Université Clermont Auvergne
Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
CNRS, UMR 6158, LIMOS, 63173 Aubière, France

Mécanismes de délégation pour les primitives de cryptographie à clé publique

Thèse

pour obtenir le

Doctorat de L'Université Clermont Auvergne

Spécialité informatique

École Doctorale des Sciences Pour l'Ingénieur

présentée et soutenue publiquement par

Xavier Bultel

le 17 Mai 2018

devant le jury composé de :

M. Sébastien Canard	Ingenieur de recherche Orange Labs	Rapporteur
M. Fabien Laguillaumie	Professeur des universités École Normale Supérieure de Lyon	Rapporteur
Mme Céline Chevalier	Maître de conférences Université Panthéon-Assas Paris 2	Examinateur
M. David Pointcheval	Directeur de recherche CNRS Département d'Informatique de l'École Normale Supérieure de Paris	Examinateur
M. Sébastien Salva	Professeur des universités Université Clermont Auvergne	Examinateur
M. Pascal Lafourcade	Maître de conférences Université Clermont Auvergne	Directeur de thèse

Remerciements

En premier lieu, je tiens à remercier Pascal Lafourcade, mon directeur de thèse, qui m'a aidé et soutenu tout au long de mes années de doctorat. Merci de m'avoir fait confiance, merci de m'avoir fait découvrir le monde merveilleux de la recherche, et merci de la grande rigueur avec laquelle tu as encadré ma thèse. Tu m'as appris énormément de choses, et ça a été un vrai plaisir de travailler avec toi. Les moments où nous *"faisions de la science"*, pour reprendre ton expression, devant un tableau surchargé de symboles, resteront mes plus beaux souvenirs de ces années de thèse.

Je voudrais aussi remercier Sébastien Canard et Fabien Laguillaumie pour le sérieux et la rigueur avec lesquels ils ont rapporté ma thèse. Leurs commentaires et leurs conseils m'ont permis d'améliorer significativement la qualité du manuscrit que vous tenez entre vos mains. Je remercie également Céline Chevalier, David Pointcheval et Sébastien Salva pour m'avoir fait l'honneur d'accepter d'être les membres de mon jury.

J'adresse mes remerciements à la chair industrielle de confiance numérique qui à permis le financement de cette thèse.

J'aimerais aussi remercier tous mes co-auteurs: Gildas Avoine, Olivier Blazy, Radu Ciucanu, Manik Lal Das, Jannik Dreier, Jean-Guillaume Dumas, Hardik Gajera, Sébastien Gambs, Malika More, Cristina Onete et Jean-Marc Robert. En particulier, je voudrais remercier David Gérault et Matthieu Giraud, mes deux petits frères de thèse. Merci beaucoup David, pour l'aide et le soutien apportés dans les moments difficiles. D'autre part, je remercie toute la petite équipe du séminaire des doctorants pour m'avoir aidé à créer et à faire vivre ce séminaire.

J'aimerais exprimer ma gratitude à tous les membres de l'équipe réseaux et protocoles, et en particulier à tous les doctorants avec qui j'ai partagé le bureau C6. Merci à Thérèse, Honoré et Malick pour m'avoir fait découvrir le poulet piqué. Merci à Déthié, Rana, Hamadoun, Jinpeng, et à tous ceux que j'oublie pour avoir contribué à la bonne ambiance qui règne dans ce bureau. Merci à Marie-Caroline et à Gauthier, les deux stagiaires (oui, j'ai bien dit stagiaires) pour avoir redécoré mon bureau et ma fenêtre avec autant de goût. Je n'oublie pas non plus Loukmen, qui mérite sa place dans ces remerciements même si son bureau est à l'*autre bout du couloir*.

Un grand merci à tous mes amis pour l'énorme soutien qu'ils m'ont apporté, parfois sans même en avoir conscience, tout au long de ces trois années de thèse. Merci aux jongleurs du mardi soir, et en particulier à Raph', Mat', Yann, Loup et Rémi. Ce fut un plaisir de pratiquer le passing avec vous. Merci à Kevin et à Mattias, et à toute la clientèle du Checkpoint Café. Merci pour toutes les parties de Smash Bros et pour tout le sel, vous n'imaginez pas à quel point vous allez me manquer. Bien sûr, merci à toi, Amandine, dite La Mèche, pour ton aide et ton soutien dans les moments les plus difficiles.

Naturellement, je remercie mes parents et mon grand frère Jean-Paul, qui m'ont toujours soutenu et encouragé tout au long de cette thèse. Enfin, je remercie chaleureusement tous ceux que j'aurais dû mentionner dans les précédents paragraphes, mais qui n'ont pas été cités, et leur présente mes plus plates excuses. Chers amis oubliés, je laisse volontairement un peu de place au bas de cette page pour pouvoir y ajouter vos noms ultérieurement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> *Pleasure has probably been the main goal all along. But I hesitate to admit it, because computer scientists want to maintain their image as hard-working individuals who deserve high salaries. Sooner or later society will realise that certain kinds of hard work are in fact admirable even though they are more fun than just about anything else.*
>
> D. E. Knuth

Contents

In this thesis, we study the mechanisms of delegation for public key cryptography primitives. In this first chapter, we introduce the basic concepts of cryptography, its applications and its challenges. We start with a short history of this science. Then, we informally recall some basic concepts of cryptography, especially protocol design and delegation. Finally, we list the contributions of this thesis.

1.1 A Little History

Cryptography is a field of computer science that deals with the protection of sensitive data. The oldest known cryptographic systems were created during antiquity, and were encryption schemes for military use. For example, the spartan used an encryption technique called the *scytale*: a strap was wrapped around a stick, and the secret message was written on it. Then the strap was unrolled, and randomly chosen letters were added on it. Thus, it was difficult to differentiate the letters of the messages from those artificially added. However, somebody who has an identical stick may wrap the strap around it, and then discover the secret message.

Since then, many other encryption systems have emerged throughout history. Until the Second World War, encryption algorithms had to be straightforward and efficient, because without computer, encryptions and decryptions were performed by humans. For practical reasons, it was impossible to spend a lot of time to encrypt or decrypt a message, and any human error of encryption would make the message unreadable. Encryption systems were substitution ciphers, *i.e.*, each letter of the message is replaced by another in the alphabet. A well-known example is Caesar's cipher, used in ancient Rome, where each letter of the message was replaced by the letter which is x places further in the alphabet. Knowing x, we can do the opposite shift and find the message. For example, with $x = 3$, the message `crypto` will be encrypted by `fubswr`. Other examples of substitution ciphers are *Polybe's square*, *Vigenère's cipher*, or *Delastelle's cipher*.

New techniques modernized cryptography during the Second World War. The Germans used an electro-mechanical encryption machine called Enigma to encrypt their messages. Humans no longer encrypted the messages, the machine automated the process, so the encryption was faster and without errors. Moreover, the encryption algorithm is more sophisticated. Breaking this encryption system becomes a crucial issue for allies, who mobilize several renowned cryptographers, including the famous Alan Turing, to study the machine. These cryptographers found how to break the Enigma cipher, which was a decisive factor in the allies victory. The cryptography issues in the Second World War raised awareness of the importance of this science.

The advent of computer science marks a turning point in the history of cryptography. In the late 1940s, Shannon introduced the theoretical foundations of cryptography by formally defining two fundamental principles: secrecy and authentication. This was the beginning of so-called modern cryptography. A few years later, in the 1970s, two major contributions revolutionized this field. The first one was the development of a symmetric encryption standard for the companies called *Data Encryption Standard* (DES) [DES77]. This encryption system was designed by IBM. Its security was based on two principles proposed by Shanon: confusion and diffusion. It becomes obsolete in 2000 because of the evolution of the power of computers, and it is replaced by the *Advanced Encryption Standard* (AES) [AES01].

The second one is public key cryptography. Until then, encryption systems required that the users know a unique secret key, which was used to both encrypt and decrypt the messages. This key allows the users to both encrypt and decrypt the messages. Thus, users had to meet physically to choose the secret key. In there pioneer paper [DH76], Diffie and Hellman give a protocol that allows two users who share no information to securely choose a secret key, even if they communicate on an insecure channel where adversaries observe the interactions. Using a similar method, Rivest, Shamir and Adelman [RSA78] introduce the first public key encryption scheme. In this encryption system, there are two different keys: the first one allows the users to encrypt messages. This key is public, *i.e.*, it is known by everybody, so anybody can encrypt a message. The second key allows its owner to decrypt the ciphertexts. This key is secret, *i.e.*, it is known only by its owner.

Since the 1970s, cryptography has become a very attractive academic research topic. Today, the features of cryptographic systems are not just about encryption and decryption. For example, homomorphic encryption, introduced by Gentry in 2009, allows computation on ciphertexts. These computations can be done by somebody who does not know the secret decryption key, such as an untrusted server. Thus, the data owner receives a ciphertext that contains only the result of the computation. Moreover, public key cryptography has enabled the design of authentication protocols and digital signatures, which has extend the diversity of cryptographic protocols.

1.2 Current Challenges

As it was seen before, cryptography only had military applications for several centuries. However, the democratization of computers allows anybody to use the cryptographic tools. These new users provide new challenges, which require development of new cryptographic protocols.

In the 1970s, some financial companies began to use encryption and authentication to protect financial transactions. Know, most of financial transactions are digital. To this day, the protection of these transactions is one of the main challenges in cryptography. The cryptographic systems are even used directly by the clients of these companies in a transparent way. For example, the bank cards or the secure payment protocols on internet use cryptography.

Advances in computing and the emergence of the Internet have led companies to automate their services and to dematerialize their data, by replacing paper documents with digital documents. On the one hand, this "digital switchover" has many benefits for companies, but on the other hand, their data are vulnerable to hackers. The protection of sensitive data in a dematerialized world requires the creation of new appropriate cryptographic protocols. For example, cryptographers designed searchable encryption protocols [BBO07], which allow to search in encrypted databases using keywords.

In recent years, dematerialization has become everyone's issue. More and more people are choosing to receive their invoices or paperwork in digital format. On the other hand, cloud computing services, proposed by Google among other companies, allow their users to use distant servers to store their documents, and thus enjoy the computation power of these servers. In this kind of application, the challenge is to protect not only the data, but also the privacy of users. Homomorphic encryption [Gen09], which allows computation on ciphertext, has many applications in this field.

Very recently, in 2013, the revelations of Edward Snowden led a renewed interest of the scientific community in the privacy. He published secret information about mass surveillance programs led by the National Security Agency. Thanks to these revelations, the general public realized the importance of protecting its privacy in a dematerialized world. Providing easy-to-use security tools accessible to everyone, even non-specialists, to encourage people to protect their digital data is a new challenge. For example, GnuPG[1] is a tool that allows a user to encrypt and sign his email using public key cryptography.

To summarize, cryptography was first used essentially by governments and armies, then it was used by economic organizations and companies. Today, everyone uses it, consciously or not. Cryptography has adapted quickly to the emergence of the Internet and the dematerialization of data, which resulted in cryptographic protocols becoming increasingly sophisticated.

1.3 Public Key Cryptography

In this thesis, we focus on public key cryptography. This field is about cryptographic primitives in which some operations can be done by anyone using a public key. On the other hand, the other operations can be done only by the users who know secret keys. Public key cryptography primitives can be classified into three basic families: encryption systems, signature systems, and authentication protocols.

Public Key Encryption: Public key encryption is used to encrypt messages, *i.e.*, encode messages in order to render them unreadable. This message can be sent on an unsecured channel, since it can not be decoded even if it is intercepted by a dishonest user. Messages are encrypted using a public key, so anyone can encrypt a message. However, only the user who knows the corresponding secret key can decrypt the message.

[1] https://www.gnupg.org

Digital Signature: The signature is a cryptographic primitive that allows a user to authenticate a message. This user, called the signer, has a secret key that allows him to sign messages. In concrete terms, the signer generates a bit string, called a signature, from his secret key and the message he wants to authenticate. Hence he is the only user who can generate valid signatures. Using the signer public key, anyone can verify the signature by using an algorithm that decides whether the signature was correctly built or not. This algorithm does not accept signatures that have not been generated from the right secret key.

Authentication Protocols: In authentication protocols, a user, called the prover, interacts with another user, called the verifier, in order to prove his or her identity. The prover uses his secret key to authenticate, and the verifier uses the prover's public key to verify that the prover authenticates correctly. Such a protocol is said to be secure when it is not possible to authenticate without the prover's secret key.

Zero-knowledge proofs of knowledge [GMR89] are cryptographic protocols that are very close to authentication schemes. In these protocols, a prover knows a secret solution to a computational problem instance. He wants to prove to a verifier that he knows this secret solution. However, the prover does not want to reveal any information about his secret to the verifier. In other words, the verifier must be convinced that the prover knows the secret solution but he must not have learned anything about it at the end of the protocol. Note that this cryptographic primitive can be used as an authentication protocol: the problem instance is the public key, and its solution is the secret key. To authenticate, the prover proves to the verifier that he knows the secret solution of the problem instance. Since the protocol does not leak any information, a malicious observer is not able to authenticate as the prover.

1.4 Design and Security Proofs

In this thesis, we propose new public key cryptographic protocols, *i.e.,* new encryption, signature and authentication systems. These protocols have additional features compared to the basic primitives we recalled in the previous section.

We focus on cryptographic protocols based on number theory. This means that the security properties of these protocols depend on algorithmic number theory problems that are supposed to be difficult to solve. A problem is said to be difficult if it is impossible to solve it effectively, *i.e.,* in a polynomial number of operations with regards to the size of the instance of the problem.

There are two main kinds of tools that can be combined to create new cryptographic protocols.

1. Operations on mathematical objects from number theory. Protocols that only use such operations are very specific, which makes them most of the time rather efficient. Their security is based on the algorithmic problems of number theory.

2. Cryptographic protocols. By combining several different protocols, new protocols with new properties can be obtained. These protocols can be instantiated with any cryptographic protocols that have the required properties. Protocols that are built only from other protocols are said to be generic. Their security is based on the security of the basic protocols they use.

One of the main challenges of cryptographic protocols is security: some information must be protected, and some operations must be impossible without the corresponding secret key. For example, an adversary must not be able to learn encrypted messages, or must not be able to generate valid signatures. Knowing if a protocol is secure or not is not obvious, so it is necessary to rigorously prove the security of the new protocols.

The first step is to formally model the security property that we want to prove. We first define a security experience from this property. Specifically, we define what it means for the adversary to break the security of a protocol, and what help he can use. Then we show that no opponent is able to break the security of the protocol, *i.e.,* to efficiently "succeed" the security experience. We prove it by contraposition. We assume that there exists an adversary able to succeed the security experience. If the protocol is based on operations from number theory, we show that a given algorithmic problem that is supposed to be hard can then be solved effectively. If the protocol is built from another secure protocol, we show that we can build an adversary that breaks the security of this protocol. In both cases, there is a contradiction, which concludes the security proof.

1.5 Secure Delegation in Cryptography

Delegation is the assignment by someone, called delegator, of some tasks to another entity, called the delegate. Moreover, tasks performed by the delegate remain under the responsibility of the delegator. For this reason, it is more interesting for the delegator to delegate only part of his capabilities.

The simplest solution is to give all the delegator secret keys to the delegate. For example, suppose that Alice wants to delegate to Bob her capability to sign messages. Alice gives her signature secret key to Bob. Then Bob is able to sign any message on behalf of Alice. However, this solution is obviously not secure. Indeed, Alice is responsible for all messages that Bob has signed using her key.

There are several primitives that allow their users to delegate some cryptographic rights securely. In this section, we present some of these primitives. Note that we do not give an exhaustive list of all the cryptographic primitives that deals with delegation, we focus on the few primitives we are working on in this thesis.

1.5.1 Proxy Re-cryptography

Proxies re-cryptography concerns primitives that allow a server to transform a cryptographic object of a user into another cryptographic object for another user. Each of the basic primitives mentioned above (*i.e.,* public key encryption, signature, and authentication protocol) has been adapted to proxy re-cryptography.

Proxy Re-encryption: Consider two users, Alice and Bob, who use a public-key encryption system. Each of them has his own public encryption key and his own secret decryption key. Suppose that Alice wants to delegate to Bob the ability to decrypt her ciphertexts on behalf of herself for a given period of time. Alice uses a server to help her. She gives the server a secret key, called a re-encryption key, that allows it to transform a ciphertext for Alice into a ciphertext for Bob. At the end of the period of time, Alice turns off the server.

Proxy Re-signature: In this primitive, Alice and Bob use a signature scheme. Alice wants to delegate to Bob the ability to sign messages on behalf of herself for a given period of time. Alice gives to the server a secret key, called a re-signature key, that allows it to transform a signature of Bob into a signature of Alice. As in proxy re-encryption, Alice turns off the server to remove the delegation.

Proxy re-authentication: This primitive allows Alice to delegate to Bob the ability to authenticate on behalf of herself. As in the other proxy re-cryptography primitives, Alice gives the server a re-authentication key that allows it to transform an authentication of Bob into an authentication of Alice. Once again, the delegation is removed when Alice turns off the server.

