell, \vec{C})$.
- Functional decryption key queries $\mathsf{QDKeyGen}(f)$: outputs the functional decryption key $\mathsf{dk}_f \leftarrow \mathsf{DKeyGen}(\mathsf{msk}, f)$;
- Corruption queries $\mathsf{QCorrupt}(i)$: outputs the encryption key ek_i;
- Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs the result β of the security game, according to the analysis given below.

The output β of the game depends on some conditions, where $\mathcal{CS}$ is the set of corrupted senders (the set of indexes i input to $\mathsf{QCorrupt}$ during the whole game), and $\mathcal{HS}$ the set of honest (non-corrupted) senders. We set the output to $\beta \leftarrow b'$, unless one of the cases below is true, in which case we set $\beta \xleftarrow{\$} \{0,1\}$:

1. some $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$-query has been asked for an index $i \in \mathcal{CS}$ with $x_i^0 \neq x_i^1$ when encryption queries have been asked for all $i \in \mathcal{HS}$;
2. for some label ℓ and for some function f asked to $\mathsf{QDKeyGen}$, there exists a pair of vectors $(\vec{x}^0 = (x_i^0)_i, \vec{x}^1 = (x_i^1)_i)$ such that $f(\vec{x}^0) \neq f(\vec{x}^1)$, when
   - $x_i^0 = x_i^1$, for all $i \in \mathcal{CS}$;
   - $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$-queries (or $\mathsf{QEncrypt}(i, x_i, \ell)$-queries if $x_i = x_i^0 = x_i^1$) have been asked for all $i \in \mathcal{HS}$;

3. for some label ℓ, a challenge query $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$ has been asked for some $i \in \mathcal{HS}$, but challenge queries $\mathsf{QLeftRight}(j, x_j^0, x_j^1, \ell)$ or encryption queries $\mathsf{QEncrypt}(j, x_j, \ell)$ have not all been asked for all $j \in \mathcal{HS}$.

4. for some $\mathsf{QFDecrypt}(f, \ell, \vec{C} = (C_i)_i)$-query, we have
   - the answer was not $\perp$;
   - there exist two vectors $\vec{x}^0$ and $\vec{x}^1$ such that for all $i \in \mathcal{CS}$: $x_i^0 = x_i^1$, for all $i \in \mathcal{HS}$: either there is a query $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$ that led to C_i, or $x_i^0 = x_i^1 = x_i$ and there is a query $\mathsf{QEncrypt}(i, x_i, \ell)$ that led to C_i.
   - the above vectors $\vec{x}^0$ and $\vec{x}^1$ satisfy $f(\vec{x}^0) \neq f(\vec{x}^1)$.

We say MCFE is $\mathtt{IND}$-secure if for any adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathtt{IND}}_{\mathsf{MCFE}}(\mathcal{A}) = |\Pr[\beta = 1|b = 1] - \Pr[\beta = 1|b = 0]|$ is negligible.

About the finalize

The two first excluded cases are situations where the adversary could trivially distinguish the encrypted vectors, they are thus considered illegitimate attacks:

1. since we are dealing with symmetric-key encryption, where the encryption key and the decryption key are the same, a $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$-query with $x_i^0 \neq x_i^1$, for $i \in \mathcal{CS}$ leaks b (either at the $\mathsf{QLeftRight}$-query time or at the corruption-time). In our stronger security model, this criteria is less restrictive, applying only when honest encryption are all queried;

2. for any functional decryption key, all the possible evaluations should not trivially allow the adversary to distinguish the ciphertexts generated through $\mathsf{QLeftRight}$-queries (on honest components), only when ciphertexts are complete;

And the last condition is classical, in the $\mathtt{CCA}$-setting, and here in the functional encryption context, as we consider illegitimate functional decryption queries on challenge ciphertexts that could result in a different values depending on the value of b. For such illegitimate attacks, the guess of the adversary is not considered (a random bit β is output). Otherwise, this is a legitimate attack, and the guess b' of the adversary is output.

Other variants

We also define several variants of the security game:

- where the adversary must announce in advance the corruption ($\mathsf{QCorrupt}$) queries: static security ($\mathtt{sta\text{-}IND^*}$/$\mathtt{sta\text{-}IND}$);
- where the adversary must announce in advance the challenge ($\mathsf{QLeftRight}$) queries: selective security ($\mathtt{sel\text{-}IND^*}$/$\mathtt{sel\text{-}IND}$);
- where the adversary is limited to one encryption/challenge query on each (i, ℓ) (later queries with the same i and ℓ will be ignored by $\mathsf{QEncrypt}$ and $\mathsf{QLeftRight}$):

without-repetition security (wtr-IND*/wtr-IND). It was a problematic characteristic of the security definition of [CDG+18a] fixed in [CDG+18b], the resulting construction is described in Section 6.2;

- where the adversary is limited to challenge queries on one particular label: 1-Label-IND*-security. The 1-Label-IND* security is exactly the same security notion as IND* where the challenge QLeftRight oracle can only be queried with the same label. Hence, as above, the index ρ of the target label ℓ^* is provided by the adversary, at the beginning, and so we can assume that all the encryption queries for $\ell^* = \ell_\rho$ are asked to the QLeftRight oracle, while the other encryption queries are asked to the QEncrypt oracle. It is known that 1-Label-IND* and IND* are equivalent [BDJR97], but the former is more convenient in the CCA security proof in Section 6.3;

About the oracle QEncrypt

Note that in the IND security game, the oracle QEncrypt can be simulated by QLeftRight, queried on input $x_i^0 = x_i^1$. However, in the IND*, this oracle gives more power to the adversary: it is not possible for the adversary to query QLeftRight on some but not all input slots, for a given label (this is the condition 3. from Finalize), but it can query QEncrypt on incomplete ciphertexts, without triggering Finalize to output a random bit. This will be helpful when going from IND* to IND security, in Section 6.1.

3.2 Decentralization

Here, we define decentralized multi-client functional encryption (DMCFE), where the generation of functional decryption key is decentralized, and only requires individual secret keys, instead of the master secret key.

3.2.1 Definition of decentralized multi-client functional encryption

In MCFE, an authority owns a master secret key msk to generate the functional decryption keys. We would like to avoid such a powerful authority, and make the scheme totally decentralized among the owners of the data (the senders). We thus define DMCFE, for decentralized multi-client functional encryption. In this context, there are n senders $(\mathcal{S}_i)_i$, for $i = 1, \ldots, n$, who will play the role of both the encrypting players and the functional decryption key generators, for a functional decryptor $\mathcal{FD}$. Of course, the senders do not trust each other and they want to control the functional decryption keys that will be generated. There may be several functional decryptors, but since they could collude and combine all the functional decryption keys, in the description below, and in the security model, we will consider only one functional decryptor $\mathcal{FD}$. We could simply use the definition of MCFE [GGG+14, GKL+13], where the setup and the functional decryption key algorithms are replaced by MPC protocols among the clients. But this could lead to a quite interactive process. We thus focus on efficient one-round key generation protocols DKeyGen that can be split in a first step DKeyGenShare that generates partial keys and the combining algorithm DKeyComb that combines partial keys into the functional decryption key.

Definition 3.3: Decentralized Multi-Client Functional Encryption

A decentralized multi-client functional encryption (DMCFE) on $\mathcal{M}$ between a set of n senders $(\mathcal{S}_i)_i$, for $i = 1, \ldots, n$, and a functional decrypter $\mathcal{FD}$ is defined by the setup protocol and four algorithms:

- $\mathsf{SetUp}(\lambda)$: this is a protocol between the senders $(\mathcal{S}_i)_i$ that generate their own secret keys sk_i and encryption keys ek_i, and eventually output the public parameters mpk;
- $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$: takes as input a user encryption key ek_i, a value x_i to encrypt, and a label ℓ, and outputs the ciphertext $C_{\ell,i}$;
- $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_f)$: takes as input a user secret key sk_i and a label ℓ_f, and outputs the partial functional decryption key $\mathsf{dk}_{f,i}$ for a function $f : \mathcal{M}^n \to \mathcal{R}$ that is described in ℓ_f;
- $\mathsf{DKeyComb}((\mathsf{dk}_{f,i})_i, \ell_f)$: takes as input the partial functional decryption keys and eventually outputs the functional decryption key dk_f;
- $\mathsf{FDecrypt}(\mathsf{dk}_f, \ell, \vec{C})$: takes as input a functional decryption key dk_f, a label ℓ, and an n-vector ciphertext $\vec{C}$, and outputs $f(\vec{x})$, if $\vec{C}$ is a valid encryption of $\vec{x} = (x_i)_i \in \mathcal{M}^n$ for the label ℓ, or $\perp$ otherwise;
- $\mathsf{Decrypt}(\mathsf{ek}_i, \ell, \vec{C})$: takes as input an encryption key ek_i, a label ℓ, and an n-vector ciphertext $\vec{C}$, and outputs x_i, if $\vec{C}$ is a valid encryption of $x_i \in \mathcal{M}$ for the label ℓ, or $\perp$ otherwise;

We make the assumption that mpk is included in all the secret and encryption keys, as well as the (partial) functional decryption keys. Similarly, the function f might be included in the (partial) functional decryption keys.

Correctness. Given $(\mathsf{mpk}, (\mathsf{sk}_i)_i, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$, for any label ℓ, any function $f : \mathcal{M}^n \to \mathcal{R}$, and any vector $\vec{x} = (x_i)_i \in \mathcal{M}^n$, if $C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$, for $i \in \{1, \ldots, n\}$, and $\mathsf{dk}_f \leftarrow \mathsf{DKeyComb}((\mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_f))_i, \ell_f)$, then we have $\mathsf{FDecrypt}(\mathsf{dk}_f, \ell, \vec{C}_\ell = (C_{\ell,i})_i) = f(\vec{x} = (x_i)_i)$. The correctness property essentially states the combined key corresponds to the functional decryption key.

3.2.2 Security of DMCFE

The IND-security model is quite similar to the previous one for MCFE (see Definition 3.2), the differences occur on the three following points.

First, for the DKeyGen protocol: the adversary has access to transcripts of the communications, thus modeled by a query $\mathsf{QDKeyGen}(i, f)$ that generates a part of the key through $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \ell_f)$, where ℓ_f is a description of f. Consequently, the QDKeyGen takes the user i as input.

- *Functional decryption key queries* $\mathsf{QDKeyGen}(i, f)$*: $\mathcal{A}$ has unlimited and adaptive access to the (non-corrupted) senders running the* $\mathsf{DKeyGenShare}(\mathsf{sk}_i, f)$

algorithm for any input function f of its choice. It is given back the partial functional decryption key $\mathsf{dk}_{f,i}$*;*

Secondly, corruption queries additionally reveal the secret keys sk_i used in the DKeyGenShare algorithm:

- *Corruptions queries* $\mathsf{QCorrupt}(i)$*: $\mathcal{A}$ can make an unlimited number of adaptive corruption queries on input index i, to get the secret and encryption keys $(\mathsf{sk}_i, \mathsf{ek}_i)$ of any sender i of its choice.*

Last point, the Finalize procedure ignores incomplete functional decryption keys: for condition (2), only functions f for which all the honest key-shares have been asked are considered:

1. some $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$-query has been asked for an index $i \in \mathcal{CS}$ with $x_i^0 \neq x_i^1$ when encryption queries have been asked for all $i \in \mathcal{HS}$;
2. for some label ℓ and for some function f asked to QDKeyGen, there exists a pair of vectors $(\vec{x}^0 = (x_i^0)_i, \vec{x}^1 = (x_i^1)_i)$ such that $f(\vec{x}^0) \neq f(\vec{x}^1)$, when
   - $x_i^0 = x_i^1$, for all $i \in \mathcal{CS}$;
   - $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$-queries (or $\mathsf{QEncrypt}(i, x_i, \ell)$-queries if $x_i = x_i^0 = x_i^1$) have been asked for all $i \in \mathcal{HS}$;
   - $\mathsf{QKeyGen}(i, f)$-queries have been asked for all $i \in \mathcal{HS}$

The critical point is the last one: the distributed key generation must guarantee that without all the shares, no information is known about the functional decryption key. In addition, the protocol must be efficient.

4 Construction of MCFE

In this section we present several constructions of inner product MCFE (IP-MCFE). The first one is generic, using a KH-PRF, and is proven $\texttt{sel-wtr-IND}^*$-secure. The two others are group-based schemes, and were described in [CDG+18a]. One can be considered as the direct instantiation of the first construction, and the security level remains the same. The other is an improvement that reaches the $\texttt{wtr-IND}^*$-security against adaptive queries and corruptions.

Contents

4.1 IP-MCFE from key-homomorphic PRF

Here we describe how to build an IP-MCFE from a KH-PRF in a black-box way.

Overview

The main idea is the same as the inner product FE (IP-FE) from [ABDP15], which consists to mask linearly the $x_{\ell,i}$ with some secret $r_{\ell,i}$. The decryption processes by computing $\sum_i (x_{\ell,i} + r_{\ell,i}) \cdot y_i$ and then removing the value $\sum_i r_{\ell,i} \cdot y_i$ with a decryption key $\mathsf{dk}_{\vec{y}}$. There are essentially two constraints on the $r_{\ell,i}$ to satisfy when building such a scheme. First, these values must be indistinguishable from uniform randomness to ensure statistical security of $x_{\ell,i}$. Secondly, they must be correlated with a same label ℓ to correctly decrypt with a same key $\mathsf{dk}_{\vec{y}}$. Eventually, they must be uncorrelated between the different labels to avoid mix-and-match attacks. A KH-PRF F fulfills these conditions if we consider the label ℓ as the input, and a user key ek_i as the key: $r_{\ell,i} = F(\mathsf{ek}_i, \ell)$.

4.1.1 Description

Our construction, using additive notation:

- $\mathsf{SetUp}(\lambda)$: outputs a KH-PRF $F : \mathcal{K} \times \mathcal{X} \longrightarrow \mathcal{Y}$ as public parameter pp, and a personal encryption key $\mathsf{ek}_i \xleftarrow{\$} \mathcal{K}$ for each users $i \in [n]$. We note $\mathsf{msk} = (\mathsf{ek}_i)_i$.
- $\mathsf{Encrypt}(x_i, \mathsf{ek}_i, \ell)$: takes as input a value x_i to encrypt, a personal encryption key ek_i, a label ℓ, and outputs the ciphertext $C_{\ell,i} = F(\mathsf{ek}_i, \ell) + x_i$.
- $\mathsf{DKeyGen}(\mathsf{msk}, \vec{y})$: takes as input msk, the vector $\vec{y}$ that characterizes the function $f_{\vec{y}}(\vec{x}) = \langle \vec{x}, \vec{y} \rangle$ and provides $\mathsf{dk}_{\vec{y}} = (\vec{y}, \langle \mathsf{msk}, \vec{y} \rangle) = (\vec{y}, \langle (\mathsf{ek}_i)_i, \vec{y} \rangle)$.
- $\mathsf{FDecrypt}(\mathsf{dk}_f, \ell, \vec{C}_\ell = (C_{\ell,j})_j)$: takes as input a decryption key dk_f, a label ℓ, and a ciphertext $\vec{C}_\ell$ to compute the value: $\langle \vec{C}_\ell, \vec{y} \rangle - F(\mathsf{dk}_{\vec{y}}, \ell)$.
- $\mathsf{Decrypt}(\mathsf{ek}_i, \ell, \vec{C}_\ell = (C_{\ell,j})_j)$: takes as input a decryption key dk_f, a label ℓ, and a ciphertext $\vec{C}_\ell$ to compute the value: $C_{\ell,i} - F(\mathsf{ek}_i, \ell)$.

Correctness. If the scalar dk in the functional decryption key $\mathsf{dk}_{\vec{y}} = (\vec{y}, dk)$ is indeed $dk = \langle (\mathsf{ek}_i)_i, \vec{y} \rangle$, then:

$$\begin{aligned} \langle \vec{C}_\ell, \vec{y} \rangle - F(\mathsf{dk}_{\vec{y}}, \ell) &= \langle (F(\mathsf{ek}_i, \ell) + x_i)_i, \vec{y} \rangle - F(\mathsf{dk}_{\vec{y}}, \ell) \\ &= F(\langle (\mathsf{ek}_i)_i, \vec{y} \rangle, \ell) + \langle \vec{x}, \vec{y} \rangle - F(\mathsf{dk}_{\vec{y}}, \ell) \\ &= \langle \vec{x}, \vec{y} \rangle \end{aligned}$$

4.1.2 Security analysis

Sadly, the security level guaranteed by this construction is inversely proportional to its simplicity: only selective in encryption queries, limited to one encryption under the pair (i, ℓ), and overall it is not secure if we suppose that the adversary can ask partial ciphertexts only.

Theorem 4.1: `sel-wtr-IND*` Security

The MCFE protocol described above (see Section 4.1) is `sel-wtr-IND*` secure if the function F is a PRF. More precisely, we have

$$\mathsf{Adv}^{\mathtt{IND}}(\mathcal{A}) \leq 2Q_\ell \cdot \mathsf{Adv}(t)^{\mathsf{PRF}}_{\mathsf{KH\text{-}PRF}},$$

for any adversary $\mathcal{A}$ running in time t, where Q_ℓ is the number of labels.

The proof of this theorem consists in a sequence of hybrid games on the labels, from $G_0 = G_{1,1,0}$ to $G_2 = G_{1,Q_\ell,5}$ where Q_ℓ is the number of labels. The idea is to pick a part of the key as a vector of $\langle \vec{x}^0 - \vec{x}^b \rangle$, then inject randomness through the KH-PRF. The following Fig. 4.1 summarizes this proof.

Games $G_{1,q,0}$, $\boxed{G_{1,q,1}}$, $\boxed{G_{1,q,2}}$, $G_{1,q,3}$, $G_{1,q,4}$

$\left(\mathsf{state}, (\ell_j, z_{j,i})_{i\in[n],j\in[Q]}\right) \leftarrow \mathcal{A}(1^\lambda, 1^n)$

where each $z_{j,i} = (x^0_{j,i}, x^1_{j,i}) \in \mathbb{Z}_p^2$, or $z_{j,i} = \perp$, which stands for no query.

Initialize PRFO: pick $\mu \xleftarrow{\$} \mathcal{K}$ Initialize PRFO as RF .

For all $i \in [n]$, $s_i \xleftarrow{\$} \mathcal{K}$, $\boxed{\text{pick } \mu \xleftarrow{\$} \mathcal{K},}$ and generate a KH-PRF $F : \mathcal{K} \times \mathcal{X} \longrightarrow \mathcal{Y}$.

$\mathsf{ek}_i := s_i$, $\boxed{\mathsf{ek}_i := s_i + \mu(x^0_{i,\ell_q} - x^b_{i,\ell_q})}$, $\mathsf{msk} := (\mathsf{ek}_i)_i$, $\mathsf{mpk} := F$.

When $z_{j,i} = (x^0_{j,i}, x^1_{j,i})$, then $C_{j,i} = \mathsf{QLeftRight}(i, x^0_{j,i}, x^1_{j,i}, \ell_j)$ for $i \in [n], j \in [Q_\ell]$.

$b' \leftarrow \mathcal{A}^{\mathsf{QDKeyGen}(\cdot),\mathsf{QCorrupt}(\cdot)}(\mathsf{mpk}, \mathsf{state})$.

Run Finalize on b'.

$\underline{\mathsf{QLeftRight}(i, x^0_i, x^1_i, \ell)}$: // $\boxed{\{G_{1,q,j}\}_{j=0,1}}$, $G_{1,q,2}$,$G_{1,q,3}$, $G_{1,q,4}$

For $b_j = 0$ when $j < q$, $j \leq q$, b else.

$\boxed{C_i := F(\mathsf{ek}_i, \ell_j) + x_i^{b_j}.}$

$C_i := F(\mathsf{ek}_i, \ell_j) + (x^0_{i,\ell_q} - x^b_{i,\ell_q})\mathsf{PRFO}(\ell) + x_i^{b_j}.$

Return $[c_i]$.

$\underline{\mathsf{QEncrypt}(i, x_i, \ell)}$: // $\{G_{1,q,j}\}_{j=0\cdots4}$

Return $\mathsf{QLeftRight}(i, x_i, x_i, \ell)$.

$\underline{\mathsf{PRFO}(\ell)}$: // $G_{1,q,2}$, $\{G_{1,q,j}\}_{j=3,4}$

Return $F(\mu, \ell)$, $\mathsf{RF}(\ell)$.

$\underline{\mathsf{QDKeyGen}(\vec{y})}$: // $\{G_{1,q,j}\}_{j=0\cdots4}$

Return $\langle \vec{s}, \vec{y} \rangle$.

$\underline{\mathsf{QCorrupt}(i)}$: // $\{G_{1,q,j}\}_{j=0\cdots4}$

Return s_i.

Figure 4.1: Summary of the hybrid-games proof system of the Theorem 4.1. On this description, PRFO plays the role of the challenger of the PRF security game, and appears only in three games.

Proof

Game G_0 : this is the selective $\texttt{wtr-IND}^*$ game, following the definition from Definition 3.2. Details are given in the Fig. 4.1.

Game $G_{1,q,0}$: this game is characterized by the encryption:

- for queries ℓ_j where $j < q$, encrypts x^0
- for queries ℓ_j where $j \geq q$, encrypts x^b

Game $G_{1,q,1}$: this game only changes the generation of msk: $\mathcal{C}$ chooses $\mathsf{msk} = \mu(\vec{x}^0_{\ell_q} - \vec{x}^1_{\ell_q}) + \vec{s}$ where $\mu \xleftarrow{\$} \mathcal{K}$ and $\vec{s} \xleftarrow{\$} \mathcal{K}^n$, and the ek_i used in the encryption are now $\mathsf{ek}_i : s_i + \mu(x^0_{i,\ell_q} - x^1_{i,\ell_q})$. Note that the decryption keys are $\mathsf{dk}_{\vec{y}} = \langle (\mathsf{ek}_i)_i, \vec{y} \rangle = \langle \vec{s}, \vec{y} \rangle$.

Game $G_{1,q,2}$: now $\mathcal{C}$ only chooses the $\vec{s}$ part of the key in the SetUp. To prepare the $\mathsf{QLeftRight}$ queries, he asks an oracle PRFO the values $F(\mu, \ell_j)$ for any query ℓ_j and returns:

- $C_i = F(s_i, \ell_j) + (x^0_{i,\ell_q} - x^1_{i,\ell_q})F(\mu, \ell_j) + x^0_{i,\ell_j}$ when $j < q$
- $C_i = F(s_i, \ell_j) + (x^0_{i,\ell_q} - x^1_{i,\ell_q})F(\mu, \ell_j) + x^b_{i,\ell_j}$ when $j \geq q$

Also, $\mathcal{C}$ can still answer other queries: for $\mathsf{QDKeyGen}$ he computes and sends $\langle \vec{s}, \vec{y} \rangle$, and for $\mathsf{QCorrupt}$ he answers s_i.

Game $G_{1,q,3}$: same as previously except that the oracle PRFO now answers $\mathsf{RF}(\ell)$ for any query ℓ, where $\mathsf{RF} : \mathcal{X} \longrightarrow \mathcal{Y}$ was randomly chosen at the initialization of PRFO.

Game $G_{1,q,4}$: this game switches the bit used in the encryption into 0 for the label ℓ_q.

Game $G_{1,q,5} = G_{1,q+1,0}$: in this game, the SetUp turns back to the original description of $G_{1,q,0}$ to start a new iteration, and we have $G_{1,q,5} = G_{1,q+1,0}$.

Game G_2 : same game as G_0 except that the bit used for encryption is 0.

The advantages for $\mathcal{A}$ are the same in $G_{1,q,0}, G_{1,q,1}$ and $G_{1,q,2}$ since the distributions of both ciphertexts and keys are not modified. The PRF makes it hard to distinguish between $G_{1,q,2}$ and $G_{1,q,3}$, $|\mathsf{Adv}_{\ell,2} - \mathsf{Adv}_{\ell,3}| = \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{KH\text{-}PRF}}(\mathcal{B})$, where $\mathcal{B}$ is an adversary playing against PRFO. The step between $G_{1,q,3}$ and $G_{1,q,4}$ is statistically secure, since PRFO has the same probability to use a function $\mathsf{RF}(\ell)$ or $\mathsf{RF}_q(\ell)$ where $\mathsf{RF}_q(\ell) = \mathsf{RF}(\ell) + \delta_{\ell_q}(\ell)$. In the last game $G_{1,q,5}$, the challenger comes back to the use of a PRF, repeating the PRF security game. Thus, the final advantage is $\mathsf{Adv}^{\texttt{IND}^*}_{\mathsf{MCFE}}(\mathcal{A}) \leq 2Q_\ell \times \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{KH\text{-}PRF}}$, where Q_ℓ is the number of labels.

Remark 1. Note that, because of the first finalize condition, the extra value $(x^0_{i,\ell_q} - x^1_{i,\ell_q})F(\mu, \ell_j)$ is null for corrupted users, so their ciphertexts are not modified in a valid simulation.

Remark 2. Also, the QEncrypt(i, x_i, ℓ) can here be answered with the value QLeftRight(i, x_i, x_i, ℓ). As explained in remark Section 3.1.2, the only difference between these two values is that QEncrypt allows the adversary to make incomplete ciphertext queries, not QLeftRight. In the security proof, the challenge ciphertexts output by QLeftRight are switched from an encryption of x_i^b to encryption of x_i^0, introducing a delta terms in the functional decryption keys. This delta term cancels out thanks to the conditions given in the Finalize procedure (for which we need complete ciphertexts). For QEncrypt queries, $x_i^b = x_i^0 = x_i^1$, which means the delta term is zero, and doesn't show up in any of the functional decryption key even for incomplete ciphertexts.

Limitations

Following the KH-PRF definition, for such a function F, we have that $\alpha F(k, \ell) = F(\alpha k, \ell)$ for any $\alpha \in \mathbb{Z}_p$. Thus, the discrete logarithm problem must hold in the image space, else F cannot be a PRF. Consequently, two problems appear here:

- if the decryption need to solve a discrete logarithm to recover the value $\langle \vec{x}, \vec{y} \rangle$, it limits the size of the outputs and decreases the efficiency of the scheme. However, we can avoid this by working on groups with subgroups that allow discrete logarithm computation.

- the group instantiation seems to be the only one for this precise construction. Such a PRF exists, since our scheme from [CDG+18a] proposes one: $F(k, \ell) = H(\ell)^k$, where $H : \{0,1\}^* \longrightarrow \mathbb{G}$ is an hash function onto the group $\mathbb{G}$. However, this PRF has the drawback to be secure in the random oracle model only.

Another assumption

There exists a family of "almost key-homomorphic" PRF which allows an operation with a bounded error. Since the method described here presents a bounded number of operations (n additions and scalar multiplications in the decryption), we can imagine to manage this noise. This PRF was first described in the paper [BLMR13], under an assumption derived from Learning With Errors, in the standard model. It was then re-developed in the paper [BP14] under the standard Learning With Errors problem, still in standard model. Finally, a recent work, [LT19], achieved a MCFE scheme from an "almost key-homomorphic" PRF family derived from these works. Their scheme is secure under the Learning With Errors assumption in the standard model. However, it loses in efficiency comparatively to the group instantiation, since this PRF uses techniques from fully homomorphic encryption.

Another interesting work from the same conference, [ABG19], builds (Decentralized) IP-MCFE from single-input IP-FE and a simple PRF in a black-box way. However, this construction mainly aims to be generic, and makes a trade-off on the size of the ciphertext, which is linear in the size of users.

4.2 IP-MCFE from groups

In this section we describe two MCFE schemes based on groups. While the first can now be seen as an instantiation of the generic construction, it was originally inspired by Abdalla *et al.* in [ABDP15]. The second improves the security from

selective to adaptive, achieving the wtr-IND* security using a similar technique as [ALS16].

Overview

This section describes two secret-key MCFE schemes initially built up from the public-key FE scheme introduced by Abdalla *et al.* [ABDP15] (itself a selectively-secure scheme). Mainly, we replace the global randomness with a hash function (modeled as a random oracle for the security analysis), in order to make the generation of the ciphertexts independent for each client. The comparison is illustrated in Figure Fig. 4.2. Note that for the final decryption to be possible, one needs the function evaluation α to be small enough, within this discrete logarithm setting. This is one limitation, which is still reasonable for real-world applications that use concrete numbers, that are not of cryptographic size.

From Section 4.2.1	MCFE	ABDP15 [ABDP15]
SetUp	Pick $(s_i)_{i\in[n]}$ at random	Pick $(s_i)_{i\in[n]}$ at random and set $V_i = [s_i]$
Encrypt	Each client i, on input (x_i, s_i, ℓ), return $[c_i] = [x_i] + s_i \cdot \mathcal{H}(\ell)$	On input $((x_i)_i, (v_i)_i)$, pick $r \xleftarrow{\$} \mathbb{Z}_p$, return $([c_0] = [r], ([c_i] = [x_i] + r \cdot V_i)_i)$
DKeyGen	On input $((y_i)_i, (s_i)_i)$, return $\mathsf{dk}_{\vec{y}} = \sum_i y_i s_i$	On input $((y_i)_i, (s_i)_i)$, return $\mathsf{dk}_{\vec{y}} = \sum_i y_i s_i$
Decrypt	Discrete logarithm on $[\alpha] = \sum_i y_i \cdot [c_i] - \mathsf{dk}_{\vec{y}} \cdot \mathcal{H}(\ell)$	Discrete logarithm on $[\alpha] = \sum_i y_i \cdot [c_i] - \mathsf{dk}_{\vec{y}} \cdot [c_0]$

Figure 4.2: Comparison of the IP-FE scheme from Abdalla *et al.* [ABDP15] and a similar MCFE obtained by introducing a hash function $\mathcal{H}$.

If we write $[c_0] = [r]$ in the single input case and $[c_0] = \mathcal{H}(\ell)$ in the multi-client case, we have $[c_i] = [x_i] + s_i \cdot [c_0]$ for $i \in [n]$ in both cases. In the public-key scheme from [ABDP15], s_i was private, and only $V_i = [s_i]$ was known to the encryptor. Since we are now dealing with private encryption, the encryptor can use s_i. Correctness then follows from

$$\begin{aligned}[\alpha] &= \sum_i y_i \cdot [c_i] - \mathsf{dk}_{\vec{y}} \cdot [c_0] = \sum_i y_i \cdot ([x_i] + s_i \cdot [c_0]) - \mathsf{dk}_{\vec{y}} \cdot [c_0] \\ &= \sum_i [x_i y_i] + (\sum_i s_i y_i) \cdot [c_0] - \mathsf{dk}_{\vec{y}} \cdot [c_0] = [\langle \vec{x}, \vec{y} \rangle]\end{aligned}$$

We now describe this MCFE scheme and prove it selectively secure under the DDH assumption.

4.2.1 Description of selective version

- SetUp(λ): takes as input the security parameter, and generates a group $\mathbb{G}$ of prime order $p \approx 2^\lambda$, $g \in \mathbb{G}$ a generator, and $\mathcal{H}$ a full-domain hash function onto $\mathbb{G}$. It also generates the encryption keys $s_i \xleftarrow{\$} \mathbb{Z}_p$, for $i = 1, \ldots, n$, and sets

$\vec{s} = (s_i)_i$. The public parameters mpk consist of $(\mathbb{G}, p, g, \mathcal{H})$, while the master secret key is $\mathsf{msk} = \vec{s}$ and the encryption keys are $\mathsf{ek}_i = s_i$ for $i = 1, \ldots, n$ (in addition to mpk, which is omitted).

- $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$: takes as input the value x_i to encrypt, under the key $\mathsf{ek}_i = s_i$ and the label ℓ. It computes $[u_\ell] := \mathcal{H}(\ell) \in \mathbb{G}$, and outputs the ciphertext $[c_i] = [u_\ell s_i + x_i] \in \mathbb{G}$.
- $\mathsf{DKeyGen}(\mathsf{msk}, \vec{y})$: takes as input $\mathsf{msk} = (s_i)_i$ and an inner product function defined by $\vec{y}$ as $f_{\vec{y}}(\vec{x}) = \langle \vec{x}, \vec{y} \rangle$, and outputs the functional decryption key $\mathsf{dk}_{\vec{y}} = (\vec{y}, \sum_i s_i y_i) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$.
- $\mathsf{FDecrypt}(\mathsf{dk}_{\vec{y}}, \ell, ([c_i])_{i \in [n]})$: takes as input a decryption key $\mathsf{dk}_{\vec{y}} = (\vec{y}, d)$, a label ℓ. It computes $[u_\ell] := \mathcal{H}(\ell)$, $[\alpha] = \sum_i y_i \cdot [c_i] - d \cdot [u_\ell]$, and eventually solves the discrete logarithm to extract and return α.
- $\mathsf{Decrypt}(\mathsf{ek}_i, \ell, ([c_j])_{j \in [n]})$: takes as input an encryption key $\mathsf{ek}_i = s_i$, a label ℓ. It computes $[u_\ell] := \mathcal{H}(\ell)$, $[\alpha] = [c_j] - \mathsf{ek}_i \cdot [u_\ell]$, and eventually solves the discrete logarithm to extract and return α.

Correctness. If the scalar dk in the functional decryption key $\mathsf{dk}_{\vec{y}} = (\vec{y}, dk)$ is indeed $dk = \langle \vec{s}, \vec{y} \rangle$, then:

$$\begin{aligned}[\alpha] &= \sum_i y_i \cdot [c_i] - d \cdot [u_\ell] = \sum_i y_i \cdot [u_\ell s_i + x_i] - [u_\ell] \cdot \sum_i s_i y_i \\ &= [u_\ell] \cdot \sum_i s_i y_i + [\sum_i x_i y_i] - [u_\ell] \cdot \sum_i s_i y_i = [\sum_i x_i y_i].\end{aligned}$$

4.2.2 Security analysis

Like Abdalla *et al.*'s original scheme [ABDP15], this protocol can only be proven secure in the weaker security model, `sel-wtr-IND`* in the MCFE context, where the adversary has to commit in advance to all of the pairs of messages for the left-or-right encryption oracle (QLeftRight-queries), and can only make one encryption per label. However, it can adaptively ask for functional decryption keys (QDKeyGen-queries) and encryption keys (QCorrupt-queries). Concretely, the challenger is provided (plaintext,label) pairs: $(x^b_{j,i}, \ell_j)_{b \in \{0,1\}, i \in [n], j \in [Q]}$, where Q is the number of query to $\mathsf{QLeftRight}(i, \cdot, \cdot, \ell_j)$, each one for a different label ℓ_j (note that in the security model, we assume each slots are queried the same number of time, on different labels). The challenge ciphertexts $C_{i,j} = \mathsf{Encrypt}(\mathsf{ek}_i, x^b_{j,i}, \ell_j)$, for the random bit b, are returned to the adversary. The adversary committing to challenge ciphertexts also limits its ability to corrupt users during the game: it must corrupt clients for which it didn't ask a ciphertext and cannot corrupt any client from which it asked a ciphertext for $x^0_{j,i} \neq x^1_{j,i}$.