For each of these primitives, a naive solution is to give Alice's secret key to the server. However, this solution is not satisfying from the security point of view. The server could decrypt or sign anything on behalf of Alice. For practical reasons, we do not want to trust the server. In the case of the proxy re-encryption, the server must be able to transform a ciphertext for Alice into a ciphertext for Bob, but he must not learn anything about the plaintext message. In the case of proxy re-signature and re-authentication, the server must not be able to sign a new message alone and authenticate on behalf of Alice.

The main advantage of proxy re-cryptography is that the delegate does not need to learn new keys or algorithms. The server transforms ciphertexts, signatures, and authentications without any help, and the delegate decrypts, signs, and authenticates exactly as he would for himself. Thus, these solutions are easy to use in practice.

1.5.2 Sanitizable Signatures

Sanitizable signatures allow the signer to delegate to somebody the ability to modify the messages without invalidating the signature. However, the delegate cannot modify the signed messages as he wants: some parts of the messages can not be modified. For example, Alice signed the message *"my shoes are red"*, and delegates to Bob the ability to change the message such that only the color can be changed. Then Bob can produce a signature of the message *"my shoes are blue"* on behalf of Alice using his sanitizer key. On the other hand, he cannot produce a signature of the message *"my pants are red"*. The verifier must not be able to guess whether the signature has been modified or not. Often, these signature schemes have a proof mechanism that allows Alice to prove whether a signature has been modified by Bob or not. It prevents Bob from abusing his power. Sanitizable signatures have many applications, especially in the privacy protection on health data. Using it, a hospital can for example change the names of patients on the signed medical records in order to anonymize them.

1.5.3 Delegation of Computation

In cryptography, the field of delegation of computation rallies primitives where a server makes computations for users. We assume that users do not trust the server. The main challenge is to allow the server to prove to the users that it performed the computations correctly.

Delegation of Hard Computation: Several works in this field concern the delegation of hard computations. Users are computationally bounded, so they need the server to do some computations that are too hard for them. However, the proof of correctness of these hard computations given by the server must be efficiently verifiable because the users are computationally bounded.

Private Function Evaluation: Another challenge in this field is the evaluation of private functions by a server. To illustrate this, consider a company that knows a secret function f, and that uses a server that evaluates the function f on the data chosen by its clients. Thus the company delegates the computation of the function to its server. Clients learn only the points of the function they ask, and must not learn additional information about f. However, the server may returns false values: the clients cannot check the server's computations because they do not know the secret function. To solve this problem, the company generates a secret key and a public verification key. After each function evaluation, the server can prove to the client the correctness of its computation thanks to the secret key. The clients can verify this proof thanks to the public verification key in order to be convinced that the data they receive is correct.

1.6 Contributions

Throughout this manuscript, we focus on several cryptographic primitives ensuring the secure delegation of rights [BDG+17, BL16, BL17a, BL17b]. We first define a new family of proxy re-cryptography primitives called proxy re-proof of knowledge [BL17b]. These primitives have applications in the delegation of authentication. Then we focus on verifiable private function evaluation protocols. We show that several protocols in the literature have critical security flaws, and we improve the existing security models [BDG+17]. In the third part of this thesis, we show how to build a generic sanitizable signature scheme that is very efficient in terms of computation time from a variant of ring signatures called verifiable ring signatures [BL17a]. Finally, we define a new public key encryption primitive where a user who encrypts messages can delegate to someone else the capability to decrypt the ciphertexts that have been encrypted within a chosen time interval [BL16].

1.6.1 Proxy Re-proof of Knowledge

In Chapter 3, we extend the concept of the proxies re-cryptography to the zero-knowledge proofs of knowledge. As mentioned before, proofs of knowledge can be used as authentication protocols. These authentication protocols have stronger security properties than basic authentication protocols: no information leaks during the protocol, and it is impossible to authenticate without full knowledge of the secret key. Similarly, proxies re-proof of knowledge can be used as proxies re-authentication that have stronger security properties than basic proxy re-authentication protocols.

Idea: Consider two users, Alice and Bob, who use a proof of knowledge protocol to authenticate to some service. Alice wants to give Bob the ability to authenticate on her behalf using a proxy. The proxy has a re-proof key that allows it to transform a proof protocol of Bob's secret into a proof protocol of Alice's secret. The proxy can not authenticate on behalf of Alice or Bob with this key only. To authenticate on behalf of Alice, Bob proves the knowledge of his secret to the proxy, which transforms it into a proof of the knowledge of Alice's secret. Such a protocol is said to be zero-knowledge when its users learn nothing while it is running.

We show how to build a practical and efficient system that manages the access rights policy to different services for a company using proxies re-proof of knowledge.

Security Flaws: There is only one proxy re-authentication in the literature. We show that it has the following security flaws. If Bob colludes with the proxy, he can guess Alice's secret key. If Alice observes the interactions between the proxy and the verifier, then she can find Bob's secret. We show that it is possible to design protocols that prevent such attacks.

Bidirectional and Unidirectional Protocols: Proxy re-cryptography primitives are separated into two main families: bidirectional and unidirectional. In bidirectional protocols, the proxy can transform proofs in both directions, *i.e.,* if he can transform Alice's proof into Bob's proof, then he can transform Bob's proof into Alice's proof. In such a protocol, the generation of the key of the proxy, called the re-proof key, requires the knowledge of both the delegator secret key and the delegate secret key. Hence it requires a trusted authority that generates the secret keys of the users, or the use of a proxy key generation protocol between the proxy, the delagate and the delegator. On the other hand, these protocols are often much less complex than unidirectional protocols.

In unidirectional protocols, the delegator can build the re-proof key by itself, using its secret key and the delegate's public key. Hence, we no longer need trusted authority, which confers stronger security properties to this kind of protocols. However, such protocols generally require more complex mathematical computations than bidirectional ones, thus there are less efficient than bidirectional protocols.

In this thesis, we formally define these two families of proxy re-proof protocols.

Interactive and Non-interactive Protocols: A proof of knowledge is said to be interactive when the prover interacts with the verifier, *i.e.,* they exchange their data one after the other. A proof of knowledge is said to be non-interactive when the prover generates only one element which is sent to the verifier. Depending on whether it uses interactive or non-interactive proof systems, a proxy re-proof protocol is said to be interactive or non-interactive. In this thesis, we formally define these two families of proxy re-proof of knowledge protocols.

Protocols Design: For each of these proxy re-proof protocol families, we give a security model. This model is based on the security model of zero-knowledge proofs. We define an additional security property to ensure that the re-proof key does not leak information about the delegator secret key and the delegate secret key.

In this thesis, we give a concrete protocol for each of the families: bidirectional interactive, unidirectional interactive, bidirectional non-interactive, and unidirectional non-interactive. We prove the security of each of these protocols.

1.6.2 Verifiable Private Function Evaluation

In Chapter 4, we focus on the security of verifiable private function evaluation protocols. In such protocols, a server evaluates a secret function on chosen input for users. Then the server proves the validity of its computation to the users, who verifies this proof thanks to a public verification key.

Cryptanalysis of Two Schemes: We first study the security of two schemes of the literature. We show a critical security flaw of these protocols: using some properties of these protocols, we prove that we can recover the secret function by interacting only once with the server, *i.e.,* by evaluating only one point of the function.

Improvement of the Security Model: Until now, the security models of this primitive assumed that the secret function was randomly chosen. We show that this security model is not realistic for some applications of the verifiable private function evaluation protocols. We propose a new security model where the adversary must guess which function is used by the server among two functions of his choice. This new model is called *indistinguishability against chosen function attacks.* We prove that one of the protocols of the literature is secure in this model.

Protocol Design: Finally, we propose a new protocol for verifiable private function evaluation, and we prove that this protocol is secure in our model. The advantages and disadvantages of the protocol of the literature and our new protocol are compared in terms of efficiency, computational complexity and security.

1.6.3 Sanitizable Signatures

In Chapter 5, we focus on a primitive called verifiable ring signature. In Chapter 6, we propose a new unlinkable sanitizable signature scheme based on the verifiable ring signatures. A sanitizable signature is said to be unlinkable when it is not possible to link the sanitized signature to the original one.

Verifiable Ring Signatures: Ring signatures allow the user to sign a message within a group such that the signer remains anonymous. Verification is done using all the public keys of the users who are in the group. A verifiable ring signature is a ring signature where a user can prove *a posteriori* whether he is the signer of a given message or not. We define a complete security model for this primitive. To the best of our knowledge, no security model was proposed for this primitive in the literature before this work. We also propose a new verifiable ring signature scheme, and we prove its security in our model.

Efficient Sanitizable Signature Scheme: We design a new generic and efficient unlinkable sanitizable signature scheme based on verifiable ring signatures. We show that the best instantiation of our scheme is twice as efficient in computational complexity as the most efficient scheme of the literature, with an equivalent level of security.

Strong Accountability: A sanitizable signature scheme is said to be accountable when the signer can prove whether a signature has been sanitized or not. We improve this property by defining strong accountability. In this model, the sanitizer can also prove whether a signature has been sanitized or not. We prove that our sanitizable signature scheme is strongly accountable.

1.6.4 A Posteriori Openable Public Key Encryption

In Chapter 7, we focus on the following problem. A user encrypts each email he sends using public keys. This user would like to be able to delegate the ability to decrypt emails that were sent during a chosen time period to a delegate. However, he does not want the delegate to be able to decrypt the other emails. As encrypting emails can sometimes seem suspicious, this property allows the user to prove that the messages he sends during the time period do not contain compromising information. On the other hand the other messages remain protected.

New Encryption Primitive: We define a new cryptographic primitive, called a posteriori public key encryption, which solves this problem. In this primitive, a user can generate a key that allows a delegate to decrypt all messages generated during a period of time chosen a posteriori. We also define a security model for this primitive that is suitable for its practical applications.

Efficient A Posteriori Public Key Encryption Scheme: The main design challenge is the size of the key given to the delegate: if this key increases proportionally to the number of messages in the interval, then the scheme is not practical. We first give a naïve solution where the size of this key is linear in the number of messages. Then we design a generic and efficient a posteriori openable public key encryption scheme where the key is of constant size, *i.e.,* it does not depend on the size of the interval.

1.7 Publications

In this section, we summarize our publications. First we recall the papers about the works presented in this manuscript, then we list the papers about other works that were conducted throughout this thesis.

1.7.1 Presented in this Manuscript

The four following papers resume the works that are presented in this manuscript.

- Xavier Bultel and Pascal Lafourcade.
  "Zero-Knowledge Proxy Re-Proof of Knowledge." (In submission)

- Xavier Bultel, Manik Lal Das, Hardik Gajera, David Gérault, Matthieu Giraud and Pascal Lafourcade.
  "Verifiable Private Polynomial Evaluation." ProvSec 2017

- Xavier Bultel and Pascal Lafourcade.
  "Unlinkable and Strongly Accountable Sanitizable Signatures from Verifiable Ring Signatures." CANS 2017

- Xavier Bultel and Pascal Lafourcade.
  "A Posteriori Openable Public Key Encryption." IFIP SEC 2016

1.7.2 Other Publications

We list exaustively the other papers published during the thesis, and we give a short abstract for each of them.

- Olivier Blazy, Xavier Bultel and Pascal Lafourcade.
  "Two Secure Anonymous Proxy-based Data Storages." SECRYPT 2016

  Abstract: Unidirectional proxy re-encryption (PRE) can be used to realize an efficient and secure shared storage. However, this type of storage does not yet protect its users' privacy: to retrieve some data a user must give his identity and his query to the proxy. We propose two secure data storage systems that allow authorized users to anonymously get access to the content of encrypted data on a storage. Each scheme corresponds to a certain economic model. In the first one, a user has to pay for each downloaded file, whereas in the second one, users pay each month a subscription to get an unlimited access to all their files. We define the models for the anonymity and the management of the users' rights. For our two schemes, we also prove, in the standard model, their security considering a classical honest-but-curious proxy.

- Olivier Blazy, Xavier Bultel and Pascal Lafourcade.
  "Anonymizable Ring Signature Without Pairing." FPS 2016

  Abstract: Ring signature is a well-known cryptographic primitive that allows any user who has a signing key to anonymously sign a message according to a group of users. Some years ago, Hoshino et al. propose a new kind of ring signature where anybody can transform a digital signature into an anonymous signature according to a chosen group of users; authors present a pairing-based construction that is secure under the gap Diffie-Hellman assumption in the random oracle model. However this scheme is quite inefficient for large group since the generation of the anonymous signature requires a number of pairing computations that is linear in the size of the group. In this paper, we give a more efficient anonymizable signature scheme without pairing. Our anonymization algorithm requires n exponentiations in a prime order group where n is the group size. Our proposal is secure under the discrete logarithm assumption in the random oracle model, which is a more standard assumption.

- Xavier Bultel and Pascal Lafourcade.
  "k-time Full Traceable Ring Signature." ARES 2016

  Abstract: Ring and group signatures allow their members to anonymously sign documents in the name of the group. In ring signatures, members manage the group themselves in an ad-hoc manner while in group signatures, a manager is required. Moreover, k-times traceable group and ring signatures [ASY06] allow anyone to publicly trace two signatures from a same user if he exceeds the *a priori* authorized number of signatures. In [CSST06], Canard *et al.* give a 1-time traceable ring signature where each member can only generate one anonymous signature. Hence, it is possible to trace any two signatures from the same user. Some other works generalize it to the k-times case, but the traceability only concerns two signatures. In this paper, we define the notion of *k-times full traceable ring signature* (k-times full traceable ring signature) such that all signatures produced by the same user are traceable if and only if he produces more than k signatures. We construct a k-times full traceable ring signature scheme called Ktrace. We extend existing formal security models of k-times linkable signatures to prove the security of Ktrace in the random oracle model. Our primitive k-times full traceable ring signature can be used to construct a k-times veto scheme or a proxy e-voting scheme that prevents denial-of-service caused by cheating users.

- Xavier Bultel, Jannik Dreier, Jean-Guillaume Dumas and Pascal Lafourcade.
  "Physical Zero-Knowledge Proofs for Akari, Takuzu, Kakuro and KenKen." FUN 2016

  Abstract: Akari, Takuzu, Kakuro and KenKen are logic games similar to Sudoku. In Akari, a labyrinth on a grid has to be light by placing lanterns, respecting various constraints. In Takuzu a grid has to be filled with 0's and 1's, while respecting certain constraints. In Kakuro a grid has to be filled with numbers such that the sums per row and column match given values; similarly in KenKen a grid has to be filled with numbers such that in given areas the product, sum, difference or quotient equals a given value. We give physical algorithms to realize zero-knowledge proofs for these games which allow a player to show that he knows a solution without revealing it. These interactive proofs can be realized with simple office material as they only rely on cards and envelopes. Moreover, we formalize our algorithms and prove their security.

- Xavier Bultel, Sébastien Gambs, David Gerault, Pascal Lafourcade, Cristina Onete and Jean-Marc Robert.
  "A Prover-Anonymous and Terrorist-Fraud Resistant Distance Bounding Protocol." WiSec 2016

  Abstract: Contactless communications have become omnipresent in our daily lives, from simple access cards to electronic passports. Such systems are particularly vulnerable to *relay attacks*, in which an adversary relays the messages from a prover to a verifier. Distance-bounding protocols were introduced to counter such attacks. Lately, there has been a very active research trend on improving the security of these protocols, but also on ensuring strong privacy properties with respect to active adversaries and malicious verifiers.

  In particular, a difficult threat to address is the *terrorist fraud*, in which a far-away prover cooperates with a nearby accomplice to fool a verifier. The usual defence against this attack is to make it impossible for the accomplice to succeed unless the prover provides him with enough information to recover his secret key and impersonate him later on. However, the mere existence of a long-term secret key is problematic with respect to privacy.

  In this paper, we propose a *novel approach* in which the prover does not leak his secret key but a reusable session key along with a group signature on it. This allows the adversary to impersonate him even without knowing his signature key. Based on this approach, we give the first distance-bounding protocol, called SPADE, integrating anonymity, revocability and provable resistance to standard threat models.

- Xavier Bultel, Sébastien Gambs, David Gerault, Pascal Lafourcade, Cristina Onete and Jean-Marc Robert.
  "SPADE : un protocole délimiteur de distance anonyme et résistant à la fraude terroriste." AlgoTel 2017

  Abstract: This paper is an extended abstract of the paper *"A Prover-Anonymous and Terrorist-Fraud Resistant Distance Bounding Protocol"* at WiSec 2016 presented above.