About the security in [CDG+18a]

Note that the security notion proven in [CDG+18a] slightly differs from the `sel-wtr-IND`* security notion we use here, in that it does not give the adversary access to a QEncrypt oracle, but only QLeftRight (see Section 3.1.2 on the role of the oracle QEncrypt, and why it generally cannot be simulated by QLeftRight in the `IND`* security game). For the same reason as in 4.1, it follows from inspection of the

security proof of [CDG+18a] that it can actually achieve the $\texttt{IND}^*$ security notion defined here.

Theorem 4.2: $\texttt{sel-wtr-IND}^*$ Security

The MCFE protocol described above (see Section 4.2.1) is $\texttt{sel-wtr-IND}^*$ secure under the DDH assumption, in the random oracle model. More precisely, we have

$$\mathsf{Adv}^{\texttt{IND}}(\mathcal{A}) \leq 2Q \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t),$$

for any adversary $\mathcal{A}$, running within time t, where Q is the number of encryption queries per slot.

Games G_0, G_1, $(G_{2.q})_{q\in[Q+1]}$

$\left(\mathsf{state}, (\ell_j, z_{j,i})_{i\in[n],j\in[Q]}\right) \leftarrow \mathcal{A}(1^\lambda, 1^n)$
where each $z_{j,i} = (x^0_{j,i}, x^1_{j,i}) \in \mathbb{Z}_p^2$, or $z_{j,i} = \perp$, which stands for no query.
$\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, for all $i \in [n]$, $s_i \xleftarrow{\$} \mathbb{Z}_p$,$\mathsf{ek}_i := s_i$, $\mathsf{msk} := (s_i)_i$, $\mathsf{mpk} := (\mathbb{G}, p, g)$.
$C_{j,i} = \mathsf{QLeftRight}(i, x^0_{j,i}, x^1_{j,i}, \ell_j)$ for $i \in [n], j \in [Q]$ such that $z_{j,i} = (x^0_{j,i}, x^1_{j,i})$.
$b' \leftarrow \mathcal{A}^{\mathsf{QDKeyGen}(\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{RO}(\cdot)}(\mathsf{mpk}, \mathsf{state})$.
Run Finalize on b'.

$\mathsf{RO}(\ell)$: // G_0, $\boxed{G_1,\ G_{2.q}}$
$[u_\ell] := \mathcal{H}(\ell)\boxed{,\ [u_\ell] := \mathsf{RF}(\ell)}$.
Return $[u_\ell]$.

$\mathsf{QLeftRight}(i, x^0_i, x^1_i, \ell)$: // G_0, G_1, $\boxed{G_{2.q}}$
$[u_\ell] := \mathsf{RO}(\ell)$,
$[c_i] := [u_\ell] \cdot s_i + [x^b_i]$
$\boxed{\text{If } \ell = \ell_j \text{ with } j < q\text{: } [c_i] := [u_\ell s_i + x^0_i]}$
Return $[c_i]$.

$\mathsf{QEncrypt}(i, x_i, \ell)$: // G_0, $\{G_{1,q,j}\}_{j=0\cdots4}$
Return $\mathsf{QLeftRight}(i, x_i, x_i, \ell)$.

$\mathsf{QDKeyGen}(\vec{y})$: // G_0, G_1, $G_{2.q}$
Return $\sum_i y_i s_i$.

$\mathsf{QCorrupt}(i)$: // G_0, G_1, $G_{2.q}$
Return s_i.

Figure 4.3: Games G_0, G_1, $(G_{2.})_{q\in[Q+1]}$, for the proof of Theorem 4.2. Here, RF is a random function onto $\mathbb{G}$, that is computed on the fly. Note that QLeftRight is only used as a subroutine of the initialization of the game and is not accessible to the adversary. In each procedure, the components inside a solid frame are only present in the games marked by a solid frame.

We proceed using hybrid games, summarized in Fig. 4.3, in a similar method as the KH-PRF-based construction: we play on the space defined by the vectors $\vec{x}^0 - \vec{x}^b$ to inject randomness and switch the bit.

Proof

Game G_0: this is the $\texttt{sel-wtr-IND}^*$ security game as given in Definition 3.2 (see the paragraph about weaker models), with all the encryption queries being sent first: they are stored in $z_{j,i} = (x^0_{j,i}, x^1_{j,i})$, for $j \in [Q]$ and $i \in [n]$, where j is for the j-th $\mathcal{H}$-query that specifies the label ℓ_j and i is for the index of the sender. If the query is not asked, we have $z_{j,i} = \perp$. Note that the hash function $\mathcal{H}$ is modeled as a random oracle RO onto $\mathbb{G}$. This is used to generate $[u_\ell] = \mathcal{H}(\ell)$.

Game G_1: we simulate the answers to any new RO query by computing a truly random element of $\mathbb{G}$, on the fly. The simulation remains perfect, so $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2: we simulate every encryption as the encryption of x^0_i instead of x^b_i.

While it is clear that in this last game the advantage of any adversary is exactly 0 since b does not appear anywhere, the gap between G_1 and G_2 will be proven using an hybrid argument on the RO-queries. We thus index the following games by q, where $q = 1, \ldots, Q$. Note that only distinct RO-queries are counted, since a second similar query is answered as the first one.

$G_{2.1}$: this is exactly game G_1. Thus, $\mathsf{Adv}_1 = \mathsf{Adv}_{2.1}$.

$G_{2.q} \rightsquigarrow G_{2.q+1}$: we change the generation of the ciphertexts from $[c_{q,i}] := [u_{\ell_q} s_i + x^b_{q,i}]$ to $[c_{q,i}] := [u_{\ell_q} s_i + x^0_{q,i}]$. We proceed in three steps:

Step 1. We use the fact that the two following distributions are identical, for any choice of γ:

$$(s_i)_{i\in[n], z_{q,i}=(x^0_{q,i}, x^b_{q,i})} \text{ and } \left(s_i + \gamma(x^0_{q,i} - x^b_{q,i})\right)_{i\in[n], z_{q,i}=(x^0_{q,i}, x^1_{q,i})},$$

where $s_i \xleftarrow{\$} \mathbb{Z}_p$, for all $i \in [n]$. This is true since the s_i are independent of the $z_{q,i}$ (we are in a selective setting, so the s_i's are generated after the $z_{q,i}$'s have been chosen). Thus, we can re-write s_i into $s_i + \gamma(x^0_{q,i} - x^b_{q,i})$ without changing the distribution of the game.

Note that when Finalize does not output a random bit $\beta \xleftarrow{\$} \{0,1\}$ independent of the guess b', γ does not appear in the outputs of $\mathsf{QCorrupt}(i)$, since it must be that $x^0_i = x^1_i$ or $z_{q,i} = \perp$, and it does not appear in the output of $\mathsf{QDKeyGen}(\vec{y})$ either, since $\sum_i s_i \cdot y_i + \sum_i \gamma(x^0_{q,i} - x^b_{q,i}) y_i$, where the gray term equals zero by Definition 3.1. The fact that γ does not appear in the outputs of these oracles will be crucial for step 2, which applies DDH on $[\gamma]$.

Step 2. We use the DDH assumption to replace the $[u_{\ell_q}\gamma]$ that appear in the output of the q-th query to $\mathsf{QLeftRight}$ queries with $[r_{\ell_q} + 1]$ with $r_{\ell_q} \xleftarrow{\$} \mathbb{Z}_p$. This is possible since the rest of the adversary view can be

generated only from $[\gamma]$ and $[r_{\ell_q}+1]$. This increases the adversary's advantage by no more than $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$. We now have:

$$\begin{aligned}[c_{q,i}] :=& [u_{\ell_q} s_i + (x^0_{q,i} - x^b_{q,i})(r_{\ell_q}+1) + x^b_{q,i}] \\ =& [u_{\ell_q} s_i + r_{\ell_q}(x^0_{q,i} - x^b_{q,i}) + x^0_{q,i} - x^b_{q,i} + x^b_{q,i}] \\ =& [u_{\ell_q} s_i + r_{\ell_q}(x^0_{q,i} - x^b_{q,i}) + x^0_{q,i}].\end{aligned}$$

Step 3. We switch $[r_{\ell_q}]$ in the output of the q-query to $\mathsf{QLeftRight}$ back to $[u_{\ell_q}\gamma]$, using the DDH assumption again. This is possible since the adversary's view is simulatable solely from $[\gamma]$, $[u_{\ell_q}]$, and $[r_{\ell_q}]$. We finally undo the distribution change on the $\vec{s}_i$, which brings us to $G_{2.q+1}$.

As a conclusion, since $G_{2.Q+1} = G_2$, we have $\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq 2Q \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$. In addition, $\mathsf{Adv}_2 = 0$, which concludes the proof.

4.2.3 Description of adaptative version

After the selective construction from [CDG+18a], we propose another construction of MCFE for inner product from the same article, adapted from the Agrawal *et al.* [ALS16] scheme in the same manner. The main contribution of this construction is re-uses a technique from this article to achieve the adaptive $\texttt{wtr-IND}^*$-security. Roughly speaking, it consists to double the size of the keys.

- $\mathsf{SetUp}(\lambda)$: takes as input the security parameter, and generates prime-order group $\mathcal{G} := (\mathbb{G}, p, P) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, and $\mathcal{H}$ a full-domain hash function onto $\mathbb{G}^2$. It also generates the encryption keys $\vec{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, for $i = 1, \ldots, n$. The public parameters mpk consist of $(\mathbb{G}, p, g, \mathcal{H})$, while the encryption keys are $\mathsf{ek}_i = \vec{s}_i$ for $i = 1, \ldots, n$, and the master secret key is $\mathsf{msk} = ((\mathsf{ek}_i)_i)$, (in addition to mpk, which is omitted).

- $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$: takes as input the value x_i to encrypt, under the key $\mathsf{ek}_i = \vec{s}_i$ and the label ℓ. It computes $[\vec{u}_\ell] := \mathcal{H}(\ell) \in \mathbb{G}^2$, and outputs the ciphertext $[c_i] = [\vec{u}_\ell^\top \vec{s}_i + x_i] \in \mathbb{G}$.

- $\mathsf{DKeyGen}(\mathsf{msk}, \vec{y})$: takes as input $\mathsf{msk} = (\vec{s}_i)_i$ and an inner product function defined by $\vec{y}$ as $f_{\vec{y}}(\vec{x}) = \langle \vec{x}, \vec{y} \rangle$, and outputs the functional decryption key $\mathsf{dk}_{\vec{y}} = (\vec{y}, \sum_i \vec{s}_i \cdot y_i) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^2$.

- $\mathsf{FDecrypt}(\mathsf{dk}_{\vec{y}}, \ell, ([c_i])_{i\in[n]})$: takes as input a functional decryption key $\mathsf{dk}_{\vec{y}} = (\vec{y}, \vec{d})$, a label ℓ, and ciphertexts. It computes $[\vec{u}_\ell] := \mathcal{H}(\ell)$, $[\alpha] = \sum_i [c_i] \cdot y_i - [\vec{u}_\ell^\top] \cdot \vec{d}$, and eventually solves the discrete logarithm to extract and return α.

- $\mathsf{Decrypt}(\mathsf{ek}_i, \ell, ([c_j])_{j\in[n]})$: takes as input an encryption key $\mathsf{ek}_i = (s_1, s_2)$, a label ℓ. It computes $[\vec{u}_\ell] := \mathcal{H}(\ell)$, $[\alpha] = [c_j] - \mathsf{ek}_i \cdot [\vec{u}_\ell]$, and eventually solves the discrete logarithm to extract and return α.

Correctness. If the scalar $\vec{d}$ in the functional decryption key $\mathsf{dk}_{\vec{y}} = (\vec{y}, \vec{d})$ is indeed $\vec{d} = (\langle \vec{s}_1, \vec{y} \rangle, \langle \vec{s}_2, \vec{y} \rangle)$, then:

$$\begin{aligned}[\alpha] &= \sum_i [c_i] \cdot y_i - [\vec{u}_\ell^\top] \cdot \vec{d} = \sum_i [\vec{u}_\ell^\top \vec{s}_i + x_i] \cdot y_i - [\vec{u}_\ell^\top] \cdot \sum_i y_i \vec{s}_i \\ &= \sum_i [\vec{u}_\ell^\top] \cdot \vec{s}_i y_i + \sum_i [x_i] \cdot y_i - [\vec{u}_\ell^\top] \cdot \sum_i y_i \vec{s}_i = [\sum_i x_i y_i].\end{aligned}$$

4.2.4 Security Analysis

We stress that the following theorem supports both adaptive encryption queries and adaptive corruptions.

Theorem 4.3: `wtr-IND*`-Security

The above MCFE protocol (see Section 4.2.3) is `wtr-IND*`-secure under the DDH assumption, in the random oracle model. More precisely, we have

$$\mathsf{Adv}^{\mathtt{IND}}(\mathcal{A}) \leq 2Q \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4Q \times t_{\mathbb{G}}) + \frac{2Q}{p},$$

for any adversary $\mathcal{A}$, running within time t, where Q is the number of (direct and indirect —asked by QEncrypt-queries—) queries to $\mathcal{H}$ (modeled as a random oracle), and $t_{\mathbb{G}}$ is the time for an exponentiation in $\mathbb{G}$.

To obtain adaptive security, we use a technique that consists of first proving perfect security in the selective variant of the involved games, then, using a guessing (a.k.a. complexity leveraging) argument, which incurs an exponential security loss, we obtain the same security guarantees in the adaptive games. Since the security in the selective game is perfect (the advantage of any adversary is exactly zero), the exponential security loss is multiplied by a zero term, and the overall adaptive security is preserved. This technique has been used before in [Wee14] in the context of attribute-based encryption, or more recently, in [AGRW17, ACF+18] in the context of multi-input IP-FE. We defer to [Wee14, Remark 1] and [AGRW17, Remark 5] for more details on this proof technique. We proceed using hybrid games, described in Fig. 4.4.

Proof

Let $\mathcal{A}$ be a PPT adversary. For any game G_{index}, we denote by $\mathsf{Adv}_{\mathsf{index}} := |\Pr[G_{\mathsf{index}}(\mathcal{A})|b = 1] - \Pr[G_{\mathsf{index}}(\mathcal{A})|b = 0]|$, where the probability is taken over the random coins of G_{index} and $\mathcal{A}$. Also, by event $G_{\mathsf{index}}(\mathcal{A})$, or just G_{index} when there is no ambiguity, we mean that the Finalize procedure in game G_{index} (defined as in Definition 3.2) returns $\beta = 1$ from the adversary's answer b' when interacting with $\mathcal{A}$.

Game G_0: this is the `IND`-security game as given in Definition 3.2. Note that the hash function $\mathcal{H}$ is modeled as a random oracle RO onto $\mathbb{G}^2$. This is essentially used to generate $[\vec{u}_\ell] = \mathcal{H}(\ell)$.

Game G_1: we simulate the answers to any new RO-query by a truly random

pair in $\mathbb{G}^2$, on the fly. The simulation remains perfect, and so $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2: we simulate the answers to any new RO-query by a truly random pair in the span of $[\vec{a}]$ for $\vec{a} := \binom{1}{a}$, with $a \xleftarrow{\$} \mathbb{Z}_p$. This uses the Multi-DDH assumption, which tightly reduces to the DDH assumption using the random-self reducibility (see Lemma Proposition 2.1): $\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t + 4Q \times t_{\mathbb{G}})$, where Q is the number of RO-queries and $t_{\mathbb{G}}$ the time for an exponentiation.

Game G_3: we simulate any QEncrypt query as the encryption of x_i^0 instead of x_i^b and go back for the answers to any new RO query by a truly random pair in $\mathbb{G}^2$.

While it is clear that in this last game the advantage of any adversary is exactly 0 since b does not appear anywhere, the gap between G_2 and G_3 will be proven using a hybrid technique on the RO-queries. We thus index the following games by q, where $q = 1, \ldots, Q$. Note that only distinct RO-queries are counted, since a second similar query is answered as the first one. We detail this proof because the technique is important (Fig. 4.5).

$G_{3.1.1}$**:** this is exactly game G_2. Thus, $\mathsf{Adv}_2 = \mathsf{Adv}_{3.1.1}$.

$G_{3.q.1} \rightsquigarrow G_{3.q.2}$**:** we first change the distribution of the output of the q-th RO-query, from uniformly random in the span of $[\vec{a}]$ to uniformly random over $\mathbb{G}^2$, using the DDH assumption. Then, we use the basis $(\binom{1}{a}, \binom{-a}{1})$ of $\mathbb{Z}_p^2$, to write a uniformly random vector over $\mathbb{Z}_p^2$ as $u_1 \cdot \vec{a} + u_2 \cdot \vec{a}^{\perp}$, where $u_1, u_2 \xleftarrow{\$} \mathbb{Z}_p$. Finally, we switch to $u_1 \cdot \vec{a} + u_2 \cdot \vec{a}^{\perp}$ where $u_1 \xleftarrow{\$} \mathbb{Z}_p$, and $u_2 \xleftarrow{\$} \mathbb{Z}_p^*$, which only changes the adversary view by a statistical distance of $1/p$: $\mathsf{Adv}_{3.q.1} - \mathsf{Adv}_{3.q.2} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t) + 1/p$. The last step with $u_2 \in \mathbb{Z}_p^*$ will be important to guarantee that $\vec{u}_\ell^{\top} \vec{a}^{\perp} \neq 0$.

$G_{3.q.2} \rightsquigarrow G_{3.q.3}$**:** we now change the generation of the ciphertext $[c_i] := [\vec{u}_\ell^{\top}] \cdot \vec{s}_i + [x_i^b]$ by $[c_i] := [\vec{u}_\ell^{\top}] \cdot \vec{s}_i + [x_i^0]$, where $[\vec{u}_\ell]$ corresponds to the q-th RO-query. We then prove this does not change the adversary's view.

Note that if the output of the q-th RO-query is not used by QEncrypt-queries, then the games $G_{3.q.2}$ and $G_{3.q.3}$ are identical. But we can show this is true too when there are RO-queries that are really involved in QEncrypt-queries, and show that $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$ in that case too, in two steps. In Step 1, we show that there exists a PPT adversary $\mathcal{B}^\star$ such that $\mathsf{Adv}_{3.q.t} = (p^2+1)^n \cdot \mathsf{Adv}^\star_{3.q.t}(\mathcal{B}^\star)$, for $t = 2, 3$, where the games $G^\star_{3.q.2}$ and $G^\star_{3.q.3}$ are selective variants of games $G_{3.q.2}$ and $G_{3.q.3}$ respectively (see Fig. 4.5), where QCorrupt queries are asked before the initialization phase. In Step 2, we show that for all PPT adversaries $\mathcal{B}^\star$, we have $\mathsf{Adv}^\star_{3.q.2}(\mathcal{B}^\star) = \mathsf{Adv}^\star_{3.q.3}(\mathcal{B}^\star)$. This will conclude the two steps.

Step 1. We build a PPT adversary $\mathcal{B}^\star$ playing against $G^\star_{3.q.t}$ for $t = 2, 3$, such that $\mathsf{Adv}_{3.q.t} = (p^2+1)^n \cdot \mathsf{Adv}^\star_{3.q.t}(\mathcal{B}^\star)$.

Adversary $\mathcal{B}^\star$ first guesses for all $i \in [n]$, $z_i \xleftarrow{\$} \mathbb{Z}_p^2 \cup \{\perp\}$, which it sends to its selective game $G^\star_{3.q.t}$. That is, each guess z_i is either a pair of values (x_i^0, x_i^1) queried to $\mathsf{QEncrypt}$, or $\perp$, which means no query to $\mathsf{QEncrypt}$. Then, it simulates $\mathcal{A}$'s view using its own oracles. When $\mathcal{B}^\star$ guesses successfully (call E that event), it simulates $\mathcal{A}$'s view exactly as in $G_{3.q.t}$. If the guess was not successful, then $\mathcal{B}^\star$ stops the simulation and outputs a random bit β. Since event E happens with probability $(p^2+1)^{-n}$ and is independent of the view of adversary $\mathcal{A}$: $\mathsf{Adv}^\star_{3.q.t}(\mathcal{B}^\star)$ is equal to

$$\left|\Pr[G^\star_{3.q.t}|b=0,E]\cdot\Pr[E]+\frac{\Pr[\neg E]}{2}-\Pr[G^\star_{3.q.t}|b=1,E]\cdot\Pr[E]-\frac{\Pr[\neg E]}{2}\right|$$
$$=\Pr[E]\cdot|\Pr[G^\star_{3.q.t}|b=0,E]-\Pr[G^\star_{3.q.t}|b=1,E]|=(p^2+1)^{-n}\cdot\mathsf{Adv}_{3.q.t}.$$

Step 2. We assume the values $(z_i)_{i\in[n]}$ sent by $\mathcal{B}^\star$ are consistent, that is, they don't make the game end and return a random bit, and Finalize on b' does not return a random bit independent of b' (call E' this event).

We show that games $G^\star_{3.q.2}$ and $G^\star_{3.q.3}$ are identically distributed, conditioned on E'. To prove it, we use the fact that the two following distributions are identical, for any choice of γ:

$$(\vec{s}_i)_{i\in[n],z_i=(x_i^0,x_i^1)} \text{ and } \left(\vec{s}_i+\vec{a}^\perp\cdot\gamma(x_i^b-x_i^0)\right)_{i\in[n],z_i=(x_i^0,x_i^1)},$$

where $\vec{a}^\perp := \binom{-a}{1} \in \mathbb{Z}_p^2$ and $\vec{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, for all $i=1,\ldots,n$. This is true since the $\vec{s}_i$ are independent of the z_i (note that this is true because we are in a selective setting, while this would not necessarily be true with adaptive $\mathsf{QEncrypt}$-queries). Thus, we can re-write $\vec{s}_i$ into $\vec{s}_i+\vec{a}^\perp\cdot\gamma(x_i^b-x_i^0)$ without changing the distribution of the game.

We now take a look at where the extra terms $\vec{a}^\perp\cdot\gamma(x_i^b-x_i^0)$ actually appear in the adversary's view:

- they do not appear in the output of $\mathsf{QCorrupt}$, because we assume event E' holds, which implies that if $z_i \neq \perp$, then i is not queried to $\mathsf{QCorrupt}$ or $x_i^1 = x_i^0$.
- they might appear in $\mathsf{QDKeyGen}(\vec{y})$ as

$$\mathsf{dk}_{\vec{y}} = \sum_{i\in[n]} \vec{s}_i\cdot y_i + \vec{a}^\perp\cdot\gamma\textstyle\sum_{i:z_i=(x_i^0,x_i^1)} y_i(x_i^b-x_i^0).$$

  But the gray term equals $\mathbf{0}$ by the constraints for E' in Definition 3.2: for all $i\in\mathcal{HS}$, $z_i\neq\perp$; if $i\in\mathcal{CS}$ and $z_i\neq\perp$, $x_i^1=x_i^0$; and $f(\vec{x}^0)=f(\vec{x}^1)$, hence $\sum_{i:z_i=(x_i^0,x_i^1)} y_i(x_i^b-x_i^0)=0$.
- eventually, they appear in the output of the $\mathsf{QEncrypt}$-queries which use $[\vec{u}_\ell]$ computed on the q-th RO-query, since for all others, the vector $[\vec{u}_\ell]$ lies in the span of $[\vec{a}]$, and $\vec{a}^\top\vec{a}^\perp=0$. We thus have $[c_i] := [\vec{u}_\ell^\top]\cdot\vec{s}_i + (x_i^b-x_i^0)\gamma[\vec{u}_\ell^\top]\vec{a}^\perp + [x_i^b]$. Since $\vec{u}_\ell^\top\vec{a}^\perp\neq 0$, we can choose $\gamma = -1/\vec{u}_\ell^\top\vec{a}^\perp \bmod p$, and then $[c_i] = [\vec{u}_\ell^\top]\cdot\vec{s}_i + [x_i^0]$, which is the encryption of x_i^0. We stress that γ is independent of the

index i, and so this simultaneously converts all the encryptions of x_i^b into encryptions of x_i^0. Finally, reverting these statistically perfect changes, we obtain that $[c_i]$ is identically distributed to $[\vec{u}_\ell^\top] \cdot \vec{s}_i + [x_i^0]$, as in game $G^\star_{3.q.3}$.

Thus, when event E' happens, the games are identically distributed. When $\neg E$ happens, the games both return $\beta \xleftarrow{\$} \{0,1\}$: $\mathsf{Adv}^\star_{3.q.2}(\mathcal{B}^\star) = \mathsf{Adv}^\star_{3.q.3}(\mathcal{B}^\star)$. As a conclusion, we get $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$.

$G_{3.q.3} \rightsquigarrow G_{3.q+1.1}$**:** this transition is the reverse of $G_{3.q.1} \rightsquigarrow G_{3.q.2}$, namely, we use the DDH assumption to switch back the distribution of $[\vec{u}_\ell]$ computed on the q-th RO-query from uniformly random over $\mathbb{G}^2$ (conditioned on the fact that $\vec{u}_\ell^\top \vec{a}^\perp \neq 0$) to uniformly random in the span of $[\vec{a}]$: $\mathsf{Adv}_{3.q.3} - \mathsf{Adv}_{3.q+1.1} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t) + 1/p$.

As a conclusion, since $G_{3.Q+1.1} = G_3$, we have $\mathsf{Adv}_2 - \mathsf{Adv}_3 \leq 2Q(\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t) + 1/p)$. In addition, $\mathsf{Adv}_3 = 0$, which concludes the proof.

Games G_0, G_1, $\boxed{G_2,\ (G_{3.q.1})_{q\in[Q+1]},\ (G_{3.q.2}, G_{3.q.3})_{q\in[Q]}}$

$\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, for all $i \in [n]$, $\vec{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, $\mathsf{ek}_i := \vec{s}_i$, $\mathsf{msk} := (\vec{s}_i)_i$, $\mathsf{mpk} := (\mathbb{G}, p, g)$.

$\boxed{a \xleftarrow{\$} \mathbb{Z}_p,\ \vec{a} := \binom{1}{a},\ \vec{a}^\perp := \binom{-a}{1}}$

Sample a full-domain hash function $\mathcal{H}$ onto $\mathbb{G}^2$, and a bit $b \xleftarrow{\$} \{0,1\}$.

$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot),\mathsf{QDKeyGen}(\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{RO}(\cdot)}(\mathsf{mpk})$.

Run Finalize on b'.

$\underline{\mathsf{RO}(\ell)}$: // G_0, $\boxed{G_1}$, $\boxed{G_2,\ G_{3.q.1},\ G_{3.q.2},\ G_{3.q.3}}$

$[\vec{u}_\ell] := \mathcal{H}(\ell)$, $\boxed{[\vec{u}_\ell] := \mathsf{RF}(\ell)}$, $\boxed{[\vec{u}_\ell] := [\vec{a} \cdot r_\ell]}$, with $r_\ell := \mathsf{RF}'(\ell)$

On the q'th (fresh) query: $[\vec{u}_\ell] := \mathsf{RF}'(\ell) \cdot \vec{a} + \mathsf{RF}''(\ell) \cdot \vec{a}^\perp$

Return $[\vec{u}_\ell]$.

$\underline{\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)}$: // G_0, G_1, G_2, $\boxed{G_{3.q.1},\ G_{3.q.2},\ G_{3.q.3}}$

$[\vec{u}_\ell] := \mathsf{RO}(\ell)$,

$[c_i] := [\vec{u}_\ell^\top] \cdot \vec{s}_i + [x_i^b]$

$\boxed{\text{If } [\vec{u}_\ell] \text{ is computed on the } j \text{ RO-query, for } j < q\text{: } [c_i] := [\vec{u}_\ell^\top] \cdot \vec{s}_i + [x_i^0]}$

If $[\vec{u}_\ell]$ is computed on the q-th RO-query: $[c_i] := [\vec{u}_\ell^\top] \cdot \vec{s}_i + [x_i^0]$

Return $[c_i]$

$\underline{\mathsf{QEncrypt}(i, x_i, \ell)}$: // G_0, $\{G_{1,q,j}\}_{j=0\cdots4}$

Return $\mathsf{QLeftRight}(i, x_i, x_i, \ell)$.

$\underline{\mathsf{QDKeyGen}(\vec{y})}$: Return $\sum_i y_i \vec{s}_i$. //G_0, G_1, G_2, $G_{3.q.1}, G_{3.q.2}, G_{3.q.3}$

$\underline{\mathsf{QCorrupt}(i)}$: Return $\vec{s}_i$. // G_0, G_1, G_2, $G_{3.q.1}, G_{3.q.2}, G_{3.q.3}$

Figure 4.4: Games for the proof of Theorem 4.3. Here, RF, RF', RF'' are random functions onto $\mathbb{G}^2$, $\mathbb{Z}_p$, and $\mathbb{Z}_p^*$, respectively, that are computed on the fly. In each procedure, the components inside a solid (dotted, gray) frame are only present in the games marked by a solid (dotted, gray) frame. The Finalize procedure is defined as in Definition 3.2.

Games $(G^{\star}_{3.q.2}, G^{\star}_{3.q.3})_{q\in[Q]}$:

$\left(\mathsf{state}, (z_i \in \mathbb{Z}_p^2 \cup \{\bot\})_{i\in[n]}\right) \leftarrow \mathcal{A}(1^\lambda, 1^n)$

$\mathcal{G} \leftarrow \mathsf{GGen}(1^\lambda)$, for all $i \in [n]$, $\vec{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, $\mathsf{ek}_i := \vec{s}_i$, $\mathsf{msk} := (\vec{s}_i)_i$, $\mathsf{mpk} := (\mathbb{G}, p, g)$.

$a \xleftarrow{\$} \mathbb{Z}_p$, $\vec{a} := \binom{1}{a}$, $\vec{a}^\bot := \binom{-a}{1}$, $b \xleftarrow{\$} \{0,1\}$.

$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot),\mathsf{QDKeyGen}(\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{RO}(\cdot)}(\mathsf{mpk}, \mathsf{state})$.

Run Finalize on b'.

$\mathsf{RO}(\ell)$: // $G^{\star}_{3.q.2}$, $G^{\star}_{3.q.3}$

$[\vec{u}_\ell] := [\vec{a} \cdot r_\ell]$, with $r_\ell := \mathsf{RF}'(\ell)$

On the q'th (fresh) query: $[\vec{u}_\ell] := [\mathsf{RF}'(\ell) \cdot \vec{a} + \mathsf{RF}''(\ell) \cdot \vec{a}^\bot]$

Return $[\vec{u}_\ell]$.

$\mathsf{QEncrypt}(i, x_i^0, x_i^1, \ell)$: // $\boxed{G^{\star}_{3.q.2}}$, $G^{\star}_{3.q.3}$

$[\vec{u}_\ell] := \mathsf{RO}(\ell)$,

$[c_i] := [\vec{u}_\ell^\top] \cdot \vec{s}_i + [x_i^b]$

If $[\vec{u}_\ell]$ is computed on the j-th RO-query with $j < q$: $[c_i] := [\vec{u}_\ell^\top] \cdot \vec{s}_i + [x_i^0]$.

If $[\vec{u}_\ell]$ is computed on the q-th RO-query, then:

- if $(x_i^0, x_i^1) \neq z_i$, the game ends and returns $\beta \xleftarrow{\$} \{0,1\}$.
- otherwise, $[c_i] := [\vec{u}_\ell^\top] \cdot \vec{s}_i \boxed{+[x_i^b]} \, +[x_i^0]$, $\mathcal{S} := \mathcal{S} \cup \{i\}$.

Return $[c_i]$.

$\mathsf{QEncrypt}(i, x_i, \ell)$: // G_0, $\{G_{1,q,j}\}_{j=0\cdots4}$

Return $\mathsf{QLeftRight}(i, x_i, x_i, \ell)$.

$\mathsf{QDKeyGen}(\vec{y})$: Return $\sum_i y_i \vec{s}_i$. //$G^{\star}_{3.q.2}, G^{\star}_{3.q.3}$

$\mathsf{QCorrupt}(i)$: // $G^{\star}_{3.q.2}, G^{\star}_{3.q.3}$

If $z_i = (x_i^0, x_i^1)$ with $x_i^0 \neq x_i^1$, the game ends, and returns $\beta \xleftarrow{\$} \{0,1\}$.

Return $\vec{s}_i$.

Figure 4.5: Games $G^{\star}_{3.q.2}$ and $G^{\star}_{3.q.3}$, with $q \in [Q]$, for the proof of Theorem 4.3. Here, RF, RF' are random functions onto $\mathbb{G}^2$, and $\mathbb{Z}_p$, respectively, that are computed on the fly. In each procedure, the components inside a solid (gray) frame are only present in the games marked by a solid (gray frame.

5 Construction of DMCFE

This chapter present two ways to build a DMCFE scheme. The first one is a black-box construction and was described in [CDG+18b]. The second (while chronologically the first) is a scheme from [CDG+18a], and relies on pairings. Both are based on the idea of an efficient secure multi-party computation applied to the construction of the decryption key.

Contents

5.1 IP-DMCFE from multi-party computation

In this section, we convert an IP-MCFE into IP-DMCFE using a layer of multi-party computation (MPC). We first describe it and its characteristics then we provide a construction from [KDK11] and a derivative. Finally, we explain how to apply this layer of a MCFE.

5.1.1 Distributed sum

In order to convert an MCFE scheme into a DMCFE, one needs to allow efficient distributed computation of the functional decryption key. In the case of our IP-MCFE, this can be seen as a particular MCFE for the unique sum function on the contributions of all the clients, since the key is $\mathsf{dk}_{\vec{y}} = (\vec{y}, \sum_i y_i \cdot \vec{s}_i)$, and namely one has to compute $\sum_i x_i = \sum_i y_i \cdot \vec{s}_i$, where the x_i's can be computed by each client. This primitive can also be considered as a two rounds secure MPC for the sum, called Private Stream Aggregation (PSA). This notion is an old primitive introduced by Shi *et al.* [SCR+11]. It is quite similar to our target DMCFE scheme, however PSA does not consider the possibility of adaptively generating different keys for different inner product evaluations, but only enables the aggregator to compute the sum of the client's data for each time period. PSA also typically involves a Differential Privacy component, which has yet to be studied in the larger setting of DMCFE. Further research on PSA has focused on achieving new properties or better efficiency [CSS12, Emu17, JL13, LC13, LC12, BJL16] but not on enabling new functionalities.

Definition 5.1: Distributed Sum

A Distributed Sum (DSum) on a group $\mathbf{G}$ among n senders is defined by three algorithms:

- $\mathsf{DSum.SetUp}(\lambda)$: Takes as input the security parameter λ. Generates the public parameters pp and the personal secret keys sk_i for $i = 1 \cdots n$.
- $\mathsf{DSum.Encode}(x_i, \ell, \mathsf{sk}_i)$: Takes the x_i value to encode, a label ℓ and the personal secret key sk_i of the user i. Returns the share $M_{\ell,i}$.
- $\mathsf{DSum.Combine}(\vec{M})$: Takes as input a vector $\vec{M} = (M_{\ell,i})_i$ of shares. Returns the value $\sum_i M_{\ell,i}$.

Correctness. For any label ℓ, we want $\Pr[\mathsf{DSum.Combine}(\vec{M}_\ell) = \sum_i x_i] = 1$, where the probability is taken over $M_{\ell,i} \leftarrow \mathsf{DSum.Encode}(x_i, \ell, \mathsf{sk}_i)$ for all $i \in [n]$, and $(\mathsf{pp}, (\mathsf{sk}_i)_i) \leftarrow \mathsf{DSum.SetUp}(\lambda)$.

Security notion. This protocol must guarantee the privacy of the x_i's, their sum possibly excepted when all the shares are known. This is the classical security notion for MPC, where the security proof is performed by simulating the view of the adversary from the output of the result: nothing when not all the shares are asked, and just the sum of the inputs when all the shares are queried. We also have to deal with the corruptions, which give the users' secret keys.

5.1.2 DSum protocol in the random oracle model

The protocol below is similar to [KDK11], with a hash function. We provide a new security analysis, which relies on the CDH problem in the Random Oracle Model.

- $\mathsf{DSum.SetUp}(\lambda)$: takes as input the security parameter λ and generates a group $\mathbb{G}$ of prime order p, with a generator g, were the CDH assumption holds. It also generates a hash function $\mathcal{H} : \{0,1\}^* \to \mathbf{G}$, for any group $\mathbf{G}$, denoted additively. Each user i, picks $t_i \xleftarrow{\$} \mathbb{Z}_p$. The public parameters pp are $(\mathbb{G}, p, g, \mathcal{H}, ([t_i])_i)$ and the personal secret keys $\mathsf{sk}_i = t_i$ for $i = 1 \cdots n$ (with the public parameters).
- $\mathsf{DSum.Encode}(x_i, \ell, \mathsf{sk}_i)$: takes the x_i value to encode, a label ℓ and the personal secret key $\mathsf{sk}_i = t_i$ of the user i, it returns $M_{\ell,i}$ computed as below, where $h_{\ell,i,j} = \mathcal{H}([t_{\min\{i,j\}}], [t_{\max\{i,j\}}], t_i \cdot [t_j], \ell) = h_{\ell,j,i}$: $M_{\ell,i} = x_i - \sum_{j<i} h_{\ell,i,j} + \sum_{j>i} h_{\ell,i,j}$.
- $\mathsf{DSum.Combine}(\vec{M} = (M_{\ell,i})_i)$: takes as input a vector $\vec{M}$ of shares. Computes and return the value $\sum_i M_{\ell,i}$.

Correctness. The correctness should show that the sum of the shares is equal to the sum of the x_i's: the former is equal to

$$\sum_i \left(x_i - \sum_{j<i} h_{\ell,i,j} + \sum_{j>i} h_{\ell,i,j} \right) = \sum_i x_i - \sum_i \sum_{j<i} h_{\ell,i,j} + \sum_i \sum_{j>i} h_{\ell,j,i}$$
$$= \sum_i x_i - \sum_i \sum_{j<i} h_{\ell,i,j} + \sum_j \sum_{i<j} h_{\ell,j,i} = \sum_i x_i$$

5.1.3 Security analysis

We now provide the proof of security for the DSum variant described above, under CDH assumption in the Random Oracle Model. We will prove that there exists a simulator that generates the view of the adversary from the output only. In this proof, we will assume static corruptions (the set $\mathcal{CS}$ of the corrupted clients is known from the beginning) and the hardness of the CDH problem. However, this construction will only tolerate up to $n-2$ corruptions, so that there are at least 2 honest users. But this is also the case for the MCFE.

Proof

W.l.o.g., we can assume that $\mathcal{HS} = \{1, \ldots, n-c\}$ and $\mathcal{CS} = \{n-c+1, \ldots, n\}$, by simply reordering the clients, when $\mathcal{CS}$ is known. We will gradually modify the behavior of the simulator, with less and less powerful queries. At the beginning, the DSum.Encode-query takes all the same inputs as in the real game, including the secret keys. At the end, it should just take the sum (when all the queries have been asked), as well as the corrupted x_j's.

Game $\mathbf{G}_0$: the simulator runs as in the real game, with known $\mathcal{CS}$.

Game $\mathbf{G}_1$: the simulator is given a group $\mathbb{G}$ with a generator g and a random pair $(X = [t]; Y = [t^2])$.

- DSum.SetUp: the simulator randomly chooses $\alpha_i \xleftarrow{\$} \mathbb{Z}_p$, for $i =$

$1, \ldots, n-c$, and defines $X_i \leftarrow X + [\alpha_i]$. This sets $t_i = t + \alpha_i$. It can also set $Y_{i,j} = \mathsf{CDH}(X_i, X_j) = Y + (\alpha_i + \alpha_j) \cdot X + [\alpha_i \alpha_j]$, for $i, j \leq n - c$. It then randomly choses $t_i \leftarrow \mathbb{Z}_p$ for $i > n - c$ and sets $X_i = [t_i]$. It can also generate all the other $Y_{i,j} = \mathsf{CDH}(X_i, X_j)$'s, using the known t_i's. It sends the X_i's as the pp, and the secret keys t_i of the corrupted users.

- $\mathsf{DSum.Encode}(x_i, \ell)$: the simulator generates all the required $h_{\ell,i,j}$ using the X_j's and $Y_{i,j}$'s, querying the hash function, and returns $M_{\ell,i} = x_i - \sum_{j<i} h_{\ell,i,j} + \sum_{j>i} h_{\ell,i,j}$.

Game $\mathbf{G}_2$: the simulator does as above, but just uses a random $Y' \xleftarrow{\$} \mathbb{G}$ instead of Y, to answer the $\mathsf{DSum.Encode}$-queries.

This can make a difference for the adversary if the latter asks for the hash function on some tuple $(X_{\min\{i,j\}}, X_{\max\{i,j\}}, \mathsf{CDH}(X_i, X_j), \ell)$, for $i, j \leq n - c$, as this will not be the value $h_{\ell,i,j}$, which has been computed using $Y_{i,j} \neq \mathsf{CDH}(X_i, X_j)$. In such a case, one can find $\mathsf{CDH}(X_i, X_j) = Y + [\alpha_i + \alpha_j] \cdot X + [\alpha_i \alpha_j]$ in the list of the hash queries, and thus extract $Y = \mathsf{CDH}(X, X)$. As a consequence, under the hardness of the square Diffie-Hellman problem (which is equivalent to the CDH problem), this simulation is indistinguishable from the previous one.

Game $\mathbf{G}_3$: the simulator does as above except for the $\mathsf{DSum.Encode}$-queries. If this is not the last-honest query under label ℓ, the simulator returns $M_{\ell,i} = -\sum_{j<i} h_{\ell,i,j} + \sum_{j>i} h_{\ell,i,j}$; for the last honest query, it returns $M_{\ell,i} = S_H - \sum_{j<i} h_{\ell,i,j} + \sum_{j>i} h_{\ell,i,j}$, where $S_H = \sum_{j \in \mathcal{HS}} x_j$.

Actually, for a label ℓ, if we denote i_ℓ the index of the honest player involved in the last query, the view of the adversary is exactly the same as if, for every $i \neq i_\ell$, we have replaced h_{ℓ,i,i_ℓ} by $h_{\ell,i,i_\ell} + x_i$ (if $i_\ell > i$) or by $h_{\ell,i,i_\ell} - x_i$ (if $i_\ell < i$). We thus replace uniformly distributed variables by other uniformly distributed variables: this simulation is perfectly indistinguishable from the previous one.

Game $\mathbf{G}_4$: the simulator now ignores the values $h_{\ell,i,j}$ for honest i, j. But for each label, it knows the corrupted x_j's, and can thus compute the values $M_{\ell,j}$ for the corrupted users, using the corrupted x_j's and secret keys. If this is not the last honest query, it returns a random $M_{\ell,i}$. For the last honest query, knowing $S = \sum_j x_j$, it outputs $M_{\ell,i} = S - \sum_{j \neq i} M_{\ell,j}$.

As in the previous analysis, if one first sets all the $h_{\ell,i,j}$, for $j \neq i_\ell$, this corresponds to define h_{ℓ,i,i_ℓ} from $M_{\ell,i}$, for $i \neq i_\ell$.

5.1.4 DSum protocol in the standard model

A variant of the protocol from Section 5.1.2 can also be described with a randomness extractor and a PRF. We then provide the security analysis under the DDH assumption and the PRF indistinguishability. More precisely, for the randomness extractor, we can use the Leftover-Hash-Lemma (see Theorem 2.1), with a random seed k in the common reference string to extract random keys K for a PRF $(\mathcal{F}_K)_K$, with a universal hash function $(H_k)_k$:

- $\mathsf{DSum.SetUp}(\lambda)$: takes as input the security parameter λ and generates a group $\mathbb{G}$ of prime order p, with a generator g. From a family of universal hash functions $(H_k)_k$ and a random key k, this define the randomness extractor $\mathcal{E}(\cdot) = H_k(\cdot)$, later used to generate the keys K of a PRF $(\mathcal{F}_K)_K$. Each user i, picks $t_i \xleftarrow{\$} \mathbb{Z}_p$. The public parameters pp are $(\mathbb{G}, p, g, \mathcal{E}, (\mathcal{F}_K)_K, ([t_i])_i)$ and the personal secret keys $\mathsf{sk}_i = t_i$ for $i = 1 \cdots n$ (with the public parameters).
- $\mathsf{DSum.Encode}(x_i, \ell, \mathsf{sk}_i)$: takes the x_i value to encode, a label ℓ and the personal secret key $\mathsf{sk}_i = t_i$ of the user i, it returns $M_{\ell,i}$ computed as below, where $h_{\ell,i,j} = \mathcal{F}_{K_{i,j}}(\ell)$ with $K_{i,j} = \mathcal{E}(t_i \cdot [t_j])$: $M_{\ell,i} = x_i - \sum_{j<i} h_{\ell,i,j} + \sum_{j>i} h_{\ell,i,j}$.
- $\mathsf{DSum.Combine}(\vec{M} = (M_{\ell,i})_i)$: takes as input a vector $\vec{M}$ of shares. Computes and return the value $\sum_i M_{\ell,i}$.

The correctness is the same as in Section 5.1.2, since it just makes use of $h_{\ell,i,j}$. The security however requires the DDH assumption, in order to guarantee the randomness of all the Diffie-Hellman values $[t_i \cdot t_j]$. The Left-over-Hash Lemma thereafter ensures the uniform and independent distributions of the $K_{i,j}$'s which then make the $h_{\ell,i,j}$'s unpredictable for all the honest i, j.

5.1.5 Security analysis

In the section Section 5.1.3, we observe that we do not exploit programmability of the random oracle, and can actually use the Decisional Diffie-Hellman assumption to prove it in the standard model. The key used $K_{i,j}$ for $\mathcal{F}$ is $\mathcal{E}([t_i t_j])$, where $\mathcal{E}$ is a randomness extractor, and the input is ℓ.

Proof

We still assume that $\mathcal{HS} = \{1, \ldots, n-c\}$.

Game $\mathbf{G}_0$: the simulator runs as in the real game, with known $\mathcal{CS}$ (assumed to be $\{n-c+1, \ldots, n\}$, without loss of generality, since we are in the static corruption setting).

Game $\mathbf{G}_1$: the simulator does as above, but just uses a random value $Y_{i,j} \xleftarrow{\$} \mathbb{G}$ instead of the key $[t_i t_j]$, when both $i \neq j \in \mathcal{HS}$, to generate the $K_{i,j}$'s to answer the $\mathsf{DSum.Encode}$-queries. After the hybrid sequence described below, the advantage for the adversary is:

$$|\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A})| \leq \frac{(n-c)^2}{2} \cdot \mathsf{Adv}^{\mathsf{ddh}}(\mathcal{B}),$$

for some adversary $\mathcal{B}$ running with a similar time as $\mathcal{A}$.

Game $\mathbf{G}_2$: the simulator now uses random keys $K_{i,j}$'s in the cases $i < j$ are both honest, and $K_{j,i} = K_{i,j}$. Because of the entropy on the $Y_{i,j}$'s, the Left-over-Hash Lemma guarantees a statistical indistinguishability with the previous game.

Game $\mathbf{G}_3$: The simulator now chooses random $h_{\ell,i,j}$ for any ℓ, in the cases $i < j$ are both honest, and $h_{\ell,j,i} = h_{\ell,i,j}$. Under the indistinguishability of

the PRF with random keys, this game is indistinguishable from $\mathbf{G}_2$. Now, the rest of the proof is similar to the previous one, with a final simulation as in above game $\mathbf{G}_4$.

Hybrid Sequence:

Here we present the hybrid games $\mathbf{H}_{i,j,k}$ between $\mathbf{G}_0$ and $\mathbf{G}_1$. An iteration of this sequence describes how to replace the value $[t_{i^*}t_{j^*}]$ used in the setup phasis, for honest $i^* < j^*$, by random $Y_{i^*,j^*} \xleftarrow{\$} \mathbb{G}$. The progression follows the lexicographical order on the pairs $(i,j) \in \mathcal{HS}$ where $i < j$, and $\mathsf{Succ}(i,j)$ denotes the next pair. It will be clear that $\mathbf{G}_0 = \mathbf{H}_{1,2,0}$ and $\mathbf{G}_1 = \mathbf{H}_{n-c-1,n-c,3}$. In addition, for all $(1,2) \leq (i^*,j^*) < (n-c-1, n-c)$, $\mathbf{H}_{i^*,j^*,3} = \mathbf{H}_{\mathsf{Succ}(i^*,j^*),0}$. We indeed insist that $K_{i,i}$ is never used, so only Diffie-Hellman values for two different keys are used.

Game $\mathbf{H}_{i^*,j^*,0}$: the simulator runs the real game, except that it additionally initializes $Y_{i,j}$ in the DSum.SetUp, used for the extracted keys $K_{i,j} = \mathcal{E}(Y_{i,j})$ during the DSum.Encode, either correctly as $[t_it_j]$ or at random:

- DSum.SetUp: after having generated the group $\mathbb{G}$ of prime order p, with a generator g, the randomness extractor $\mathcal{E}(\cdot)$, and the PRF $(\mathcal{F}_K)_K$, the simulator generates the secret keys $t_i \xleftarrow{\$} \mathbb{Z}_p$ and sets $X_i \leftarrow [t_i]$, for all i. Then it defines:
  - for $(i,j) < (i^*,j^*)$, where $i < j$ are both honest, pick a random element $Y_{i,j} \xleftarrow{\$} \mathbb{G}$
  - for $(i,j) \geq (i^*,j^*)$, where $i < j$ are both honest, set $Y_{i,j} \leftarrow [t_it_j]$
  - for (i,j) where $i < j$ and some of them is corrupted, set $Y_{i,j} \leftarrow [t_it_j]$
  - for (i,j) where $i > j$, set $Y_{i,j} \leftarrow Y_{j,i}$

  It sends the X_i's as the pp, and the secret keys t_i of the corrupted users.

Game $\mathbf{H}_{i^*,j^*,1}$: for $i^* < j^*$, the simulator is given a group $\mathbb{G}$ with a generator g and a random Diffie-Hellman tuple $(X = [x], Y = [y], Z = [xy])$.

- DSum.SetUp: it uses the above group $\mathbb{G}$ and generator g, and generates $\mathcal{E}$ and $(\mathcal{F}_K)_K$. For the indices i^*, j^*, the simulator defines $X_{i^*} \leftarrow X$ and $X_{j^*} \leftarrow Y$. This sets $t_{i^*} \leftarrow x$ and $t_{j^*} \leftarrow y$. It can also set $Y_{i^*,j^*} = \mathsf{CDH}(X_{i^*}, X_{j^*}) = Z$. It then randomly chooses $t_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \neq i^*, j^*$ and sets $X_i \leftarrow [t_i]$. It can also generate $Y_{i,j} = \mathsf{CDH}(X_i, X_j)$, using the known t_i, for $(i,j) > (i^*,j^*)$ and $i < j$. The cases $(i,j) < (i^*,j^*)$ for $i < j$ and the cases $i > j$ remain unchanged. It sends the X_i's as the pp, and the secret keys t_i of the corrupted users.

  The view of the adversary remains the same.

Game $\mathbf{H}_{i^*,j^*,2}$: for $i^* < j^*$, the simulator is given a random tuple $(X = [x], Y = [y], Z \xleftarrow{\$} \mathbb{G})$, and does as above. Under the hardness of the

Decisional Diffie-Hellman problem, this simulation is indistinguishable from the previous one.

Game $\mathbf{H}_{i^*,j^*,3}$: this is quite similar to game $\mathbf{H}_{i^*,j^*,0}$, but with difference for $(i,j) = (i^*, j^*)$:

- DSum.SetUp: after having generated the group $\mathbb{G}$ of prime order p, with a generator g, the randomness extractor $\mathcal{E}(\cdot)$, and the PRF $(\mathcal{F}_K)_K$, the simulator generates the secret keys $t_i \xleftarrow{\$} \mathbb{Z}_p$ and sets $X_i \leftarrow [t_i]$, for all i. Then it defines:
  - for $(i,j) \leq (i^*, j^*)$, where $i < j$ are both honest, pick a random element $Y_{i,j} \xleftarrow{\$} \mathbb{G}$
  - for $(i,j) > (i^*, j^*)$, where $i < j$ are both honest, set $Y_{i,j} \leftarrow [t_i t_j]$
  - for (i,j) where $i < j$ and some of them is corrupted, set $Y_{i,j} \leftarrow [t_i t_j]$
  - for (i,j) where $i > j$, set $Y_{i,j} \leftarrow Y_{j,i}$

  The view of the adversary does not change.

Starting from $(1,2)$ up to $(n-c-1, n-c)$, there are $(n-c)(n-c-1)/2$ cases with $i^* < j^*$ which involve the DDH assumption, hence the conclusion.

5.1.6 Application to IP-DMCFE

Finally, one can generically convert an IP-MCFE into an IP-DMCFE, when $\mathsf{dk}_{\vec{y}} = \left(\vec{y}, \vec{d_{\vec{y}}}\right)$, where $\vec{d_{\vec{y}}} = \sum_i x_i$, with the x_i's computed by each client, as $x_i \leftarrow y_i \cdot \vec{s}_i$ in [CDG+18a], by letting the clients generating the DSum secret keys at the setup time, and the label is the vector $\vec{y}$:

- $\mathsf{SetUp}(\lambda)$: runs the set-up of the IP-MCFE scheme plus $\mathsf{DSum.SetUp}(\lambda)$.
- $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \vec{y})$: outputs $M_{\vec{y},i} \leftarrow \mathsf{DSum.Encode}(x_i, \vec{y}, \mathsf{sk}_i)$.
- $\mathsf{DKeyComb}((M_{\vec{y},i})_i, \vec{y})$: outputs the functional decryption key $\mathsf{dk}_{\vec{y}} = \left(\vec{y}, \vec{d_{\vec{y}}}\right)$, where $\vec{d_{\vec{y}}}$ is publicly computed as $\mathsf{DSum.Combine}((M_{\vec{y},i})_i)$.

In the last simulated game, we can now show that all the $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \vec{y})$-queries are simulated at random, excepted the last query that requires a DKeyGen-query to the IP-MCFE scheme to get the sum and program the output. Hence, unless all the queries are asked, the functional decryption key is unknown. However, this proof system forces statical security in the final DMCFE, meaning that the adversary decides the set of corrupted users on the beginning of the security game (Definition 3.2).

Remark : This technique is in the same vein as the idea introduced in a recent independent work [ACF+19]. In that work, the SetUp is fully decentralized by definition, and their construction uses a two-round MPC scheme in a same way as we do with the DSum: one round in the SetUp to output public parameters and another in the decryption key generation to compute the key. This potentially generalizes our method to several other functionalities.

5.2 IP-DMCFE from pairings

While the transformation using DSum is quite generic, our original decentralized scheme from [CDG+18a] relies on pairing groups. The idea, however, remains the same: using an existing MCFE, but making the decryption key with a MPC scheme. In this section, the decryption key is build with another MCFE scheme.

Overview

Our construction from Section 4.2.1 (or Section 4.2.3 if we double s_i) uses functional decryption keys $\mathsf{dk}_{\vec{y}} = (\vec{y}, \sum_i s_i y_i) = (\vec{y}, dk)$, where $dk = \langle \vec{s}, \vec{y} \rangle = \sum_i s_i y_i = \langle \vec{t}, \vec{1} \rangle$, with $t_i = s_i y_i$, for $i = 1, \ldots, n$, and $\vec{1} = (1, \ldots, 1)$. Hence, one can split $\mathsf{msk} = \vec{s}$ into $\mathsf{msk}_i = s_i$ and define $F(\vec{t}) = \langle \vec{t}, \vec{1} \rangle$. We could thus wish to use one of the MCFE constructions for inner product from Section 4.2 to describe a DMCFE for inner product. However, this is not straightforward as those schemes only allow small results for the function evaluations, since a discrete logarithm has to be computed. While, for real-life applications, it might be reasonable to assume the plaintexts and any evaluations on them are small enough, it is impossible to recover such a large scalar as $dk = \langle \vec{s}, \vec{y} \rangle$, which comes up when we use our scheme to encrypt encryption keys. Nevertheless, following this idea we can overcome the concern above with pairings: one can only recover $[dk]$, but using a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, one can use our MCFE in both $\mathbb{G}_1$ and $\mathbb{G}_2$. This allows us to compute the functional decryption in $\mathbb{G}_T$, to get $[\langle \vec{x}, \vec{y} \rangle]_T$, which is decryptable as $\langle \vec{x}, \vec{y} \rangle$ is small enough.

5.2.1 Construction

Let us describe the new construction, using an asymmetric pairing group, as in Section 2.1.

- $\mathsf{SetUp}(\lambda)$: generates $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e) \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$. Samples two full-domain hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ onto $\mathbb{G}_1^2$ and $\mathbb{G}_2^2$ respectively. Each sender $\mathcal{S}_i$ generates $\vec{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$ for all $i \in [n]$, and interactively generates $\mathbf{T}_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$ such that $\sum_{i\in[n]} \mathbf{T}_i = \mathbf{0}$. One then sets $\mathsf{mpk} \leftarrow (\mathcal{PG}, \mathcal{H}_1, \mathcal{H}_2)$, and for $i = 1, \ldots, n$, $\mathsf{ek}_i = \vec{s}_i$, $\mathsf{sk}_i = (\vec{s}_i, \mathbf{T}_i)$.

- $\mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$: takes as input the value x_i to encrypt, under the key $\mathsf{ek}_i = \vec{s}_i$ and the label ℓ. It computes $[\vec{u}_\ell]_1 := \mathcal{H}_1(\ell) \in \mathbb{G}_1^2$, and outputs the ciphertext $[c_i]_1 = [\vec{u}_\ell^\top \vec{s}_i + x_i]_1 \in \mathbb{G}_1$.

- $\mathsf{DKeyGenShare}(\mathsf{sk}_i, \vec{y})$: on input $\vec{y} \in \mathbb{Z}_p^n$ that defines the function $f_{\vec{y}}(\vec{x}) = \langle \vec{x}, \vec{y} \rangle$, and the secret key $\mathsf{sk}_i = (\vec{s}_i, \mathbf{T}_i)$, it computes $[\vec{v}_{\vec{y}}]_2 := \mathcal{H}_2(\vec{y}) \in \mathbb{G}_2^2$, $[\vec{d}_i]_2 := [y_i \cdot \vec{s}_i + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$, and returns the partial decryption key as $\mathsf{dk}_{\vec{y},i} := ([\vec{d}_i]_2)$.

- $\mathsf{DKeyComb}((\mathsf{dk}_{\vec{y},i})_{i\in[n]}, \vec{y})$: the partial decryption keys $\left(\mathsf{dk}_{\vec{y},i} = ([\vec{d}_i]_2)\right)_{i\in[n]}$, lead to $\mathsf{dk}_{\vec{y}} := (\vec{y}, [\vec{d}]_2)$, where $[\vec{d}]_2 = \sum_{i\in[n]} [\vec{d}_i]_2$.

- $\mathsf{Decrypt}(\mathsf{dk}_{\vec{y}}, \ell, ([c_i]_1)_{i\in[n]})$: on input the decryption key $\mathsf{dk}_{\vec{y}} = [\vec{d}]_2$, the label ℓ, and ciphertexts $([c_i]_1)_{i\in[n]}$, it computes $[\alpha]_T := \sum_{i\in[n]} e([c_i]_1, [y_i]_2) - e([\vec{u}_\ell]_1^\top, [\vec{d}]_2)$, and eventually solve the discrete logarithm in basis $[1]_T$ to extract and return α.

Correctness: Let $\vec{x}, \vec{y} \in \mathbb{Z}_p^n$, we have:

$$[\vec{d}]_2 = \sum_{i\in[n]} [\vec{d_i}]_2 = \sum_{i\in[n]} [y_i \cdot \vec{s}_i + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$$
$$= [\sum_{i\in[n]} y_i \cdot \vec{s}_i]_2 + [\vec{v}_{\vec{y}}]_2 \cdot \sum_{i\in[n]} \mathbf{T}_i = [\sum_{i\in[n]} y_i \cdot \vec{s}_i]_2.$$

Thus:

$$[\alpha]_T := \sum_{i\in[n]} e([c_i]_1, [y_i]_2) - e([\vec{u}_\ell]_1^\top, [\vec{d}]_2)$$
$$= \sum_i [(\vec{u}_\ell^\top \vec{s}_i + x_i) y_i]_T - [\sum_{i\in[n]} y_i \vec{u}_\ell^\top \vec{s}_i]_T = [\sum_i x_i y_i]_T.$$

Compatibility

Our constructions from Section 4.2 use a prime-order group, while this one uses pairing groups. Since the latter use the former as a building block, we must use groups that are compatible with each other. Notice that one can generate a prime-order group either with $\mathcal{G} := (\mathbb{G}, p, P) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, but also using $\mathcal{PG} := (\mathbb{G}_1, \mathbb{G}_2, p, P_1, P_2, e) \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, and setting $\mathbb{G} := \mathbb{G}_1$. This is possible here because we use asymmetric pairings and rely on the SXDH assumption in the pairing group, which is DDH in $\mathbb{G}_1$ and $\mathbb{G}_2$ (see Section 2.3).

5.2.2 Security analysis

Theorem 5.1: `sta-wtr-IND*`-Security

The above DMCFE protocol (see Section 5.2.1) is `sta-wtr-IND*`-secure under the SXDH assumption, in the random oracle model. Namely, for any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:

$$\mathsf{Adv}^{\mathtt{IND}}(\mathcal{A}) \leq 2Q_1 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + 2Q_2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + \frac{2Q_1 + 2Q_2}{p}$$
$$+ \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t + 4Q_1 \times t_{\mathbb{G}_1}) + 2 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + 4Q_2 \times t_{\mathbb{G}_2}),$$

where Q_1 and Q_2 are the number of (direct and indirect) queries to $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively (modeled as random oracles). The former being asked by QLeftRight-queries and the latter being asked by QDKeyGen-queries.

We stress that the above Theorem supports adaptive encryption queries, but static corruptions only. To prove it, we proceed using hybrid games, described in Fig. 5.1.

Proof

Game G_0: this is the `sta-wtr-IND*`-security game as given in Section 3.2.2, but with the set $\mathcal{CS}$ of corrupted senders known from the beginning. Note that the hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ are modeled as random oracles. The former is used to generate $[\vec{u}_\ell]_1 := \mathcal{H}_1(\ell) \in \mathbb{G}_1^2$ and the latter $[\vec{v}_{\vec{y}}]_2 := \mathcal{H}_2(\vec{y}) \in \mathbb{G}_2^2$.

Game G_1: we replace the hash function $\mathcal{H}_2$ by a random oracle RO_2 that generates random pairs from $\mathbb{G}_2^2$ on the fly. In addition, for any $\mathsf{QDKeyGen}$-query on a corrupted index $i \in \mathcal{CS}$, one generates the partial functional decryption key by itself, without explicitly querying $\mathsf{QDKeyGen}$. Hence, we can assume that $\mathcal{A}$ does not query $\mathsf{QCorrupt}$ and $\mathsf{QDKeyGen}$ on the same indices $i \in [n]$. The simulation remains perfect, and so $\mathsf{Adv}_0 = \mathsf{Adv}_1$.

Game G_2: now, the outputs of RO_2 are uniformly random in the span of $[\vec{b}]_2$ for $\vec{b} := \binom{1}{a'}$, with $a' \xleftarrow{\$} \mathbb{Z}_p$. As in the previous proof, we have $\mathsf{Adv}_1 - \mathsf{Adv}_2 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + 4Q_2 \times t_{\mathbb{G}_2})$, where Q_2 is the number of RO_2-queries and $t_{\mathbb{G}_2}$ the time for an exponentiation.

Game G_3: we replace all the partial key decryption answers by $\mathsf{dk}_{\vec{y},i} := [y_i \cdot \vec{s}_i + \vec{w}_i \cdot (\vec{b}^\perp)^\top \vec{v}_{\vec{y}} + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$, for new $\vec{w}_i \xleftarrow{\$} \mathbb{Z}_p^2$, such that $\sum_i \vec{w}_i = \mathbf{0}$, for each $\vec{y}$. This sum being among the honest clients, we need to know the last queried honest client to set this sum to zero. Hence the requirement to know the set of honest clients, and thus just security against static corruptions. We show below that $\mathsf{Adv}_2 = \mathsf{Adv}_3$.

Game G_4: we switch back the distribution of all the vectors $[\vec{v}_{\vec{y}}]_2$ output by RO_2, from uniformly random in the span of $[\vec{b}]_2$, to uniformly random over $\mathbb{G}_2^2$, thus back to $\mathcal{H}_2(\vec{y})$. This transition is reverse to the two first transitions of this proof: $\mathsf{Adv}_3 - \mathsf{Adv}_4 \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t + 4Q_2 \times t_{\mathbb{G}_2})$.

In order to prove the gap between G_2 and G_3, we do a new hybrid proof:

Game $G_{3.1.1}$: this is exactly game G_2. Thus, $\mathsf{Adv}_2 = \mathsf{Adv}_{3.1.1}$.

$G_{3.q.1} \rightsquigarrow G_{3.q.2}$**:** as in the previous proof, we first change the distribution of the output of the q-th RO_2-query, from uniformly random in the span of $[\vec{b}]$ to uniformly random over $\mathbb{G}^2$, using the DDH assumption. Then, we use the basis $(\binom{1}{a'}, \binom{-a'}{1})$ of $\mathbb{Z}_p^2$, to write a uniformly random vector over $\mathbb{Z}_p^2$ as $v_1 \cdot \vec{b} + v_2 \cdot \vec{b}^\perp$, where $v_1, v_2 \xleftarrow{\$} \mathbb{Z}_p$. Finally, we switch to $v_1 \cdot \vec{b} + v_2 \cdot \vec{b}^\perp$ where $v_1 \xleftarrow{\$} \mathbb{Z}_p$, and $v_2 \xleftarrow{\$} \mathbb{Z}_p^*$, which only changes the adversary view by a statistical distance of $1/p$: $\mathsf{Adv}_{3.q.1} - \mathsf{Adv}_{3.q.2} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t) + 1/p$. The last step with $v_2 \in \mathbb{Z}_p^*$ will be important to guarantee that $\vec{v}_{\vec{y}}^\top \vec{b}^\perp \neq 0$.

$G_{3.q.2} \rightsquigarrow G_{3.q.3}$**:** we now change the simulation of $\mathsf{dk}_{\vec{y},i}$ from $\mathsf{dk}_{\vec{y},i} = [y_i \cdot \vec{s}_i + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$ to $\mathsf{dk}_{\vec{y},i} = [y_i \cdot \vec{s}_i + \mathsf{RF}_i(\vec{y}) + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$, with some RF_i functions onto $\mathbb{Z}_p^2$ such that $\sum_i \mathsf{RF}_i(\vec{y}) = 0$ for any input $\vec{y}$. We prove $\mathsf{Adv}_{3.q.2} = \mathsf{Adv}_{3.q.3}$.

To this aim, we use the fact that the two following distributions are identical, for any choice of $\vec{w}_i \xleftarrow{\$} \mathbb{Z}_p^2$, such that $\sum_i \vec{w}_i = \mathbf{0}$:

$$(\mathbf{T}_i)_{i \in \mathcal{HS}} \text{ and } (\mathbf{T}_i + \vec{w}_i (\vec{b}^\perp)^\top)_{i \in \mathcal{HS}},$$

where for all $i \in [n]$, $\mathbf{T}_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$ such that $\sum_i \mathbf{T}_i = \mathbf{0}$, and $\vec{b}^\perp := \binom{-a'}{1}$.