- Gildas Avoine, Xavier Bultel, Sébastien Gambs, David Gerault, Pascal Lafourcade, Cristina Onete and Jean-Marc Robert.
  "A Terrorist-fraud Resistant and Extractor-free Anonymous Distance-Bounding Protocol." ASIA CCS 2017

  Abstract: Distance-bounding protocols have been introduced to thwart relay attacks against contactless authentication protocols. In this context, *verifiers* have to authenticate the credentials of untrusted *provers*. Unfortunately, these protocols are themselves subject to complex threats such as *terrorist-fraud* attacks, in which a malicious prover helps an accomplice to authenticate. Provably guaranteeing the resistance of distance-bounding protocols to these attacks is a complex task due to the potential interactions between the malicious prover and his accomplice. The classical countermeasures usually assume that rational provers want to protect their long-term authentication credentials, even with respect to their accomplices. Thus, terrorist-fraud resistant protocols generally rely on artificial *extraction mechanisms*, ensuring that an accomplice can retrieve the credential of his partnering prover if he is able to authenticate with a non-negligible probability. However, these artificial mechanisms are usually required only for sake of the proofs and are complex to deploy.

- Xavier Bultel, Radu Ciucanu, Matthieu Giraud and Pascal Lafourcade.
  "Secure Matrix Multiplication with MapReduce." ARES 2017

  Abstract: The MapReduce programming paradigm allows to process big data sets in parallel on a large cluster of commodity machines. The MapReduce users often outsource their data and computations to a public cloud provider. We focus on the fundamental problem of matrix multiplication, and address the inherent security and privacy concerns that occur when outsourcing to a public cloud. Our goal is to enhance the two state-of-the-art algorithms for MapReduce matrix multiplication with privacy guarantees such as: none of the nodes storing an input matrix can learn the other input matrix or the output matrix, and moreover, none of the nodes computing an intermediate result can learn the input or the output matrices. To achieve our goal, we rely on the well-known Paillier's cryptosystem and we use its partially homomorphic property to develop efficient algorithms that satisfy our problem statement. We develop two different approaches called secure-Private and Collision-Resistant-Secure-Private, and compare their trade-offs with respect to three fundamental criteria: computation cost, communication cost, and privacy guarantees. Finally, we give security proofs of our protocols.

- Xavier Bultel, Jannik Dreier, Pascal Lafourcade and Malika More.
  "How to explain modern security concepts to your children." Cryptologia (2017)

  Abstract: At the main cryptography conference CRYPTO in 1989, Quisquater et al. published a paper showing how to explain the complex notion of zero-knowledge proof in a simpler way that children can understand. In the same line of work, we present simple and intuitive explanations of various modern security concepts and technologies, including symmetric encryption, public key encryption, homomorphic encryption, intruder models (CPA, CCA1, CCA2) and security properties (OW, IND, NM). The explanations given in this paper may also serve in demystifying such complex security notions for non-expert adults.

Chapter 2

Technical Introduction

Contents

In this chapter we recall the cryptographic notions used throughout this thesis. We begin by the mathematical background, next we define the cryptographic assumptions used to evaluate the security of our schemes. Finally, we present some fundamental cryptographic primitives classified in the following families: hash functions, public key encryptions, proofs of knowledge and signatures. For each of them, we give a formal definition, a security model and a concrete instanciation. When no reference is specified, the definitions and theorems given in this chapter refer to the book *Foundations of Cryptography* [Gol01, Gol04].

As a preamble to this chapter, let us recall the principle of provable security. This approach allows us to formally prove that a given cryptographic scheme have some security properties. The first step is to formally define this property: we build an algorithm called the *experiment* that proposes some *challenge* to a polynomial time algorithm called the *adversary*. The challenge can be viewed as an algorithmic problem that the adversary tries to solve; if the adversary solves this problem, then we say that *it wins the experiment.* Sometimes, to help it, the adversary has access to some black-boxes called *oracles* that allows it to learn some information that it cannot compute by himself (for example, an oracle that decrypts some ciphertexts using a key that the adversary does not know). The scheme is secure according to the property when there is no adversary that wins the experiment with a *non-negligible* (*i.e.,* significantly high) probability in polynomial time. Often, this kind of result is demonstrated under the hypothesis that a mathematical problem is hard to solve, or under the hypothesis that another cryptographic scheme is secure.

For example, consider an encryption scheme and a number-theory based problem. We will show how to prove that it is not possible to decrypt a ciphertext, without the corresponding secret key, under the hypothesis that our number-theory based problem is hard to solve. We define an experiment where the challenge is to decrypt the encryption of a randomly picked message using a randomly picked secret key. We then prove by reduction that if our number-theory based problem is hard to solve, then there is no adversary that wins the experiment with a non-negligible probability. Most of the time, such a property is often proved by contraposition: if there exists an adversary $\mathscr{A}$ that wins the experiment with a non-negligible probability, then we show how to build an algorithm from $\mathscr{A}$ that solves the number-theory based problem with non-negligible probability in polynomial time, which contradicts our assumption. Hence, such adversary does not exist.

2.1 Mathematical Background

We first recall some fundamental notions of complexity.

Definition 1 (Polynomial-Time Algorithm) *Let k be an integer. An algorithm $\mathscr{A}$ whose inputs are generated by a function* Set *is said to be* polynomial-time *when for any input value* I_k *generated by* $\mathsf{Set}(k)$*, there exists a polynomial t such that the execution time of $\mathscr{A}(\mathrm{I}_k)$ is bounded by $|t(k)|$.*

Often, if $\mathscr{A}$ is a cryptographic algorithm, the parameter k is said to be the security parameter.

Definition 2 (Negligible function) *A function $f : \mathbb{N} \to \mathbb{R}$ is negligible when for all positive polynomial t there exists an integer $n \in \mathbb{N}$ such that for all $x > n$:*

$$|f(x)| < \frac{1}{t(x)}$$

In the following, we recall some fundamental notions of algebra.

Definition 3 (Group) *A* group *is a couple $(\mathbb{G}, \cdot)$ where* G *is a set and $\cdot$ is an operation of $\mathbb{G}^2 \to \mathbb{G}$ that combines any $(a, b) \in \mathbb{G}^2$ to from an element of $\mathbb{G}$ denoted $a \cdot b$ such that the three following properties hold:*

Associativity: *For all $a \in \mathbb{G}, b \in \mathbb{G}$ and $c \in \mathbb{G}$ then $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*
Identity element: *There exists a unique element $1 \in \mathbb{G}$ called* the identity element *such that for all $a \in \mathbb{G}$, $1 \cdot a = a \cdot 1 = a$.*
Inverse element: *For all $a \in \mathbb{G}$, there exists a unique element $\frac{1}{a} \in \mathbb{G}$ such that $a \cdot \frac{1}{a} = \frac{1}{a} \cdot a = 1$.*

Moreover, we say that a group is commutative *when for any $a \in \mathbb{G}$ and $b \in \mathbb{G}$, it holds that $a \cdot b = b \cdot a$. A group is said to be* finite *when $\mathbb{G}$ has a finite number of elements. The number of elements of $\mathbb{G}$ is called the* order *of the group and is denoted by $|\mathbb{G}|$. When it is clear in the context, we denote by $\mathbb{G}$ the group $(\mathbb{G}, \cdot)$*

Definition 4 (Field) *A* field *is a triplet $(\mathbb{G}, +, \cdot)$ where* G *is a set and + (resp. $\cdot$) is an operation of $\mathbb{G}^2 \to \mathbb{G}$ that combines any $(a, b) \in \mathbb{G}^2$ to from an element of $\mathbb{G}$ denoted $a + b$ (resp. $a \cdot b$) such that the three following properties hold:*

Associativity: *For all $a \in \mathbb{G}, b \in \mathbb{G}$ and $c \in \mathbb{G}$ then $(a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*

commutativity: *For all $a \in \mathbb{G}$ and $b \in \mathbb{G}$ then $(a+b) = (b+a)$ and $a \cdot b = b \cdot a$.*

Identity element: *There exists two unique elements $0 \in \mathbb{G}$ and $1 \in \mathbb{G}$ such that for all $a \in \mathbb{G}$, $0 + a = a + 0 = a$ and $1 \cdot a = a \cdot 1 = a$.*

Multiplicative inverse element: *For all $a \in \mathbb{G}$, there exists a unique element $\frac{1}{a} \in \mathbb{G}$ such that $a \cdot \frac{1}{a} = \frac{1}{a} \cdot a = 1$.*

Additive inverse element: *For all $a \in \mathbb{G}$, there exists a unique element $(-a) \in \mathbb{G}$ such that $a + (-a) = (-a) + a = 0$.*

Distributivity: *For all $a \in \mathbb{G}, b \in \mathbb{G}$ and $c \in \mathbb{G}$ then $a \cdot (b+c) = a \cdot b + a \cdot c$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*

Moreover, we say that a group is commutative *when for any $a \in \mathbb{G}$ and $b \in \mathbb{G}$, it holds that $a \cdot b = b \cdot a$. A group is said to be* finite *when $\mathbb{G}$ has a finite number of elements. The number of elements of $\mathbb{G}$ is called the* order *of the group and is denoted by $|\mathbb{G}|$. When it is clear in the context, we denote by $\mathbb{G}$ the group $(\mathbb{G}, \cdot)$*

Definition 5 (Generator and cyclic group) *Let $(\mathbb{G}, \cdot)$ be a finite group. We denote by g^x the* discrete exponentiation *of g by x where $g \in \mathbb{G}$ and $x \in \mathbb{Z}$:*

$$g^x = \underbrace{g \cdot g \cdot g \cdots g}_{x \text{ times}}$$

We say that g is a generator of $(\mathbb{G}, \cdot)$ when $\mathbb{G} = \{g^x | x \in \mathbb{Z}\}$. A group that admits a generator is called a cyclic group.

Definition 6 (Homomorphism) *Let $(\mathbb{G}_1, \cdot)$ and $(\mathbb{G}_2, \times)$ be two groups. A* group homomorphism *between $(\mathbb{G}_1, \cdot)$ and $(\mathbb{G}_2, \times)$ is a function $f : \mathbb{G}_1 \to \mathbb{G}_2$ such that for all $a \in \mathbb{G}_1$ and $b \in \mathbb{G}_2$, it holds that $f(a \cdot b) = f(a) \times f(b)$. If f is a bijective function, then we say that f is an* isomorphism*, and that $(\mathbb{G}_1, \cdot)$ and $(\mathbb{G}_2, \times)$ are* isomorphic.

Property 1 *We denote by $\mathbb{Z}/p\mathbb{Z}$, or $\mathbb{Z}_p$, the set of the integers modulo p. If $(\mathbb{G}, \cdot)$ is a cyclic group of order p then $(\mathbb{G}, \cdot)$ is commutative, and $(\mathbb{G}, \cdot)$ and $(\mathbb{Z}/p\mathbb{Z}, +)$ are* isomorphic. *Moreover, if p is prime then all elements of $(\mathbb{G}, \cdot)$ except the identity element are generators.*

Definition 7 (Bilinear pairing) *Let $(\mathbb{G}_1, \cdot), (\mathbb{G}_2, \cdot)$ and $(\mathbb{G}_T, \cdot)$ be three groups of prime order p. A* bilinear pairing map *$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a function that satisfies the following properties for all $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$:*

Bilinearity: *For all $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p^*$, then $e(g_1^a, g_2^b) = e(g_1, g_2)^{a \cdot b}$.*

Non-degeneracy: $(e(g_1, g_2) = 1) \Rightarrow ((g_1 = 1)$ *or* $(g_2 = 1))$.

Computability: *There exists a polynomial time algorithm to compute e.*

Moreover, bilinear pairings are classified in the three following families:

Type 1: $\mathbb{G}_1 = \mathbb{G}_2$*, such a pairing is said to be* symmetric.

Type 2: $\mathbb{G}_1 \neq \mathbb{G}_2$ *and there exists a polynomial time homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$.*

Type 3: $\mathbb{G}_1 \neq \mathbb{G}_2$ *and there is no polynomial time homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Definition 8 (Lagrange's interpolation) *Let k be an integer and F be a field. For all $i \in [\![0, l]\!]$, let $(x_i, y_i) \in \mathrm{F}^2$ such that for all $(i_1, i_2) \in [\![0, l]\!]^2$, $x_{i_1} \neq x_{i_2}$. There exists one and only one polynomial f of degree at most l such that for all $i \in [\![0, n]\!]$, $f(x_i) = y_i$. This polynomial is given by Lagrange's interpolation formula:*

$$f(x) = \sum_{i=0}^{l} \left(y_i \cdot \prod_{j=0, j \neq i}^{l} \frac{x - x_j}{x_i - x_j} \right)$$

2.2 Notations

- We denote by $a||b$ the concatenation of two bit strings a and b.
- We denote by $r \xleftarrow{\$} \mathrm{S}$ the random draw of r into the uniform distribution on S.
- Let X be a *Probabilistic Polynomial-Time* algorithm (PPT), we denote by $z \leftarrow \mathrm{X}(x; r)$ the execution of X where x is the input, z is the output, and r is a random tape. When it is clear from the context, we omit the parameter r and simply use $\mathrm{X}(x)$.
- $\textsc{poly}(k)$ denotes the set of all the algorithms that are polynomialy bounded in the security parameter k.
- $\mathrm{X}^{\mathrm{Y}(\cdot)}(x)$ denotes that the algorithm X has access to the algorithm Y as a black box. Y is said to be *an oracle.*
- A *protocol* P involves at least two *entities* that are modeled by probabilistic polynomial-time algorithms interacting together. $\mathsf{P}\langle \mathrm{X}(x); \mathrm{Y}(y)\rangle$ denotes the execution of the protocol P between the entities X and Y using respective inputs x and y.
- $\mathtt{out}_{\mathrm{X}}(\mathsf{P}\langle \mathrm{X}(x); \mathrm{Y}(y)\rangle)$ returns the output of the entity X at the end of the execution of the protocol P. We also denote by $\mathtt{view}_{\mathrm{X}}(\mathsf{P}\langle \mathrm{X}(x); \mathrm{Y}(y)\rangle)$ all the values sent and received by X throughout the execution of the protocol P. We note that out and view can be used on several entities, for example, $\mathtt{out}_{\mathrm{X,Z}}(\mathsf{P}\langle \mathrm{X}(x); \mathrm{Y}(y); \mathrm{Z}(z)\rangle)$ returns the couple $(o_{\mathrm{X}}, o_{\mathrm{Z}})$ where o_{X} (resp. o_{Z}) is the output of the entity X (resp. Z) at the end of the execution of the protocol P, and $\mathtt{view}_{\mathrm{X,Z}}(\mathsf{P}\langle \mathrm{X}(x); \mathrm{Y}(y); \mathrm{Z}(z)\rangle)$ returns the couple $(v_{\mathrm{X}}, v_{\mathrm{Z}})$ where v_{X} (resp. v_{Z}) is the set of all the values sent and received by X (resp. Z) throughout of the execution of the protocol P. Finally, $\mathtt{trans}(\mathsf{P}\langle \mathrm{X}(x); \mathrm{Y}(y)\rangle)$ returns the full transcript of the protocol, *i.e.*, all the values sent and received by all the entities.
- By convention, we use the symbol $*$ to denote a dishonest user in a protocol: for example, $\mathsf{P}\langle \mathrm{X}(x); \mathrm{Y}^*(y)\rangle$ denotes that X honestly runs the protocol but Y^* does not. If several dishonest entities *collude* then each of theses entities has access to all information known by all other dishonest entities. When it is not precised, we assume that dishonest entities do not collude.
- Let $\mathscr{R}$ be a binary relation, $\mathscr{L}_{\mathscr{R}}$ denotes the language defined by $\mathscr{L}_{\mathscr{R}} = \{a | \exists\, b, (a,b) \in \mathscr{R}\}$.
- We denote the set of polynomials with coefficients in the field F by F[X] and we denote the set of all $f \in \mathrm{F[X]}$ of degree l by $\mathrm{F[X]}_l$.

2.3 Cryptographic Assumptions

We present several number-theory based computational assumptions that are assumed to be true throughout this thesis.

Definition 9 (Discrete Logarithm assumption [DH76]) *Let p be a prime number generated according to a security parameter $k \in \mathbb{N}$. Let $\mathbb{G}$ be a multiplicative group of order p, and $g \in \mathbb{G}$ be a generator. The* discrete logarithm assumption *(DL) in $(\mathbb{G}, p, g)$ states that there exists a negligible function ϵ such that for all $\mathscr{A} \in \textsc{poly}(k)$:*

$$\Pr\left[x \xleftarrow{\$} \mathbb{Z}_p^*; x' \leftarrow \mathscr{A}(g^x) : x = x'\right] = \epsilon(k)$$

Vocabulary: If there exists an algorithm $\mathscr{A} \in \textsc{poly}(k)$ such that $\epsilon(k)$ is not negligible, then we say that $\mathscr{A}$ *breaks the DL assumption*. If $\mathscr{A}(g^x)$ returns x' such that $x = x'$, we say that $\mathscr{A}$ *wins the DL experiments*. Moreover, $\epsilon(k)$ is said to be the *advantage of* $\mathscr{A}$ *against the DL experiment*. The same vocabulary will be used in a similar way for the other assumptions we will define in this section.