The extra terms $(\vec{w}_i (\vec{b}^\perp)^\top)_{i \in \mathcal{HS}}$ only appear in the output of the queries to $\mathsf{QDKeyGen}$ which use the vector $[\vec{v}_{\vec{y}}]_2$ computed on the q-th RO_2-query (if there are such queries), because for all other queries, $[\vec{v}_{\vec{y}}]_2$ lies in the span

of $[\vec{b}]_2$, and $\vec{b}^\top \vec{b}^\perp = 0$. We thus have $\mathsf{dk}_{\vec{y},i} := [y_i \cdot \vec{s}_i + \vec{w}_i \cdot (\vec{b}^\perp)^\top \vec{v}_{\vec{y}} + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$. Since $\vec{v}_{\vec{y}}$ is such that $\vec{v}_{\vec{y}}^\top \vec{b}^\perp \neq 0$, $(\vec{b}^\perp)^\top \vec{v}_{\vec{y}} \neq 0$. In that case, the vectors $\vec{w}_i \cdot (\vec{b}^\perp)^\top \vec{v}_{\vec{y}}$ are uniformly random over $\mathbb{Z}_p^2$ such that $\sum_i \vec{w}_i \cdot (\vec{b}^\perp)^\top \vec{v}_{\vec{y}} = \mathbf{0}$, which is as in $G_{3.q.3}$, by setting $\mathsf{RF}_i(\vec{y}) := \vec{w}_i \cdot (\vec{b}^\perp)^\top \vec{v}_{\vec{y}}$.

$G_{3.q.3} \rightsquigarrow G_{3.q+1.1}$**:** this transition is the reverse of $G_{3.q.1} \rightsquigarrow G_{3.q.2}$, namely, we use the DDH assumption to switch back the distribution of $[\vec{v}_{\vec{y}}]_2$ to uniformly random in the span of $[\vec{b}]_2$: $\mathsf{Adv}_{3.q.3} - \mathsf{Adv}_{3.q+1.1} \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_2}(t) + 1/p$.

Then one can note that $G_{3.Q_2+1.1} = G_3$, but also that in Game G_4, all the $\mathsf{dk}_{\vec{y},i}$ output by $\mathsf{QDKeyGen}$ can be computed only knowing $\sum_{i\in[n]} \vec{s}_i \cdot y_i$, which is exactly the functional decryption key $\mathsf{dk}_{\vec{y}}$ from our MCFE in Section 4.2.3. This follows from the fact that the values $\mathsf{RF}_i(\vec{y})$ perfectly mask the vectors $\vec{s}_i \cdot y_i$, up to revealing $\sum_{i\in[n]} \vec{s}_i \cdot y_i$ (as the RF_i must sum up to the zero function). Thus, we can reduce to the $\mathtt{IND}$-security of the MCFE from Section 4.2.3 (or even $\mathtt{sta\text{-}IND}$-security) by designing an adversary $\mathcal{B}$ against the MCFE from Section 4.2.3: Adversary $\mathcal{B}$ first samples $\mathbf{T}_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$ for all $i \in [n]$, such that $\sum_{i\in[n]} \mathbf{T}_i = \mathbf{0}$. It sends $\mathcal{CS}$ given by $\mathcal{A}$ (set of static corruptions), then it receives mpk from the MCFE security game, as well as the secret keys $\vec{s}_i$ for $i \in \mathcal{CS}$. It forwards mpk as well as $(\vec{s}_i, \mathbf{T}_i)$ for $i \in \mathcal{CS}$ to $\mathcal{A}$. Then

- $\mathcal{B}$ answers oracle calls to RO_1, RO_2 and $\mathsf{QEncrypt}$ from $\mathcal{A}$ using its own oracles.
- To answer $\mathsf{QDKeyGen}(i, \vec{y})$: if i is the last non-corrupted index for $\vec{y}$, $\mathcal{B}$ queries its own $\mathsf{QDKeyGen}$ oracle on $\vec{y}$, to get $\mathsf{dk}_{\vec{y}} := \sum_i \vec{s}_i \cdot y_i \in \mathbb{Z}_p^2$, computes $[\vec{v}_{\vec{y}}]_2 := \mathcal{H}_2(\vec{y})$, and returns $\mathsf{dk}_{\vec{y},i} := [\mathsf{dk}_{\vec{y}} + \mathsf{RF}_i(\vec{y}) + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$ to $\mathcal{A}$. Otherwise, it computes $[\vec{v}_{\vec{y}}]_2 := \mathcal{H}_2(\vec{y})$, and returns $\mathsf{dk}_{\vec{y},i} := [\mathsf{RF}_i(\vec{y}) + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$ to $\mathcal{A}$. The random functions RF_i are computed on the fly, such that their sum is the zero function.

We stress that this last simulation requires to know $\mathcal{CS}$ and $\mathcal{HS}$, hence static corruptions only. From this reduction, one gets

$$\mathsf{Adv}_4 \leq 2Q_1 \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}_1}(t + 4Q_1 \times t_{\mathbb{G}_1}) + \frac{2Q_1}{p},$$

where Q_1 denotes the number of calls to RO_1, $t_{\mathbb{G}_1}$ denotes the time to compute an exponentiation in $\mathbb{G}_1$. This concludes the proof.

Games G_0, G_1, $\boxed{G_2,\ (G_{3.q.1})_{q\in[Q_{\mathsf{dk}}+1]},\ \boxed{(G_{3.q.2}, G_{3.q.3})_{q\in[Q_{\mathsf{dk}}]}}}$, G_4

$\mathcal{PG} \leftarrow \mathsf{PGGen}(1^\lambda)$, $\forall i \in [n]$: $\vec{s}_i \xleftarrow{\$} \mathbb{Z}_p^2$, $\mathbf{T}_i \xleftarrow{\$} \mathbb{Z}_p^{2\times 2}$, such that $\sum_{i\in[n]} \mathbf{T}_i = \mathbf{0}$
$\mathsf{ek}_i := \vec{s}_i$, $\mathsf{sk}_i := (\vec{s}_i, \mathbf{T}_i)$, $\mathsf{mpk} := (\mathbb{G}, p, g)$.
$\boxed{a' \xleftarrow{\$} \mathbb{Z}_p,\ \vec{b} := \binom{1}{a'}}$
Sample full-domain hash functions $\mathcal{H}_1$ onto $\mathbb{G}_1^2$ and $\mathcal{H}_2$ onto $\mathbb{G}_2^2$.
Sample a bit $b \xleftarrow{\$} \{0,1\}$.
$b' \leftarrow \mathcal{A}^{\mathsf{QEncrypt}(\cdot,\cdot,\cdot,\cdot),\mathsf{QDKeyGen}(\cdot,\cdot),\mathsf{QCorrupt}(\cdot),\mathsf{RO}_1(\cdot),\mathsf{RO}_2(\cdot)}(\mathsf{mpk})$.
Run Finalize on b'.

$\underline{\mathsf{RO}_1(\ell)}$: // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$
Return $\mathcal{H}_1(\ell)$.

$\underline{\mathsf{RO}_2(\vec{y})}$: // G_0, $\boxed{G_1}$, $\boxed{G_2,\ G_{3.q.1},\ \boxed{G_{3.q.2},\ G_{3.q.3}}}$, G_4
$[\vec{v}_{\vec{y}}]_2 := \mathcal{H}_2(\vec{y})$, $\boxed{[\vec{v}_{\vec{y}}]_2 := \mathsf{RF}(\vec{y})}$, $\boxed{[\vec{v}_{\vec{y}}]_2 := [\vec{b} \cdot t_{\vec{y}}]_2, \text{ with } t_{\vec{y}} := \mathsf{RF}'(\vec{y})}$.
On the q-th RO_2-query: $[\vec{v}_{\vec{y}}]_2 := \mathsf{RF}(\vec{y})$.
Return $[\vec{v}_{\vec{y}}]_2$.

$\underline{\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)}$: // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$, G_4
$[\vec{u}_\ell]_1 := \mathsf{RO}_1(\ell)$.
$[c_i]_1 := [\vec{u}_\ell^\top]_1 \cdot \vec{s}_i + [x_i^b]_1$.
Return $[c_i]$.

$\underline{\mathsf{QEncrypt}(i, x_i, \ell)}$: // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$, G_4
$[c_i]_1 := \mathsf{QLeftRight}(i, x_i, x_i, \ell)$
Return $[c_i]$

$\underline{\mathsf{QDKeyGen}(\vec{y}, i)}$: //$G_0$, G_1, G_2, $\boxed{G_{3.q.1},\ G_{3.q.2},\ G_{3.q.3}}$, $\boxed{G_4}$
Compute $[\vec{v}_{\vec{y}}]_2 := \mathsf{RO}_2(\vec{y})$, $\mathsf{dk}_{\vec{y},i} := [y_i \cdot \vec{s}_i + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$, set $\mathcal{S} := \mathcal{S} \cup \{i\}$.
If $[\vec{v}_{\vec{y}}]_2$ is computed on the j-th RO_2-query, for $j < q$:
$\mathsf{dk}_{\vec{y},i} := [y_i \cdot \vec{s}_i + \mathsf{RF}_i(\vec{y}) + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$.
If $[\vec{v}_{\vec{y}}]_2$ is computed on the q-th RO_2-query:
$\mathsf{dk}_{\vec{y},i} := [y_i \cdot \vec{s}_i + \mathsf{RF}_i(\vec{y}) + \mathbf{T}_i \vec{v}_{\vec{y}}]_2$.
$\boxed{\mathsf{dk}_{\vec{y},i} := [y_i \cdot \vec{s}_i + \mathsf{RF}_i(\vec{y}) + \mathbf{T}_i \vec{v}_{\vec{y}}]_2.}$
Return $\mathsf{dk}_{\vec{y},i}$.

$\underline{\mathsf{QCorrupt}(i)}$: Return $(\vec{s}_i, \mathbf{T}_i)$. // G_0, G_1, G_2, $G_{3.q.1}$, $G_{3.q.2}$, $G_{3.q.3}$, G_4

Figure 5.1: Games for the proof of Theorem 5.1. Here, RF, RF' are random functions onto $\mathbb{G}_2^2$ and $\mathbb{Z}_p$, respectively, that are computed on the fly. The RF_i are random functions conditioned on the fact that $\sum_{i\in[n]} \mathsf{RF}_i$ is the zero function. In each procedure, the components inside a solid (dotted, gray) frame are only present in the games marked by a solid (dotted, gray) frame. The Finalize procedure is defined as in Section 3.2.2.

6 Modular security for MCFE

Our work [CDG+18a] gave a construction that only satisfies the weaker wtr-IND* security definition for Inner Product. In this part we propose several techniques from [CDG+18b] to increase the security of a MCFE scheme in multiple ways. Especially, they permit to modify the wtr-IND*-security of the MCFE scheme from Section 4.2.3 into IND-CCA-security.

On a first time, we show how to go, from any variant of IND*, to the same variant of IND, using an extra Secret Sharing Encapsulation, in Section 6.1. This transformation strengthen the security model by allowing the adversary to query encryptions, but not necessarily all of them, leading to incomplete ciphertexts.

Secondly, we show how to allow several encryptions under a same label (X-wtr-IND-Y to X-IND-Y) for Inner Product, in Section 6.2, by adding a layer of single-input functional encryption for Inner Product. This transformation specifically relies on the IP-MCFE scheme from [CDG+18a], described in Section 4.2.3.

Finally, we briefly show in Section 6.3 how to achieve CCA security (additional functional decryption queries) from any X-IND-Y scheme, in a generic way.

We remind these contributions are summed up in Fig. 1.3, and all the different security notions we go into through this chapter are given in Definition 3.2.

Contents

6.1 From IND* to IND with secret sharing encapsulation

As explained in Section 3.1.2, the main difference between the IND notion and the IND*, is that incomplete ciphertexts were considered illegitimate. This was with the intuition that no adversary should use it since this leaks no information. But actually, an adversary could exploit that in the real-life, especially in the scheme from [CDG+18a] which use homomorphic properties. The IND notion requires the scheme to actually leak nothing in such a case.

In this section, we present a generic layer, called the secret sharing encapsulation (SSE), that we will use to encapsulate ciphertexts from a MCFE scheme. It allows a user to recover the ciphertexts from the n senders only when he gets the contributions of all the servers. That is, if one sender did not send anything, the user cannot get any information from any of the ciphertexts of the other senders. More concretely, a share of a key $S_{\ell,i}$ is generated for each user $i \in [n]$ and each label ℓ. Unless all the shares $S_{i,\ell}$ have been generated, the encapsulation keys are random and mask all the ciphertexts.

After giving the definition of SSE, we provide a construction whose security is based on the DBDH assumption, and we show how to apply it on any MCFE scheme.

6.1.1 Definitions for secret sharing encapsulation

Definition 6.1: Secret Sharing Encapsulation

A Secret Sharing Encapsulation (SSE) on $\mathcal{K}$ over a set of n senders is defined by four algorithms:

- $\mathsf{SSE.SetUp}(\lambda)$: on input a security parameter λ, generates the public parameters $\mathsf{pk_{sse}}$ and the personal encryption keys are $\mathsf{ek_{sse},}_i$ for all $i \in [n]$.
- $\mathsf{SSE.Encaps}(\mathsf{pk_{sse}}, \ell)$: on input $\mathsf{pk_{sse}}$ and the label ℓ, outputs a ciphertext C_ℓ and an encapsulation key $K_\ell \in \mathcal{K}$.
- $\mathsf{SSE.Share}(\mathsf{ek_{sse},}_i, \ell)$: on input $\mathsf{ek_{sse},}_i$ and the label ℓ, outputs the share $S_{\ell,i}$.
- $\mathsf{SSE.Decaps}(\mathsf{pk_{sse}}, (S_{\ell,i})_{i\in[n]}, \ell, C_\ell)$: on input all the shares $S_{\ell,i}$ for all $i \in [n]$, a label ℓ, and a ciphertext C_ℓ, outputs the encapsulation key K_ℓ.

Correctness. For any label ℓ, we have: $\Pr[\mathsf{SSE.Decaps}(\mathsf{pk_{sse}}, (S_{\ell,i})_{i\in[n]}, \ell, C_\ell) = K_\ell] = 1$, where the probability is taken over $\left(\mathsf{pk_{sse}}, (\mathsf{ek_{sse},}_i)_{i\in[n]}\right) \leftarrow \mathsf{SSE.SetUp}(\lambda)$, $(C_\ell, K_\ell) \leftarrow \mathsf{SSE.Encaps}(\mathsf{pk_{sse}}, \ell)$, and $S_{\ell,i} \leftarrow \mathsf{SSE.Share}(\mathsf{ek_{sse},}_i, \ell)$ for all $i \in [n]$.

Indistinguishability. We want to show that the encapsulated keys are indistinguishable from random if not all the shares are known to the adversary. We could define a Real-or-Random security game [BDJR97] for all the masks. Instead, we limit the Real-or-Random queries to one label only, and for all the other labels, the adversary can do the encapsulation by itself, since it just uses a public key. This is well-known that a hybrid proof among the label indices (the order they appear in the game)

shows that the One-Label security is equivalent to the Many-Label security. The One-Label definition will be enough for our applications.

Definition 6.2: `1-Label-IND`-security game for SSE

Let us consider an SSE scheme over a set of n senders. We define the following security game against a challenger $\mathcal{C}$, where the adversary has unlimited and adaptive access to the oracles QRealRandom, QShare, and QCorrupt described below.

- Initialize(i^*): the adversary announces an index $i^*[n]$. The challenger $\mathcal{C}$ runs the setup algorithm $(\mathsf{pk}_{\mathsf{sse}}, (\mathsf{ek}_{\mathsf{sse},i})_{i\in[n]}) \leftarrow \mathsf{SetUp}(\lambda)$ and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It provides $\mathsf{pk}_{\mathsf{sse}}$ to the adversary $\mathcal{A}$.
- Challenge queries QRealRandom(ℓ): outputs a ciphertext C_ℓ, together with an encapsulation key K_ℓ^b, where $(C_\ell, K_\ell^0) \leftarrow \mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell)$, and $K_\ell^1 \xleftarrow{\$} \mathcal{K}$, where $\mathcal{K}$ is the encapsulation key space.
- Sharing queries QShare(i, ℓ): outputs $S_{\ell,i} \leftarrow \mathsf{SSE.Share}(\mathsf{ek}_{\mathsf{sse},i}, \ell)$.
- Corruption queries QCorrupt(i): outputs the encapsulation key $\mathsf{ek}_{\mathsf{sse},i}$.
- Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs this $\beta \leftarrow b'$ if the following condition is satisfied: QRealRandom is only queried on at most one label ℓ^* and i^* was not queried to QCorrupt and (i^*, ℓ^*) was not queried to QShare. If this condition is not satisfied, Finalize outputs a random bit β.

We say this SSE is `1-Label-IND`-secure if for any PPT adversary $\mathcal{A}$, its advantage $\mathsf{Adv}_{\mathsf{SSE}}^{\texttt{1-Label-IND}}(\mathcal{A}) = |\Pr[\beta = 1|b = 1] - \Pr[\beta = 1|b = 0]|$ is negligible.

We can also define the weaker static variant, where corruptions are known from the beginning.

6.1.2 Construction

Let us exhibit a concrete construct for our main tool SSE, in the random oracle model, under the DBDH assumption.

- $\mathsf{SSE.SetUp}(\lambda)$: takes as input a security parameter λ and generates $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p_{\mathsf{sse}}, P_1, P_2, e) \xleftarrow{\$} \mathsf{PGGen}_{\mathsf{sse}}(1^\lambda)$. Generates a full domain hash function $\mathcal{H}_{\mathsf{sse}}$ from $\{0,1\}^\lambda$ into $\mathbb{G}_1$. It also generates $\vec{t} \xleftarrow{\$} \mathbb{Z}_p^n$. The public parameters $\mathsf{pk}_{\mathsf{sse}}$ consist of $(\mathcal{PG}, \mathcal{H}_{\mathsf{sse}}, [\vec{t}]_2)$, and the personal encapsulation keys are $\mathsf{ek}_{\mathsf{sse},i} = t_i$, the i-th coordinate of $\vec{t}$.
- $\mathsf{SSE.Share}(\mathsf{ek}_{\mathsf{sse},i}, \ell)$: takes as input the key $\mathsf{ek}_{\mathsf{sse},i} = t_i$ and the label ℓ and outputs the share $S_{\ell,i} = t_i \cdot \mathcal{H}_{\mathsf{sse}}(\ell) \in \mathbb{G}_1$.
- $\mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell)$: takes as input the public key $\mathsf{pk}_{\mathsf{sse}} = (\mathcal{PG}, \mathcal{H}_{\mathsf{sse}}, [\vec{t}]_2)$ and the label ℓ, samples $r \xleftarrow{\$} \mathbb{Z}_p$, and outputs the ciphertext C_ℓ and the encapsulation key K_ℓ defined as: $C_\ell = [r]_2, K_\ell = e(\mathcal{H}_{\mathsf{sse}}(\ell), [r\sum_{j\in[n]} t_j]_2)$.

- $\mathsf{SSE.Decaps}(\mathsf{pk}_{\mathsf{sse}}, (S_{\ell,i})_{i\in[n]}, \ell, C_\ell)$: takes as input all the shares $S_{\ell,i}$ for all $i \in [n]$, a label ℓ and a ciphertext C_ℓ, and outputs an encapsulation key
$$K_\ell = e\Big(\sum_j S_{\ell,j}, C_\ell\Big).$$

We stress here that K_ℓ is not unique for each label ℓ: whereas $S_{\ell,i}$ deterministically depends on ℓ and the client i, K_ℓ is randomized by the random coins r in C_ℓ. Hence, with all the shares, using a specific C_ℓ, one can recover the associated K_ℓ.

Correctness. It follows from the fact that the above decapsulated key K_ℓ is equal to:
$$e\Big(\sum_j t_j \mathcal{H}_{\mathsf{sse}}(\ell), [r]_2\Big) = e\Big(\mathcal{H}_{\mathsf{sse}}(\ell), [r \cdot \sum_j t_j]_2\Big),$$
where the pair (C_ℓ, K_ℓ) has been generated by $\mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell)$ with random r.

6.1.3 Security analysis

The intuition for the security is that given all the $S_{\ell,i} = t_i \cdot \mathcal{H}_{\mathsf{sse}}(\ell)$ for a label ℓ, one can recover the masks $K_\ell = e(\mathcal{H}_{\mathsf{sse}}(\ell), [r \sum_j t_j]_2)$ using $C_\ell = [r]_2$. However if $S_{\ell,i}$ is missing for one slot i, then all the encapsulation keys K_ℓ are pseudo-random, from the DBDH assumption.

Theorem 6.1

Let $\mathcal{A}$ be a PPT adversary against the security of the above SSE. We can build a PPT adversary $\mathcal{B}$ against the q_r-fold DBDH such that:
$$\mathsf{Adv}^{\mathtt{1\text{-}Label\text{-}IND}}_{\mathsf{SSE}}(\mathcal{A}) \leq (1 + q_h) \cdot \mathsf{Adv}^{q_r\text{-}\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{B}),$$
where q_h denotes the number of $\mathcal{H}_{\mathsf{sse}}$ queries (explicit or implicit) and q_r the number of challenge-queries to the QRealRandom oracle.

Applying Lemma Lemma 2.1, one can reduce the security to the DBDH assumption. Here we describe the construction of such an adversary:

Proof

$\mathcal{B}$ receives a q_r-fold DBDH challenge $\Big(\mathcal{PG}, [a]_1, [b]_1, [b]_2, \{[c_i]_2, [s_i]_T\}_{i\in[q_r]}\Big)$, where q_r denotes the number of queries of $\mathcal{A}$ to its oracle QRealRandom, and receives $i^\star \in [n]$ from $\mathcal{A}$.

Then, $\mathcal{B}$ guesses $\rho \xleftarrow{\$} \{0, \ldots, q_h\}$. Intuitively, ρ is a guess on when the random oracle is going to be queried on $\ell^\star$, the first label used as input to QRealRandom (without loss of generality, we can assume QRealRandom is queried at least once by $\mathcal{A}$, otherwise the security is trivially satisfied), with $\rho = 0$ indicating that the adversary never queries $\mathcal{H}_{\mathsf{sse}}$ on $\ell^\star$ before querying QRealRandom.

Then, $\mathcal{B}$ samples $t_i \xleftarrow{\$} \mathbb{Z}_p$ and sets $\mathsf{ek}_{\mathsf{sse},i} := t_i$ for all $i \in [n]$, $i \neq i^\star$, and sets $[t_{i^\star}]_2 = [b]_2$. It returns $\mathsf{pk}_{\mathsf{sse}} = (\mathcal{PG}, [\vec{t}]_2)$ to $\mathcal{A}$.

For any query $\mathsf{QCorrupt}(i)$: if $i \neq i^\star$, $\mathcal{B}$ returns $\mathsf{ek}_{\mathsf{sse},i}$, otherwise $\mathcal{B}$ stops simulating the experiment for $\mathcal{A}$ and returns 0 to its own experiment.

For any query to the random oracle $\mathcal{H}_{\mathsf{sse}}$, if this the ρ'th new query, then $\mathcal{B}$

sets $\mathcal{H}_{\mathsf{sse}}(\ell_\rho) := [a]_1$. For others queries, $\mathcal{B}$ outputs $[h]_1$ for a random $h \xleftarrow{\$} \mathbb{Z}_p$. $\mathcal{B}$ keeps track of the queries and outputs to the random oracle $\mathcal{H}_{\mathsf{sse}}$, so that it answers two identical queries with the same output.

For any query to $\mathsf{QRealRandom}(\ell)$: if ℓ has never been queried to the random oracle $\mathcal{H}_{\mathsf{sse}}$ before (directly, or indirectly via QShare) and $\rho = 0$, then $\mathcal{B}$ sets $\mathcal{H}_{\mathsf{sse}}(\ell) := [a]_1$; if ℓ was queried to random oracle as the ρ'th new query (again, we consider direct and indirect queries to $\mathcal{H}_{\mathsf{sse}}$, the latter coming from QShare), then we already have $\mathcal{H}_{\mathsf{sse}}(\ell) = [a]_1$. In both cases, $\mathcal{B}$ sets $C_\ell \leftarrow [c_j]_2$, for the next index j in the q_r-fold DBDH instance, computes $K_\ell \leftarrow [s_j]_T + e([a]_1, (\sum_{i \neq i^\star} t_i) \cdot [c_j]_2)$, and returns (C_ℓ, K_ℓ) to $\mathcal{A}$. Otherwise, the guess ρ was incorrect: $\mathcal{B}$ stops simulating the experiment for $\mathcal{A}$, and returns 0 to its own experiment. Moreover, if $\mathcal{A}$ ever calls $\mathsf{QRealRandom}$ on different labels ℓ, then $\mathcal{B}$ stops simulating this experiment for $\mathcal{A}$ and returns 0 to its own experiment.

For any query to $\mathsf{QShare}(i, \ell)$: if the random has been called on ℓ, then $\mathcal{B}$ uses the already computed input $\mathcal{H}_{\mathsf{sse}}(\ell)$; otherwise, it computes $\mathcal{H}_{\mathsf{sse}}(\ell)$ for the first time as explained above. If $i = i^\star$ and $\ell = \ell_\rho$, then $\mathcal{B}$ stops simulating the experiment for $\mathcal{A}$ and returns 0 to its own experiment. Otherwise, that means either $i \neq i^\star$, in which case $\mathcal{B}$ knows $t_i \in \mathbb{Z}_p$, or $\ell \neq \ell_\rho$, in which case $\mathcal{B}$ the discrete logarithm of $\mathcal{H}_{\mathsf{sse}}(\ell)$. In both cases, $\mathcal{B}$ can compute $S_{\ell,i} := t_i \cdot \mathcal{H}_{\mathsf{sse}}(\ell) \in \mathbb{G}_1$, which it returns to $\mathcal{A}$.

At the end of the experiment, $\mathcal{B}$ receives the output α from $\mathcal{A}$. If its guess ρ was correct, $\mathcal{B}$ outputs α to its own experiment, otherwise, it ignores α and returns 0.

When $\mathcal{B}$'s guess is incorrect, it returns 0 to its experiment. Otherwise, when it is given as input a real q_r-fold DBDH challenge, that is $s_j = abc_j$ for all indices $j \in [q_r]$, then $\mathcal{B}$ simulates the $\texttt{1-Label-IND}$ security game with $b = 0$. Indeed, since $b = t_{i^\star}$, for the j-th query to $\mathsf{QRealRandom}$, we have:

$$\begin{aligned}
K_{\ell^\star} &= [s_j]_T + e([a]_1, (\sum_{i \neq i^\star} t_i) \cdot [c_j]_2) = [abc_j]_T + e([a]_1, (\sum_{i \neq i^\star} t_i) \cdot [c_j]_2) \\
&= e([a]_1, [bc_j]_2) + e([a]_1, (\sum_{i \neq i^\star} t_i) \cdot [c_j]_2) = e([a]_1, [bc_j]_2 + (\sum_{i \neq i^\star} t_i) \cdot [c_j]_2) \\
&= e([a]_1, (b + \sum_{i \neq i^\star} t_i) \cdot [c_j]_2) = e([a]_1, \sum_i t_i \cdot [c_j]_2) = e(\mathcal{H}_{\mathsf{sse}}(\ell^\star), \sum_i t_i \cdot [c_j]_2)
\end{aligned}$$

where $C_{\ell^\star} = [c_j]_2$. When given as input a random q_r-fold DBDH challenge, the simulation corresponds to the case $b = 1$. Finally, we conclude using the fact that the guess ρ is correct with probability exactly $\frac{1}{q_h+1}$.

6.1.4 Generic construction of IND-secure MCFE

We now show how we can enhance the security of any MCFE using a SSE as defined in Definition 6.1. Namely, we show that the following construction is $\texttt{IND}$ secure if the underlying MCFE is $\texttt{IND}^*$-secure, thereby removing the complete-ciphertext restriction, as incomplete ciphertexts do not leak any information thanks to the SSE layer.

Let $\mathsf{MCFE} = (\mathsf{SetUp}, \mathsf{Encrypt}, \mathsf{DKeyGen}, \mathsf{FDecrypt}, \mathsf{Decrypt})$ be a MCFE scheme, $\mathsf{SSE} = (\mathsf{SSE.SetUp}, \mathsf{SSE.Encaps}, \mathsf{SSE.Decaps})$ be a SSE scheme, and $\mathsf{SKE} = (\mathsf{SEnc}, \mathsf{SDec})$ be a Symmetric Key Encryption scheme (Definition 2.5) with same key space

as SSE, and whose message space is the ciphertext space of MCFE. We define $\mathsf{MCFE}' = (\mathsf{SetUp}', \mathsf{Encrypt}', \mathsf{DKeyGen}', \mathsf{FDecrypt}', \mathsf{Decrypt}')$ as follows:

- $\mathsf{SetUp}'(\lambda)$: it executes $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ and $(\mathsf{pk}_{\mathsf{sse}}, (\mathsf{ek}_{\mathsf{sse},i})_i) \leftarrow \mathsf{SSE.SetUp}(\lambda)$. The public parameters mpk' consist of $\mathsf{mpk} \cup \mathsf{pk}_{\mathsf{sse}}$, while the encryption keys are $\mathsf{ek}'_i = \mathsf{ek}_i \cup \mathsf{ek}_{\mathsf{sse},i}$ for $i = 1, \ldots, n$, and the master secret key is $\mathsf{msk}' = \mathsf{msk}$.

- $\mathsf{Encrypt}'(\mathsf{ek}'_i, x_i, \ell)$: it parses the encryption key ek'_i as $\mathsf{ek}_i \cup \mathsf{ek}_{\mathsf{sse},i}$, runs $C_{\ell,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x_i, \ell)$, executes $(D_{\ell,i}, K_{\ell,i}) \leftarrow \mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell)$, and $S_{\ell,i} \leftarrow \mathsf{SSE.Share}(\mathsf{ek}_{\mathsf{sse},i}, \ell)$. The ciphertext $C'_{\ell,i}$ is then set to the tuple $(E_{\ell,i} = \mathsf{SEnc}(K_{\ell,i}, C_{\ell,i}), D_{\ell,i}, S_{\ell,i})$.

- $\mathsf{DKeyGen}'(\mathsf{msk}', f)$: with $\mathsf{msk} = \mathsf{msk}'$, it runs $\mathsf{dk}_f = \mathsf{DKeyGen}(\mathsf{msk}, f)$.

- $\mathsf{FDecrypt}'(\mathsf{dk}_f, \ell, (C'_{\ell,i})_{i\in[n]})$: takes as input a functional decryption key dk_f, a label ℓ, and ciphertexts $(C'_{\ell,i} = (E_{\ell,i}, D_{\ell,i}, S_{\ell,i}))_{i\in[n]}$. It operates in two steps; first it applies $\mathsf{SSE.Decaps}_{\mathsf{sse}}(\mathsf{pk}_{\mathsf{sse}}, (S_{\ell,j})_{j\in[n]}, \ell, D_{\ell,i})$ on all the ciphertexts $D_{\ell,i}$ to get all the encapsulation keys $K_{\ell,i}$'s and thus all the plaintexts $C_{\ell,i}$'s using SDec on $E_{\ell,i}$. Then it runs $\mathsf{FDecrypt}(\mathsf{dk}_f, \ell, (C_{\ell,i})_{i\in[n]})$.

- $\mathsf{Decrypt}'(\mathsf{ek}_i, \ell, (C'_{\ell,i})_{i\in[n]})$: takes as input an encryption key ek_i, a label ℓ, and ciphertexts $(C'_{\ell,i} = (E_{\ell,i}, D_{\ell,i}, S_{\ell,i}))_{i\in[n]}$. It operates in two steps; first it applies $\mathsf{SSE.Decaps}_{\mathsf{sse}}(\mathsf{pk}_{\mathsf{sse}}, (S_{\ell,j})_{j\in[n]}, \ell, D_{\ell,i})$ on all the ciphertexts $D_{\ell,i}$ to get all the encapsulation keys $K_{\ell,i}$'s and thus all the plaintexts $C_{\ell,i}$'s using SDec on $E_{\ell,i}$. Then it runs $\mathsf{Decrypt}(\mathsf{ek}_i, \ell, (C_{\ell,i})_{i\in[n]})$.

Remark : Because the DSum protocol (described in Section 5.1), the SSE (described in Section 6.1) and the original MCFE from [CDG+18a] (recalled in Section 4.2.3) all have a decentralized setup, the resulting scheme also enjoys the same property, thereby completely getting rid of the need for a trusted party generating the private keys.

6.1.5 Security analysis

We show that this generic construction MCFE' achieves IND-security, assuming the underlying MCFE is $\mathtt{IND}^*$-secure (see Definition 3.2), SSE is 1-Label-IND-secure (see Definition 6.2), and the symmetric encryption is one-time secure (see definition in Definition 2.6). More precisely, we can state the following security result:

Theorem 6.2

For any adversary $\mathcal{A}$ running within time t, against the IND-security of the above MCFE',

$$\mathsf{Adv}^{\mathtt{IND}}_{\mathsf{MCFE}'}(\mathcal{A}) \leq (n+1) \cdot L \times \begin{pmatrix} \mathsf{Adv}^{\mathtt{IND}^*}_{\mathsf{MCFE}}(t) + 2 \cdot \mathsf{Adv}^{\mathtt{1\text{-}Label\text{-}IND}}_{\mathsf{SSE}}(t') \\ +q_e \cdot \mathsf{Adv}^{\mathsf{OT}}_{\mathsf{SKE}}(t'') \end{pmatrix},$$

with t' and t'' quite close to t, where L is the total number of labels queried to the oracle $\mathsf{QLeftRight}'$, and q_e is the maximum number of queries to $\mathsf{QLeftRight}'$

for a given label. In addition $\mathsf{Adv}(t)$, for any security notion, is the maximum advantage an algorithm can get within time t.

We stress that this security result keeps all the properties of the basic MCFE and the SSE schemes:

- if MCFE and SSE are both secure against adaptive corruptions, MCFE' is also IND against adaptive corruptions;
- if MCFE is secure with repetitions (see Section 6.2), MCFE' is also IND with repetitions.

The proof uses a hybrid argument that goes over all the labels $\ell_1, \ldots, \ell_L$ used as input to the queries $\mathcal{A}$ makes to the $\mathsf{QLeftRight}'$ oracle.

Proof

We define the following hybrid games, for all $\rho = 0, \ldots, L$:

Game $\mathbf{G}_\rho$: This hybrid game outputs right answers for the $\mathsf{QLeftRight}'$-queries involving the first ρ labels, and left answers for the other labels, to the IND-adversary $\mathcal{A}$, as follows:

- Initialize: it gets the global parameters $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_{i\in[n]}) \leftarrow \mathsf{SetUp}(\lambda)$, $(\mathsf{pk}_{\mathsf{sse}}, (\mathsf{ek}_{\mathsf{sse},i})_{i\in[n]}) \leftarrow \mathsf{SSE.SetUp}(\lambda)$ and it returns the public ones $\mathsf{mpk}' = \mathsf{mpk} \cup \mathsf{pk}_{\mathsf{sse}}$ to the adversary $\mathcal{A}$.
- $\mathsf{QEncrypt}'(i, x, \ell_j)$: it returns $\mathsf{Encrypt}'(\mathsf{ek}_i, x, \ell_j)$.
- $\mathsf{QLeftRight}'(i, x^0, x^1, \ell_j)$: if $j \leq \rho$, it returns $\mathsf{Encrypt}'(\mathsf{ek}_i, x^1, \ell_j)$, if $j > \rho$, it returns $\mathsf{Encrypt}'(\mathsf{ek}_i, x^0, \ell_j)$.
- $\mathsf{QDKeyGen}'(f)$: it returns $\mathsf{DKeyGen}'(\mathsf{msk}, f)$.
- $\mathsf{QCorrupt}'(i)$: it returns $\mathsf{ek}'_i = \mathsf{ek}_i \cup \mathsf{ek}_{\mathsf{sse},i}$;
- Finalize: as in Definition 3.2, for IND-security.

For any hybrid game $\mathbf{G}_\rho$, we denote by $\mathsf{Adv}_{\mathbf{G}_\rho}(\mathcal{A}) := \Pr[\beta = 1]$, where β is the output of Finalize. Note that $\mathsf{Adv}^{\mathtt{IND}}_{\mathsf{MCFE}'}(\mathcal{A}) = |\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_L}(\mathcal{A})|$. Lemma 6.1 states that for all $i \in [L]$, $|\mathsf{Adv}_{\mathbf{G}_{i-1}}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_i}(\mathcal{A})|$ is negligible, which concludes the proof.

Lemma 6.1

For any adversary $\mathcal{A}$ against the IND-security of the above MCFE', for all $\rho \in [L]$, there exist PPT adversaries $\mathcal{B}_\rho$, $\mathcal{B}'_\rho$, and $\mathcal{B}''_\rho$ such that :

$$|\mathsf{Adv}_{\mathbf{G}_{\rho-1}}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_\rho}(\mathcal{A})| \leq (n+1) \cdot \left(\begin{array}{l} \mathsf{Adv}^{\mathtt{IND}^*}_{\mathsf{MCFE}}(\mathcal{B}_\rho) + \\ 2 \cdot \mathsf{Adv}^{\mathtt{1\text{-}Label\text{-}IND}}_{\mathsf{SSE}}(\mathcal{B}'_\rho) + q_e \cdot \mathsf{Adv}^{\mathsf{OT}}_{\mathsf{SKE}}(\mathcal{B}''_\rho) \end{array} \right)$$

Proof

Actually, two cases can happen between games $\mathbf{G}_{\rho-1}$ and $\mathbf{G}_\rho$, for each $\rho \in \{1,\ldots,L\}$: either all the honest components of the ciphertext are generated under ℓ_ρ or not all of them. We first make the guess, and then deal with the two cases: if they are all generated (for honest clients), this is the simple $\mathtt{IND}^*$ security game for the underlying MCFE, otherwise there is an honest index i^* for which the ciphertext has not been generated, and the SSE scheme will help, together with the symmetric encryption scheme:

Guess of the case for the ℓ_ρ: We define a new sequence of hybrid games $\mathbf{G}^*_\rho$ for all $\rho = 0,\ldots,n$, which is exactly as above, except that a guess for the missing honest-client ciphertext i^* under ℓ_ρ is performed ($i^* = 0$ means that all the honest-client ciphertexts are expected to be generated under ℓ_ρ):

- Initialize: it first makes a guess for $i^* \xleftarrow{\$} \{0,\ldots,n\}$, and then does as in $\mathbf{G}_\rho$.
- $\mathsf{QEncrypt}'(i,x,\ell_j)$, $\mathsf{QLeftRight}'(i,x^0,x^1,\ell_j)$, $\mathsf{QDKeyGen}'(f)$, $\mathsf{QCorrupt}'(i)$, as in $\mathbf{G}_\rho$.
- Finalize: as in $\mathbf{G}_\rho$, except if
  - $i^* = 0$, but not all the honest ciphertexts under ℓ_ρ have been asked.
  - $i^* \neq 0$, but client i^* is corrupted.
  - $i^* \neq 0$, but the i^*-th client ciphertext has been asked under ℓ_ρ.

  in which cases a random bit is output.

Since $\mathbf{G}^*_\rho$ and $\mathbf{G}_\rho$ are the same when the guess incorrect, which happens with probability exactly $1/(n+1)$, for any adversary $\mathcal{A}$: $\mathsf{Adv}_{\mathbf{G}_\rho}(\mathcal{A}) = (n+1)\cdot\mathsf{Adv}_{\mathbf{G}^*_\rho}(\mathcal{A})$.

All the ciphertexts are generated under ℓ_ρ: Under the condition that $\mathcal{A}$ asks for all the honest ciphertexts under ℓ_ρ, which means the correct guess is $i^* = 0$, we build a PPT adversary $\mathcal{B}_\rho$ against the $\mathtt{IND}^*$ security of MCFE such that $|\mathsf{Adv}_{\mathbf{G}^*_{\rho-1}}(\mathcal{A} \wedge i^* = 0) - \mathsf{Adv}_{\mathbf{G}^*_\rho}(\mathcal{A} \wedge i^* = 0)| \leq \mathsf{Adv}^{\mathtt{IND}^*}_{\mathsf{MCFE}}(\mathcal{B}_\rho)$. $\mathcal{B}_\rho$ simulates the $\mathtt{IND}$-adversary $\mathcal{A}$'s view as follows:

- Initialize: it obtains mpk from its own $\mathtt{IND}^*$-security game for MCFE, samples $(\mathsf{pk}_{\mathsf{sse}}, (\mathsf{ek}_{\mathsf{sse},i})_{i\in[n]}) \leftarrow \mathsf{SSE.SetUp}(\lambda)$ and returns $\mathsf{mpk}' = \mathsf{mpk} \cup \mathsf{pk}_{\mathsf{sse}}$ to the adversary $\mathcal{A}$.
- $\mathsf{QEncrypt}'(i,x,\ell_j)$: it uses its own encryption oracle $\mathsf{QEncrypt}$ to get $C_{\ell_j,i} \leftarrow \mathsf{QEncrypt}(i,x,\ell_j)$. Then, it computes $(D_{\ell_j,i}, K_{\ell_j,i}) \leftarrow \mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell_j)$, and $S_{\ell_j,i} \leftarrow \mathsf{SSE.Share}(\mathsf{ek}_{\mathsf{sse},i}, \ell_j)$. Eventually, it computes and returns the ciphertext $(E_{\ell_j,i} = \mathsf{SEnc}(K_{\ell_j,i}, C_{\ell_j,i}), D_{\ell_j}, S_{\ell_j,i})$.
- $\mathsf{QLeftRight}'(i,x^0,x^1,\ell_j)$:
  - if $j < \rho$, it uses its own encryption oracle $\mathsf{QEncrypt}$ to get the ciphertext $C_{\ell_j,i} \leftarrow \mathsf{QEncrypt}(i,x^1,\ell_j)$.

– if $j > \rho$, it uses its own encryption oracle $\mathsf{QEncrypt}$ to get the ciphertext $C_{\ell_j,i} \leftarrow \mathsf{QEncrypt}(i, x^0, \ell_j)$.

– if $j = \rho$, then it uses its own left-or-right encryption oracle to get the ciphertext $C_{\ell_j,i} \leftarrow \mathsf{QLeftRight}(i, x^0, x^1, \ell_\rho)$.

Then, it computes the encapsulation $(D_{\ell_j,i}, K_{\ell_j,i}) \leftarrow \mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell_j)$ and the share $S_{\ell_j,i} \leftarrow \mathsf{SSE.Share}(\mathsf{ek}_{\mathsf{sse},i}, \ell_j)$. Eventually, it returns the ciphertext $(E_{\ell_j,i} = \mathsf{SEnc}(K_{\ell_j,i}, C_{\ell_j,i}), D_{\ell_j,i}, S_{\ell_j,i})$.

- $\mathsf{QCorrupt}'(i)$: it uses its own corruption oracle to get $\mathsf{ek}_i \leftarrow \mathsf{QCorrupt}(i)$, and returns $\mathsf{ek}'_i = \mathsf{ek}_i \cup \mathsf{ek}_{\mathsf{sse},i}$.

- Finalize: $\mathcal{B}_\rho$ checks whether all the honest ciphertexts under ℓ_ρ have been asked. If not, it ignores $\mathcal{A}$'s guess and sends a uniformly random bit $\beta \xleftarrow{\$} \{0,1\}$. Otherwise, it forwards $\mathcal{A}$'s guess to the Finalize procedure of the $\mathtt{IND}^*$-security game.

When the guess $i^* = 0$ is correct, the queries $\mathcal{B}_\rho$ makes to its $\mathsf{QLeftRight}$ oracle are valid, i.e. they don't make the Finalize procedure output a uniformly random bit (independent of $\mathcal{B}_\rho$'s guess). Indeed, if $\mathsf{QLeftRight}(i, \cdot, \cdot, \ell_\rho)$ is queried for some $i \in [n]$, then for all slots $j \in \mathcal{HS}$, $\mathsf{QLeftRight}(j, \cdot, \cdot, \ell_\rho)$ is also queried. Thus, we can use the $\mathtt{IND}^*$ security of MCFE to switch $\mathsf{Encrypt}'(\mathsf{ek}_i, \vec{x}^0, \ell_\rho)$ as in game $\mathbf{G}^*_{\rho-1}$ to $\mathsf{Encrypt}'(\mathsf{ek}_i, \vec{x}^1, \ell_\rho)$ as in game $\mathbf{G}^*_\rho$.

Some ciphertexts are missing under ℓ_ρ: For $\beta \in \{0,1\}$, we define the game $\mathbf{H}_{\rho,\beta}$ as $\mathbf{G}^*_\rho$, except that when $i^* \neq 0$, $\mathsf{QEncrypt}'(i, x, \ell_\rho)$ encrypts x and $\mathsf{QLeftRight}'(i, x^0, x^1, \ell_\rho)$ encrypts x^β in $C_{\ell_\rho,i}$, then they both generate the encapsulation $(D_{\ell_\rho,i}, K_{\ell_\rho,i}) \leftarrow \mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell_\rho)$ and the share $S_{\ell_\rho,i} \leftarrow \mathsf{SSE.Share}(\mathsf{ek}_{\mathsf{sse},i}, \ell_\rho)$, sample a fresh key $K'_{\ell_\rho,i} \xleftarrow{\$} \mathcal{K}$ at random in the key space, and return the ciphertext $(E_{\ell_\rho,i} = \mathsf{SEnc}(K'_{\ell_\rho,i}, C_{\ell_\rho,i}), D_{\ell_\rho,i}, S_{\ell_\rho,i})$.

Now, we build PPT adversaries $\mathcal{B}_{\rho,0}$ and $\mathcal{B}_{\rho,1}$ against the $\mathtt{1\text{-}Label\text{-}IND}$-security of the SSE such that

$$|\mathsf{Adv}_{\mathbf{G}^*_{\rho-1}}(\mathcal{A} \wedge i^* \neq 0) - \mathsf{Adv}_{\mathbf{H}_{\rho,0}}(\mathcal{A} \wedge i^* \neq 0)| \leq \mathsf{Adv}^{\mathtt{1\text{-}Label\text{-}IND}}_{\mathsf{SSE}}(\mathcal{B}_{\rho,0})$$
$$\text{and } |\mathsf{Adv}_{\mathbf{G}^*_{\rho}}(\mathcal{A} \wedge i^* \neq 0) - \mathsf{Adv}_{\mathbf{H}_{\rho,1}}(\mathcal{A} \wedge i^* \neq 0)| \leq \mathsf{Adv}^{\mathtt{1\text{-}Label\text{-}IND}}_{\mathsf{SSE}}(\mathcal{B}_{\rho,1}).$$

Let $\beta \in \{0,1\}$. We proceed to describe $\mathcal{B}_{\rho,\beta}$. First, $\mathcal{B}_{\rho,\beta}$ samples the guess $i^* \xleftarrow{\$} \{0, \ldots, n\}$. If $i^* = 0$, then $\mathcal{B}_{\rho,\beta}$ behaves exactly as the challenger in the game $\mathbf{G}^*_{\rho-1+\beta}$. Otherwise, it does the following, using the $\mathtt{1\text{-}Label\text{-}IND}$-security game against SSE:

- Initialize: it generates $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_{i\in[n]}) \leftarrow \mathsf{SetUp}(\lambda)$, and sends i^* to receive $\mathsf{pk}_{\mathsf{sse}}$ from its own $\mathtt{1\text{-}Label\text{-}IND}$ challenger for SSE. It returns $\mathsf{mpk}' = \mathsf{mpk} \cup \mathsf{pk}_{\mathsf{sse}}$ to the adversary $\mathcal{A}$.

- $\mathsf{QEncrypt}'(i, x, \ell_j)$: it can compute $C_{\ell_j,i} \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, x, \ell_j)$. Then, it call its own oracle to get $S_{\ell_j,i} \leftarrow \mathsf{QShare}(i, \ell_j)$. If $j \neq \rho$, it computes $(D_{\ell_j,i}, K_{\ell_j,i}) \leftarrow \mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell_j)$; if $j = \rho$, it calls $(D_{\ell_\rho,i}, K_{\ell_\rho,i}) \leftarrow \mathsf{QRealRandom}(\ell_\rho)$. Eventually, it returns the ciphertext

$$E_{\ell_j,i} = (\mathsf{SEnc}(K_{\ell_j,i}, C_{\ell_j,i}), D_{\ell_j,i}, S_{\ell_j,i})$$

- $\mathsf{QLeftRight}'(i, x^0, x^1, \ell_j)$: if $j < \rho$, it computes $C_{\ell_j,i} = \mathsf{Encrypt}(\mathsf{ek}_i, x^1, \ell_j)$; if $j > \rho$, it computes $C_{\ell_j,i} = \mathsf{Encrypt}(\mathsf{ek}_i, x^0, \ell_j)$; and if $j = \rho$, it computes $C_{\ell_j,i} = \mathsf{Encrypt}(\mathsf{ek}_i, x^\beta, \ell_j)$. Then it calls its own oracle to get $S_{\ell_j,i} = \mathsf{QShare}(i, \ell_j)$. If $j \neq \rho$, it computes $(D_{\ell_j,i}, K_{\ell_j,i}) \leftarrow \mathsf{SSE.Encaps}(\mathsf{pk}_{\mathsf{sse}}, \ell_j)$; if $j = \rho$, it calls $(D_{\ell_\rho,i}, K_{\ell_\rho,i}) \leftarrow \mathsf{QRealRandom}(\ell_\rho)$. Eventually, it returns the ciphertext $(E_{\ell_j,i} = \mathsf{SEnc}(K_{\ell_j,i}, C_{\ell_j,i}), D_{\ell_j,i}, S_{\ell_j,i})$.

- $\mathsf{QDKeyGen}'(f)$: it runs and returns $\mathsf{DKeyGen}(\mathsf{msk}, f)$.

- $\mathsf{QCorrupt}'(i)$: it uses its own corruption oracle to get $\mathsf{ek}_{\mathsf{sse},i} \leftarrow \mathsf{QCorrupt}(i)$, and returns $\mathsf{ek}'_i = \mathsf{ek}_i \cup \mathsf{ek}_{\mathsf{sse},i}$.

- Finalize: $\mathcal{B}_{\rho,\beta}$ checks whether the ciphertext for the i^*-th client has been asked under ℓ_ρ, or corrupted. If so, it ignores $\mathcal{A}$'s guess and sends a uniformly random bit $\beta \xleftarrow{\$} \{0,1\}$. Otherwise, it forwards $\mathcal{A}$'s guess to the Finalize procedure of the $\mathtt{IND}^*$-security game.

Game $\mathbf{G}_\rho$, which encrypts x^1 under ℓ_ρ, just differs from $\mathbf{H}_{\rho,1}$ with real vs. random keys K_{ℓ_ρ}, as emulated by $\mathcal{B}_{\rho,1}$, according to the real-or-random behavior of the $\mathtt{1\text{-}Label\text{-}IND}$ game for SSE. Game $\mathbf{G}_{\rho-1}$, which encrypts x^0 under ℓ_ρ, just differs from $\mathbf{H}_{\rho,0}$ with real vs. random keys K_{ℓ_ρ}, as emulated by $\mathcal{B}_{\rho,0}$, according to the real-or-random behavior of the $\mathtt{1\text{-}Label\text{-}IND}$ game for SSE. Note that if adversary $\mathcal{A}$ makes queries that satisfy the conditions required by the Finalize procedure from the game $\mathtt{IND}^*$ of MCFE', and that the guess $i^* \neq 0$ is correct, then the queries of $\mathcal{B}_{\rho,\beta}$ satisfy the conditions required by the $\mathtt{1\text{-}Label\text{-}IND}$ security game for SSE, namely, $\mathsf{QRealRandom}$ is only queried on one label ℓ_ρ, $\mathsf{QCorrupt}$ is never queried on i^*, and QShare is never queried on (i^*, ℓ_ρ).

Note that in the case the guess $i^* \neq 0$ is correct, in the $\mathtt{IND}$-security game, the adversary can ask $\mathsf{QLeftRight}$ queries on users $i \in \mathcal{CS}$ for the label ℓ_ρ, which was not allowed in the original $\mathtt{IND}^*$-security game (and would lead to a random output). The reason is that security here relies on the $\mathtt{1\text{-}Label\text{-}IND}$ security of the SSE for the label ℓ_ρ. In the simulation, no matter what is encrypted in $C_{\ell_\rho,i}$ under the label ℓ_ρ, the $\mathsf{QRealRandom}$ algorithm provides randomness and makes the ciphertexts $C_{\ell_\rho,i}$ impossible to recover.

Since the encapsulation keys K_{ℓ_ρ} are uniformly random in games $\mathbf{H}_{\rho,0}$ and $\mathbf{H}_{\rho,1}$, we can use the one-time security of SKE, for each ciphertext for the label ℓ_ρ, to obtain a PPT adversary $\mathcal{B}''_\rho$ such that:

$$|\mathsf{Adv}_{\mathbf{H}_{\rho,0}}(\mathcal{A} \wedge i^* \neq 0) - \mathsf{Adv}_{\mathbf{H}_{\rho,1}}(\mathcal{A} \wedge i^* \neq 0)| \leq q_e \cdot \mathsf{Adv}^{\mathsf{OT}}_{\mathsf{SKE}}(\mathcal{B}''_\rho),$$

where q_e denotes maximum number of ciphertexts generated by the $\mathsf{QLeftRight}$ oracle for a given label.

Putting everything together, for the case $i^* \neq 0$, we obtain PPT adversaries $\mathcal{B}'_\rho$ and $\mathcal{B}''_\rho$ such that: $|\mathsf{Adv}_{\mathbf{G}^*_{\rho-1}}(\mathcal{A} \wedge i^* \neq 0) - \mathsf{Adv}_{\mathbf{G}^*_\rho}(\mathcal{A} \wedge i^* \neq 0)|$ is upper-bounded by $2 \cdot \mathsf{Adv}^{\mathtt{1\text{-}Label\text{-}IND}}_{\mathsf{SSE}}(\mathcal{B}'_\rho) + q_e \cdot \mathsf{Adv}^{\mathsf{OT}}_{\mathsf{SKE}}(\mathcal{B}''_\rho))$. Since for any game $\mathbf{G}$ and any adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathbf{G}}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}}(\mathcal{A} \wedge i^* = 0) + \mathsf{Adv}_{\mathbf{G}}(\mathcal{A} \wedge i^* \neq 0)$, this concludes the proof of Lemma 6.1.

6.2 Bridge between MIFE and MCFE

In this section, we add a layer of IP-FE on top of the IP-MCFE from Section 4.2.3, to remove the restriction of having a unique challenge ciphertext per client and per label. This way, our IP-MCFE really captures the MIFE formalism on one label (see Section 3.1.1) in the case of the inner product functionality.

Overview

Our construction works for any IP-FE that is compatible with the IP-MCFE from Section 4.2.3, namely, an IP-FE whose message space is the ciphertext space of the IP-MCFE.

For correctness, we exploit the fact that decryption of the IP-MCFE computes the inner product of the ciphertext together with the decryption keys. Also, we make use of the fact that the encryption algorithm can act on vectors of group elements, in $\mathbb{G}^m$, where $\mathbb{G}$ is a prime-order group, as opposed to vectors over $\mathbb{Z}$. Decryption recovers the inner product in the group $\mathbb{G}$, without any restriction on the size of the input of the encryption and decryption key generation algorithms. Namely, the message space of IP-FE is $\mathbb{G}^m$, for some dimension m, its decryption key space is $\mathbb{Z}_p^m$, where p is the order of $\mathbb{G}$, and for any $[\vec{x}] \in \mathbb{G}^m$, $\vec{y} \in \mathbb{Z}_p^m$, $\mathsf{IP.Dec}(\mathsf{ct}, \mathsf{dk}_{\vec{y}}) = [\vec{x}^\top \vec{y}]$ with probability one, where $\mathsf{ct} \leftarrow \mathsf{IP.Encrypt}(\mathsf{IP.msk}, [\vec{x}])$, $\mathsf{dk}_{\vec{y}} \leftarrow \mathsf{IP.DKeyGen}(\mathsf{IP.msk}, \vec{y})$, and $(\mathsf{IP.mpk}, \mathsf{IP.msk}) \leftarrow \mathsf{IP.SetUp}(\lambda)$

For security, we exploit the fact that this IP-MCFE is linearly homomorphic, in the sense that given an input $\vec{x}$, one can publicly maul an encryption of $\vec{x}'$ into an encryption of $\vec{x} + \vec{x}'$. This is used to bootstrap the security from one to many challenge ciphertexts per (user,label) pair, similarly to [AGRW17, ACF+18] in the context of multi-input IP-FE. In fact, [ACF+18] uses a one-time secure MIFE as inner layer, and a single-input IP-FE as outer layer, while we use an IP-MCFE as inner layer, and an IP-FE as outer layer. The main technical challenge is to handle the case of (adaptive) corruptions, which are not considered in [AGRW17, ACF+18] (even in the static case where corruptions are known beforehand).

Remark. Notice finally that using the IP-FE from [ALS16], the IP-MCFE from Section 4.2.3, and adding the SSE scheme from Section 6.1.2, one gets an IP-MCFE that is IND-secure, with repetitions and with adaptive corruptions.

6.2.1 Extension of IP-MCFE to vectors

We first extend the IP-MCFE from Section 4.2.3 to handle vectors as inputs of the encryption algorithm for each client, instead of just scalars.

- $\mathsf{SetUp}(\lambda)$: samples $\mathcal{G} := (\mathbb{G}, p, P) \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, a full-domain hash function $\mathcal{H}$ onto $\mathbb{G}^2$, $\mathbf{S}_i \xleftarrow{\$} \mathbb{Z}_p^{m \times 2}$, for $i = 1, \ldots, n$. Returns the public key $\mathsf{mpk} := (\mathcal{G}, \mathcal{H})$, encryption keys $\mathsf{ek}_i = \mathbf{S}_i$ for $i = 1, \ldots, n$, and the master secret key $\mathsf{msk} = ((\mathbf{S}_i)_i)$, (in addition to mpk, which is omitted).

- $\mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i, \ell)$: Takes as input the value $\vec{x}_i \in \mathbb{Z}_p^m$ to encrypt, under the key $\mathsf{ek}_i = \mathbf{S}_i$ and the label ℓ. It computes $[\vec{u}_\ell] := \mathcal{H}(\ell) \in \mathbb{G}^2$, and outputs the ciphertext $[\vec{c}_i] = [\mathbf{S}_i \vec{u}_\ell + \vec{x}_i] \in \mathbb{G}^m$.

- $\mathsf{DKeyGen}(\mathsf{msk}, \vec{y})$: Takes as input $\mathsf{msk} = (\mathbf{S}_i)_i$ and an inner product function defined by $\vec{y} \in \mathbb{Z}_p^{m \cdot n}$ as $f_{\vec{y}}(\vec{x}) = \langle \vec{x}, \vec{y} \rangle$,where $\vec{x} = (\vec{x}_1 \| \cdots \| \vec{x}_n) \in \mathbb{Z}_p^{nm}$, and outputs the functional decryption key $\mathsf{dk}_{\vec{y}} = \left(\vec{y}, \sum_i \mathbf{S}_i^\top \vec{y}_i\right) \in \mathbb{Z}_p^{mn} \times \mathbb{Z}_p^2$.

- $\mathsf{FDecrypt}(\mathsf{dk}_{\vec{y}}, \ell, ([\vec{c}_i])_{i \in [n]})$: Takes as input a functional decryption key $\mathsf{dk}_{\vec{y}} = (\vec{y}, \vec{d})$, a label ℓ, and ciphertexts. It computes $[\vec{u}_\ell] := \mathcal{H}(\ell)$ and returns $[\alpha] = \sum_i [\vec{c}_i]^\top \vec{y}_i - [\vec{u}_\ell]^\top \vec{d}$.

- $\mathsf{Decrypt}(\mathsf{ek}_i, \ell, ([\vec{c}_i])_{i \in [n]})$: Takes as input the encryption key $\mathsf{ek}_i = \mathbf{S}_i$, a label ℓ, and ciphertexts. It computes $[\vec{u}_\ell] := \mathcal{H}(\ell)$ and returns $[\vec{x}_{\ell,i}] = [\vec{c}_i] - [\mathbf{S}_i \vec{u}_\ell]$.

Correctness. One can check that:

$$\begin{aligned}[\alpha] &= \sum_i [\vec{c}_i]^\top \vec{y}_i - [\vec{u}_\ell]^\top \vec{d} = \sum_i [\mathbf{S}_i \vec{u}_\ell + \vec{x}_i]^\top \vec{y}_i - [\vec{u}_\ell]^\top \sum_i \mathbf{S}_i^\top \vec{y}_i \\ &= \sum_i [\mathbf{S}_i \vec{u}_\ell]^\top \vec{y}_i + [\vec{x}_i]^\top \vec{y}_i - \sum_i [\mathbf{S}_i \vec{u}_\ell]^\top \vec{y}_i = \sum_i [\vec{x}_i]^\top \vec{y}_i = [\vec{x}^\top \vec{y}] = [\langle \vec{x}, \vec{y} \rangle].\end{aligned}$$

For security, we will use the two following properties of the IP-MCFE from Section 4.2.3:

- *Linear Homomorphism of ciphertexts*: for any $i \in [n]$, $\vec{x}_i, \vec{x}'_i \in \mathbb{Z}_p$, and any label ℓ, we have $[\vec{c}_i] + [\vec{x}'_i] = \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i + \vec{x}'_i, \ell)$, where $[\vec{c}_i] = \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i, \ell)$.

- *Deterministic Encryption*: in particular, together with the linear homomorphism of ciphertexts, this implies that for any $\vec{x}_i, \vec{x}'_i \in \mathbb{Z}_p^m$ and any label ℓ, we have: $\mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i, \ell) - \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}'_i, \ell) = [\vec{x}_i - \vec{x}'_i]$.

6.2.2 Construction of IND-secure IP-MCFE with repetitions

We now give our generic construction to obtain security with repetitions. Let $\mathsf{MCFE} = (\mathsf{SetUp}, \mathsf{Encrypt}, \mathsf{DKeyGen}, \mathsf{FDecrypt}, \mathsf{Decrypt})$ be the above IP-MCFE scheme and $\mathsf{IP\text{-}FE} = (\mathsf{IP.SetUp}, \mathsf{IP.Encrypt}, \mathsf{IP.DKeyGen}, \mathsf{IP.Dec})$ be a single-input IP-FE (as defined in Definition 2.15) whose message space is the ciphertext space of MCFE. We define a new $\mathsf{MCFE}' = (\mathsf{SetUp}', \mathsf{Encrypt}', \mathsf{DKeyGen}', \mathsf{Decrypt}')$ as follows:

- $\mathsf{SetUp}'(\lambda)$: it executes $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ as well as, for $i = 1, \ldots, n$, $(\mathsf{IP.mpk}_i, \mathsf{IP.msk}_i) \leftarrow \mathsf{IP.SetUp}(\lambda)$. The encryption keys are $\mathsf{ek}'_i = (\mathsf{ek}_i, \mathsf{IP.msk}_i)$ for all $i = 1, \ldots, n$, the public key is $\mathsf{mpk}' := (\mathsf{mpk}, \{\mathsf{IP.mpk}_i\}_i)$, and the master secret key is $\mathsf{msk}' = (\mathsf{msk}, \{\mathsf{IP.msk}_i\}_i)$.

- $\mathsf{Encrypt}'(\mathsf{ek}'_i, \vec{x}_i, \ell)$: it parses the encryption key ek'_i as $(\mathsf{ek}_i, \mathsf{IP.msk}_i)$, runs $[\vec{c}_{i,\ell}] \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i, \ell)$, and returns $C'_{\ell,i} := \mathsf{IP.Encrypt}(\mathsf{IP.msk}_i, [\vec{c}_{i,\ell}])$.

- $\mathsf{DKeyGen}'(\mathsf{msk}', \vec{y})$: on input $\vec{y} := (\vec{y}_1 \| \cdots \| \vec{y}_n) \in \mathbb{Z}_p^{nm}$, it computes $\mathsf{dk}_{\vec{y}} = \mathsf{DKeyGen}(\mathsf{msk}, \vec{y})$, and for all $i \in [n]$: $\mathsf{dk}_{\vec{y}_i} = \mathsf{IP.DKeyGen}(\mathsf{IP.msk}_i, \vec{y}_i)$. It returns $\mathsf{dk}'_{\vec{y}} = \left(\mathsf{dk}_{\vec{y}}, \{\mathsf{dk}_{\vec{y}_i}\}_{i \in [n]}\right)$.

The three above algorithms are enough to show the security (as proven below), which holds with respect to any IP-MCFE that satisfies the Linearly Homomorphism of ciphertexts, and deterministic encryption, as defined above. However, correctness only holds for the particular IP-MCFE from Section 4.2.3, where decryption computes

the inner product between ciphertexts and decryption keys. That prevents from a generic transformation. We now prove correctness when using the IP-MCFE from Section 4.2.3 in MCFE':

- $\mathsf{FDecrypt}'(\mathsf{dk}'_{\vec{y}}, \ell, (C'_{\ell,i})_{i\in[n]})$: takes as input a functional decryption key $\mathsf{dk}'_{\vec{y}} = \left(\mathsf{dk}_{\vec{y}}, \{\mathsf{dk}_{\vec{y}_i}\}_{i\in[n]}\right)$, where $\mathsf{dk}_{\vec{y}} = (\vec{y}, \vec{d} = \sum_i \mathbf{S}_i^\top \vec{y}_i)$, a label ℓ, and ciphertexts $(C'_{\ell,i})_{i\in[n]}$. First, it computes $[d_{i,\ell}] = \mathsf{IP.Dec}(\mathsf{dk}_{\vec{y}_i}, C'_{\ell,i})$ for $i = 1 \cdots n$. Then it computes $[\vec{u}_\ell] = \mathcal{H}(\ell)$, and computes $[\alpha] = [\sum_i d_{i,\ell}] - \vec{d}^\top[\vec{u}_\ell]$. Finally, it returns the discrete logarithm $\alpha \in \mathbb{Z}_p$.
- $\mathsf{Decrypt}'(\mathsf{ek}'_i, \ell, (C'_{\ell,i})_{i\in[n]})$: takes as input an encryption key $\mathsf{ek}'_i = (\mathsf{ek}_i, \mathsf{IP.msk}_i)$, a label ℓ, and ciphertexts $(C'_{\ell,i})_{i\in[n]}$. For $k \in [m]$, computes $\mathsf{dk}_k = \mathsf{IP.DKeyGen}(\mathsf{IP.msk}_i, (\delta_{k,j})_j)$, then $[c_i] = \mathsf{IP.Dec}(\mathsf{dk}_k, C'_{\ell,i})$, and finally $\vec{x}_i = \mathsf{Decrypt}(\mathsf{ek}_i, \ell, [\vec{c}_i])$.

Correctness. By correctness of the IP-FE, we have for all $i \in [n]$, and any label ℓ: $[d_{i,\ell}] = [\langle \vec{y}_i, \vec{x}_i + \mathbf{S}_i \vec{u}_\ell \rangle] = [\langle \vec{y}_i, \vec{x}_i \rangle] + \langle \vec{y}_i, \mathbf{S}_i \rangle \cdot [\vec{u}_\ell]$. Thus, $\sum_i [d_{i,\ell}] = [\langle \vec{y}, \vec{x} \rangle] + (\sum_i \vec{y}_i^\top \mathbf{S}_i) \cdot [\vec{u}_\ell]$. Since $\vec{d} = \sum_i \mathbf{S}_i^\top \vec{y}_i$, we have $\sum_i [d_{i,\ell}] = [\langle \vec{y}, \vec{x} \rangle] + \vec{d}^\top [\vec{u}_\ell]$, hence $\alpha = \langle \vec{x}, \vec{y} \rangle$.