Definition 10 (Computational Diffie-Hellman assumption [DH76]) *Let p be a prime number generated according to a security parameter $k \in \mathbb{N}$. Let $\mathbb{G}$ be a multiplicative group of order p, and $g \in \mathbb{G}$ be a generator. The* Computational Diffie-Hellman assumption *(CDH) in $(\mathbb{G}, p, g)$ states that there exists a negligible function ϵ such that for all $\mathscr{A} \in \textsc{poly}(k)$:*

$$\Pr\left[(x,y) \xleftarrow{\$} (\mathbb{Z}_p^*)^2; h \leftarrow \mathscr{A}(g^x, g^y) : h = g^{x\cdot y}\right] = \epsilon(k)$$

Definition 11 (Divisible Computational Diffie-Hellman assumption [BDZ03]) *Let p be a prime number generated according to a security parameter $k \in \mathbb{N}$. Let $\mathbb{G}$ be a multiplicative group of order p, and $g \in \mathbb{G}$ be a generator. The* Divisible Computational Diffie-Hellman assumption *(DCDH) in $(\mathbb{G}, p, g)$ states that there exists a negligible function ϵ such that for all $\mathscr{A} \in \textsc{poly}(k)$:*

$$\Pr\left[(x,y) \xleftarrow{\$} (\mathbb{Z}_p^*)^2; h \leftarrow \mathscr{A}(g^x, g^y) : h = g^{x/y}\right] = \epsilon(k)$$

The DCDH assumption is equivalent to the CDH assumption [BDZ03].

Definition 12 (Decisional Diffie-Hellman assumption [Bon98]) *Let p be a prime number generated according to a security parameter $k \in \mathbb{N}$. Let $\mathbb{G}$ be a multiplicative group of order p, and $g \in \mathbb{G}$ be a generator. The* Decisional Diffie-Hellman assumption *(DDH) in $(\mathbb{G}, p, g)$ states that there exists a negligible function ϵ such that for all $\mathscr{A} \in \textsc{poly}(k)$:*

$$\left|\Pr\left[(x,y,z_0) \xleftarrow{\$} (\mathbb{Z}_p^*)^3; z_1 = x\cdot y; b \xleftarrow{\$} \{0,1\}; b' \leftarrow \mathscr{A}(g^x, g^y, g^{z_b}) : b = b'\right] - \frac{1}{2}\right| = \epsilon(k)$$

Definition 13 (t-Strong Diffie-Hellman assumption [BB04]) *Let p be a prime number generated according to a security parameter $k \in \mathbb{N}$. Let $t \in \mathbb{N}$ be an integer, $\mathbb{G}$ be a multiplicative group of order p, and $g \in \mathbb{G}$ be a generator. The* t-Square Decisional Diffie-Hellman assumption *(t-SDH) in $(\mathbb{G}, p, g)$ states that there exists a negligible function ϵ such that for all $\mathscr{A} \in \textsc{poly}(k)$:*

$$\Pr\left[x \xleftarrow{\$} \mathbb{Z}_p^*; (c,h) \leftarrow \mathscr{A}\left((g^{x^i})_{0\le i\le t}\right) : \left(c \in \mathbb{Z}_p \backslash \{-x\}\right) \wedge \left(h = g^{\frac{1}{x+c}}\right)\right] = \epsilon(k)$$

Remark 1 *Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three groups of prime order p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear pairing of type 2 or 3 and $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be two group elements. The DDH assumption does not hold in both $(\mathbb{G}_1, p, g_1)$ and $(\mathbb{G}_2, p, g_2)$. We show how to break the DDH assumption when e is a type 1 pairing, i.e., $\mathbb{G} = \mathbb{G}_1 = \mathbb{G}_2$ and $g = g_1 = g_2$: knowing g^x, g^y and g^{z_b}, we can check whether $e(g^x, g^y) = e(g, g^{z_b})$, which is equivalent to check whether $g^{x\cdot y} = g^{z_b}$ because $e(g^x, g^y) = e(g, g^{x\cdot y})$. We can show that the DDH assumption does not hold when e is a type 2 pairing in a similar way.*

Definition 14 (Bilinear Computational Diffie-Hellman assumption [Jou04]) *Let a prime number p generated according to a security parameter $k \in \mathbb{N}$ be. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three groups of order p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear pairing and $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be two group elements. The* Bilinear Computational Diffie-Hellman assumption *(BCDH) in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ states that there exists a negligible function ϵ such that for all $\mathscr{A} \in \textsc{poly}(k)$ and for all $(\alpha, \beta, \gamma) \in \{1,2\}^3$:*

$$\Pr\left[(x,y,z) \xleftarrow{\$} (\mathbb{Z}_p^*)^3; h \leftarrow \mathscr{A}(g_\alpha^x, g_\beta^y, g_\gamma^z) : h = e(g_1, g_2)^{x\cdot y\cdot z}\right] = \epsilon(k)$$

Definition 15 (Bilinear Decisional Diffie-Hellman assumption [Jou04]) *Let p be a prime number generated according to a security parameter $k \in \mathbb{N}$. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three groups of order p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear pairing and $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be two group elements.*

The Decisional Diffie-Hellman assumption *(BDDH) in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ *states that there exists a negligible function* ϵ *such that for all* $\mathcal{A} \in \text{POLY}(k)$ *and for all* $(\alpha, \beta, \gamma) \in \{1, 2\}^3$*:*

$$\left| \Pr\left[\begin{array}{c} (x, y, z) \xleftarrow{\$} (\mathbb{Z}_p^*)^3; h_0 \xleftarrow{\$} \mathbb{G}_T; h_1 = e(g_1, g_2)^{x \cdot y \cdot z}; \\ b \xleftarrow{\$} \{0, 1\}; b' \leftarrow \mathcal{A}(g_\alpha^x, g_\beta^y, g_\gamma^z, h_b) \end{array} : b = b' \right] - \frac{1}{2} \right| = \epsilon(k)$$

Definition 16 (Fixed Argument Pairing Inversion assumption [GHV08]) *Let p be a prime number generated according to a security parameter* $k \in \mathbb{N}$*. Let* $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$ *be three groups of order* p*,* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *be a non-degenerate bilinear pairing and* $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$ *be two group elements.The* Fixed Argument Pairing Inversion 1 *assumption (FAPI1) in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ *states that there exists a negligible function* ϵ *such that for all* $\mathcal{A} \in \text{POLY}(k)$*:*

$$\Pr\left[X \xleftarrow{\$} \mathbb{G}_1; X' \leftarrow \mathcal{A}(e(X, g_2)) : X = X' \right] = \epsilon(k)$$

The Fixed Argument Pairing Inversion 2 *assumption (FAPI2) in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ *states that there exists a negligible function* ϵ *such that for all* $\mathcal{A} \in \text{POLY}(k)$*:*

$$\Pr\left[X \xleftarrow{\$} \mathbb{G}_2; X' \leftarrow \mathcal{A}(e(g_1, X)) : X = X' \right] = \epsilon(k)$$

We define a variant of the DL assumption in a bilinear mapping that is computationally equivalent to the DL assumption.

Definition 17 (Bilinear Discrete Logarithm Variant assumption) *Let p be a prime number generated according to a security parameter* $k \in \mathbb{N}$*. Let* $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$ *be three groups of order* p*,* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *be a non-degenerate type 2 bilinear pairing and* $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$ *be two group elements. The* Bilinear Discrete Logarithm Variant *(BDLV) in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ *states that there exists a negligible function* ϵ *such that for all* $\mathcal{A} \in \text{POLY}(k)$*:*

$$\Pr\left[x \xleftarrow{\$} \mathbb{Z}_p^*; x' \leftarrow \mathcal{A}(g_1^x, g_2^{\frac{1}{x}}) : x = x' \right] \leq \epsilon(k)$$

Theorem 1 *The BDLV assumption in* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ *holds under the DL assumption in* $(\mathbb{G}_2, p, g_2)$*.*

Proof: Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three groups of prime order p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a type 2 non-degenerate bilinear pairing and $g_2 \in \mathbb{G}_2$ be a generator. Suppose that there exists a polynomial time algorithm $\mathcal{A}$ that breaks the BDLV assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ for any g_1 chosen in the uniform distribution on $\mathbb{G}_1$. We construct the polynomial time algorithm $\mathcal{B}$ that breaks the DL assumption in $(\mathbb{G}_2, p, g_2)$. $\mathcal{B}$ receives $h_2 = g_2^x$ as input. Since e is a type 2 pairing, there exists a computationally efficient morphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$. $\mathcal{B}$ picks $a \in \mathbb{Z}_p^*$ and computes $h_1 = \phi(g_2^a)$ and $g_1 = \phi(h_2^a) = \phi((g_2^a)^x) = h_1^x$. Thus, $g_1^{1/x} = h_1$, and (h_1, h_2) is an instance of BDLV in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ with regards to the solution $y = 1/x$. $\mathcal{B}$ runs $y' \leftarrow \mathcal{A}(h_1, h_2)$ and returns $y' = 1/x'$. We observe that $y' = y \Leftrightarrow x' = 1/y = x$. We conclude that $\mathcal{B}$ breaks DL in $(\mathbb{G}_2, p, g_2)$ in polynomial time with the same probability that $\mathcal{A}$ breaks BDLV in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. □

Note that this proof requires the hypothesis that e is a type 2 pairing. To the best of our knowledge, there is neither known reduction to DL in $(\mathbb{G}_2, g_2)$ where e is a type 3 pairing, nor known polynomial time algorithm that breaks BDLV.

2.4 Hash Function

In this section, we recall the definition of hash functions and the random oracle model.

2.4.1 Formal Definition

A hash function transforms a given bit-string into a fixed size hash value. Such a function is secure when it is hard to inverse it, and when it is hard to find two bit-strings that correspond to the same hash value.

Definition 18 (Hash function) *A* hash function *is a deterministic function* $\mathrm{H} : \{0,1\}^* \rightarrow \mathcal{I}$*. Such a function is said to be* secure *for the security parameter* $k \in \mathbb{N}$ *when* H *verifies the two following properties:*

Pre-image resistance: *For all algorithm* $\mathcal{A} \in \textsc{poly}(k)$*, there exists a negligible function* ϵ *such that:*

$$\Pr\left[h \xleftarrow{\$} \mathcal{I}; m \leftarrow \mathcal{A}(\mathrm{H}, h) : \mathrm{H}(m) = h\right] \leq \epsilon(k)$$

Collusion resistance: *For all algorithm* $\mathcal{A} \in \textsc{poly}(k)$*, there exists a negligible function* ϵ *such that:*

$$\Pr\left[(m_0, m_1) \leftarrow \mathcal{A}(\mathrm{H}) : \mathrm{H}(m_0) = \mathrm{H}(m_1)\right] \leq \epsilon(k)$$

SHA-2 [Dan15] is an example of family of hash functions considered secure in practice.

2.4.2 The Random Oracle Model

In this thesis, some security properties are proven in the *random oracle model, i.e.,* under the assumption that each hash function is replaced by an oracle that responds to a given bit-string with a random value chosen from uniform distribution on the output domain of the hash function. Proofs that do not use this heuristic are said in the *standard model.* Most of the time, a security proof in the random oracle model is sufficient to justify the security of a cryptographic protocol in practice, even if from a theoretical point of view, a proof of security in the standard model is stronger (and more difficult to obtain) than a proof in the random oracle model.

Definition 19 (Random Oracle Model [BR93]) *We say that a cryptographic scheme is secure in the* random oracle model *(ROM) when its security proof requires the hypothesis that any hash function* $\mathrm{H} : \{0,1\}^* \rightarrow \mathcal{I}$ *can be replaced by a black box* $\mathrm{H}(\cdot)$ *(called the* random oracle*) defined as follows, where* m_i *(resp.* h_i*) is the* i^{th} *input (resp. output) of* $\mathrm{H}(\cdot)$*:*

Random oracle $\mathrm{H}(\cdot)$**:** *On the* i^{th} *input* m_i*, if there exists* $j \leq i$ *such that* $m_j = m_i$*, this oracle sets* $h_i = h_j$*, else it picks* $h_i \xleftarrow{\$} \mathcal{I}$*. It returns* h_i*.*

2.5 Public Key Encryption

Public key encryption allows the users to encrypt messages thanks to public keys. However, the corresponding secret key is required to decrypt the ciphertexts.

2.5.1 Formal Definition

Definition 20 (Public Key Encryption) *A* Public Key Encryption *scheme (PKE) is a tuple of polynomial time algorithms* $(\mathsf{Set}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*

$\mathsf{Set}(k)$**:** *It returns a setup* set*, a set of messages* $\mathcal{M}$ *and a set of random coins* $\mathcal{C}$*.*
$\mathsf{Gen}(\mathsf{set})$**:** *It returns a public/private key pair* $(\mathsf{pk}, \mathsf{sk})$*.*
$\mathsf{Enc}_{\mathsf{pk}}(m, r)$**:** *It returns a ciphertext* c*.*
$\mathsf{Dec}_{\mathsf{sk}}(c)$**:** *It returns a plaintext* m *or the bottom symbol* $\perp$*.*

Moreover a PKE is said to be correct *when the following equation holds for any* $k \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{c} (\mathsf{set}, \mathcal{M}, \mathcal{C}) \leftarrow \mathsf{Set}(k); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{set}); \\ m \xleftarrow{\$} \mathcal{M}; r \xleftarrow{\$} \mathcal{C}; \\ c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m, r); m' \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c) \end{array} : m' = m\right] = 1$$

Remark 2 *In practice, the setup algorithm* $\mathsf{Set}(k)$ *is run once, and the outputted setup* set *is publicly known. As it is clear from the context, we omit it in the description of both algorithms* Enc *and* Dec*, although it is obviously used as input by these algorithms. This remark can be applied to the other primitives defined throughout this document.*

Remark 3 *In Definition 20,* Enc *is described as a deterministic algorithm that takes as input a public key* pk*, a message* m *and a random coin* r*. Most of the time, in order to make the notation less cluttered, we assume that the algorithm is probabilistic, i.e., it is able to generate the random value* r *by itself. In this thesis, we will define properties that require that public key encryptions are formalized as in Definition 20. However, in the other cryptographic primitives we will define, all algorithms are assumed to be probabilistic.*

2.5.2 Security Against Chosen Plaintext Attack

The *indistinguishability against chosen-plaintext attack* is the basic security requirement for public key encryptions. Informally, it implies that a ciphertext leaks no information about the corresponding plaintext. Consider an adversary that chooses two messages m_0 and m_1, and that receives the encryption of one of the two messages. If such an adversary is not able to guess the chosen message with a significant probability (*i.e.,* significantly different to $1/2$, the probability of guessing at random), then the public key encryption scheme is said to be indistinguishable against chosen-plaintext attack.

Definition 21 (Indistinguishability against Chosen-Plaintext Attack [GM84]) *Let the tuple of algorithms* $\mathrm{E} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE,* $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$ *be a two-party algorithm and* $k \in \mathbb{N}$ *be a security parameter. We define the IND-CPA experiment for the adversary* $\mathscr{A}$ *against* E *as follows.*

```
$\mathsf{Exp}^{\textbf{IND-CPA}}_{\mathrm{E},\mathscr{A}}(k)$:
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{set}, \mathscr{M}, \mathscr{C}) \leftarrow \mathsf{Set}(k)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{set})$
$(m_0, m_1, st) \leftarrow \mathscr{A}_1(\mathsf{set}, \mathsf{pk})$
$r \xleftarrow{\$} \mathscr{C}$
$c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b, r)$
$b' \leftarrow \mathscr{A}_2(\mathsf{st}, \mathsf{pk}, c)$
return $(b = b')$
```

We define the IND-CPA advantage of $\mathscr{A}$ *against* E *as follows:*

$$\mathsf{Adv}^{\textit{IND-CPA}}_{\mathrm{E},\mathscr{A}}(k) = \left|\Pr\left[1 \leftarrow \mathsf{Exp}^{\textit{IND-CPA}}_{\mathrm{E},\mathscr{A}}(k)\right] - \frac{1}{2}\right|$$

We define the IND-CPA advantage against E *as follows:*

$$\mathsf{Adv}^{\textit{IND-CPA}}_{\mathrm{E}}(k) = \max_{\mathscr{A} \in \mathrm{POLY}(k)^2} \left\{\mathsf{Adv}^{\textit{IND-CPA}}_{\mathrm{E},\mathscr{A}}(k)\right\}$$

E *is said to be* indistinguishable against chosen-plaintext attack*, or IND-CPA secure, when the advantage* $\mathsf{Adv}^{\textit{IND-CPA}}_{\mathrm{E}}(k)$ *is negligible.*

The *indistinguishability against multiple chosen-plaintexts attack* is an extension of the previous property where the adversary receives several public keys and chooses several couples of messages (m_0, m_1). For each of them, the adversary receives the encryption of m_b where $b \in \{0,1\}$ (the same b is used each time). The goal of the adversary is to guess b with a probability that is significantly different to $1/2$.