6.2.3 Security analysis

In this section, we provide the proof that the MCFE' described above achieves 1-Label-IND*-security (Section 3.1.2), using the wtr-IND*-security of the MCFE from Section 4.2.3, assuming the IP-FE is IND-secure (concrete instances of which are given in [ALS16]). We can state the following security result:

Theorem 6.3

For any adversary $\mathcal{A}$, against the 1-Label-IND*-security of the above MCFE',

$$\mathsf{Adv}^{\text{1-Label-IND}^*}_{\mathsf{MCFE}'}(\mathcal{A}) \leq \mathsf{Adv}^{\text{wtr-IND}^*}_{\mathsf{MCFE}}(t') + n \cdot \mathsf{Adv}^{\text{IND}}_{\mathsf{IP\text{-}FE}}(t''),$$

where both t' and t'' are close to the running time t of $\mathcal{A}$.

We prove how the scheme from Section 6.2.2 achieves the 1-Label-IND*-security. As explained in Definition 3.2, the 1-Label-IND*-security game for MCFE is exactly the IND*-security game where only one label ℓ^* is allowed in the challenge QLeftRight oracle, defined by its index ρ, at the initialization step. We assume that all the other encryption queries are asked to the QEncrypt. The proof uses a series of hybrid games, defined below:

Proof

For any game $\mathbf{G}$, we denote $\mathsf{Adv}_{\mathbf{G}}(\mathcal{A})$ the advantage of $\mathcal{A}$ in the game $\mathbf{G}$, that is, the probability that the procedure Finalize in the game $\mathbf{G}$ outputs 1. For any user $i \in [n]$, we denote by Q_i the number of queries to the oracle $\mathsf{QLeftRight}'$ containing the user i, that is, of the form: $\mathsf{QLeftRight}'(i, \vec{x}_i^{k,0}, \vec{x}_i^{k,1}, \ell)$, for $k \in \{1, \ldots, Q_i\}$. When all the Q_i's are 1, there is no repetition, but here we are dealing with repetitions. The counter k numbers the repetitions.

Game $\mathbf{G}_\beta$: For any $\beta \in \{0,1\}$, we define the following game, where multiple plaintexts can be queried for the same user i and the same label. We use a counter k, which starts at 1 to number the queries $(\vec{x}_i^{k,0}, \vec{x}_i^{k,1})$, under the label $\ell^* = \ell_\rho$. We do not keep track of the queries under other labels (as in previous definitions).

- Initialize(ρ): it generates $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_{i\in[n]}) \leftarrow \mathsf{SetUp}(\lambda)$, and for all $i \in [n]$, $(\mathsf{IP.mpk}_i, \mathsf{IP.msk}_i) \leftarrow \mathsf{IP.SetUp}(\lambda)$. It returns $\mathsf{mpk}' := (\mathsf{mpk}, (\mathsf{IP.mpk}_i)_{i\in[n]})$ to the adversary $\mathcal{A}$.
- $\mathsf{QEncrypt}'(i, \vec{x}_i, \ell_j)$: it first computes $[\vec{c}_i] \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i, \ell_j)$, and returns $\mathsf{IP.Enc}(\mathsf{msk}_i, [\vec{c}_i])$.
- $\mathsf{QLeftRight}'(i, \vec{x}_i^{k,0}, \vec{x}_i^{k,1}, \ell_\rho)$: it computes $[\vec{c}_i^k] \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{k,\beta}, \ell_\rho)$, and returns $\mathsf{IP.Enc}(\mathsf{msk}_i, [\vec{c}_i^k])$.
- $\mathsf{QDKeyGen}'(\vec{y})$: on input $\vec{y} := (\vec{y}_1 \| \cdots \| \vec{y}_n) \in \mathbb{Z}_p^{nm}$, it first computes $\mathsf{dk}_{\vec{y}} = \mathsf{DKeyGen}(\mathsf{msk}, \vec{y})$, and then, for all $i \in [n]$: $\mathsf{dk}_{\vec{y}_i} = \mathsf{IP.DKeyGen}(\mathsf{msk}_i, \vec{y}_i)$. It returns $\mathsf{dk}'_{\vec{y}} = \left(\mathsf{dk}_{\vec{y}}, \{\mathsf{dk}_{\vec{y}_i}\}_{i\in[n]}\right)$.
- $\mathsf{QCorrupt}'(i)$: on input a user $i \in [n]$, it returns $(\mathsf{ek}_i, \mathsf{IP.msk}_i)$.
- Finalize: as in Definition 3.2.

Note that:

$$\mathsf{Adv}^{\texttt{1-Label-IND}^*}_{\mathsf{MCFE}'}(\mathcal{A}) = |\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A})|.$$

Game $\mathbf{H}_0$: Now we consider the game $\mathbf{H}_0$ defined exactly as $\mathbf{G}_0$, except in $\mathsf{QLeftRight}'(i, \vec{x}_i^{k,0}, \vec{x}_i^{k,1}, \ell_\rho)$, one computes $[\vec{c}_i^k] \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{k,0} + \vec{x}_i^{1,1} - \vec{x}_i^{1,0}, \ell_\rho)$. Then it returns $\mathsf{IP.Enc}(\mathsf{IP.msk}_i, [\vec{c}_i^k])$. The transition from $\mathbf{G}_0$ and $\mathbf{H}_0$ uses $\texttt{1-Label-IND}^*$ security and the linear homomorphism of the ciphertexts of MCFE. Namely, we build a PPT adversary $\mathcal{B}$ against the $\texttt{1-Label-IND}^*$ security of MCFE such that:

$$|\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{H}_0}(\mathcal{A})| \leq \mathsf{Adv}^{\texttt{1-Label-IND}^*}_{\mathsf{MCFE}}(\mathcal{B}).$$

$\mathcal{B}$ simulates the view of the $\texttt{1-Label-IND}^*$-adversary $\mathcal{A}$ against MCFE' as follows:

- Initialize(ρ): after having sent ρ, it gets mpk from its $\texttt{1-Label-IND}^*$ challenger. For all $i \in [n]$, $(\mathsf{IP.mpk}_i, \mathsf{IP.msk}_i) \leftarrow \mathsf{IP.SetUp}(\lambda)$, and it returns $\mathsf{mpk}' := (\mathsf{mpk}, (\mathsf{IP.mpk}_i)_{i\in[n]})$ to the adversary $\mathcal{A}$.
- $\mathsf{QEncrypt}'(i, \vec{x}_i, \ell_j)$: it first computes $[\vec{c}_i] \leftarrow \mathsf{QEncrypt}(i, \vec{x}_i, \ell_j)$, and returns $\mathsf{IP.Enc}(\mathsf{msk}_i, [\vec{c}_i])$.
- $\mathsf{QLeftRight}'(i, \vec{x}_i^{k,0}, \vec{x}_i^{k,1}, \ell_\rho)$: for $k = 1$, i.e. the first query for user i, $\mathcal{B}$ queries its own QLeftRight oracle to get the value $[\vec{c}_i^1] = \mathsf{QLeftRight}(i, \vec{x}_i^{k,0}, \vec{x}_i^{k,1}, \ell_\rho)$, otherwise it computes $[\vec{c}_i^k] := [\vec{c}_i^1] + [\vec{x}_i^{k,0} - \vec{x}_i^{1,0}]$. It then returns $\mathsf{IP.Encrypt}(\mathsf{IP.msk}_i, [\vec{c}_i^k])$ to $\mathcal{A}$.

- $\mathsf{QDKeyGen}'(\vec{y})$: on input $\vec{y} := (\vec{y}_1 \| \cdots \| \vec{y}_n) \in \mathbb{Z}_p^{nm}$, it first computes $\mathsf{dk}_{\vec{y}} = \mathsf{DKeyGen}(\mathsf{msk}, \vec{y})$, and then, for all $i \in [n]$: $\mathsf{dk}_{\vec{y}_i} = \mathsf{IP.DKeyGen}(\mathsf{msk}_i, \vec{y}_i)$. It returns $\mathsf{dk}'_{\vec{y}} = \left(\mathsf{dk}_{\vec{y}}, \{\mathsf{dk}_{\vec{y}_i}\}_{i\in[n]}\right)$.
- $\mathsf{QCorrupt}'(i)$: $\mathcal{B}$ queries its own oracle to obtain $\mathsf{ek}_i \leftarrow \mathsf{QCorrupt}(i)$, and returns $(\mathsf{ek}_i, \mathsf{IP.msk}_i)$ to $\mathcal{A}$.
- Finalize: $\mathcal{B}$ verifies that the conditions in Definition 3.2 are satisfied; if they are, it forwards the guess b' of $\mathcal{A}$, otherwise, it sends a random bit to its own Finalize oracle.

Note that the constraints $\mathcal{B}$ has to verify in the finalize procedure, and namely for condition (2), might look exponential for general functionalities. But in the case of inner product, one just has to look at spanned vector subspaces. Namely, all queries $(i, \vec{x}_i^{k_i,0}, \vec{x}_i^{k_i,1}, \ell_\rho)_{i\in[n],k_i\in[Q_i]}$ to $\mathsf{QLeftRight}'$ and all queries $\vec{y} := (\vec{y}_1 \| \cdots \| \vec{y}_n)$ to $\mathsf{QDKeyGen}'$ must satisfy: $\sum_i \langle \vec{x}_i^{k_i,0}, \vec{y}_i \rangle = \sum_i \langle \vec{x}_i^{k_i,1}, \vec{y}_i \rangle$. This is an exponential number of linear equations, but, as noted in [AGRW17], it suffices to verify the linearly independent equations, of which there can be at most $n \cdot m$. This can be done efficiently given the queries.

One can note that, for the label $\ell_\rho = \ell^*$, $[\vec{c}_i^1]$ received by $\mathcal{B}$ is actually $[\vec{c}_i^1] = \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{1,b}, \ell^*)$, where b is the random bit chosen by the $\texttt{1-Label-IND}^*$ security game for MCFE that $\mathcal{B}$ is interacting with. By linear homomorphism of the ciphertexts of MCFE, for all $k \in [Q_i]$, we have: $[\vec{c}_i^k] = \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{1,b}, \ell^*) + [\vec{x}_i^{k,0} - \vec{x}_i^{1,0}] = \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{k,0} + \vec{x}_i^{1,b} - \vec{x}_i^{1,0}, \ell^*)$. So, when $b = 0$, $\mathcal{B}$ simulates $\mathbf{G}_0$, while it simulates $\mathbf{H}_0$ when $b = 1$, which proves $|\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{H}_0}(\mathcal{A})| \leq \mathsf{Adv}^{\texttt{1-Label-IND}^*}_{\mathsf{MCFE}}(\mathcal{B})$.

We define the following hybrid games $\mathbf{H}_r$, for all $r \in [n]$, as $\mathbf{H}_0$, except for $\mathsf{QLeftRight}'(i, \vec{x}_i^{k,0}, \vec{x}_i^{k,1}, \ell_\rho)$: for all $i \leq r$, it sets $[\vec{c}_i^k] \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{k,1}, \ell_\rho)$, instead of $[\vec{c}_i^k] \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{k,0} + \vec{x}_i^{1,1} - \vec{x}_i^{1,0}, \ell_k)$, and returns $\mathsf{IP.Enc}(\mathsf{msk}_i, [\vec{c}_i^k])$. Note that this definition is compatible with $\mathbf{H}_0$ defined previously, and $\mathbf{H}_n$ is $\mathbf{G}_1$. Thus, it suffices to build a PPT adversary $\mathcal{B}_r$ for all $r \in [n]$, against the IND-security of the IP-FE, such that:

$$|\mathsf{Adv}_{\mathbf{H}_{r-1}}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{H}_r}(\mathcal{A})| \leq \mathsf{Adv}^{\mathtt{IND}}_{\mathsf{IP\text{-}FE}}(\mathcal{B}_r).$$

We distinguish two cases. The first case occurs when $\mathcal{A}$ queries the user r to its oracle $\mathsf{QCorrupt}'$. Then, conditioned on the event that Finalize doesn't output a random bit, it must be the case that for all $k \in [Q_r]$, $\vec{x}_r^{k,0} = \vec{x}_r^{k,1}$. If we call E this first case, we have: $\mathsf{Adv}_{\mathbf{H}_{r-1}}(\mathcal{A} \wedge E) = \mathsf{Adv}_{\mathbf{H}_r}(\mathcal{A} \wedge E)$. The second case corresponds to the event $\neg E$: $\mathcal{A}$ does not query $\mathsf{QCorrupt}'$ on r. We build a PPT adversary $\mathcal{B}$ such that $|\mathsf{Adv}_{\mathbf{H}_{r-1}}(\mathcal{A}|\neg E) - \mathsf{Adv}_{\mathbf{H}_r}(\mathcal{A}|\neg E) \leq \mathsf{Adv}^{\mathtt{IND}}_{\mathsf{IP\text{-}FE}}(\mathcal{B}_r)$, which implies that $|\mathsf{Adv}_{\mathbf{H}_{r-1}}(\mathcal{A} \wedge \neg E) - \mathsf{Adv}_{\mathbf{H}_r}(\mathcal{A} \wedge \neg E) \leq \mathsf{Adv}^{\mathtt{IND}}_{\mathsf{IP\text{-}FE}}(\mathcal{B}_r)$. We conclude using the fact that for any game $\mathbf{G}$ and event E: $\mathsf{Adv}_{\mathbf{G}}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}}(\mathcal{A} \wedge E) + \mathsf{Adv}_{\mathbf{G}}(\mathcal{A} \wedge \neg E)$.

We now proceed to describe $\mathcal{B}_r$, which simulates the view of the $\texttt{1-Label-IND}^*$ adversary $\mathcal{A}$ against MCFE' as follows:

- Initialize(ρ): $\mathcal{B}_r$ obtains $\mathsf{IP.mpk}_r$ from its own Initialize oracle, and generates $(\mathsf{IP.mpk}_i, \mathsf{IP.msk}_i) \leftarrow \mathsf{IP.SetUp}(\lambda)$ for all $i \neq r$, $(\mathsf{mpk}, \mathsf{msk}, (\mathsf{ek}_i)_i) \leftarrow \mathsf{SetUp}(\lambda)$ and returns $\mathsf{mpk}' := (\mathsf{mpk}, (\mathsf{IP.mpk}_i)_i)$ to $\mathcal{A}$.
- $\mathsf{QEncrypt}'(i, \vec{x}_i, \ell_j)$: it computes $[\vec{c}_i] \leftarrow \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i, \ell_j)$. If $i \neq r$, it returns $\mathsf{IP.Enc}(\mathsf{msk}_i, [\vec{c}_i])$; if $i = r$, it returns $\mathsf{QLeftRight}([\vec{c}_i], [\vec{c}_i])$.
- $\mathsf{QLeftRight}'(i, \vec{x}_i^{k,0}, \vec{x}_i^{k,1}, \ell_\rho)$: $\mathcal{B}$ computes $[\vec{c}_i^{k,0}] = \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{k,1}, \ell_\rho)$ and $[\vec{c}_i^{k,1}] = \mathsf{Encrypt}(\mathsf{ek}_i, \vec{x}_i^{k,0} + \vec{x}_i^{1,1} - \vec{x}_i^{1,0}, \ell_\rho)$, and uses its own $\mathsf{QLeftRight}$ oracle to output the ciphertext to $\mathcal{A}$
  - if $i < r$, it outputs $\mathsf{IP.Enc}(\mathsf{msk}_i, [\vec{c}_i^{k,0}])$.
  - if $i > r$, it outputs $\mathsf{IP.Enc}(\mathsf{msk}_i, [\vec{c}_i^{k,1}])$.
  - if $i = r$, it outputs $\mathsf{QLeftRight}([\vec{c}_i^{k,0}], [\vec{c}_i^{k,1}])$.
- $\mathsf{QDKeyGen}'(\vec{y})$: on input $\vec{y} := (\vec{y}_1 \| \cdots \| \vec{y}_n) \in \mathbb{Z}_p^{nm}$, $\mathcal{B}_r$ computes $\mathsf{dk}_{\vec{y}} = \mathsf{DKeyGen}(\mathsf{msk}, \vec{y})$, for all $i \neq r$: it computes $\mathsf{dk}_{\vec{y}_i} = \mathsf{IP.DKeyGen}(\mathsf{msk}_i, \vec{y}_i)$, and it queries its $\mathsf{QDKeyGen}$ oracle to obtain $\mathsf{QDKeyGen}(\vec{y}_r)$. It returns $\mathsf{dk}'_{\vec{y}} = \left(\mathsf{dk}_{\vec{y}}, \{\mathsf{dk}_{\vec{y}_i}\}_{i \in [n]}\right)$ to $\mathcal{A}$.
- $\mathsf{QCorrupt}(i)$: if $i = r$, $\mathcal{B}$ aborts the simulation and sends a random bit to its Finalize oracle. Otherwise, it returns $\mathsf{IP.msk}_i$.
- Finalize: $\mathcal{B}_r$ verifies that the conditions in Definition 3.2 are satisfied; if they are, it forwards the guess b' of $\mathcal{A}$, otherwise, it sends a random bit to its own Finalize oracle.

Note that when the random bit b used by the $\mathtt{IND}$-security game of IP-FE that $\mathcal{B}_r$ is interacting with is equal to 0, then, $\mathcal{B}_r$ simulates the game $\mathbf{H}_r$ to $\mathcal{A}$; otherwise, it simulates the game $\mathbf{H}_{r-1}$. In particular, the condition of the Finalize from Definition 3.2 implies that for all queries $(i, \vec{x}_i^{k_i,0}, \vec{x}_i^{k_i,1}, \ell_\rho)$ to $\mathsf{QLeftRight}'$, we have: $\sum_i \langle \vec{x}_i^{k_i,0}, \vec{y}_i \rangle = \sum_i \langle \vec{x}_i^{k_i,1}, \vec{y}_i \rangle$ for all $k_i \in [Q_i]$. Thus, we have in particular, for all $k \in [Q_r]$:

$$\begin{aligned}
&\langle \vec{x}_r^{k,0} - \vec{x}_\rho^{1,0}, \vec{y}_r \rangle = \langle \vec{x}_r^{k,1} - \vec{x}_\rho^{1,1}, \vec{y}_r \rangle \Rightarrow \\
&\langle \vec{x}_r^{k,0} + \vec{x}_r^{1,1} - \vec{x}_\rho^{1,0}, \vec{y}_r \rangle = \langle \vec{x}_r^{k,1} + \vec{x}_r^{1,1} - \vec{x}_\rho^{1,1}, \vec{y}_r \rangle \Rightarrow \\
&\langle \vec{c}_r^{k,0}, \vec{y}_r \rangle = \langle \vec{c}_r^{k,1}, \vec{y}_r \rangle,
\end{aligned}$$

where $[\vec{c}_r^{k,0}] = \mathsf{Encrypt}(\mathsf{ek}_r, (\vec{x}_r^{k,0} + \vec{x}_r^{1,1} - \vec{x}_r^{1,0}), \ell_\rho)$ and $[\vec{c}_r^{k,1}] = \mathsf{Encrypt}(\mathsf{ek}_r, (\vec{x}_r^{k,1} + \vec{x}_r^{1,1} - \vec{x}_r^{1,1}), \ell_\rho)$. The last implication uses the structural properties of the IP-MCFE scheme, namely, the property of linear homomorphism, and deterministic encryption. The last equality corresponds exactly to the condition to prevent the Finalize oracle from the $\mathtt{IND}$ security game of the IP-FE from outputting a random bit (see Definition 2.16).

This proves $|\mathsf{Adv}_{\mathbf{H}_{r-1}}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{H}_r}(\mathcal{A})| \leq \mathsf{Adv}^{\mathtt{IND}}_{\mathsf{IP\text{-}FE}}(\mathcal{B}_r)$, and concludes the security proof.

6.3 CCA security from signatures

In this section we briefly describe how to achieve Chosen-Ciphertext ($\mathtt{CCA}$) security in a black-box way, where the adversary may have access to functional decryption queries, without learning the functional decryption key, but just the result. Indeed,

while using public-key cryptography techniques, MCFE remains a symmetric-key primitive, where the client encryption key not only allows encryption, but decryption. Using the additional decryption algorithm, we show that adding a signature (Definition 2.7) simply provides CCA-security.

6.3.1 Description

More precisely, given an IND-secure MCFE scheme $\mathsf{MCFE} = (\mathsf{MCFE.SetUp}, \mathsf{MCFE.Encrypt}, \mathsf{MCFE.KeyGen}, \mathsf{MCFE.FDecrypt}, \mathsf{MCFE.Decrypt})$, a strongly unforgeable signature $(\mathsf{S.SetUp}, \mathsf{S.Sign}, \mathsf{S.Verif})$, we generically build an IND-CCA-secure scheme $\widetilde{\mathsf{MCFE}}$, where the changes are as follows:

- $\widetilde{\mathsf{MCFE}}.\mathsf{SetUp}(\lambda)$: executes $(\mathsf{msk}, (\mathsf{ek}_i)_i, \mathsf{pp}) \leftarrow \mathsf{MCFE.SetUp}(\lambda)$ and, for any $i \in [n]$, sets the signature parameters $(\mathsf{S.key}_i, \mathsf{S.pp}_i) \leftarrow \mathsf{S.SetUp}(\lambda)$ for each user i. $\widetilde{\mathsf{msk}} = (\mathsf{msk}, (\mathsf{S.key}_i)_i)$, for all i sets $\widetilde{\mathsf{ek}_i} = (\mathsf{ek}_i, \mathsf{S.key}_i)$, and finally $\widetilde{\mathsf{pp}} = (\mathsf{pp}, (\mathsf{S.pp}_i)_i)$.
- $\widetilde{\mathsf{MCFE}}.\mathsf{Encrypt}(\widetilde{\mathsf{ek}_i}, x_i, \ell)$: computes $A_{\ell,i} \leftarrow \mathsf{MCFE.Encrypt}(\mathsf{ek}_i, x_i, \ell)$ and the signature $S_{\ell,i} \leftarrow \mathsf{S.Sign}(\mathsf{S.key}_i, (A_{\ell,i}, \ell))$ to return $C_{\ell,i} = (A_{\ell,i}, S_{\ell,i})$.
- $\widetilde{\mathsf{MCFE}}.\mathsf{FDecrypt}(\mathsf{dk}_f, \ell, \vec{C})$: given a ciphertext $\vec{C} = (A_i, S_i)_i$, computes the values $\mathsf{S.Verif}(\mathsf{S.pp}_i, (A_i, \ell), S_i)$ for any user i. If all the signatures are valid, returns $\alpha \leftarrow \mathsf{MCFE.FDecrypt}(\mathsf{dk}_f, \ell, (A_i)_i)$, otherwise, returns $\perp$.
- $\widetilde{\mathsf{MCFE}}.\mathsf{Decrypt}(\widetilde{\mathsf{ek}_i}, \ell, \vec{C})$: given a ciphertext $\vec{C} = (A_i, S_i)_i$, computes the values $\mathsf{S.Verif}(\mathsf{S.pp}_i, (A_i, \ell), S_i)$ for any user i. If all the signatures are valid, returns $x_i \leftarrow \mathsf{MCFE.Decrypt}(\mathsf{ek}_i, \ell, (A_i)_i)$, otherwise, returns $\perp$.

$\widetilde{\mathsf{MCFE}}.\mathsf{DKeyGen}$ remains the same as $\mathsf{MCFE.DKeyGen}$. In the following section we provide the full security proof.

6.3.2 Security analysis

Here we formally prove the scheme described in Section 6.3 to achieve the IND−CCA security level using the strong unforgeability (see Definition 2.8) property of the signature. The proof consists of a sequence of games starting from $\mathbf{G}_0$, playing the real security game described as above, to $\mathbf{G}_2$ where we are back to the basic IND-security game, without decryption queries.

Proof

Game $\mathbf{G}_0$: the real security game.

- Initialize: $\mathcal{C}$ runs both the algorithms $\mathsf{MCFE.SetUp}(\lambda)$ and $\mathsf{S.SetUp}(\lambda)$, to get respectively $(\mathsf{msk}, (\mathsf{ek}_i)_i, \mathsf{pp})$ and $(\mathsf{S.key}_i, \mathsf{S.pp}_i)_i$. Finally, $\mathcal{C}$ chooses a bit $b \xleftarrow{\$} \{0,1\}$ and provides $\mathcal{A}$ the public parameters $\mathsf{mpk} = (\mathsf{pp}, (\mathsf{S.pp}_i)_i)$.
- Encryption queries $\mathsf{QEncrypt}(i, x_i, \ell)$: for each encryption query, $\mathcal{C}$ computes $A_{\ell,i} \leftarrow \mathsf{MCFE.Encrypt}(\mathsf{ek}_i, x_i, \ell)$ and the signature $S_{\ell,i} \leftarrow \mathsf{S.Sign}(\mathsf{S.key}_i, (A_{\ell,i}, \ell))$ to return $\mathcal{A}$ the value $C_{\ell,i} = (A_{\ell,i}, S_{\ell,i})$.

- Challenge queries $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$: for each encryption query, $\mathcal{C}$ computes $A_{\ell,i} \leftarrow \mathsf{MCFE.Encrypt}(\mathsf{ek}_i, x_i^b, \ell)$ and the signature $S_{\ell,i} \leftarrow \mathsf{S.Sign}(\mathsf{S.key}_i, (A_{\ell,i}, \ell))$ to return $\mathcal{A}$ the value $C_{\ell,i} = (A_{\ell,i}, S_{\ell,i})$.
- Functional decryption key queries $\mathsf{QDKeyGen}(f)$: for each functional key query, $\mathcal{C}$ returns $\mathsf{dk}_f \leftarrow \mathsf{MCFE.QDKeyGen}(\mathsf{msk}, f)$.
- Corruption queries $\mathsf{QCorrupt}(i)$: for each corruption query, $\mathcal{C}$ returns $\widetilde{\mathsf{ek}_i} = (\mathsf{ek}_i, \mathsf{S.key}_i)$.
- Functional decryption queries $\mathsf{QFDecrypt}(\vec{C}, \ell, f)$: for each functional decryption query, $\mathcal{C}$ first checks the validity of the signatures parts S_i of C_i. If all the signatures are valid, recovers the decryption function key dk_f and runs $\mathsf{MCFE.FDecrypt}(\mathsf{dk}_f, \ell, \vec{C})$.

Game $\mathbf{G}_1$: in this game, $\mathcal{C}$ does as above, but rejects (outputting $\perp$) upon a query $\mathsf{QFDecrypt}(f, \ell, \vec{C})$ for which there is $i \in \mathcal{HS}$, and C_i is not the output of $\mathsf{QEncrypt}$ or $\mathsf{QLeftRight}$ on previous queries. This makes a difference with game $\mathbf{G}_0$ only if such C_i contains a valid signature, which would be a forgery. By simply guessing which client will be impersonated (which is correct with probability greater than $1/n$), one can output a forgery as soon as a difference happens:

$$|\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) - \mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A})| \leq n \times \mathsf{Adv}^{\mathtt{SUF}}_{\mathsf{S}}(t).$$

Now we build a PPT adversary $\mathcal{B}$ such that: $\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) \leq \mathsf{Adv}^{\mathtt{IND}}_{\mathsf{MCFE}}(\mathcal{B})$, that is, we rely on the IND-CPA security of the underlying MCFE to conclude the CCA security proof. $\mathcal{B}$ samples the signature parameters $(\mathsf{S.key}_i, \mathsf{S.pp}_i)_i \leftarrow \mathsf{S.SetUp}(\lambda)$ for all $i \in [n]$, and uses its own oracle to simulate the oracles $\mathsf{QEncrypt}$, $\mathsf{QLeftRight}$, $\mathsf{QDKeyGen}$ and $\mathsf{QCorrupt}$. It answers queries $\mathsf{QFDecrypt}(f, \ell, \vec{C})$ as follows. If for some $i \in \mathcal{HS}$, C_i is not the output of $\mathsf{QEncrypt}$ or $\mathsf{QLeftRight}$ on previous queries, then output $\perp$. Otherwise, that means that for all $i \in \mathcal{HS}$, either C_i is the output of $\mathsf{QEncrypt}(i, x_i, \ell)$, in which case $\mathcal{B}$ sets $\widetilde{x_i} := x_i$; either C_i is the output of $\mathsf{QLeftRight}(i, x_i^0, x_i^1, \ell)$, in which case $\mathcal{B}$ sets $\widetilde{x_i} := x_i^0$. For all corrupted slots $i \in [n] \setminus \mathcal{HS}$, $\mathcal{B}$ computes $\widetilde{x_i} \leftarrow \mathsf{Decrypt}(\mathsf{ek}_i, \ell, \vec{C})$, which it can do since it knows ek_i, and $\vec{C}$ is a complete valid ciphertext. It finally outputs $f(\widetilde{\vec{x}} = (\widetilde{x_i})_i)$ to $\mathcal{A}$, which is correctly simulated, since the Finalize procedure imposes that the function f doesn't distinguish between x_i^0 and x_i^1 queried to $\mathsf{QLeftRight}$, that is, it doesn't matter whether we choose $\widetilde{x_i} := x_i^0$ or $\widetilde{x_i} := x_i^1$. This allows $\mathcal{B}$ to properly simulate the oracle $\mathsf{QFDecrypt}$ without knowing which bit b is chosen by $\mathsf{QLeftRight}$ in its own experiment.

7 Delegation of access control

In this last chapter, we describe how to delegate the access control through functional encryption, the material from this chapter comes from the work [CPP17].

In a first part, we define the attribute-based key encapsulation mechanism primitive, a mix of CP-ABE and KEM (see Definition 2.19 and Definition 2.9 respectively), and the homomorphic policy property. We discuss about the security behind those notions.

In the second part, we discuss about a construction of a new LSSS (Definition 2.14) from two independent LSSS, following combinations of their shares.

In the last part, we provide an example with an adaptation of the Lewko-Waters scheme from [LW11], and we explain how to exploit homomorphic policy property using the LSSS techniques from the second part.

Contents

7.1 Preliminaries

In this part we derive a key encapsulation mechanism from the usual attribute-based encryption, we define the homomorphic-policy property and the associated security notions, and we explain how schemes based on this formalism can solve the access control delegation problem described in the introduction.

7.1.1 Attribute-based key encapsulation mechanism

We first extend ABE to attribute-based key encapsulation mechanism (ABKEM), where the ciphertext encapsulates a session key, later used to encrypt the message, in a symmetric way.

Definition 7.1: ABKEM

An attribute-based key encapsulation mechanism (ABKEM) over an attribute space $\mathfrak{A}$ is defined by four algorithms:

- $\mathsf{SetUp}(\lambda)$: takes as input the security parameter, and outputs the master secret key msk and the public key pk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, \mathsf{a})$: takes as input the master secret key msk, the identity id of a player, and an attribute $\mathsf{a} \in \mathfrak{A}$, to output the private decryption key $\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}}$ for this attribute a.
- $\mathsf{Encaps}(\mathsf{pk}, p)$: takes as input the public key pk and a policy p, to output a key K and an encapsulation E of this key.
- $\mathsf{Decaps}(\mathsf{dk}, E)$: takes as input a decryption key and an encapsulation E, to output the encapsulated key K or $\perp$.

Correctness. For any $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$, $\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, \mathsf{a})$ for $\mathsf{a} \in \mathsf{A}$, and $(K, E) \leftarrow \mathsf{Encaps}(\mathsf{pk}, p)$, $\mathsf{Decaps}(\mathsf{dk}_{\mathsf{id}}, E) = K$ if A satisfies the policy p.

Note that the decryption key will indifferently mean a key $\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}}$ for a specific user id and a specific attribute a, or a set $\mathsf{dk}_{\mathsf{id}}^{\mathsf{A}}$ of keys specific to a user id, but for many attributes $\mathsf{a} \in \mathsf{A} \subset \mathfrak{A}$.

Indistinguishability. The main security property is the usual indistinguishability (IND), which should prevent collusions of adaptively chosen players, that can also get decryption keys for adaptively chosen attributes:

Definition 7.2: IND for ABKEM

Let us consider an ABKEM over an attribute space $\mathfrak{A}$. No adversary $\mathcal{A}$ should be able to break the following security game against a challenger:

- Initialization: the challenger runs the setup $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$, and provides pk to the adversary $\mathcal{A}$.

- Key Queries $\mathsf{QKeyGen}$: the adversary $\mathcal{A}$ can ask KeyGen-queries, for any id and any attribute a of its choice to get $\mathsf{dk}_{\mathsf{id}}^{\mathsf{a}}$.
- Challenge $\mathsf{Challenge}$: the adversary $\mathcal{A}$ provides a policy p to the challenger that runs $(K, E) \leftarrow \mathsf{Encaps}(\mathsf{pk}, p)$, and sets $K_b \leftarrow K$ and K_{1-b} as a random key, for a random bit b. It provides (E, K_0, K_1) to the adversary.
- Key Queries $\mathsf{QKeyGen}$: the adversary $\mathcal{A}$ can again ask KeyGen-queries of its choice.
- Finalize: the adversary $\mathcal{A}$ outputs its guess b' on the bit b.

We also define the event Cheat, which means that a user (with some identity id) owns a set of attributes A (the set of all the attributes a asked to a Key Query for id) that satisfies p: in such a case, the adversary can trivially guess b. Hence, we only allow adversaries such that $\Pr[\mathsf{Cheat}] = 0$. We then define $\mathsf{Adv}^{\mathtt{ind}}(\mathcal{A}) = |2 \times \Pr[b' = b] - 1|$, and say that an ABKEM is (t, ε)-adaptively secure if no adversary $\mathcal{A}$ running within time t can get $\mathsf{Adv}^{\mathtt{ind}}(\mathcal{A}) \geq \varepsilon$.

We stress that everything is adaptive in this definition: the identities and the attributes asked to the key queries, and the policy asked for the challenge query. However, we are in the chosen-plaintext scenario, without access to any decryption/decapsulation oracle.

7.1.2 Homomorphic-policy

While $\mathsf{CP\text{-}ABE}$ allows to specify the policy at the encryption time, which is also the case for our definition of ABKEM, the sender may not be aware of the policy yet. We thus suggest to exploit an homomorphic property on the policy: we would like to allow the derivation of an encapsulation of K under a combination $p = p_1 \wedge p_2$ or $p = p_1 \vee p_2$ from the encapsulations of K under the policies p_1 and p_2 on the attributes in $\mathfrak{A}$, without knowing K (which has already been used to encrypt the payload).