Definition 22 (Indistinguishability against multiple Chosen-Plaintexts Attack [BBM00]) *Let* $\mathrm{E} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE,* $\mathscr{A}$ *be an algorithm,* $\alpha \in \mathbb{N}$ *and* $\beta \in \mathbb{N}$ *be two integers and* $k \in \mathbb{N}$ *be a security parameter. For all* i *in* $[\![1,\beta]\!]$ *,let the following oracle be:*

$\mathsf{Enc}_{\mathsf{pk}_i}(\mathsf{LR}_b(\cdot,\cdot),*)$**:** *On input* (m_0, m_1)*, this algorithm picks* $r \xleftarrow{\$} \mathscr{C}$*, runs* $c \leftarrow \mathsf{Enc}_{\mathsf{pk}_i}(m_b, r)$ *and returns* c*. This oracle cannot be called more than* α *times.*

We define the IND-CPA$_{\alpha,\beta}$ *experiment for the adversary* $\mathscr{A}$ *against* E *as follows.*

```
$\mathsf{Exp}^{\textbf{IND-CPA}_{\alpha,\beta}}_{\mathrm{E},\mathscr{A}}(k)$:
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{set}, \mathscr{M}, \mathscr{C}) \leftarrow \mathsf{Set}(k)$
$\forall i \in [\![1,\beta]\!]], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{set})$
$b' \leftarrow \mathscr{A}^{\mathsf{Enc}_{\mathsf{pk}_1}(\mathsf{LR}_b(\cdot,\cdot),*),\ldots,\mathsf{Enc}_{\mathsf{pk}_\beta}(\mathsf{LR}_b(\cdot,\cdot),*)}(\mathsf{set}, \mathsf{pk}_1, \ldots \mathsf{pk}_\beta)$
return $(b = b')$
```

We define the IND-CPA$_{\alpha,\beta}$ *advantage of* $\mathscr{A}$ *against* E *as follows:*

$$\mathsf{Adv}^{\textit{IND-CPA}_{\alpha,\beta}}_{\mathrm{E},\mathscr{A}}(k) = \left|\Pr\left[1 \leftarrow \mathsf{Exp}^{\textit{IND-CPA}_{\alpha,\beta}}_{\mathrm{E},\mathscr{A}}(k)\right] - \frac{1}{2}\right|$$

We define the IND-CPA$_{\alpha,\beta}$ *advantage against* E *as follows:*

$$\mathsf{Adv}^{\textit{IND-CPA}_{\alpha,\beta}}_{\mathrm{E}}(k) = \max_{\mathscr{A} \in \mathrm{POLY}(k)} \left\{\mathsf{Adv}^{\textit{IND-CPA}_{\alpha,\beta}}_{\mathrm{E},\mathscr{A}}(k)\right\}$$

E *is said to be* (α,β)-indistinguishable against multiple chosen-plaintext attack*, or* IND-CPA$_{\alpha,\beta}$ *secure, when* $\mathsf{Adv}^{\textit{IND-CPA}_{\alpha,\beta}}_{\mathrm{E}}(k)$ *is negligible.*

If $\beta = 1$*, then we ommit it in the notation. we denote* IND-CPA$_{\alpha,1}$ *(resp.* $\mathsf{Exp}^{\textit{IND-CPA}_{\alpha,1}}_{\mathrm{E},\mathscr{A}}(k)$*,* $\mathsf{Adv}^{\textit{IND-CPA}_{\alpha,1}}_{\mathrm{E},\mathscr{A}}(k)$ *and* $\mathsf{Adv}^{\textit{IND-CPA}_{\alpha,1}}_{\mathrm{E}}(k)$*) by* IND-CPA$_\alpha$ *(resp.* $\mathsf{Exp}^{\textit{IND-CPA}_{\alpha}}_{\mathrm{E},\mathscr{A}}(k)$*,* $\mathsf{Adv}^{\textit{IND-CPA}_{\alpha}}_{\mathrm{E},\mathscr{A}}(k)$ *and* $\mathsf{Adv}^{\textit{IND-CPA}_{\alpha}}_{\mathrm{E}}(k)$*), and we say that* E *is* α-indistinguishable against multiple chosen-plaintext attack*, or* IND-CPA$_\alpha$ *secure, when* $\mathsf{Adv}^{\textit{IND-CPA}_{\alpha}}_{\mathrm{E}}(k)$ *is negligible*

In [BBM00], Bellare, Boldyreva and Micali show that the indistinguishability against chosen-plaintext attack is equivalent to the indistinguishability against multiple chosen-plaintexts attack:

Theorem 2 *Let* E *be a PKE.* E *is* IND-CPA *secure if and only if* E *is* IND-CPA$_{t(k)}$ *for any polynomial* t*.*

2.5.3 Additional Properties

We introduce the notion of *Random Coin Decryptable* (RCD) public key encryption. Each ciphertext outputted by the encryption algorithm depends to a random coin (denoted by r in Definition 33). Informally, a public key encryption scheme is said to be a RCD if it is possible to decrypt a ciphertext using this random coin as a secret key. This primitive is a kind of PKE with *double decryption* mechanism (DD-PKE) which is defined in [GH08]. Actually, a RCD public key encryption is a DD-PKE where the second secret key is the random coin and where this key is used once.

Definition 23 (Random Coin Decryptable PKE) *A probabilistic PKE is* Random Coin Decryptable *(RCD) if:*

1. *there exists a polynomial time algorithm* CDec *defined as follows:*

   $\mathsf{CDec}_r(c, \mathsf{pk})$**:** *It returns a plaintext* m *or a bottom symbol* $\perp$*.*

2. *the following equation holds for any $k \in \mathbb{N}$:*

$$\Pr\left[\begin{array}{c}(\mathsf{set}, \mathcal{M}) \leftarrow \mathsf{Set}(k); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{set}); \\ m \xleftarrow{\$} \mathcal{M}; r \xleftarrow{\$} \mathcal{C}; \\ c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m, r); m' \leftarrow \mathsf{CDec}_r(c, \mathsf{pk})\end{array} : m' = m\right] = 1$$

We introduce the concepts of *valid key pair* and of *verifiable key* public key encryption. Informally, a key pair $(\mathsf{pk}, \mathsf{sk})$ is valid when any message encrypted by pk will be correctly decrypted by sk. A public key encryption is said to be *verifiable key* when it is possible to check whether a key pair is valid or not.

Definition 24 (Verifiable Key PKE) *Let* $\mathrm{E} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE. We say that a key pair* $(\mathsf{pk}, \mathsf{sk})$ *is* valid *for* E *according to the setup* $(\mathsf{set}, \mathcal{M}, \mathcal{C})$ *when for any message $m \in \mathcal{M}$ and any random coin $r \in \mathcal{C}$:*

$$\Pr\left[c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m, r); m' \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c) : m' = m\right] = 1$$

We say that E *is* verifiable-key *(VK) when there exists an algorithm* Ver *such that* $1 \leftarrow \mathsf{Ver}(\mathsf{pk}, \mathsf{sk})$ *if and only if* $(\mathsf{pk}, \mathsf{sk})$ *is valid for* E *according to the setup* $(\mathsf{set}, \mathcal{M}, \mathcal{C})$.

2.5.4 ElGamal Encryption Scheme

The ElGamal Cryptosystem is a well known public key encryption scheme based on the discret logarithm problem. It is known to be IND-CPA secure under the decisional Diffie-Hellman assumption.

Definition 25 (ElGamal Cryptosystem [ElG85]) *The ElGamal cryptosystem scheme* $\mathsf{ElGamal} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a PKE defined as follows:*

$\mathsf{Set}(k)$**:** *It generates a prime order group setup* $\mathsf{set} = (\mathbb{G}, p, g)$*, sets* $\mathcal{M} = \mathbb{G}$ *and* $\mathcal{C} = \mathbb{Z}_p^*$*, and returns* $(\mathsf{set}, \mathcal{M}, \mathcal{C})$.

$\mathsf{Gen}(\mathsf{set})$**:** *It picks* $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$*, sets* $\mathsf{pk} = g^{\mathsf{sk}}$ *and returns* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Enc}_{\mathsf{pk}}(m, r)$**:** *It returns* $c = (g^r, \mathsf{pk}^r \cdot m)$.

$\mathsf{Dec}_{\mathsf{sk}}(c)$**:** *It parses* $c = (c_1, c_2)$ *and returns* $m = \frac{c_2}{c_1^{\mathsf{sk}}}$.

Theorem 3 ElGamal *is IND-CPA secure under the DDH assumption.*

The proof is given in [Sho04]. Finally, we show that the ElGamal encryption is both random coin decryptable and verifiable key.

Theorem 4 ElGamal *is RCD and VK.*

Proof: To show that ElGamal is RCD, we show how to build the algorithm CDec:

$\mathsf{CDec}_r(c, \mathsf{pk})$**:** It parses $c = (c_1, c_2)$ and returns $m = \frac{c_2}{\mathsf{pk}^r}$.

Clearly, this algorithm returns the message m for any ciphertext $(c_1, c_2) = (g^r, \mathsf{pk}^r \cdot m)$ with probability 1. On the other hand, to show that ElGamal is VK, we show how to build its verification algorithm Ver:

$\mathsf{Ver}(\mathsf{pk}, \mathsf{sk})$**:** It returns 1 if and only if $g^{\mathsf{sk}} = \mathsf{pk}$.

For any ciphertext $(c_1, c_2) = (g^r, \mathsf{pk}^r \cdot m)$, the equation $g^{\mathsf{sk}} = \mathsf{pk}$ implies that $m = \frac{c_2}{c_1^{\mathsf{sk}}}$, which concludes the proof. □

Many other public key encryption schemes in the literature can easily be proven to be RCD in a similar way, *e.g.* [CS03, ABR98, FO13]. Moreover, in many PKE, the public key is generated from the secret key in a deterministic way as in the ElGamal cryptosystem (for instance, [CS03, ABR98]). It is easy to see that this kind of encryption scheme is always VK.

2.6 Proof of Knowledge

A *zero-knowledge proof of knowledge* is a protocol that allows a prover to convince a verifier that it knows the secret solution sk of some public instance pk of a computational problem. Such a protocol has the following properties:

- The *correctness* assures that if the prover knows the solution sk, then he is able to convince the verifier.
- The *soundness* assures that if pk is not a valid instance of the computational problem, then the prover is not able to convince the verifier.
- The *validity* assures that if the prover does not know the secret sk, then he is not able to convince the verifier.
- Finally, the *zero-knowledge* property assures that no information about the secret solution sk leaks during the protocol.

2.6.1 Formal Definitions

Definition 26 (Interactive proof systems [GMR89]) *An* interactive proof *system (IP) is a couple* $\mathrm{P} = (\mathsf{Set}, \mathsf{Proof})$ *defined as follows:*

$\mathsf{Set}(k)$**:** *It is an algorithm that returns a setup* set *and a binary relation* $\mathscr{R}$*.*

$\mathsf{Proof}\langle \mathrm{P}(\mathsf{sk});\mathrm{V}(\mathsf{pk})\rangle$**:** *It is a two-party protocol between two probabilistic polynomial time algorithms* P *(the* prover*) and* V *(the* verifier*). At the end of the protocol,* P *returns the bottom symbol* $\perp$ *and* V *returns a bit* b*.*

Moreover, we define the following properties:

Completeness: *An IP is* complete *when for any* $k \in \mathbb{N}$, $(\mathsf{set},\mathscr{R}) \leftarrow \mathsf{Set}(k)$ *and* $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}$*, the following equation holds:*

$$\Pr\left[b \leftarrow \mathtt{out}_{\mathrm{V}}\left(\mathsf{Proof}\langle \mathrm{P}(\mathsf{sk});\mathrm{V}(\mathsf{pk})\rangle\right) : b = 1\right] = 1$$

Soundness: *An IP is* sound *when for any* $k \in \mathbb{N}$, $(\mathsf{set},\mathscr{R}) \leftarrow \mathsf{Set}(k)$ *and* $\mathsf{pk}^* \notin \mathscr{L}_{\mathscr{R}}$*, there exists a negligible function* ϵ *such that for any dishonest prover* $\mathrm{P}^* \in \mathrm{POLY}(k)$*:*

$$\Pr\left[b \leftarrow \mathtt{out}_{\mathrm{V}}\left(\mathsf{Proof}\langle \mathrm{P}^*(\mathsf{pk}^*);\mathrm{V}(\mathsf{pk}^*)\rangle\right) : b = 1\right] \leq \epsilon(k)$$

Validity [BG93]: *An IP is* valid*, or is an* interactive proof of knowledge *(POK), when for any* $k \in \mathbb{N}$ *and* $(\mathsf{set},\mathscr{R}) \leftarrow \mathsf{Set}(k)$*, there exists a polynomial time algorithm* K *called* the knowledge extractor *such that for any (possibly dishonest and unbounded) prover* P^**, any* $\mathsf{pk} \in \mathscr{L}_{\mathscr{R}}$ *and any bit-string* $\mathsf{in} \in \{0,1\}^*$*, there exist a negligible function* ϵ *and a polynomial* t *such that:*

$$\Pr\left[sk \leftarrow \mathsf{K}^{\mathrm{P}^*(\mathsf{in})}(\mathsf{pk}) : (\mathsf{pk},\mathsf{sk}) \in \mathscr{R}\right] \geq t\left(\Pr\left[b \leftarrow \mathtt{out}_{\mathrm{V}}\left(\mathsf{Proof}\langle \mathrm{P}^*(\mathsf{in});\mathrm{V}(\mathsf{pk})\rangle\right) : b = 1\right]\right) - \epsilon(k)$$

Zero-knowledge: *An IP is* Zero-Knowledge *(IZKP), when for any* $k \in \mathbb{N}$*, any* $(\mathsf{set},\mathscr{R}) \leftarrow \mathsf{Set}(k)$*, any* $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}$ *and any (possibly dishonest) verifier* $\mathrm{V}^* \in \mathrm{POLY}(k)$*, there exists a polynomial time algorithm* Sim *called* the simulator *such that for any* $\alpha \in \{0,1\}^*$*, the following equation holds:*

$$\Pr\left[\alpha_* \leftarrow \mathtt{view}_{\mathrm{V}^*}\left(\mathsf{Proof}\langle \mathrm{P}(\mathsf{sk});\mathrm{V}^*(\mathsf{pk})\rangle\right) : \alpha = \alpha_*\right] = \Pr\left[\alpha_* \leftarrow \mathsf{Sim}(\mathsf{pk}) : \alpha = \alpha_*\right]$$

Honest-verifier Zero-Knowledge: *This property is the same as the* Zero-knowledge *one except that the verifier is* honest*, i.e.,* V *correctly runs the protocol.*

When there is no interaction between the prover and the verifier, such a proof system is called *non-interactive proof systems.*

Definition 27 (Non-interactive proof systems [GMR89]) *A* non-interactive proof *system (NIP) is a tuple of algorithms* $\mathsf{P} = (\mathsf{Set}, \mathsf{Pro}, \mathsf{Ver})$ *defined as follows:*

$\mathsf{Set}(k)$**:** *It returns a setup* set *and a binary relation* $\mathscr{R}$.
$\mathsf{Pro}(\mathsf{sk}, \mathsf{pk})$**:** *It returns a proof* π.
$\mathsf{Ver}(\mathsf{pk}, \pi)$**:** *It returns a bit* b.

Moreover, we define the following properties:

Completeness: *A NIP is* complete *when for any* $k \in \mathbb{N}$, $(\mathsf{set}, \mathscr{R}) \leftarrow \mathsf{Set}(k)$ *and* $(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}$*, the following equation holds:*

$$\Pr\left[\pi \leftarrow \mathsf{Pro}(\mathsf{sk}, \mathsf{pk}); b \leftarrow \mathsf{Ver}(\mathsf{pk}, \pi) : b = 1\right] = 1$$

Soundness: *An NIP is* sound *when for any* $k \in \mathbb{N}$ *and* $(\mathsf{set}, \mathscr{R}) \leftarrow \mathsf{Set}(k)$*, there exists a negligible function* ϵ *such that for any algorithm* $\mathscr{A} \in \textsc{poly}(k)$*:*

$$\Pr\left[(\mathsf{pk}^*, \pi) \leftarrow \mathscr{A}(\mathsf{set}); b \leftarrow \mathsf{Ver}(\mathsf{pk}^*, \pi) : (b = 1) \wedge (\mathsf{pk}^* \not\in \mathscr{L}_{\mathscr{R}})\right] \leq \epsilon(k)$$

(Non-Adaptive) Validity: *An NIP is (non-adaptive)* valid*, or is a (non-adaptive)* non-interactive proof of knowledge *(NIPOK), when for any* $k \in \mathbb{N}$ *and* $(\mathsf{set}, \mathscr{R}) \leftarrow \mathsf{Set}(k)$*, there exists a polynomial time algorithm* K *called* the knowledge extractor *such that for any (possibly unbounded) algorithm* $\mathscr{A}$*, and any* $\mathsf{pk} \in \mathscr{L}_{\mathscr{R}}$*, there exist a negligible function* ϵ *and a polynomial* t *such that:*

$$\Pr\left[sk \leftarrow \mathsf{K}^{\mathscr{A}(\mathsf{pk})}(\mathsf{pk}) : (\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}\right] \geq t\left(\Pr\left[\pi \leftarrow \mathscr{A}(\mathsf{pk}); b \leftarrow \mathsf{Ver}(\mathsf{pk}, \pi) : b = 1\right]\right) - \epsilon(k)$$

Zero-knowledge: *An NIP is* Zero-Knowledge *(NIZKP), when for any* $k \in \mathbb{N}$*, any* $(\mathsf{set}, \mathscr{R}) \leftarrow \mathsf{Set}(k)$ *and any* $(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}$*, there exists a polynomial time algorithm* Sim *called* the simulator *such that for any* $\alpha \in \{0,1\}^*$*, the following equation holds:*

$$\Pr\left[\alpha_* \leftarrow \mathsf{Pro}(\mathsf{sk}, \mathsf{pk}) : \alpha = \alpha_*\right] = \Pr\left[\alpha_* \leftarrow \mathsf{Sim}(\mathsf{pk}) : \alpha = \alpha_*\right]$$

Remark 4 *In Definition 27, we define the non-adaptive validity of non-interactive proofs of knowledge. There exists a stronger definition of validity where the algorithm* $\mathscr{A}$ *has access to more information than* pk *[BPW12], especially proofs for chosen statments* $\mathsf{pk}' \in \mathscr{L}_{\mathscr{R}}$ *such that* $\mathsf{pk}' \neq \mathsf{pk}$*. In [BPW12], Bernhard et al. show that several cryptographic protocols are not secure because they use non-adaptive valid proofs. However, the authors show that non-adaptive validity is sufficient for several applications, as identification protocols, or signatures based on non-interactive proofs of knowledge. Since in this thesis, we only focus on non-interactive proofs for such applications, then the non-adaptive validity is sufficient for our purposes.*

Note that validity implies soundness. Indeed, if a proof system is valid, then for any prover that have a non-negligible probability to success a proof on the instance pk, there is a knowldege extractor that returns sk such that $(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}$ with non-negligible probability, which implies that $\mathsf{pk} \in \mathscr{L}_{\mathscr{R}}$.

Theorem 5 *Let* P *be a (N)IP. If* P *is valid (or is non-adaptive valid), then* P *is sound.*

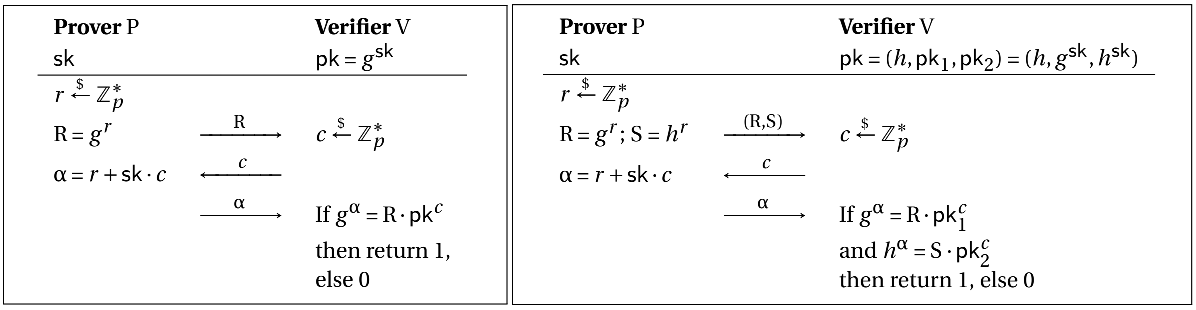

Prover P
sk
Verifier V
$\mathsf{pk} = g^{\mathsf{sk}}$
$r \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{R} = g^r$
$\xrightarrow{\mathrm{R}}$
$c \xleftarrow{\$} \mathbb{Z}_p^*$
$\alpha = r + \mathsf{sk} \cdot c$
$\xleftarrow{c}$
$\xrightarrow{\alpha}$
If $g^\alpha = \mathrm{R} \cdot \mathsf{pk}^c$
then return 1,
else 0
Prover P
sk
Verifier V
$\mathsf{pk} = (h, \mathsf{pk}_1, \mathsf{pk}_2) = (h, g^{\mathsf{sk}}, h^{\mathsf{sk}})$
$r \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{R} = g^r; \mathrm{S} = h^r$
$\xrightarrow{(\mathrm{R},\mathrm{S})}$
$c \xleftarrow{\$} \mathbb{Z}_p^*$
$\alpha = r + \mathsf{sk} \cdot c$
$\xleftarrow{c}$
$\xrightarrow{\alpha}$
If $g^\alpha = \mathrm{R} \cdot \mathsf{pk}_1^c$
and $h^\alpha = \mathrm{S} \cdot \mathsf{pk}_2^c$
then return 1, else 0

Figure 2.1: Protocols Proof of Schnorr (left) and LogEq (right).

2.6.2 Schnorr Interactive Proof System

The following interactive proof system, called the *Schnorr protocol*, is a well known zero-knowledge proof of knowledge that allows some prover to prove that it knows the discret logarithm of an element of a prime order group.

Definition 28 (Schnorr interactive proof system [Sch90]) *The Schnorr proof system* Schnorr = (Set, Proof) *is a zero-knowledge interactive proof of knowledge system where* Proof *is defined as in Fig. 2.1 and where* Set *is defined as follows:*

Set(k)**:** *It generates a prime order group setup* set $= (\mathbb{G}, p, g)$ *and returns* $(\mathsf{set}, \mathscr{R})$, *where* $(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R} \Leftrightarrow (\mathsf{sk} \in \mathbb{Z}_p^*) \wedge (g^{\mathsf{sk}} = \mathsf{pk})$.

2.6.3 An Interactive Proof System For Discrete Logarithm Equality

We show a zero-knowledge proof of knowledge, called LogEq, that allows some prover to prove that two elements of a prime order group have the same discrete logarithm, and to prove that he knows this discrete logarithm.

Definition 29 (LogEq **interactive proof system [CP93])** *The* LogEq *interactive proof system* LogEq = (Set, Proof) *is a zero-knowledge interactive proof of knowledge system where* Proof *is defined as in Fig. 2.1 and where* Set *is defined as follows:*

Set(k)**:** *It generates a prime order group setup* set $= (\mathbb{G}, p, g)$ *and returns* $(\mathsf{set}, \mathscr{R})$, *where the following equivalence holds for* $\mathsf{pk} = (h, \mathsf{pk}_1, \mathsf{pk}_2)$*:*

$$(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R} \Leftrightarrow \left(\mathsf{pk} \in \mathbb{G}^3\right) \wedge \left(\mathsf{pk}_1 = g^{\mathsf{sk}}\right) \wedge \left(\mathsf{pk}_2 = h^{\mathsf{sk}}\right)$$

2.6.4 Sigma Protocols

A *sigma protocol* is an interactive proof of knowledge where the proof protocol is in three steps: the prover sends a *commitment* to the verifier, the verifier sends a *challenge* to the prover, and the verifier sends a *response* to the verifier. Note that Schnorr and LogEq are simga protocols.

Definition 30 (Sigma protocol [FKI06]) *An interactive proof* P = (Set, Proof) *is a* sigma protocol *when there exist a triplet of polynomial time algorithms* (Commit, Response, Check) *and a challenge set* $\mathscr{C}$ *such that the protocol* Proof *can be describe as in Fig. 2.2.*

Theorem 6 Schnorr *is a sigma protocol, moreover, it is complete, sound, valid and honest verifier zero-knowledge.*

The proof of this theorem is given in [Sch90].

Theorem 7 LogEq *is a sigma protocol, moreover, it is complete, sound, valid and honest verifier zero-knowledge.*

The proof of this theorem is given in [CP93].

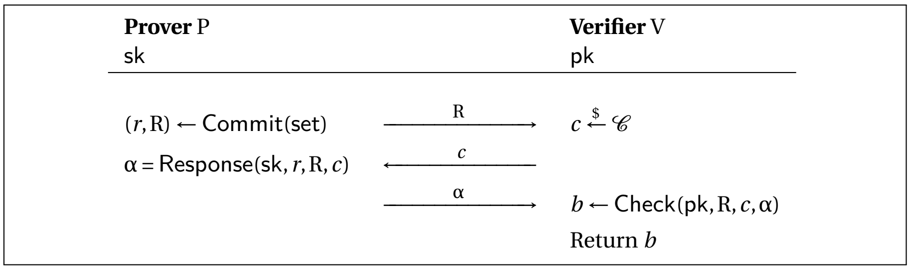

Prover P
sk
Verifier V
pk
$(r, \mathrm{R}) \leftarrow \mathsf{Commit}(\mathsf{set})$
R
$c \overset{\$}{\leftarrow} \mathscr{C}$
$\alpha = \mathsf{Response}(\mathsf{sk}, r, \mathrm{R}, c)$
c
α
$b \leftarrow \mathsf{Check}(\mathsf{pk}, \mathrm{R}, c, \alpha)$
Return b

Figure 2.2: Protocol Proof of a sigma protocol.

2.6.5 The Fiat-Shamir Transformation

The *Fiat-Shamir transformation* is a method to change a sigma protocol into a non-interactive proof system for the same language in the random oracle model. The idea is to use the hash of the commitment and the public statment as challenge. Since the prover can compute the challenge by himself, he does not need to interact with the verifier.

Definition 31 (Fiat-Shamir Transformation [FS87]) *Let* P *be a sigma protocol defined as in Def. 30. The* Fiat-Shamir Transformation *(FST) of* P *is the non-interactive proof system* $\mathrm{P}^{\mathrm{NI}} = (\mathsf{Set}, \mathsf{Pro}, \mathsf{Ver})$ *defined as follows:*

$\mathsf{Set}(k)$**:** *It generates* $(\mathsf{set}, \mathscr{R})$ *using the setup algorithm of* P*, generates a hash function* $\mathrm{H} : \{0,1\}^* \to \mathscr{C}$ *and returns* $((\mathsf{set}, \mathrm{H}), \mathscr{R})$.

$\mathsf{Pro}(\mathsf{sk}, \mathsf{pk})$**:** *It runs* $(r, \mathrm{R}) \leftarrow \mathsf{Commit}(\mathsf{set})$ *and* $\alpha = \mathsf{Response}(\mathsf{sk}, r, \mathrm{R}, \mathrm{H}(\mathrm{R}||\mathsf{pk}))$. *It returns* $\pi = (\mathrm{R}, \alpha)$.

$\mathsf{Ver}(\mathsf{pk}, \pi)$**:** *It parses* $\pi = (\mathrm{R}, \alpha)$*, runs* $b \leftarrow \mathsf{Check}(\mathsf{pk}, \mathrm{R}, \mathrm{H}(\mathrm{R}||\mathsf{pk}), \alpha)$ *and returns* b

Theorem 8 *Let* P *be an interactive proof system that is complete, sound, valid, honest-verifier zero-knowledge and that is a sigma protocol, then the Fiat-Shamir transformation of* P *is a non-interactive proof system that is complete, sound, valid and zero-knowledge.*

The proof of this theorem is given in [FS87].

Remark 5 *The Fiat-Shamir transformation achieves adaptive validity (see Remark 4). There exists a weaker version of this transformation where the prover hashs only the commitment without the public statment, which is non-adaptive valid but not adaptive valid [BPW12]. In the following, we show the difference between the two versions of the Fiat-Shamir transformation on the Schnorr protocol in term of security. We show how to build an adaptive adversary* $\mathscr{A}$ *that forges a valid non-interactive proof of a statment* $\mathsf{pk} = g^{\mathsf{sk}}$ *without knowing* sk *in the weak version of the transformation. The algorithm* $\mathscr{A}$ *chooses a scalar* a *and computes* $\mathsf{pk}' = \mathsf{pk} \cdot g^a$*, then it receives a proof for this statment* $\pi = (\mathrm{R}, z)$ *, where* $z = r + (\mathsf{sk} + a) \cdot \mathrm{H}(\mathrm{R})$. *It sets* $z' = z - a \cdot \mathrm{H}(\mathrm{R})$ *and returns the proof* $\pi' = (\mathrm{R}, z')$. *Since* $z' = z - a \cdot \mathrm{H}(\mathrm{R}) = r + (\mathsf{sk} + a) \cdot \mathrm{H}(\mathrm{R}) - a \cdot \mathrm{H}(\mathrm{R}) = r + \mathsf{sk} \cdot \mathrm{H}(\mathrm{R})$*, then* π' *is a valid proof for the statment* pk. *On the other hand, this attack no longer holds when the statment* pk' *is hashed together with* R.

2.6.6 The Cramer-Damgård-Schoenmakers Transformation

The following definition requires some notions of *secret sharing* [Sha79]. A (t, n)-threshold secret sharing scheme allows to share a secret into n shares such that at least t shares are required to recover the secret. Note that less than t shares do not leak any information about the secret. A n-secret sharing scheme is a (t, n)-threshold secret sharing scheme where $t = n$, *i.e.*, all shares are required to recover the secret. Let p be a prime number, we show how to share a secret $s \in \mathbb{Z}_p^*$ into n shares $(s_i)_{1 \leq i \leq n} \in (\mathbb{Z}_p^*)^n$. The first $n-1$ shares are chosen at random: $\forall\ i \in [\![1, n-1]\!], s_i \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. The last share is computed as follows:

$$s_n = \left(\prod_{i=1}^{n-1} \frac{1}{s_i} \right) \cdot s$$

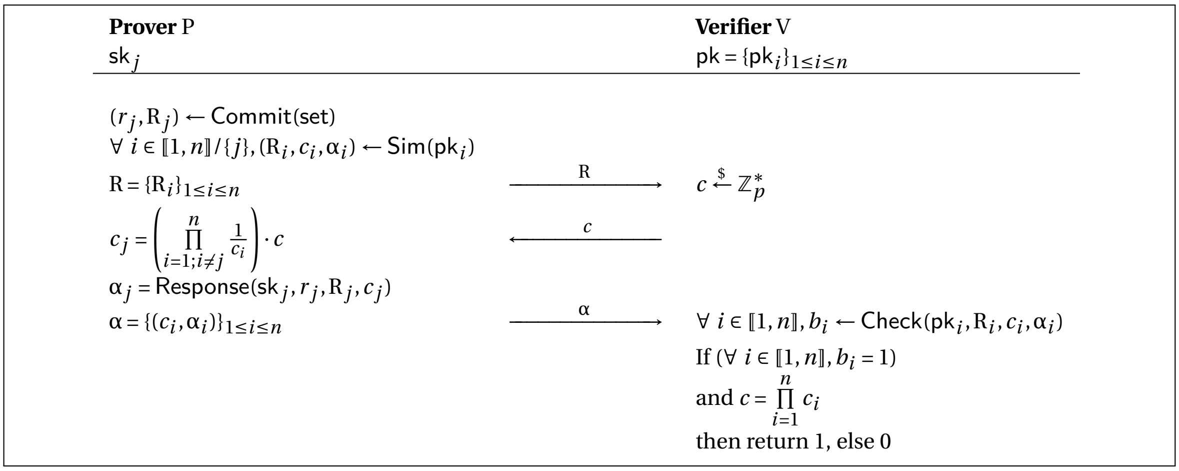

Prover P
sk_j
Verifier V
$\mathsf{pk} = \{\mathsf{pk}_i\}_{1\leq i\leq n}$
$(r_j, \mathrm{R}_j) \leftarrow \mathsf{Commit}(\mathsf{set})$
$\forall\ i \in [\![1,n]\!]/\{j\}, (\mathrm{R}_i, c_i, \alpha_i) \leftarrow \mathsf{Sim}(\mathsf{pk}_i)$
$\mathrm{R} = \{\mathrm{R}_i\}_{1\leq i\leq n}$
$\xrightarrow{\mathrm{R}}$
$c \xleftarrow{\$} \mathbb{Z}_p^*$
$c_j = \left(\prod_{i=1;i\neq j}^{n} \frac{1}{c_i}\right)\cdot c$
$\xleftarrow{c}$
$\alpha_j = \mathsf{Response}(\mathsf{sk}_j, r_j, \mathrm{R}_j, c_j)$
$\alpha = \{(c_i, \alpha_i)\}_{1\leq i\leq n}$
$\xrightarrow{\alpha}$
$\forall\ i \in [\![1,n]\!], b_i \leftarrow \mathsf{Check}(\mathsf{pk}_i, \mathrm{R}_i, c_i, \alpha_i)$
If $(\forall\ i \in [\![1,n]\!], b_i = 1)$
and $c = \prod_{i=1}^{n} c_i$
then return 1, else 0

Figure 2.3: Protocol Proof of the Cramer-Damgård-Schoenmakers transformation.

Since the shares are chosen at random, then less than n shares do not leaks any information about s. On the other hand, a user that knows the n shares retrieves the secret by computing:

$$s = \prod_{i=1}^{n} s_i$$

The *Cramer-Damgård-Schoenmakers transformation* is a way to change a proof of knowledge of the secret solution of some problem instance into a proof of knowledge of v secret solutions out of n intances (for two given integers v and n). This transformation requires the hypothesis that the proof system is a zero-knowledge sigma protocol. The original Transformation requires a $(n+1-v, n)$ threshold secret sharing scheme, however, we present a simplified version of this transformation where $v = 1$, then it only requires a n-secret sharing. Moreover, all the proofs of knowledge we use throughout this thesis are sigma protocols where the set of challenges is the set $\mathbb{Z}_p^*$ for some prime number p. The following definition recall the Cramer-Damgård-Schoenmakers transformation in the specific case where the set of challenges is $\mathbb{Z}_p^*$ and the secret sharing scheme used is the one given previously.