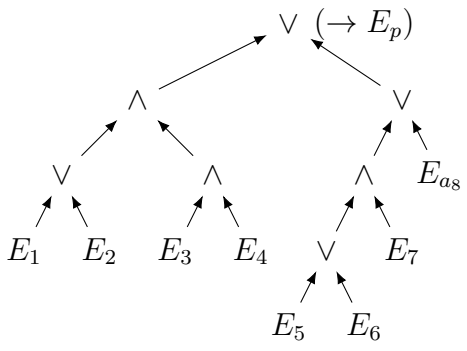

Figure 7.1: Derivation of E_p from a set of encapsulations $\{E_i\}$, for the policy $p = ((\mathsf{a}_1 \vee \mathsf{a}_2) \wedge (\mathsf{a}_3 \wedge \mathsf{a}_4)) \vee (((\mathsf{a}_5 \vee \mathsf{a}_6) \wedge \mathsf{a}_7) \vee \mathsf{a}_8)$

With such an homomorphism on the policies, from the encapsulations of a common key K under all the attributes $\mathsf{a} \in \mathfrak{A}$, one could publicly generate an encapsulation of

K under any policy on $\mathfrak{A}$: as illustrated above on Fig. 7.1, from the encapsulations $\{E_i\}_i$ of K for the attributes $\mathfrak{A} = \{\mathsf{a}_i\}$, one can derive the encapsulation E_p of K under any policy p, encoded as a binary tree with AND ($\wedge$) and OR ($\vee$) gates. Again, we only consider monotone policies, hence the absence of NOT gates. On attributes, if one wants to consider the negation (or absence) of some attribute a, one has to define a second attribute a' that is exclusive with a, so that, if $p = (\mathsf{a})$, then $\neg p = (\mathsf{a}')$.

To achieve this goal, we just need to be able to combine two encapsulations of K under p_1 and p_2 in order to derive the encapsulation of K under $p_\vee = p_1 \vee p_2$ and under $p_\wedge = p_1 \wedge p_2$. The global encapsulation under a more general policy can then be recursively built.

Definition 7.3: Homomorphic-Policy ABKEM

An Homomorphic-Policy Attribute-Based Key Encapsulation Mechanism (denoted HP-ABKEM) over an attribute space $\mathfrak{A}$ is an ABKEM (see Definition 7.1), with a more specific encapsulation algorithm and two additional algorithms for the homomorphism:

- $\mathsf{Encaps}(\mathsf{pk}, P)$: takes as input the public key pk, a list of policies $P = (p_i)_i$, to output a key K and the encapsulations E_i of this key under the policies p_i's.
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{gate}, E_1, E_2)$: takes as input the public key pk as well as two encapsulations E_1 and E_2, and a gate $\mathsf{gate} \in \{\wedge, \vee\}$, to output an encapsulation under the combination of the initial policies for E_1 and E_2.
- $\mathsf{Rand}(\mathsf{pk}, E)$: takes as input the public key pk as well as an encapsulation, to output a new encapsulation (of the same key under the same policy).

The intuition behind the new Encaps algorithm is that we want to be able to encapsulate the same key K under various policies. We thus opt for an encapsulation algorithm that takes as input all the policies that will be combined later.

Correctness. The correctness properties are:

- if $(E_i)_i \leftarrow \mathsf{Encaps}(\mathsf{pk}, (p_i)_i)$ are common encapsulations of a key K under the p_i's, then for any i, j, $E \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{gate}, E_i, E_j)$ is an encapsulation of the same key K, but under the policy $p = p_i \ \mathsf{gate} \ p_j$.
- for any encapsulation E of some key K under a policy p, $E' \leftarrow \mathsf{Rand}(\mathsf{pk}, E)$ follows the same distribution as a fresh encapsulation of K under the policy p.

Note that we do not expect the combination to hide the structure of the initial encapsulations. The randomization will do this work, but there is no need to do it at each step, hence the separation of the two processes: one will iteratively combine the encapsulations in order to obtain the encapsulation under the appropriate policy, and then the randomization will finalize the process. Fig. 7.2 illustrates this fact: combining and randomizing at each step leads to exactly the same distribution of the root encapsulation as combining at each step and randomizing at the last step only.

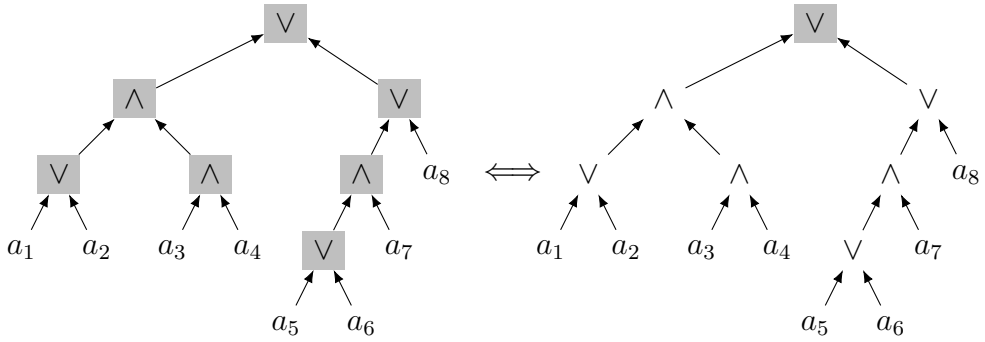

Figure 7.2: Randomization process in combination

7.1.3 Security

As explained in the Pay-TV scenario in the introduction, we have three players: the content provider (or the sender), the manager of the access policy (or the combiner) and the receiver. We recall that we do not consider the security manager here since it is a necessary trusted party, potentially played by the sender. We thus expect the sender to encapsulate a key K under each attribute, and to encrypt the payload under K; the combiner then generates the encapsulation of K under the appropriate policy; so that only the legitimate receivers can decapsulate and decrypt the payload.

When the adversary plays the role of the receivers, the required security notion is exactly the previous indistinguishability: given several keys for various attributes, and even several identities (to model collusions), an adversary should not be able to get any information about a key encapsulated under a policy that is not satisfies by any of the users under its control. We stress that this indistinguishability game (`IND`) models the resistance against the collusion of receivers. But both the sender and the combiner are considered honest.

On the other hand, the sender may not totally trust the combiner and may want to limit the risk in case the combiner would be corrupted: while the former sends K encapsulated under many attributes (or more generally many policies), the latter should not be able to distinguish K from a random key, in order to guarantee to privacy of the content encrypted under K. Hence the new indistinguishability game with multiple encapsulations (`m-IND`), but without being able to get any decryption key, hence the no-key attack (`NKA`). Since the adversary does not have access to any decryption key, this security scenario does not allow the combiner to collude with anybody, and namely not with any receiver.

Definition 7.4: `m-IND-NKA` for ABKEM

Let us consider an ABKEM over an attribute space $\mathfrak{A}$. No adversary $\mathcal{A}$ should be able to break the following security game against a challenger:

- Initialization: the challenger runs the setup $(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$ and provides pk to the adversary $\mathcal{A}$;

- Challenge: the challenger runs $(K, (E_i)_i) \leftarrow \mathsf{Encaps}(\mathsf{pk}, \mathfrak{A})$, and sets $K_b \leftarrow K$ and K_{1-b} as a random key, for a random bit b. It provides $((E_i)_i, K_0, K_1)$ to the adversary;
- Finalize: the adversary $\mathcal{A}$ outputs its guess b' on the bit b.

We then define $\mathsf{Adv}^{\texttt{m-ind-nka}}(\mathcal{A}) = |2 \times \Pr[b' = b] - 1|$, and say that an ABKEM is (t, ε)-m-IND if no adversary $\mathcal{A}$ running within time t can get $\mathsf{Adv}^{\texttt{m-ind-nka}}(\mathcal{A}) \geq \varepsilon$.

We stress that now, nothing is adaptive, since the adversary cannot get decryption keys, but gets the encapsulations of the same key K under all the individual attributes. We also remain in the chosen-plaintext scenario, without access to any decryption/decapsulation oracle. In addition, since the adversary is the combiner that receives the key K encapsulated under every attribute, we do not allow any collusion with a user: any attribute would be enough to get K and break the security game.

On can note that in the real-life, such a combiner would not be a critical party since it does not know any long-term secret. Of course, it will learn ephemeral encapsulations that would allow any receiver (with attributes that satisfy the final policy or not) to decapsulate the session key and to decrypt the content. But a short-term corruption will just leak the content during a short period, and not for ever.

7.2 An appropriate LSSS construction

The ABKEM we aim to build with the homomorphic policy functionality allows a very fine grained access control. The classical method to realize a such control used in ABE is the monotone span program (Definition 2.12), achieved by a linear secret sharing scheme (Definition 2.14).

Linear secret sharing is a secret sharing (Definition 2.13) defined with a matrix. In order to share $s \in \mathbb{K}$, one chooses $v_2, \ldots, v_n \xleftarrow{\$} \mathbb{K}$ and sets $\vec{v} \leftarrow (s, v_2, \ldots, v_n)^t$, then the share-vector is $\vec{\nu} \leftarrow \mathbf{A} \cdot \vec{v}$. One would like to be able to reconstruct s from a few coordinates of this share-vector is $\vec{\nu}$. Being able to find such a vector $\vec{c}$ with support I is equivalent to reconstruct s for the players satisfying $\rho(I)$ only: $\sum_{i \in I} c_i \cdot \nu_i = \sum_{i=1}^{m} c_i \cdot \nu_i = \vec{c}^t \cdot \vec{\nu} = \vec{c}^t \cdot \mathbf{A} \cdot \vec{v} = (1, 0, \ldots, 0) \cdot \vec{v} = s$.

Lewko-Waters LSSS construction

To give an example, we can refer to the LSSS proposed by Lewko-Waters [LW11]. It generates the matrix $\mathbf{A}$ and the map ρ from any monotone policy p that is encoded as a boolean tree, with binary AND and OR gates. One does not need to handle NOT gates, since one only considers monotone policies. We recall in Fig. 7.3 this construction from a predicate: let p be a predicate, we build the corresponding binary tree and we apply the following algorithm to build the share-generating matrix $\mathbf{A}$.

However, this chapter aims to describe another way to build such a LSSS matrix, more adapted to the homomorphic policy functionality. This one assemble several independents LSSS into one. More precisely, we detail a construction of the LSSS, from a boolean tree (with only OR and AND gates), in an iterative way.

Input: Predicate binary tree
$c \leftarrow 1$;
$\mathbf{A} \leftarrow [\varnothing]$;
Associate (1) to the root ;
Do a Depth First Search in the tree, from the root, **and** ;
for *unvisited nodes* **do**
  Set v to the vector associated to the node ;
  if *node* $= \vee$ **then**
    Associate $\underbrace{v||(0,...,0)}_{c}$ to the children ;
  end
  if *node* $= \wedge$ **then**
    $c \leftarrow c + 1$;
    Associate $\underbrace{v||(0,...,0,1)}_{c}$ to one children ;
    Associate $\underbrace{(0,...,0,-1)}_{c}$ to the other children ;
  end
end
Create the matrix $\mathbf{A}$ with the rows corresponding to the vectors associated to the leaves (extend the vectors with 0 at the end when necessary) ;
Output: $\mathbf{A}$

Figure 7.3: Conversion of predicate binary tree into an LSSS share-generating matrix, method used in [LW11].

Iterative LSSS construction

First, we have to start from an LSSS for a simple policy $p = (\mathsf{a}_i)$, for some i (*i.e.*, a unique attribute): $\mathbf{A}_i = (1)$ and $\rho(1) = i$. Then we explain how to combine two policies p_1 and p_2, represented by the LSSS's $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$ respectively, into the policies $p_\wedge = p_1 \wedge p_2$ and $p_\vee = p_1 \vee p_2$ with LSSS's $(\mathbf{A}_\wedge, \rho_\wedge)$ and $(\mathbf{A}_\vee, \rho_\vee)$ respectively.

In the following, for any $\mathbf{A}$, we denote $\mathbf{A}^1$ the first column et $\mathbf{A}^*$ the matrix $\mathbf{A}$ without the first column (*i.e.*, $\mathbf{A} = \begin{pmatrix}\mathbf{A}^1 & \mathbf{A}^*\end{pmatrix}$).

Proposition 7.1

Let $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$ be two LSSS's for the policies p_1 and p_2. Then we can build the LSSS's $(\mathbf{A}_\wedge, \rho_\wedge)$ and $(\mathbf{A}_\vee, \rho_\vee)$ for the policies $p_\wedge = p_1 \wedge p_2$ and $p_\vee = p_1 \vee p_2$ as follows

$$\mathbf{A}_\vee = \begin{pmatrix} \mathbf{A}_1^1 & \mathbf{A}_1^* & 0 \\ \mathbf{A}_2^1 & 0 & \mathbf{A}_2^* \end{pmatrix} \quad \mathbf{A}_\wedge = \begin{pmatrix} \mathbf{A}_1^1 & \mathbf{A}_1^1 & \mathbf{A}_1^* & 0 \\ 0 & -\mathbf{A}_2^1 & 0 & -\mathbf{A}_2^* \end{pmatrix}$$

If we label the rows of the matrices from 1 to $m_1 + m_2$, where $\mathbf{A}_1 \in \mathbb{K}^{m_1 \times n_1}$ and

$\mathbf{A}_2 \in \mathbb{K}^{m_2 \times n_2}$, we have

$$\rho_\wedge = \rho_\vee : x \mapsto \begin{cases} \rho_1(x), \text{ if } x \leq m_1 \\ \rho_2(x - m_1), \text{ if } x \geq m_1 + 1 \end{cases}$$

This construction is not really new, since it was described in [NN04] in a more generic way. But we need this explicit description for the security analysis of our ABKEM. The correctness of this LSSS construction is provided in the following proof. Up to a re-ordering of the rows and columns of the matrices, this is also the same construction obtained from the algorithm presented in Fig. 7.3 from [LW11]. A brief comparison of the two methods is indeed proposed after the proof of Proposition 7.1.

Proof

Let us now prove the combinations of LSSS lead to LSSS for the combined policies.

First, let us remark that with the initialization $\mathbf{A} = (1)$ and $\rho(1) = i$ for a simple policy $p = (\mathsf{a}_i)$, this is indeed an LSSS:

- $\nu_1 = s$, where ν_1 is the share of the player i (or with attribute a_i). The secret s can be revocered;
- Without ν_1, no information is leaked about s.

Disjunctions. We assume that for some attributes A, there is I such that $\rho(I) \subset A$ satisfies the policy $p_1 \vee p_2$. This also means there is $I' \subseteq I$ such that $\rho(I') \subset A$ satisfies the policy p_b, for $b = 1$ or $b = 2$: from the LSSS $(\mathbf{A}_b, \rho_b)$, there is $\vec{c}_b$ with support I' such that $\vec{c}_b^t \cdot \mathbf{A}_b = (1, 0, \ldots, 0)$. Let us take the other vector $\vec{c}_{3-b}^t \leftarrow (0, \ldots, 0)^t$ and build:

$$\vec{c} = \begin{pmatrix} (2-b)\vec{c}_1 \\ (b-1)\vec{c}_2 \end{pmatrix} \left(= \begin{pmatrix} \vec{c}_1 \\ 0 \end{pmatrix} \text{ or } = \begin{pmatrix} 0 \\ \vec{c}_2 \end{pmatrix} \right).$$

This vector also has the same support $I' \subseteq I$ as $\vec{c}_b$. The first component of $\vec{c}^t \cdot \mathbf{A}_\vee$ is $(2-b)\vec{c}_1^t \cdot \mathbf{A}_1^1 + (b-1)\vec{c}_2 \cdot \mathbf{A}_2^1$, which is $\vec{c}_1^t \cdot \mathbf{A}_1^1 = 1$ if $b = 1$, or $\vec{c}_2 \cdot \mathbf{A}_2^1 = 1$ if $b = 2$. The next block of $n_1 - 1$ components is $(2-b)\vec{c}_1^t \cdot \mathbf{A}_1^*$, which is clearly $(0, \ldots, 0)$ if $b = 2$, and $\vec{c}_1^t \cdot \mathbf{A}_1^* = (0, \ldots, 0)$ if $b = 1$. The last block of $n_2 - 1$ components is also also $(0, \ldots, 0)$.

In the other direction, in order to be able to distinguish s from a random value, given the coordinates of $\vec{\nu}$ in I, one must be able to find $\vec{c}$ such that $\vec{c}^t \cdot \mathbf{A}_\vee = (1, 0, \ldots, 0)$, with support I. Of course, we can split $\vec{c} = \begin{pmatrix} \vec{c}_1 \\ \vec{c}_2 \end{pmatrix}$.

$$\vec{c}^t \cdot \mathbf{A}_\vee = (\vec{c}_1^t \cdot \mathbf{A}_1^1 + \vec{c}_2^t \cdot \mathbf{A}_2^1, \vec{c}_1^t \cdot \mathbf{A}_1^*, \vec{c}_2^t \cdot \mathbf{A}_2^*).$$

As a consequence, $\vec{c}_1^t \cdot \mathbf{A}_1^1 + \vec{c}_2^t \cdot \mathbf{A}_2^1 = 1$ and $\vec{c}_1^t \cdot \mathbf{A}_1^* = (0, \ldots, 0)$ and $\vec{c}_2^t \cdot \mathbf{A}_2^*) = (0, \ldots, 0)$. At least, one of the two elements $\vec{c}_1^t \cdot \mathbf{A}_1^1$ or $\vec{c}_2^t \cdot \mathbf{A}_2^1$ is non-zero. Let us assume this is the first one, and it is equal to $\alpha \in \mathbb{K}^*$: the vector $\vec{c'}_1 \leftarrow \alpha^{-1} \cdot \vec{c}_1$ satisfies $\vec{c'}_1^t \cdot \mathbf{A}_1 = (1, 0, \ldots, 0)$. This vector has a support $I' \subseteq I$, and thus (from

the LSSS property) $\rho(I')$ must satisfy the policy p_1, and so does $\rho(I)$ because of the monotonicity.

Conjunctions. We assume that for some attributes A, there is I such that $\rho(I) \subset A$ satisfies the policy $p_1 \wedge p_2$. This also means there is $I_b \subseteq I$ such that $\rho(I_b) \subset A$ satisfies the policy p_b, for $b = 1, 2$: from the LSSS $(\mathbf{A}_b, \rho_b)$, there is $\vec{c}_b$ with support I_b such that $\vec{c}_b^t \cdot \mathbf{A}_b = (1, 0, \ldots, 0)$. Let us build $\vec{c} = \begin{pmatrix} \vec{c}_1 \\ \vec{c}_2 \end{pmatrix}$.

$$\begin{aligned} \vec{c}^t \cdot \mathbf{A}_\wedge &= (\vec{c}_1^t \cdot \mathbf{A}_1^1, \vec{c}_1^t \cdot \mathbf{A}_1^1 - \vec{c}_2^t \cdot \mathbf{A}_2^1, \vec{c}_1^t \cdot \mathbf{A}_1^*, -\vec{c}_2^t \cdot \mathbf{A}_2^*) \\ &= (1, 1-1, (0, \ldots, 0), (0, \ldots, 0)) = (1, 0, 0, \ldots, 0) \end{aligned}$$

This vector $\vec{c}$ has a support $I_1 \cup I_2 \subseteq I$.

In the other direction, in order to be able to distinguish s from a random value, given the coordinates of $\vec{\nu}$ in I, one must be able to find $\vec{c}$ such that $\vec{c}^t \cdot \mathbf{A}_\wedge = (1, 0, \ldots, 0)$. Of course, we can split $\vec{c} = \begin{pmatrix} \vec{c}_1 \\ \vec{c}_2 \end{pmatrix}$:

$$\vec{c}^t \cdot \mathbf{A}_\wedge = (\vec{c}_1^t \cdot \mathbf{A}_1^1, \vec{c}_1^t \cdot \mathbf{A}_1^1 - \vec{c}_2^t \cdot \mathbf{A}_2^1, \vec{c}_1^t \cdot \mathbf{A}_1^*, -\vec{c}_2^t \cdot \mathbf{A}_2^*).$$

Then $\vec{c}_1^t \cdot \mathbf{A}_1^1 = 1$ and $\vec{c}_1^t \cdot \mathbf{A}_1^1 - \vec{c}_2^t \cdot \mathbf{A}_2^1 = 0$, which also implies $\vec{c}_2^t \cdot \mathbf{A}_2^1 = 1$; $\vec{c}_1^t \cdot \mathbf{A}_1^* = (0, \ldots, 0)$ and $\vec{c}_2^t \cdot \mathbf{A}_2^* = (0, \ldots, 0)$.

We thus have both $\vec{c}_1^t \cdot \mathbf{A}_1 = (1, 0, \ldots, 0)$ and $\vec{c}_2^t \cdot \mathbf{A}_2 = (1, 0, \ldots, 0)$ and the supports of $\vec{c}_1$ and $\vec{c}_2$ are I_1 and I_2 included in I. From the LSSS property $\rho(I_b)$ must satisfy the policy p_b, for $b = 1, 2$, and so does $\rho(I)$ because of the monotonicity. As a consequence, $\rho(I)$ satisfies the policy $p_1 \wedge p_2$.

About the two constructions

To summarize, we illustrate here these methods to construct the LSSS for the policy $p = ((\mathsf{a}_1 \vee \mathsf{a}_2) \wedge (\mathsf{a}_3 \wedge \mathsf{a}_4)) \vee (((\mathsf{a}_5 \vee \mathsf{a}_6) \wedge \mathsf{a}_7) \vee \mathsf{a}_8)$.

Using Lewko-Waters' algorithm, we get the tree and the matrix described on Fig. 7.4. This algorithm explores the tree following nodes corresponding to binary gates (i) and associates vectors with values in $\{-1, 0, 1\}$ to the child nodes. The vectors linked to leaves form the final matrix. Using our combination of matrices, starting from the leaves, associated to the matrix $[1]$, and going go back to the root, we obtain the construction on Fig. 7.5.

The matrices obtained are equivalent: columns can just differ with the sign. Indeed, if there exists $\vec{c}$ such that $\vec{c}^t \cdot \mathbf{A} = (1, 0, \ldots, 0)$, changing the sign of a column of $\mathbf{A}$ does not impact the support of $\vec{c}$: changing the sign of the first column needs changing the sign of $\vec{c}$, while changing the sign of the other columns does not impact $\vec{c}$ at all. The matrices also have the same size: m rows for $n+1$ columns where m is the numbers of literals in the predicate p and n is the number of AND gates. If this is well known for the matrices made from Lewko-Waters's method, it is easy to see it with our method too, by induction. Let $\mathbf{A}_1$ and $\mathbf{A}_2$ be two LSSS matrices, with respectively n_1 and n_2 columns, for the policies p_1 and p_2, with respectively n_1 and n_2 AND gates. Following our construction, the new LSSS matrices $\mathbf{A}_\vee$ and $\mathbf{A}_\wedge$ have respectively $n_1 + n_2$ and $n_1 + n_2 + 1$ columns, and their associated policies $p_1 \vee p_2$ and $p_1 \wedge p_2$ respectively have $n_1 + n_2$ and $n_1 + n_2 + 1$ AND gates. Note that we start

from the atomic predicates $p = (a_i)$, and we assume them all being distinct.

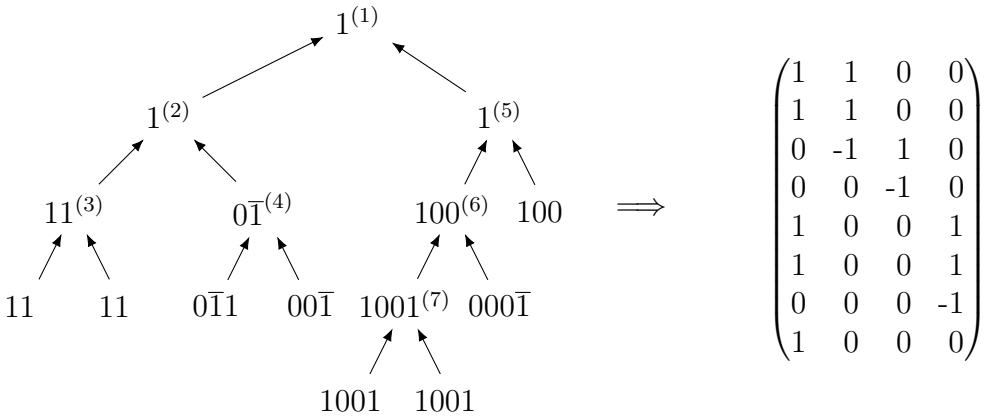

Figure 7.4: Lewko-Waters's construction applied on the policy p, using the method exposed in Fig. 7.3. The indexes represent the state of the run in the tree, and the resulting vectors are the leaves which are then filled with zeros to make the matrix. Also, we denote $\overline{1} := -1$ for the sake of clarity.

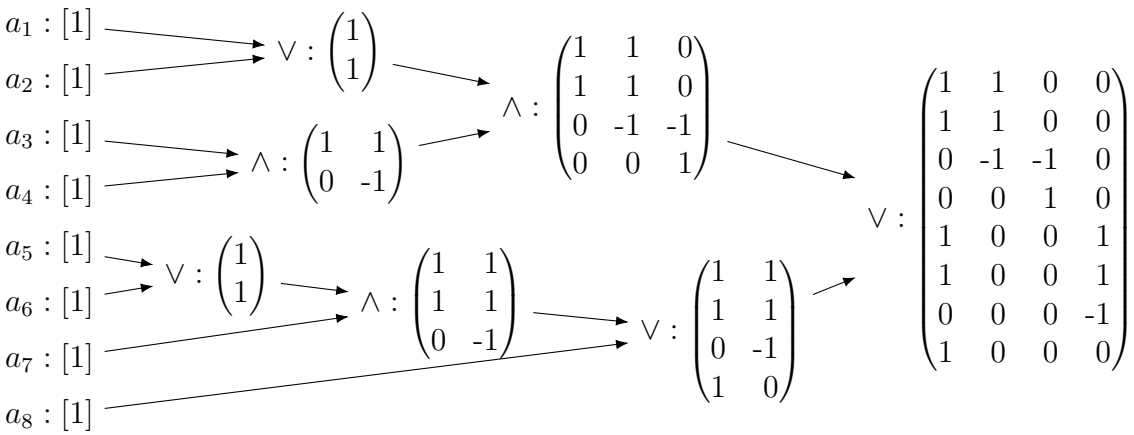

Figure 7.5: Matrix-based iterative construction of p, following the method described in Proposition 7.1.

7.3 Practical example

Finally, we provide an example that details the application of the iterative LSSS construction on an ABKEM to achieve homomorphic policy property. To this aim, we need an ABKEM scheme allowing linear combinations on the shares ν_i (where $\vec{\nu} = \mathbf{A} \cdot \vec{v}$) encoded in the encapsulations; it is necessary to make use of the LSSS properties. Thus, we present here a revised version of a CP-ABE scheme from [LW11]. First, for the sake of simplicity, we do not exploit the decentralized version and so all the attributes are managed by the same entity (but we could keep the decentralized

version). Second, for the homomorphic property, we consider a KEM instead of an encryption scheme, which just encaps a session key. However, we still use an LSSS to realize the access policy and pairing techniques to ensure collusion resistance. More precisely, we use a symmetric pairing $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$, where the groups $\mathbb{G}$ and $\mathbb{G}_T$ will be of composite order $N = q_1 q_2 q_3$, with three large prime integers q_1, q_2, and q_3. Let us describe our variant of ABKEM.

7.3.1 Description

- $\mathsf{SetUp}(\lambda)$: one first generates symmetric pairing groups $(\mathbb{G}, \mathbb{G}_T, [g], N, e) \leftarrow \mathsf{PGGen}(\lambda)$ of composite order $N = q_1 q_2 q_3$. One also generates a generator $[g_1]$ of the subgroup $\mathbb{G}_1 \subset \mathbb{G}$ of order q_1 and a hash function $\mathcal{H} : \{0,1\}^* \longrightarrow \mathbb{G}$. We also denote $G = e([g_1], [g_1]) = [g_1^2]_T \in \mathbb{G}_T$. Then, for each attribute a, the authority specifies the pair of secret/public keys, respectively $\mathsf{sk}_\mathsf{a} = (\alpha_\mathsf{a}, y_\mathsf{a})$ and $\mathsf{pk}_\mathsf{a} = (G_\mathsf{a} = \alpha_\mathsf{a} \cdot G, [g_\mathsf{a}] = y_\mathsf{a} \cdot [g_1])$. The master secret key msk is the concatenation of the sk_a's, and the public key pk contains N, $[g_1]$ and $\mathcal{H}$, together with the concatenation of the pk_a's.

- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}, \mathsf{a})$: from $\mathsf{msk} = \{\mathsf{sk}_\mathsf{a}\}$, id and a, the authority outputs $\mathsf{dk}_\mathsf{id}^\mathsf{a} = \alpha_\mathsf{a} \cdot [g_1] + y_\mathsf{a} \cdot \mathcal{H}(\mathsf{id})$.

- $\mathsf{Encaps}(\mathsf{pk}, P)$: from the public key pk and a set P of policies, one first chooses some random $s \xleftarrow{\$} \mathbb{Z}_N$ and sets the symmetric encapsulated key $K \leftarrow s \cdot G$. Then, for each $p \in P$, we process the following encapsulation: from the LSSS matrix $\mathbf{A} \in \mathbb{K}^{m \times n}$ and the associated labeling map ρ onto the attributes describing the access structure defined by the policy p, we set $\vec{v} = (s, v_2, \ldots, v_n)$ and $\vec{w} = (0, w_2, \ldots, w_n)$, with $v_k, w_k \xleftarrow{\$} \mathbb{Z}_N$ for $k = 2, \ldots, n$ and $\vec{r} \xleftarrow{\$} \mathbb{Z}_N^m$. We build the share vectors $\vec{\nu} = \mathbf{A} \cdot \vec{v}$ and $\vec{\omega} = \mathbf{A} \cdot \vec{w}$. Eventually, for each line $x \in \{1, \ldots, m\}$ of the matrix $\mathbf{A}$, we construct the encapsulation using the keys $\mathsf{pk}_{\rho(x)} = (G_{\rho(x)}, [g_{\rho(x)}])$ associated to the attribute $\mathsf{a}_x = \rho(x)$ involved in the policy p:

$$E_{1,x} = \nu_x \cdot G + r_x \cdot G_{\rho(x)} \qquad E_{2,x} = r_x \cdot [g_1] \qquad E_{3,x} = \omega_x \cdot [g_1] + r_x \cdot [g_{\rho(x)}]$$

  The algorithm returns $E_p = \{(E_{1,x}, E_{2,x}, E_{3,x})\}_x$ for each $p \in P$.

- $\mathsf{Decaps}(\mathsf{dk}_\mathsf{id}, E_p)$, where $\mathsf{dk}_\mathsf{id} = (\mathsf{dk}_\mathsf{id}^\mathsf{a})$ for the attributes owned by id: first, the user must find a vector $\vec{c} \in \mathbb{K}^m$ such that $\vec{c}^t \cdot \mathbf{A} = (1, 0, \ldots, 0)$ and the support I of the non-zero components of $\vec{c}$ links to a set of attributes owned by the user. Then, for each $x \in I$, the user computes $F_x = E_{1,x} + e(H(\mathsf{id}), E_{3,x}) - e(\mathsf{dk}_\mathsf{id}^{\rho(x)}, E_{2,x})$. He finally gets K by combining with the vector $\vec{c}$: $K \leftarrow \sum_{x \in I} c_x \cdot F_x$.

Correctness. The latter reconstruction works since:

$$\sum_{x \in I} c_x \cdot \nu_x = \sum_{x=1}^{m} c_x \cdot \nu_x = \langle \vec{c}, \vec{\nu} \rangle = \vec{c}^t \cdot \mathbf{A} \cdot \vec{v} = (1, 0, \ldots, 0) \cdot \vec{v} = s$$

$$\sum_{x \in I} c_x \cdot \omega_x = \sum_{x=1}^{m} c_x \cdot \omega_x = \langle \vec{c}, \vec{\omega} \rangle = \vec{c}^t \cdot \mathbf{A} \cdot \vec{w} = (1, 0, \ldots, 0) \cdot \vec{w} = 0$$

In addition, for each $x \in I$,

$$F_x = E_{1,x} + e(H(\mathsf{id}), E_{3,x}) - e(\mathsf{dk}_{\mathsf{id}}^{\rho(x)}, E_{2,x}) = \nu_x \cdot G + \omega_x \cdot e(H(\mathsf{id}), [g_1]).$$

And so, the final combination leads to

$$\begin{aligned}\sum_{x \in I} c_x \cdot F_x &= \sum_{x \in I} c_x \cdot (\nu_x \cdot G + \omega_x \cdot e(H(\mathsf{id}), [g_1])) \\ &= \langle \vec{c}, \vec{\nu} \rangle \cdot G + \langle \vec{c}, \vec{\omega} \rangle \cdot e(H(\mathsf{id}), [g_1]) = s \cdot G.\end{aligned}$$

Note that this scheme produces ciphertexts decryptable when at least k attributes overlapped between a ciphertext and a private key. While they showed that this primitive is useful for error-tolerant encryption with biometrics, the lack of expressibility limits its applicability when more general policy are required. Consequently, the map ρ needs to be an injection. In practice, this is not a real issue, since one can simply duplicate the attributes and provide multiple keys to users.

7.3.2 Security analysis

In [LW11], Lewko and Waters proved their ABE scheme to be indistinguishable under several assumptions in the composite-order pairing setting recalled here (Section 7.3.2), and the condition that ρ is injective. This easily leads to the IND security for the above variant of ABKEM, even for adaptive KeyGen-queries. Hence, this ABKEM construction achieves the IND security level. We now show that, the m-IND-NKA security of the modified ABKEM can also be based on the IND security of Lewko-Waters scheme.

Theorem 7.1

The IND security level of Lewko-Waters implies the m-IND-NKA security of the modified ABKEM.

Proof

As highlighted in Fig. 7.6, the two security games are quite similar, the main differences appear in the challenge phase, and the lack of key-queries in the latter. If one looks at the above construction of the LSSS-matrix, for $p = \mathsf{a}_1 \vee \ldots \vee \mathsf{a}_k$, then $\mathbf{A} = (1, \ldots, 1)^t$ and $\vec{\nu} = (s, \ldots, s)^t$: from an encapsulation E of the key $K = s \cdot G$ under the policy p, one can easily extract the encapsulations E_i of the same K, under the policies $p_i = (\mathsf{a}_i)$ respectively: indeed, each triple $(E_{1,x}, E_{2,x}, E_{3,x})$ is a simple encapsulation of K under $\mathsf{a}_x = \rho(x)$.

This remark is true for every conjunction $p_f = \bigvee p_i$ where the policies p_i's do not share any attribute. Note that the triples $(E_{1,x}, E_{2,x}, E_{3,x})$ involved in the decryption of a policy p_i are those associated to the attributes which appears in this policy. The choice of these triples is given by the vector c. Consequently, we can easily convert the challenger's answer from one game to another by concatenating/separating the ciphertext(s) by following this policy decomposition. Because of the lack of key-queries in the m-IND-NKA security game, we can just build an adversary $\mathcal{B}$ for the IND game from an adversary $\mathcal{A}$ of the m-IND-NKA game. More precisely, if an adversary $\mathcal{A}$ has an advantage $\mathsf{Adv}^{\mathtt{m\text{-}ind\text{-}nka}}(\mathcal{A}) = \varepsilon$ in the m-IND-NKA game for the policies $(p_j)_j$, one can

construct an adversary $\mathcal{B}$ with the same advantage $\mathsf{Adv}^{\mathtt{ind}}(\mathcal{B}) = \varepsilon$ in the `IND` game for the policy $p_f = \bigvee p_i$.

	`IND` game	`m-IND-NKA` game
$\underline{\mathsf{SetUp}(\lambda)}$:	$(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$	$(\mathsf{msk}, \mathsf{pk}) \leftarrow \mathsf{SetUp}(\lambda)$
$\underline{\mathsf{QKeyGen}(\mathsf{id}_i, \mathsf{a}_i)}$:	$\mathsf{dk}_{id_i}^{\mathsf{a}_i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}_i, \mathsf{a}_i)$	none
$\underline{\mathsf{Challenge}(p)}$:	$p = \bigvee p_i$ $(K, E) \leftarrow \mathsf{Encaps}(\mathsf{pk}, p)$ $K_b \leftarrow K$, $K_{1-b} \xleftarrow{\$} \mathcal{K}$ (E, K_0, K_1)	$p = (p_j)_j$ $(K, (E_j)_j) \leftarrow \mathsf{Encaps}(\mathsf{pk}, (p_j)_j)$ $K_b \leftarrow K$, $K_{1-b} \xleftarrow{\$} \mathcal{K}$ $((E_j)_j, K_0, K_1)$
$\underline{\mathsf{QKeyGen}(\mathsf{id}_i, \mathsf{a}_i)}$:	$\mathsf{dk}_{id_i}^{\mathsf{a}_i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}_i, \mathsf{a}_i)$	none
$\underline{\mathsf{Finalize}}$:	(E, C_β, M_0, M_1)	$((E_j)_j, C_\beta, M_0, M_1)$

Figure 7.6: Comparison between the original `IND` security game for ABKEM and the `m-IND-NKA` game. The p_i's are different policies that do not share any attribute, and $\mathcal{K}$ is the key space.

Remark. As already noted, Lewko and Waters [LW11] assume a one-use restriction on attributes throughout the proof: this means that the row-labeling map ρ of the challenge ciphertext access matrix $(\mathbf{A}, \rho)$ must be injective. The reason is that, if an attribute is used twice in the access matrix, then there will appear an implicit relation between the randomnesses associated to the corresponding two lines of the matrix and the proof does not go through anymore. To overcome this issue, Lewko and Waters suggested to associate k independent attributes to any attribute a, where k is an upper-bound on the number of repetitions of an attribute in a policy. Thus, it is important to note our ABKEM derivation inherently has the same limitation.

Composite group assumptions

While we do not recall the proof of the original CP-ABE scheme from [LW11], we describe the assumptions used. Unlike the other work presented in this thesis, this scheme uses non standard assumptions, based on pairing groups with an order $N = p_1 p_2 p_3$ where the p_i are primes.

Definition 7.5: Composite group assumption 1

Given a group generator $\mathcal{G}$, let be the following distribution:

$$\mathbb{G} = (N = p_1 p_2 p_3, e : G \times G \to G_T) \xleftarrow{\$} \mathcal{G}$$

$$g_1 \xleftarrow{\$} G_{p_1}$$

$$D = (\mathbb{G}, g_1)$$
$$T_0 \xleftarrow{\$} G, T_1 \xleftarrow{\$} G_{p_1}$$

This assumption says that with the distribution D but without the knowledge of the factorization of N, we can't decide if $T_\beta \xleftarrow{\$} \{T_0, T_1\}$ is in G or G_{p_1}.

Definition 7.6: Composite group assumption 2

Given a group generator $\mathcal{G}$, let be the following distribution:

$$\mathbb{G} = (N = p_1 p_2 p_3, e : G \times G \to G_T) \xleftarrow{\$} \mathcal{G}$$
$$g_1, X_1 \xleftarrow{\$} G_{p_1}, X_2 \xleftarrow{\$} G_{p_2}, g_3 \xleftarrow{\$} G_{p_3}$$
$$D = (\mathbb{G}, g_1, g_3, X_1 X_2)$$
$$T_0 \xleftarrow{\$} G_{p_1}, T_1 \xleftarrow{\$} G_{p_1 p_2}$$

This assumption says that with the distribution D but without the knowledge of the factorization of N, we can't decide if $T_\beta \xleftarrow{\$} \{T_0, T_1\}$ is in G_{p_1} or $G_{p_1 p_2}$.

Definition 7.7: Composite group assumption 3

Given a group generator $\mathcal{G}$, let be the following distribution:

$$\mathbb{G} = (N = p_1 p_2 p_3, e : G \times G \to G_T) \xleftarrow{\$} \mathcal{G}$$
$$g_1, X_1 \xleftarrow{\$} G_{p_1}, Y_2 \xleftarrow{\$} G_{p_2}, X_3, Y_3 \xleftarrow{\$} G_{p_3}$$
$$D = (\mathbb{G}, g_1, X_1 X_3, Y_2 Y_3)$$
$$T_0 \xleftarrow{\$} G_{p_1 p_2}, T_1 \xleftarrow{\$} G_{p_1 p_3}$$

This assumption says that with the distribution D but without the knowledge of the factorization of N, we can't decide if $T_\beta \xleftarrow{\$} \{T_0, T_1\}$ is in $G_{p_1 p_2}$ or $G_{p_1 p_3}$.

Definition 7.8: Composite group assumption 4

Given a group generator $\mathcal{G}$, let be the following distribution:

$$\mathbb{G} = (N = p_1 p_2 p_3, e : G \times G \to G_T) \xleftarrow{\$} \mathcal{G}$$
$$g_1, X_1 \xleftarrow{\$} G_{p_1}, g_2 \xleftarrow{\$} G_{p_2}, g_3 \xleftarrow{\$} G_{p_3}, (a, b, c, d) \xleftarrow{\$} \mathbb{Z}_N$$
$$D = (\mathbb{G}, g_1, g_2, g_3, g_1^a, g_1^b g_3^b, g_1^c, g_1^{ac} g_3^d)$$
$$T_0 = e(g_1, g_1)^{abc}, T_1 \xleftarrow{\$} G_T$$

This assumption says that with the distribution D but without the knowledge of the factorization of N, we can't decide if $T_\beta \xleftarrow{\$} \{T_0, T_1\}$ is in the G_T or a subgroup of G_T generated by $e(g_1, g_1)$.

7.3.3 Achieving homomorphic policy

Eventually, we now show how the iterative construction of the LSSS can be exploited in our ABKEM (Section 7.3.1) to fulfill the homomorphic-policy property.

We recall that in this scheme, $\vec{\nu} = \mathbf{A} \cdot \vec{v}$ is a secret sharing of a random scalar s, while $\vec{\omega} = \mathbf{A} \cdot \vec{w}$ is a secret sharing of 0, the components ν_x and ω_x being hidden in $E_{1,x}$ and $E_{3,x}$ by $G_{\rho(x)}^{r_x}$ and $g_{\rho(x)}^{r_x}$ respectively. Also, note that the encapsulated key is not exactly s but $s \cdot G$.

Shares combinations

We first describe the share combinations that are needed to achieve our goal; because of the linear property of the LSSS, by concatenating or by adding the shares, we either obtain the OR or the AND policies of two encapsulations $E^{(1)}$ and $E^{(2)}$:

Share-Vectors		**Encapsulations**
$\begin{pmatrix}\vec{\nu}_1\\ \vec{\nu}_2\end{pmatrix}$	$\longleftrightarrow$	$E^{(1)} \cup E^{(2)}$
$\vec{\nu}_1 + \vec{\nu}_2$	$\longleftrightarrow$	$E^{(1)} + E^{(2)}$

Of course, the same applies on the shares $\vec{\omega}$ of 0, but we focus on the shares $\vec{\nu}$ of the random s

One secret under two policies. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of the same secret value $K = s \cdot G$ under the policies p_1 and p_2, represented by the LSSS $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$. The construction thus used the share-vectors $\vec{\nu}_i = (\nu_{i,1}, \ldots, \nu_{i,m_i}) = \mathbf{A}_i \cdot \vec{v}_i$, with $\vec{v}_i = (s, v_{i,2}, \ldots, v_{i,n_i})^t$, for $i = 1, 2$. Using

$$\mathbf{A}_\vee = \begin{pmatrix}\mathbf{A}_1^1 & \mathbf{A}_1^* & 0\\ \mathbf{A}_2^1 & 0 & \mathbf{A}_2^*\end{pmatrix} \text{ and } \vec{v} = (s, v_{1,2}, \ldots, v_{1,n_1}, v_{2,2}, \ldots, v_{2,n_2})^t,$$

one gets $\vec{\nu} = \begin{pmatrix}\vec{\nu}_1\\ \vec{\nu}_2\end{pmatrix}$. From attributes satisfying p_i, under the LSSS property, one can efficiently find a vector $\vec{c}_i = (c_{i,1}, \ldots, c_{i,m_i})^t \in \mathbb{K}^m$ such that $\vec{c}_i^t \cdot \mathbf{A}_i = (1, 0, \ldots, 0)$. By multiplying this vector on the appropriate half of $\vec{\nu}$, one can get s:

$$(c_{1,1}, \ldots, c_{1,m_1}, 0, \ldots, 0) \cdot \vec{\nu} = \vec{c}_1^t \cdot \vec{\nu}_1 = s$$
$$(0, \ldots, 0, c_{2,1}, \ldots, c_{2,m_2}) \cdot \vec{\nu} = \vec{c}_2^t \cdot \vec{\nu}_2 = s.$$

It will be used for the disjunction of policies in Section 7.3.3.

Two secrets under different policies. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of two secret values $K_1 = s_1 \cdot G$ and $K_2 = s_2 \cdot G$ under the policies p_1 and p_2, represented by the LSSS $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$. The construction thus used the share-vectors $\vec{\nu}_i = (\nu_{i,1}, \ldots, \nu_{i,m_i}) = \mathbf{A}_i \cdot \vec{v}_i$, with $\vec{v}_i = (s_i, v_{i,2}, \ldots, v_{i,n_i})^t$, for $i = 1, 2$. Using

$$\mathbf{A}_\wedge = \begin{pmatrix}\mathbf{A}_1^1 & \mathbf{A}_1^1 & \mathbf{A}_1^* & 0\\ 0 & -\mathbf{A}_2^1 & 0 & -\mathbf{A}_2^*\end{pmatrix} \text{ and } \vec{v} = (s_1 + s_2, -s_2, v_{1,2}, \ldots, v_{1,n_1}, v_{2,2}, \ldots, v_{2,n_2})^t,$$

one gets again $\vec{\nu} = \begin{pmatrix}\vec{\nu}_1\\ \vec{\nu}_2\end{pmatrix}$. This combination will be used for the conjunction of policies in Section 7.3.3, but only with the same secret. Note that the produced encapsulation

must be randomized to perform the new policy, otherwise there is a colluding attack: with independent keys for each policy, two players can independently get s_1 and s_2, and can then combine them to get $s_1 + s_2$.

Two secrets under the same policy. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of two secret values $K_1 = s_1 \cdot G$ and $K_2 = s_2 \cdot G$ under the same policy p, represented by the LSSS $(\mathbf{A}, \rho)$. The construction thus used the share-vectors $\vec{\nu}_1$ and $\vec{\nu}_2$ of the random scalars s_1 and s_2 respectively under the same policy p. Then, one can see $\vec{\nu} = \vec{\nu}_1 + \vec{\nu}_2$ as a share-vector of $s = s_1 + s_2$ under the policy p, since $\vec{\nu} = \mathbf{A} \cdot (\vec{v}_1 + \vec{v}_2)$. Indeed, from attributes satisfying p, one can efficiently find a vector $\vec{c} \in \mathbb{K}^m$ such that $\vec{c}^t \cdot \mathbf{A} = (1, 0, \ldots, 0)$:

$$\vec{c}^t \cdot \vec{\nu} = \vec{c}^t \cdot \mathbf{A} \cdot (\vec{v}_1 + \vec{v}_2) = (1, 0 \ldots, 0) \cdot (\vec{v}_1 + \vec{v}_2) = s_1 + s_2.$$

This combination will be used for the randomization in Section 7.3.3, with $s_2 = 0$.

Encapsulations combinations

Let us now see how this impacts on the encapsulations, when one wants to do disjunctions and conjunctions of policies.

Disjunctions. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of the same key $K = s \cdot G$ under the policies p_1 and p_2, represented by the LSSS $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$. We want to make an encapsulation of K under the policy $p_1 \vee p_2$. Using the construction of the share-vectors from Section 7.3.3, which applies on both $\vec{\nu}_1, \vec{\nu}_2$ and $\vec{\omega}_1, \vec{\omega}_2$, we know that the resulting encapsulation should use

$$\vec{\nu} = \begin{pmatrix} \vec{\nu}_1 \\ \vec{\nu}_2 \end{pmatrix} \quad \vec{\omega} = \begin{pmatrix} \vec{\omega}_1 \\ \vec{\omega}_2 \end{pmatrix}.$$

Therefore, the resulting encapsulation is $E_{p_1 \vee p_2} = \{(E^{(1)}_{j,x}, E^{(2)}_{j,x})_{j=1,2,3}\}_{x \in \mathfrak{A}}$.

Conjunctions. Let us be given two encapsulations $E^{(1)}$ and $E^{(2)}$ of the same key $K = s \cdot G$ under the policies p_1 and p_2, represented by the LSSS $(\mathbf{A}_1, \rho_1)$ and $(\mathbf{A}_2, \rho_2)$. We want to make an encapsulation of K under the policy $p_1 \wedge p_2$. Using the construction of the share-vectors from Section 7.3.3, which applies on both $\vec{\nu}_1, \vec{\nu}_2$ and $\vec{\omega}_1, \vec{\omega}_2$, we know that the resulting encapsulation should use

$$\vec{\nu} = \begin{pmatrix} \vec{\nu}_1 \\ \vec{\nu}_2 \end{pmatrix} \quad \vec{\omega} = \begin{pmatrix} \vec{\omega}_1 \\ \vec{\omega}_2 \end{pmatrix}.$$

However, this will contain the key $2 \cdot K = 2s \cdot G$. We thus have to halve: the resulting encapsulation is $E_{p_1 \wedge p_2} = \{((\frac{1}{2} \cdot E^{(1)}_{j,x}), (\frac{1}{2} \cdot E^{(2)}_{j,x}))_{j=1,2,3}\}_{x \in \mathfrak{A}}$. Note that even if in the Lewko-Waters' construction there is a modulus $N = q_1 q_2 q_3$ that is hard to factor, this is the order of the group. Hence $\frac{1}{2} \cdot [g] = \alpha \cdot [g]$ where $\alpha = (N+1)/2$.

As already noted, collusion is possible. But this is even worse in this case since we are using $s = s_1 = s_2$: just satisfying one of the two policies, one can recover $\frac{1}{2} \cdot K = \frac{s}{2} \cdot G$, which thereafter easily leads to K. We thus need to randomize the encapsulation, in order to glue together the policies.

Randomization. If one looks in details the description of the Encaps algorithm, there are 4 kinds of randomness:

- s, that defined the encapsulated key $K = s \cdot G$;
- $v_k, w_k \xleftarrow{\$} \mathbb{Z}_N$ for $k = 2, \ldots, n$, to define $\vec{v}$ and $\vec{w}$;
- $\vec{r} \xleftarrow{\$} \mathbb{Z}_N^m$.

Let us start from any encapsulation $E^{(1)}$ of K under a policy p, with

$$E^{(1)}_{1,x} = \nu^{(1)}_x \cdot G + r^{(1)}_x \cdot G_{\rho(x)} \quad E^{(1)}_{2,x} = r^{(1)}_x \cdot [g_1] \quad E^{(1)}_{3,x} = \omega^{(1)}_x \cdot [g_1] + r^{(1)}_x \cdot [g_{\rho(x)}]$$

for each $\mathsf{a}_x = \rho(x)$ involved in the policy p, where $\vec{\nu}^{(1)} = \mathbf{A} \cdot \vec{v}^{(1)}$ and $\vec{\omega}^{(1)} = \mathbf{A} \cdot \vec{w}^{(1)}$. We now define a new fresh encapsulation $E^{(2)}$:

$$E^{(2)}_{1,x} = \nu^{(2)}_x \cdot G + r^{(2)}_x \cdot G_{\rho(x)} \quad E^{(2)}_{2,x} = r^{(2)}_x \cdot [g_1] \quad E^{(2)}_{3,x} = \omega^{(2)}_x \cdot [g_1] + r^{(2)}_x \cdot [g_{\rho(x)}]$$

where $\vec{\nu}^{(2)} = \mathbf{A} \cdot \vec{v}^{(2)}$ and $\vec{\omega}^{(2)} = \mathbf{A} \cdot \vec{w}^{(2)}$, for $\vec{v}^{(2)} = (0, v'_2, \ldots, v'_n)^t$ and $\vec{w}^{(2)} = (0, w'_2, \ldots, w'_n)^t$, with $v'_k, w'_k \xleftarrow{\$} \mathbb{Z}_N$ for $k = 2, \ldots, n$, and $\vec{r}^{(2)} \xleftarrow{\$} \mathbb{Z}_N^m$. This is actually a fresh random encapsulation of $K^{(2)} = e([g], [g]) = [1]_T$ under the policy p. It can be computed from the public key pk that contains N, $[g_1]$, and the keys $\mathsf{pk}_\mathsf{a} = (G_\mathsf{a}, [g_\mathsf{a}])$, for all the attributes, as would be generated a fresh encapsulation of $K = [1]_T$. Eventually, the new encapsulation $E = \{(E^{(1)}_{1,x} + E^{(2)}_{1,x}, E^{(1)}_{2,x} + E^{(2)}_{2,x}, E^{(1)}_{3,x} + E^{(2)}_{3,x})\}_x$ is a truly random encapsulation of the same K under the policy p, and so looks like a fresh encapsulation.

8 Conclusion

This thesis presented two ways to delegate the access control to encrypted data, both based on functional encryption.

The first chapters describe how to allow a distributed control of data analysis with the decentralization of multi-client functional encryption; after defining the main definitions and security notions in Chapter 3, we provide practical schemes in Chapter 4. Finally, we describe black-box combinations or generic ideas to achieve a non-interactive decentralized construction and improve security, respectively in Chapter 6 and Chapter 5. All this, with standard assumptions in the random oracle model.

The last chapter (Chapter 7) describes a contribution on a different delegation problem. It allows to separate the roles of the sender and the access right manager. This is a quite useful property for the Pay-TV context, since the access right manager does not have access anymore to the content payload. The distribution to the subscribers can be performed by a weakly trusted party.

Open questions. Through MCFE, we aim to build analysis tools for aggregated data. Our work makes an important step towards this goal, but also left open several questions:

- A first point is the security. Our work is based on groups, uses pairings, and is only proven secure in the random oracle model. It would be interesting to know if we can generalize the ideas presented in this thesis to other assumptions, especially lattices-based. Getting rid of the random oracle model also seems to be an important gap to pass.

  In Section 4.1.2 we cite an ongoing work [LT19] that deals with these two points. This article describes a scheme built with something close to a KH-PRF, based on Learning With Errors, and is proven secure in the standard model. However, it implies an heavy trade-off on the practicability.

- As noticed in the Section 4.1.2, our IP-MCFE constructions are based on groups where the discrete logarithm problem holds, and the size of the numbers we encrypt is limited by the necessity to solve one in the decryption. At first sight, this constraint may not be problematic for financial or marketing applications, where they work on relatively little values. However it is a real limitation for cryptographic applications since one can not re-use the functionality as a black-box.

A solution to re-use the framework described in this thesis could be using groups that allow discrete logarithm on sub-groups. Thus, the use of groups from Paillier's cryptosystem [Pai99] $(\mathbb{Z}/N^2\mathbb{Z})^*$ seems legitimate. More recently, another recent work [CLT18] uses specific kind of groups with this property in an IP-FE context.

- A problem not tackled in this work is the dynamic setting. In a practical application, an external user could manifest his wish to join the protocol. This case involves several problems, the first being the creation of the secrets for this new user. A joining algorithm is equivalent to a personal set-up, and must not concern the other participants, so we must imagine a fully decentralized, non-interactive set-up from the beginning. Note that our work offers this possibility. Also, it is the goal of the recent paper [ACF+19] in the MIFE context.

  Another point related to the dynamic setting is the forward secrecy. When a new user is added to an existing group, should he be able to apply a decryption key on the old ciphertexts. In the other way, should we allow a user with an old decryption key to work on a ciphertext from a more recent set of participants? We could add the group as input to the decryption key generation and encryption algorithms to handle these questions, or maybe pair the MCFE with a finer grained access control, similarly to [DP19].

- Another functionality to study is the function-hiding. Briefly explained, we can suppose a case where a user is given a decryption key that does not leak the function encoded inside. This is already possible with the MIFE through the paper [ACF+18], and the technique seems applicable to MCFE. However it remains a problem in the decentralized context.

  In the case of the DMCFE, even if the users do not know the precise function when they make the decryption key, they must use a common time stamp to forbid combination of pieces from different keys. Consequently, no one is fully aware of the function, the key requester only knows the time stamp, and the senders know this time stamp and their personal values encoded in their key parts.

- Last point, but not the least, is about the others function we could potentially describe through MCFE. In this work we described a framework principally aiming inner product functionality, and a natural extension would be higher degree polynomials. We know today, since the paper [LT17], that trilinear maps would imply strong theoretical results in cryptography. Furthermore, functional encryption for quadratic functions is now known possible for single input functional encryption through the work [BCFG17], a scheme that relies on the use of pairings. Therefore, the important question is to know if quadratic functionality is possible in a (decentralized) multi-client context.

  In an attempt to answer positively, an idea to explore is to re-use an additive mask $\vec{c}_\ell = \vec{x} + \vec{s}_\ell$, apply the function A: $\vec{c}_\ell \cdot A \cdot \vec{c}_\ell$ and use function-hiding to remove the $\vec{x} \cdot A \cdot \vec{s}_\ell$ terms, while we take away $\vec{s}_\ell A \vec{s}_\ell$ using a key, analogously to IP-FE schemes.

Finally, on homomorphic-policy, we leave open the question of a generalization of the method used in [CPP17] to others ABE schemes with more standard assumptions.

Abbreviations

PPT	Probabilistic Polynomial Time
IND	Indistinguability
CCA	Chosen Cipher Attack
GGen/PGGen	(Pairing-friendly) Group Generator
CDH	Computational Diffie Hellman
DDH	Decisional Diffie Hellman
SXDH	Symmetric eXternal Diffie Hellman
DBDH	Decisional Bilinear Diffie Hellman
PRF/KH-PRF	(Key Homomorphic) Pseudo Random Function
LSSS	Linear Secret Sharing Scheme
SSE	Secret Sharing Encapsulation
SKE	Symmetric Key Encryption
FE/MIFE	(Multi-Input) Functional Encryption
MCFE/DMCFE	(Decentralized) Multi-Client Functional Encryption
IP	Inner Product
MPC	Multi-Party Computation
PSA	Private Stream Aggregation
DSum	Distributed Sum
ABE/CP-ABE	(Ciphertext-Policy) Attribute-Based Encryption
KEM/ABKEM	(Attribute-Based) Key Encapsulation Mechanism
HP	Homomorphic-Policy

Figure list

Bibliography

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015.

[ABG19] Michel Abdalla, Fabrice Benhamouda, and Romain Gay. From single-input to multi-client inner-product funtional encryption. ASIACRYPT 2019, 2019.

[ABKW19] Michel Abdalla, Fabrice Benhamouda, Markulf Kohlweiss, and Hendrik Waldner. Decentralizing inner-product functional encryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 128–157. Springer, Heidelberg, April 2019.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015.

[ACF+18] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 597–627. Springer, Heidelberg, August 2018.

[ACF+19] Shweta Agrawal, Michael Clear, Ophir Frieder, Sanjam Garg, Adam O'Neill, and Justin Thaler. Ad hoc multi-input functional encryption. Cryptology ePrint Archive, Report 2019/356, 2019. https://eprint.iacr.org/2019/356.

[AGRW17] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input inner-product functional encryption from pairings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 601–626. Springer, Heidelberg, April / May 2017.

[AL10] Nuttapong Attrapadung and Benoît Libert. Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 384–402. Springer, Heidelberg, May 2010.

[ALdP11] Nuttapong Attrapadung, Benoît Libert, and Elie de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 90–108. Springer, Heidelberg, March 2011.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016.

[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. Springer, Heidelberg, August 2017.

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997.

[BDK+11] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2011.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[BGJS16] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. Verifiable functional encryption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 557–587. Springer, Heidelberg, December 2016.

[BJL16] Fabrice Benhamouda, Marc Joye, and Benoît Libert. A new framework for privacy-preserving aggregation of time-series data. *ACM Trans. Inf. Syst. Secur.*, 18(3):10:1–10:21, 2016.

[BKS16] Zvika Brakerski, Ilan Komargodski, and Gil Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 852–880. Springer, Heidelberg, May 2016.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, August 2013.

[Boy99] Victor Boyko. On the security properties of OAEP as an all-or-nothing transform. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 503–518. Springer, Heidelberg, August 1999.

[BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 353–370. Springer, Heidelberg, August 2014.

[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. Springer, Heidelberg, March 2011.

[CCL+13] Cheng Chen, Jie Chen, Hoon Wei Lim, Zhenfeng Zhang, Dengguo Feng, San Ling, and Huaxiong Wang. Fully secure attribute-based systems with short ciphertexts/signatures and threshold access structures. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 50–67. Springer, Heidelberg, February / March 2013.

[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. Springer, Heidelberg, December 2018.

[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/2019/020.

[CDH+00] Ran Canetti, Yevgeniy Dodis, Shai Halevi, Eyal Kushilevitz, and Amit Sahai. Exposure-resilient functions and all-or-nothing transforms. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 453–469. Springer, Heidelberg, May 2000.

[CLT18] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 733–764. Springer, Heidelberg, December 2018.

[CPP17] Jérémy Chotard, Duong Hieu Phan, and David Pointcheval. Homomorphic-policy attribute-based key encapsulation mechanisms. In Phong Q. Nguyen and Jianying Zhou, editors, *ISC 2017*, volume 10599 of *LNCS*, pages 155–172. Springer, Heidelberg, November 2017.

[CSS12] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In Angelos D. Keromytis, editor, *FC 2012*, volume 7397 of *LNCS*, pages 200–214. Springer, Heidelberg, February / March 2012.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DOT18] Pratish Datta, Tatsuaki Okamoto, and Junichi Tomida. Full-hiding (unbounded) multi-input inner product functional encryption from the k-linear assumption. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 245–277. Springer, Heidelberg, March 2018.

[DP19] Edouard Dufour Sans and David Pointcheval. Unbounded inner-product functional encryption, with succinct keys. ACNS 2019, 2019. https://eprint.iacr.org/2018/487.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013.

[Emu17] Keita Emura. Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. In *Australasian Conference on Information Security and Privacy*, pages 193–213. Springer, 2017.

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Heidelberg, August 1994.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999.

[Gay16] Romain Gay. Functional encryption for quadratic functions, and applications to predicate encryption. Cryptology ePrint Archive, Report 2016/1106, 2016. http://eprint.iacr.org/2016/1106.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[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. Springer, Heidelberg, May 2014.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GJPS08] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 579–591. Springer, Heidelberg, July 2008.

[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. http://eprint.iacr.org/2013/774.

[GKP+13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013.

[GKP+13b] 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. ACM Press, June 2013.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HLR10] Javier Herranz, Fabien Laguillaumie, and Carla Ràfols. Constant size ciphertexts in threshold attribute-based encryption. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 19–34. Springer, Heidelberg, May 2010.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC*, pages 12–24. ACM Press, May 1989.

[JL13] Marc Joye and Benoît Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In Ahmad-Reza Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, pages 111–125. Springer, Heidelberg, April 2013.

[KDK11] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. In Simone Fischer-Hübner and Nicholas Hopper, editors, *Privacy Enhancing Technologies - 11th International Symposium, PETS '11*, volume 6794 of *LNCS*, pages 175–191. Springer, 2011.

[KW19] Lucas Kowalczyk and Hoeteck Wee. Compact adaptively secure ABE for $\mathsf{NC}s^1$ from k-lin. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 3–33. Springer, Heidelberg, May 2019.

[LC12] Qinghua Li and Guohong Cao. Efficient and privacy-preserving data aggregation in mobile sensing. In *ICNP 2012*, pages 1–10. IEEE Computer Society, 2012.

[LC13] Qinghua Li and Guohong Cao. Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In Emiliano De Cristofaro and Matthew K. Wright, editors, *PETS 2013*, volume 7981 of *LNCS*, pages 60–81. Springer, Heidelberg, July 2013.

[LL16] Kwangsu Lee and Dong Hoon Lee. Two-input functional encryption for inner products from bilinear maps. *IACR Cryptology ePrint Archive*, 2016:432, 2016.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 62–91. Springer, Heidelberg, May / June 2010.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 630–660. Springer, Heidelberg, August 2017.

[LT19] Benoit Libert and Radu Titiu. Multi-client functional encryption for linear functions in the standard model from lwe. ASIACRYPT 2019, 2019.

[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. Springer, Heidelberg, May 2011.

[NN04] Ventzislav Nikov and Svetla Nikova. New monotone span programs from old. Cryptology ePrint Archive, Report 2004/282, 2004. http://eprint.iacr.org/2004/282.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/2010/556.

[OSW07] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 195–203. ACM Press, October 2007.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 191–208. Springer, Heidelberg, August 2010.

[OT12] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure unbounded inner-product and attribute-based encryption. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 349–366. Springer, Heidelberg, December 2012.

[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. Springer, Heidelberg, May 1999.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Riv97] Ronald L. Rivest. All-or-nothing encryption and the package transform. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 210–218. Springer, Heidelberg, January 1997.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[RW13] Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 463–474. ACM Press, November 2013.

[SCR+11] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *NDSS 2011*. The Internet Society, February 2011.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[Wat15] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 678–697. Springer, Heidelberg, August 2015.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 616–637. Springer, Heidelberg, February 2014.

[YAHK14] Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. A framework and compact constructions for non-monotonic attribute-based encryption. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 275–292. Springer, Heidelberg, March 2014.

Résumé

Le chiffrement fonctionnel est un paradigme récent qui généralise le chiffrement à clef publique classique. Cette formalisation a pour objectif de réguler plus finement le contrôle d'accès aux données chiffrées, ainsi que l'information dévoilée par le déchiffrement. Cette thèse étudie des possibilités de délégation au travers de ces deux aspects.

Dans un premier temps, nous travaillons dans un contexte multi-client : plusieurs utilisateurs fournissent chacun une donnée personnelle chiffrée, et une entité souhaite extraire de l'information de ces données. Notre contribution consiste à permettre à ces utilisateurs de donner, ou refuser, leur accord à cette extraction. Pour ce faire, nous décrivons des constructions de chiffrement fonctionnel multi-utilisateur, puis nous definissons plusieurs niveaux de sécurité et fournissons des méthodes pour les atteindre. Enfin, principal objectif de ces travaux, nous décentralisons la fabrication de la clef de déchiffrement, pour qu'une personne souhaitant une clef de déchiffrement ait besoin de l'accord de tous pour l'obtenir. Toutes les instantiations proposées dans ces travaux sont utilisables en pratique.

Dans second temps, nous considérons une autre problématique dans laquelle un producteur de contenu vidéo cherche à déléguer la distribution de sa création, sans la révéler. Notre solution est un mécanisme d'encapsulation de clef, dérivé du chiffrement par attributs, avec une propriété particulière. Ce producteur l'utilise pour encapsuler la clef du flux vidéo sous plusieurs attributs, et fournit les encapsulations au distributeur. Celui-ci peut alors utiliser la propriété pour combiner les encapsulations et en définir les conditions d'accès à sa guise.

Mots clés

cryptographie, chiffrement fonctionnel, délégation, multi-utilisateur, décentralisation

Abstract

Functional encryption is a recent paradigm that generalizes the classical public key encryption. This formalization aims to finely manage both the access control to the encrypted data, and the information revealed by the decryption. This thesis studies possibilities of delegation through these two sides.

First, we deal with a multi-client context : several users provide each one an encryption of personal data, and an entity wishes to extract information from the aggregate of those inputs. Our contribution in this environment consists to provide these clients the possibility to give, or refuse, their consent for such an extraction. To this aim, we describe constructions of multi-client functional encryption. We then formalize several levels of security and provide methods to reach them. Eventually, we decentralize the construction of the functional decryption key, so that one needs the agreement of all clients to get a functional decryption key. All this, in a practical way.

Second, we consider a more specific case where a video content provider wishes to delegate the distribution of his creation, but without revealing it. Our solution is a key encapsulation mechanism, derived from attribute-based encryption, with a particular property. The provider uses it to encapsulate the key of the encrypted stream under several attributes, and provides the encapsulations to the distributor. This "content manager" can then use the property to combine the encapsulations and make a new one under the access policy of his choice.

Keywords

cryptography, functional encryption, delegation, multi-client, decentralization