Definition 32 (Cramer-Damgård-Schoenmakers Transformation [CDS94]**)** *Let* $\mathrm{P} = (\mathsf{Set}, \mathsf{Proof})$ *be a sigma protocol defined as in Def. 30 where the set of challenge* $\mathscr{C}$ *is equals to* $\mathbb{Z}_p^*$ *for some prime number* p*, and* $n \in \mathbb{N}$ *be an integer. The* n-Cramer-Damgård-Schoenmakers Transformation *(*n*-CDST) of* P *is the interactive proof system* $\mathrm{P}_n = (\mathsf{Set}_n, \mathsf{Proof}_n)$ *where:*

$\mathsf{Set}_n(k)$**:** *It runs* $(\mathsf{set}, \mathscr{R}) \leftarrow \mathsf{Set}(k)$*. Since* P *is zero-knowledge, there exists a polynomial time simulator* Sim*. Let* $\mathscr{S}$ *be the following set:*

$$\mathscr{S} = \{s | \mathsf{Pr}[(\mathrm{R}, c, \alpha) \leftarrow \mathsf{Sim}(s); b \leftarrow \mathsf{Check}(s, \mathrm{R}, c, \alpha) : b = 1] = 1\}$$

This algorithm returns $(\mathsf{set}, \mathscr{R}_n)$ *where the following equivalence holds for any* $\mathsf{pk} = \{\mathsf{pk}_i\}_{1\leq i\leq n}$*:*

$$(\mathsf{pk}, \mathsf{sk}_j) \in \mathscr{R}_n \Leftrightarrow \left(\mathsf{pk} \in \mathscr{S}^n\right) \wedge \left(\exists\ j \in [\![1,n]\!], ((\mathsf{pk}_j, \mathsf{sk}_j) \in \mathscr{R})\right)$$

$\mathsf{Proof}_n\langle \mathrm{P}(\mathsf{sk}_j); \mathrm{V}(\mathsf{pk})\rangle$**:** *It is the protocol in Fig. 2.3, where* Sim *is the simulator of* P.

Theorem 9 *Let* P *be an interactive proof system that is complete, sound, valid, zero-knowledge (resp. honest-verifier zero-knowledge) and that is a sigma protocol, then the* n*-Cramer-Damgård-Schoenmakers Transformation of* P *is an interactive proof system that is complete, sound, valid, zero-knowledge (resp. honest-verifier zero-knowledge) and that is a sigma protocol.*

This theorem is proven in [CDS94].

2.7 Digital Signature

Digital signatures are public-key cryptographic schemes of message authentication. The secret key allows a user to sign messages, and the public key allows anybody to verify the validity of these signatures.

2.7.1 Formal Definition

Definition 33 (Digital signature [DH76]) *A* Digital signature *(DS) is a tuple of polynomial time algorithms* $(\mathsf{Set}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *defined as follows:*

$\mathsf{Set}(k)$**:** *It returns a setup* set *and a set of messages* $\mathcal{M}$.
$\mathsf{Gen}(\mathsf{set})$**:** *It returns a verification/signing key pair* $(\mathsf{pk}, \mathsf{sk})$.
$\mathsf{DSig}_{\mathsf{sk}}(m)$**:** *It returns a signature* σ.
$\mathsf{Ver}_{\mathsf{pk}}(m, \sigma)$**:** *It returns a bit* b.

Moreover a DS is said to be correct *when the following equation holds for any* $k \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{c}(\mathsf{set},\mathcal{M}) \leftarrow \mathsf{Set}(k); (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{set}); m \overset{\$}{\leftarrow} \mathcal{M} \\ \sigma \leftarrow \mathsf{DSig}_{\mathsf{sk}}(m); b \leftarrow \mathsf{Ver}_{\mathsf{pk}}(m,\sigma)\end{array} : b = 1\right] = 1$$

Finally, a DS is said to be probabilistic *(resp.* deteministic*) when the algorithm* Sig *is probabilistic (resp. deterministic).*

2.7.2 Security Against Chosen Message Attacks

The unforgeability against chosen-message attack is the basic security requirement of digital signatures. Consider an adversary that knows a public key pk, and that has access to an oracle that signs chosen messages using the corresponding secret key sk, the digital signature scheme is unforgeable against chosen-message attack when no adversary is able to forge a valid signature of a chosen message according to the public key pk.

Definition 34 (Existential Unforgeability against Chosen-Message Attack [GMR88]) *Let* $\mathsf{S} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *be a DS,* $\mathcal{A}$ *be an algorithm and* $k \in \mathbb{N}$ *be a security parameter. Let the following oracle be:*

$\mathsf{DSig}_{\mathsf{sk}}(\cdot)$**:** *On input* m*, it runs* $s \leftarrow \mathsf{DSig}_{\mathsf{sk}}(m)$ *and returns* s*.*

We define the EUF-CMA experiment for the adversary $\mathcal{A}$ *against* S *as follows, where* σ_i *is the* i^{th} *signature outputted by the oracle* $\mathsf{DSig}_{\mathsf{sk}}(\cdot)$ *and* q_{Sig} *is the number of queries asked by* $\mathcal{A}$ *to this oracle:*

```
$\mathsf{Exp}^{\mathbf{EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}(k)$:
$(\mathsf{set},\mathcal{M}) \leftarrow \mathsf{Set}(k)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{set})$
$(m,\sigma) \leftarrow \mathcal{A}^{\mathsf{DSig}_{\mathsf{sk}}(\cdot)}(\mathsf{set},\mathsf{pk})$
$b \leftarrow \mathsf{Ver}_{\mathsf{pk}}(m,\sigma)$
if $(\forall\ i \in [\![1, q_{\mathsf{Sig}}]\!], \sigma \neq \sigma_i)$, then return $b$
else return $0$
```

We define the EUF-CMA advantage of $\mathcal{A}$ *against* S *as follows:*

$$\mathsf{Adv}^{\mathit{EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathit{EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}(k)\right]$$

We define the EUF-CMA advantage against S *as follows:*

$$\mathsf{Adv}^{\mathit{EUF\text{-}CMA}}_{\mathsf{S}}(k) = \max_{\mathcal{A} \in \mathrm{POLY}(k)} \left\{\mathsf{Adv}^{\mathit{EUF\text{-}CMA}}_{\mathsf{S},\mathcal{A}}(k)\right\}$$

S *is said to be* unforgeable against chosen-message attack*, or EUF-CMA secure, when the advantage* $\mathsf{Adv}^{\mathit{EUF\text{-}CMA}}_{\mathsf{S}}(k)$ *is negligible.*

2.7.3 Schnorr Signature

The Schnorr signature is a well known digital signature scheme that is EUF-CMA secure. As in the ElGamal encryption, the secret key is the discrete logarithm of the public key. The signature algorithm is build as the proof algorithm of the Fiat-Shamir transformation of the Schnorr interactive proof system, except that the challenge is the hash of the message in addition to the commitment.

Definition 35 (Schnorr Signature [Sch90]) *The (probabilistic) Schnorr signature scheme* $\mathsf{Schnorr} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *is a DS defined as follows:*

$\mathsf{Set}(k)$**:** *It generates a prime order group setup* $(\mathbb{G}, p, g)$ *and a hash function* $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p^*$, *sets* $\mathcal{M} = \{0,1\}^*$ *and* $\mathsf{set} = (\mathbb{G}, p, g, \mathrm{H})$, *and returns* $(\mathsf{set}, \mathcal{M})$.

$\mathsf{Gen}(\mathsf{set})$**:** *It picks* $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$, *sets* $\mathsf{pk} = g^{\mathsf{sk}}$ *and returns* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{DSig}_{\mathsf{sk}}(m)$**:** *It picks* $r \xleftarrow{\$} \mathbb{Z}_p^*$ *and returns* $\sigma = (g^r, r + \mathsf{sk} \cdot \mathrm{H}(g^r || m))$.

$\mathsf{Ver}_{\mathsf{pk}}(m, s)$**:** *It parses* $\sigma = (\mathrm{R}, z)$ *and returns* 1 *if and only if* $g^z = \mathrm{R} \cdot \mathsf{pk}^{\mathrm{H}(\mathrm{R}||m)}$.

The deterministic Schnorr signature scheme $\mathsf{Schnorr} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{DSig}, \mathsf{Ver})$ *is a deterministic DS where* $\mathsf{Set}, \mathsf{Gen},$ *and* Ver *are defined as in the probabilistic scheme and where* DSig *is defined as follows:*

$\mathsf{DSig}_{\mathsf{sk}}(m)$**:** *It computes* $r = \mathrm{H}(\mathsf{sk}||m)$ *and Returns* $s = (g^r, r + \mathsf{sk} \cdot \mathrm{H}(g^r||m))$.

Theorem 10 $\mathsf{Schnorr}$ *is EUF-CMA secure under the DL assumption in the random oracle model.*

The proof of this theorem is given in [PS96].

Chapter 3

Delegation of Authentication Using A Proxy

Contents

Zero-Knowledge Proxies Re-Proof of Knowledge were introduced by Blaze *et al.* in 1998 together with two other well known primitives of proxy re-cryptography, namely *proxy re-encryption* (PRE) and *proxy re-signature* (PRS). A proxy re-proof of knowledge allows a proxy to transform a proof of knowledge of Alice's secret into proof of knowledge of Bob's secret using a *re-proof* key. PRE and PRS have been largely studied in the last decade, but surprisingly, no result about proxies re-proof of knowledge has been published since the pioneer paper of Blaze *et al.*. We first show the insecurity of this scheme: just by observing the communications, Alice can deduce Bob's secret key. Then we give (i) definitions of the different families of proxy re-proof of knowledge (bidirectional/unidirectional and interactive/non-interactive) (ii) a formal security model for these primitives and (iii) a concrete construction for each family. Moreover, we show that proxy re-proof of knowledge has some applications in the delegation of authentication. The proxy can be used to efficiently and securely manage the access policy to several external services that require a public key authentication.

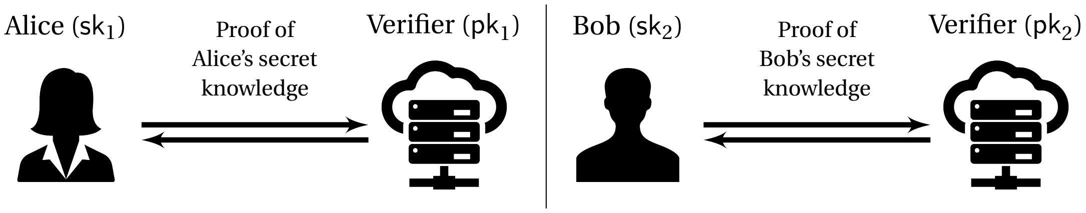

Alice (sk_1)
Proof of Alice's secret knowledge
Verifier (pk_1)
Bob (sk_2)
Proof of Bob's secret knowledge
Verifier (pk_2)

Figure 3.1: Authentication using proofs of knowledge.

3.1 Introduction

In this section, we introduce the *proxy re-cryptography*, and particularly *Proxies Re-Proof of Knowledge* (PRP). We give a state of the art of this research area and the applications of PRP schemes.

3.1.1 Proxy Re-Cryptography

Proxy Re-Encryption (PRE), Proxy Re-Signature (PRS) and proxy re-Proof of knowledge (PRP), also called proxy re-identification, are three useful cryptographic primitives that using a proxy allow *a delegator* (Bob) to delegate his decryption, his signature or his proof rights to *a delegate* (Alice). In PRE, the proxy transforms (*i.e.,* it *re-encrypts*) a message encrypted for Bob into another one encrypted for Alice using a *re-encryption* key. In PRS, the proxy transforms (*i.e.,* it *re-signs*) a message signed by Alice into a message signed with Bob's secret key using a *re-signature* key. In PRP, the proxy transforms (*i.e.,* it *re-proves*) a proof of knowledge of the secret key of Alice into another one of the secret key of Bob using a *re-proof* key. These three primitives have been introduced in 1998 by Blaze, Bleumer and Strauss in [BBS98]. For all of them, a naïve solution is to give Bob's secret key to the proxy. For example, in the case of the proxy re-encryption, the proxy decrypts Bob's ciphertext using Bob's secret key, then it re-encrypts the plaintext using Alice's public key. Note that the proxy must be trusted since it can decrypt any Bob's message. In the case of the re-proof, Alice runs the proof protocol with the proxy using her secret key, the proxy verifies Alice's proof, then the proxy runs the proof algorithm with the verifier using Bob's secret key. These naïve schemes require a fully trusted proxy, however, the aim of proxy re-cryptography is to consider a *semi-trusted* proxy that does not collude with Alice or Bob but cannot act alone as Bob.

A few years later, in 2003, Ivan and Dodis revisit the proxy re-cryptography by introducing the notion of bidirectional and unidirectional proxies, in [ID03]. All schemes given by Blaze *et al.* are bidirectional in the sense that knowing the re-key that delegates Bob's capabilities to Alice, the proxy can easily compute a re-key to delegate Alice's capabilities to Bob. However, this property induces a serious security weakness, because the delegator Bob is able to recover the secret key of Alice by colluding with the proxy. Using bilinear pairings, Ivan *et al.* propose unidirectional proxy schemes in the sense that the delegator Bob is able to compute the re-key alone, *i.e.,* using only his secret key and the public key of Alice. Then the re-key leaks no information about the secret key of Alice and cannot be used to delegate Alice's power to Bob. In addition, Ivan *et al.* give a formal treatment for bidirectional and unidirectional proxy re-encryptions and proxy re-signatures. However, they do not study the case of unidirectionality in proxy re-proof.

Unidirectional PRE and PRS caused a renewed interest in the field of proxy cryptography. For a decade, this topic has became very attractive and has led to numerous publications [BBS98, ID03, AFGH05, LV08b, GA07, LHC10, SFZ$^+$10, LV08a, AH05, CP08]. However, surprisingly, PRP has not received the same attention from the community. Actually, the scheme given in [BBS98] is the only one in the literature and there does not exist any formal model for this primitive. Moreover, this scheme has the following security weakness: if Alice and the verifier collude then they can recover the secret key of Bob.

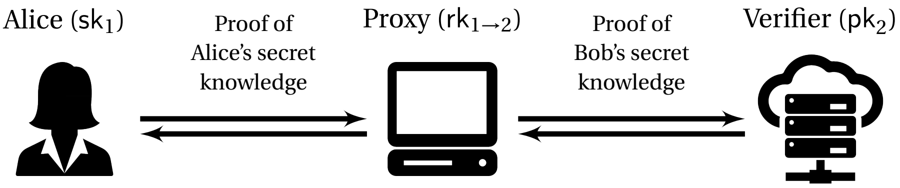

Alice (sk_1)
Proof of Alice's secret knowledge
Proxy ($\mathsf{rk}_{1\to 2}$)
Proof of Bob's secret knowledge
Verifier (pk_2)

Figure 3.2: Delegation of authentication using a proxy.

3.1.2 Functionalities

The goal of proxy re-proof is to solve the following problem, where Alice and Bob know respectively the secret/public keys $(\mathsf{pk}_1,\mathsf{sk}_1) \in \mathscr{R}_1$ and $(\mathsf{pk}_2,\mathsf{sk}_2) \in \mathscr{R}_2$ such that both $\mathscr{R}_1$ and $\mathscr{R}_2$ are binary relations. Using a proof of knowledge, Alice (resp. Bob) can prove the knowledge of his (resp. her) secret key sk_1 (resp. sk_2) in a zero-knowledge way to some verifier that knows the corresponding public key pk_1 (rsp. pk_2), as it is shown in the schemes given in Figure 3.1. Bob does not want to reveal his secret to Alice, but he would like to delegate Alice the power to prove the knowledge of the secret sk_2 without his help. However, it is impossible to prove the knowledge of something we do not know. Then Bob uses a server, called *the proxy*, that knows a *re-proof key* $\mathsf{rk}_{1\to 2}$: Alice and the proxy are able to prove together the knowledge of sk_2 in a zero-knowledge way and Alice uses exactly the same protocol as when she proves the knowledge of her secret. To sum up, the proxy has to transform Alice's proof of knowledge of sk_1 into a proof of knowledge of sk_2 for some verifier. The scheme given in Figure 3.2 resumes the different interactions between Alice, the proxy and the verifier. The proxy has no way to learn any information about Alice's and Bob's keys while it does not collude with Alice or Bob. However, as in the other re-cryptographic primitives, the proxy is not fully untrusted in the sense that we assume that the proxy and Alice does not collude, *i.e.*, the proxy does not give the re-proof key to Alice, and Alice does not give her secret sk_1 to the proxy. We say that the proxy is *semi-trusted*. Note that the *semi-trust* limitation is inherent to any primitive of proxy re-cryptography for any scenario where the re-key must be protected, as in proxy re-encryption or in proxy re-signature.

We distinguish two families of PRP, namely unidirectional and bidirectional schemes. On the other hand, proofs of knowledge are split also into two categories: the interactive and the non-interactive proofs. Thus, PRP are classified into the following categories:

Interactive/Non-interactive: a PRP can be interactive or non-interactive depending on the type of the two proofs of knowledge.

Bidirectional/Unidirectional: a PRP is bidirectional when the re-proof key that delegates Bob's rights to Alice can be used by the proxy to delegate Alice's rights to Bob. On the other hand, a PRP is unidirectional when the delegator Bob is able to generate the re-proof key alone using the public key of Alice as input.

Security requirements: We define the zero-knowledge property for our primitive: informally, a proxy re-proof is zero-knowledge when the protocol between the delegate Alice, the proxy and the verifier leaks no information to the different entities. More precisely, a collusion between the proxy and the verifier leaks nothing about the secret of Alice and a collusion between Alice and the verifier leaks nothing about the re-proof key.

3.1.3 Applications

Proxy re-proofs can be used to provide a practical mechanism of authentication delegation. To illustrate this, we consider that Bob works for a company. In his job, he manipulates sensitive data and he must often authenticate using his secret key. The authentication mechanism used by the company is based on zero-knowledge proof. Bob proves that he knows his secret key from a public key to authenticate. Although he is a hard-worker, he decides to take some holidays. He

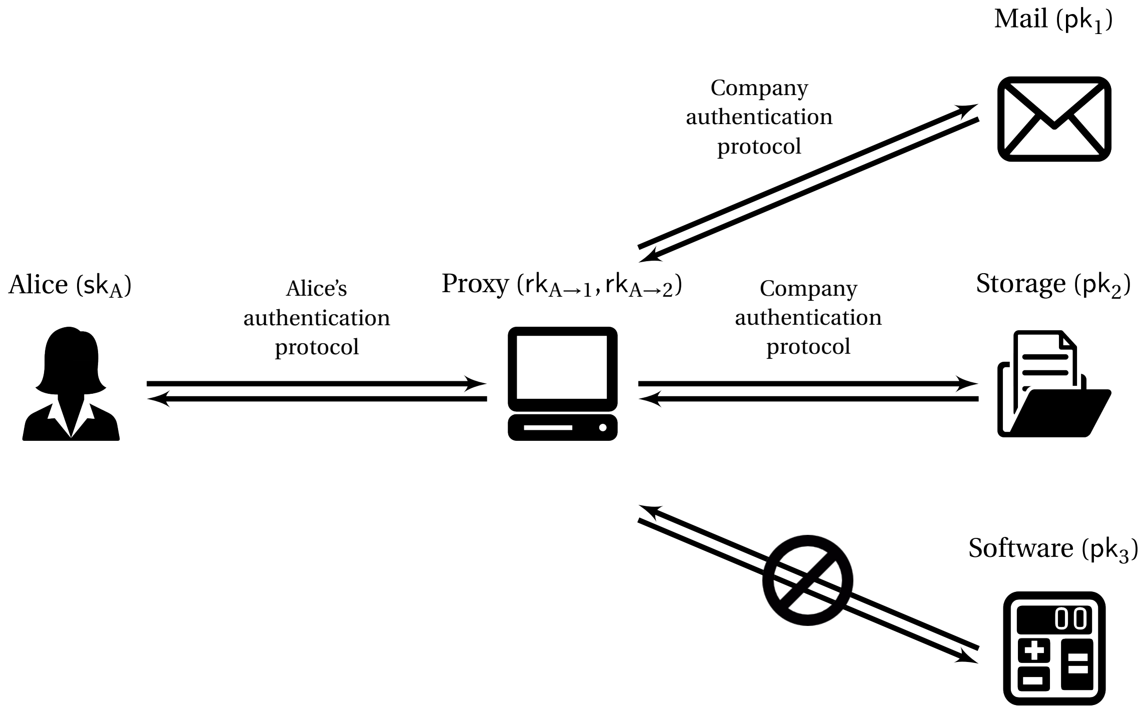

Mail (pk_1)
Company authentication protocol
Alice (sk_A)
Alice's authentication protocol
Proxy ($\mathsf{rk}_{\mathrm{A}\to 1}, \mathsf{rk}_{\mathrm{A}\to 2}$)
Company authentication protocol
Storage (pk_2)
Software (pk_3)
00

Figure 3.3: Access control using a proxy.

has to delegate his access rights to a colleague, called Alice. However, Bob does not want to reveal his secret to Alice, and he wants to be able to cancel the delegation after his holidays. To solve this problem, he uses a proxy re-proof which allows Alice to authenticate as Bob using his secret key during the holidays. Thus, using the same algorithm and the same secret, Alice is able to authenticate under Bob's identity as long as she has access to the proxy.

Another possible application of proxy re-ZKP is access control management. For instance a company has subscribed to several external services like emails, cloud storages or softwares. Each of them requires an authentication based on proofs of knowledge. The company would like to use a proxy to manage which employee has access to which service according to the following constraints:

- Each employee has only one authentication key pair, and runs the same protocol to authenticate to any service.
- The company has only one account to each service, *i.e.,* each service always uses the same verification public key to authenticate any employees.
- The proxy is not able to access to the services alone, *i.e.,* without interacting with some employee.
- An employee cannot access to an unauthorized service, even if he colludes with the proxy.

For example, consider an employee called Alice who has access to the email server and to the cloud storage, but who cannot access to the software. In Figure 3.3 we show how to use a proxy re-proof of knowledge to manage the access right of Alice to the different services. The secret key of Alice is sk_A and the public verification key of the email server (resp. storage, software) is pk_1 (resp. pk_2, pk_3). The proxy has the re-proof keys $\mathsf{rk}_{\mathrm{A}\to 1}$ and $\mathsf{rk}_{\mathrm{A}\to 2}$. To access to the email server, Alice runs her authentication protocol (*i.e.,* she proves the knowledge of sk_A) with the proxy, and the proxy runs the company authentication protocol (*i.e.,* it proves the knowledge of sk_1) with the email server. The proxy *transforms* Alice's authentication protocol into the company authentication protocol for the email server using the re-proof key $\mathsf{rk}_{\mathrm{A}\to 1}$. Since the proxy also knows the key $\mathsf{rk}_{\mathrm{A}\to 2}$, Alice can access to the storage in a similar way. However, Alice cannot access to the software: neither

Alice nor the proxy knows the secret key sk_3, and the proxy does not know the corresponding re-proof key that allows to transform Alice's authentication protocol into the company authentication protocol for the software.

The main avantage of this solution is that the company can manage its access policy alone, without given new authentication keys to the manager of the services, and without changing the secret/public keys of its employees. Each user just needs to know his own authentication material. Moreover, using the logs of the proxy, the company can trace the connections to the different services of its employees. In addition, the two interactive schemes that will be defined in this chapter are based on Schnorr's protocol, which is one of the most used authentication protocols in practice.

Note that we can use proxies re-signature instead of proxies re-proof for these applications: in this case, the authentication protocol consists of signing a random message chosen by the verifier. The main advantage of proxy re-proofs is that these security properties (validity, zero-knowledge) are stronger than the security properties of proxy re-signatures.

3.1.4 Some Proxy Re-Cryptography Schemes

In this section, we present a simple instantiation of each of the main primitives of proxy re-cryptography. First we describe a bidirectional and a unidirectional proxy re-encryption scheme, secondly we describe a bidirectional and a unidirectional proxy re-signature scheme, and finally, we present a bidirectional zero-knowledge proxy re-proof of knowledge scheme which is indeed the only proxy re-proof scheme in the literature. We also show that this scheme has an insufficient security level for practical applications.

Bidirectional proxy re-encryption. The first bidirectional proxy re-encryption scheme was proposed by Blaze *et al.* in [BBS98]. Let Bob's key pair $(\mathsf{pk}_1,\mathsf{sk}_1)$ (resp. Alice's key pair $(\mathsf{pk}_2,\mathsf{sk}_2)$) be such that $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$ (resp. $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$) where g is a generator of a prime order group. Consider the following ElGamal encryption variant: to encrypt a message for Bob (using pk_1), a user picks a random number r and computes $c_1 = (\mathsf{pk}_1^r, g^r\cdot m)$. Bob computes $\frac{(g^r\cdot m)}{(\mathsf{pk}_1^r)^{1/\mathsf{sk}_1}} = m$ to decrypt the message. The re-encryption key is $\mathsf{rk} = \mathsf{sk}_2/\mathsf{sk}_1$.The proxy transforms the encrypted message of Bob c_1 into an encrypted message of Alice c_2 using rk by computing $c_2 = ((\mathsf{pk}_1^r)^{\mathsf{rk}}, g^r\cdot m)$. We observe that:

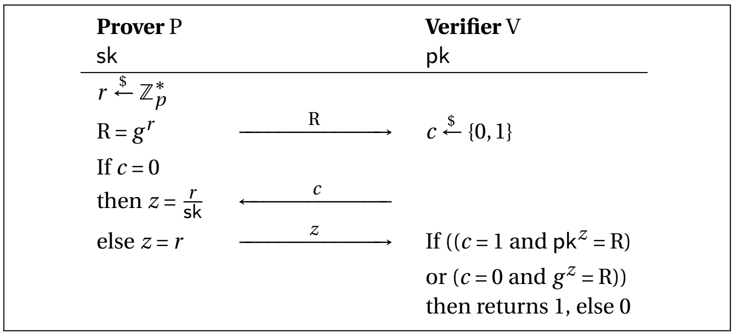
$$(\mathsf{pk}_1^r)^{\mathsf{rk}} = g^{\mathsf{sk}_1\cdot r\cdot\mathsf{rk}} = g^{\mathsf{sk}_1\cdot r\cdot\frac{\mathsf{sk}_2}{\mathsf{sk}_1}} = g^{\mathsf{sk}_2\cdot r} = \mathsf{pk}_2^r$$

Finally, Alice decrypts $c_2 = (\mathsf{pk}_2^r, g^r\cdot m)$ by computing $\frac{(g^r\cdot m)}{(\mathsf{pk}_2^r)^{1/\mathsf{sk}_2}} = m$. Note that the proxy can compute $\mathsf{rk}' = 1/\mathsf{rk}$. We remark that $\mathsf{rk}' = \mathsf{sk}_1/\mathsf{sk}_2$ is the re-proof key that allows to transform Alice's ciphertexts into Bob's ciphertexts, then this scheme is bidirectional.

Unidirectional proxy re-encryption. Using bilinear pairings on the bidirectional scheme of Blaze *et al.*, Ivan *et al.* design the following unidirectional proxy re-encryption scheme [ID03]. Let $e:\mathbb{G}\times\mathbb{G}\to\mathbb{G}_T$ be a bilinear pairing. Users key pairs are defined as in the previous scheme, but the re-encryption key is $\mathsf{rk} = g^{\mathsf{sk}_2/\mathsf{sk}_1}$. Note that Bob is able to build rk alone using Alice's public key $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$ and his secret key sk_1 by computing $\mathsf{pk}_2^{1/\mathsf{sk}_1} = g^{\mathsf{sk}_2/\mathsf{sk}_1}$, then the scheme is unidirectional. To encrypt a message for Bob (using pk_1), a user picks a random number r and computes $c_1 = (\mathsf{pk}_1^r, e(g,g)^r\cdot m)$. Bob computes $\frac{(e(g,g)^r\cdot m)}{(e(\mathsf{pk}_1^r,g))^{1/\mathsf{sk}_1}} = m$ to decrypt the message. We show how the proxy transforms the encrypted message of Bob c_1 into an encrypted message of Alice c_2 using rk: it computes $c_2 = (e(\mathsf{pk}_1^r,\mathsf{rk}), e(g,g)^r\cdot m)$. We observe that:

$$e(\mathsf{pk}_1^r,\mathsf{rk}) = e(g^{\mathsf{sk}_1\cdot r}, g^{\frac{\mathsf{sk}_2}{\mathsf{sk}_1}}) = e(g^{\mathsf{sk}_1\cdot r\cdot\frac{\mathsf{sk}_2}{\mathsf{sk}_1}}, g) = e(g^{\mathsf{sk}_2\cdot r}, g) = e(\mathsf{pk}_2^r, g)$$

Finally, Alice decrypts $c_2 = (e(\mathsf{pk}_2^r,g), e(g,g)^r\cdot m)$ by computing $\frac{(e(g,g)^r\cdot m)}{(e(\mathsf{pk}_1^r,g))^{1/\mathsf{sk}_1}} = m$.



Prover P
sk
Verifier V
pk
$r \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{R} = g^r$
R
$c \xleftarrow{\$} \{0,1\}$
If $c = 0$
then $z = \frac{r}{\mathsf{sk}}$
c
else $z = r$
z
If $((c = 1$ and $\mathsf{pk}^z = \mathrm{R})$
or $(c = 0$ and $g^z = \mathrm{R}))$
then returns 1, else 0

Figure 3.4: Identification protocol of Blaze *et al.* given in [BBS98]

Bidirectional proxy re-signature. A simple bidirectional proxy re-signature scheme was proposed by Ateniese *et al.* in [AH05]. This scheme is based on Boneh–Lynn–Shacham (BLS) signature scheme [BLS04]. Let H be a hash function and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathrm{T}$ be a bilinear pairing. Let Bob's key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$ (resp. Alice's key pair $(\mathsf{pk}_2, \mathsf{sk}_2)$) be such that $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$ (resp. $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$) where g is a generator of $\mathbb{G}$. To sign a message, Bob computes $\sigma_1 = \mathrm{H}(m)^{\mathsf{sk}_1}$. To verify this signature, a user checks that $e(\sigma_1, g) = e(\mathsf{pk}_1, \mathrm{H}(m))$. The re-signature key is $\mathsf{rk} = \mathsf{sk}_2/\mathsf{sk}_1$. The proxy transforms the signature of Bob σ_1 into a signature of Alice σ_2 using rk by computing $\sigma_2 = \sigma_1^{\mathsf{rk}} = \mathrm{H}(m)^{\mathsf{sk}_1 \cdot \frac{\mathsf{sk}_2}{\mathsf{sk}_1}} = \mathrm{H}(m)^{\mathsf{sk}_2}$.

Unidirectional proxy re-signature. In [AH05], Ateniese *et al.* also propose a unidirectional proxy re-signature scheme. Let H be a hash function and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathrm{T}$ be a bilinear pairing. Let Bob's key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$ (resp. Alice's key pair $(\mathsf{pk}_2, \mathsf{sk}_2)$) be such that $\mathsf{pk}_1 = (g^{\mathsf{sk}_1}, h^{1/\mathsf{sk}_1})$ (resp. $\mathsf{pk}_2 = (g^{\mathsf{sk}_2}, h^{1/\mathsf{sk}_2})$) where g and h are two generators of $\mathbb{G}$. To sign a message, Alice picks a random value r, computes $\mathrm{R} = h^r$, $s = \mathsf{sk}_2 \cdot (\mathrm{H}(m||\mathrm{R}) + r)$ and output the signature $\sigma_1 = (\mathrm{R}, s)$. To verify this signature, a user checks that $e(g, h^s) = e(\mathsf{pk}_2, \mathrm{R} \cdot h^{\mathrm{H}(m||\mathrm{R})})$. The re-signature key is $\mathsf{rk} = h^{\mathsf{sk}_1/\mathsf{sk}_2}$. Note that Bob generates the re-signature key by himself by computing $(h^{1/\mathsf{sk}_2})^{\mathsf{sk}_1}$ using Alice's public key $\mathsf{pk}_2 = (g^{\mathsf{sk}_2}, h^{1/\mathsf{sk}_2})$ and his secret key sk_1. The proxy transforms the signature of Alice $\sigma_1 = (\mathrm{R}, s)$ into a signature of Alice σ_2 using rk by computing $\mathrm{S} = \mathsf{rk}^s$ and $\sigma_2 = (\mathrm{R}, \mathrm{S})$. We observe that:

$$\mathrm{S} = \mathsf{rk}^s = h^{\frac{\mathsf{sk}_1}{\mathsf{sk}_2} \cdot \mathsf{sk}_2 \cdot (\mathrm{H}(m||\mathrm{R}) + r)} = h^{\mathsf{sk}_1 \cdot (\mathrm{H}(m||\mathrm{R}) + r)}$$

Finally, a user verifies this signature $\sigma_2 = (\mathrm{R}, \mathrm{S})$ by checking that $e(g, \mathrm{S}) = e(\mathsf{pk}_1, \mathrm{R} \cdot h^{\mathrm{H}(m||\mathrm{R})})$.

Note that this scheme has the following security weakness: knowing the second parts of σ_2 (*i.e.,* $\mathrm{S} = h^{\mathsf{sk}_1 \cdot (\mathrm{H}(m||\mathrm{R}) + r)}$), her secret key sk_2 and the value r she uses to generate the signature σ_1, Alice computes:

$$\mathrm{S}^{\frac{1}{\mathsf{sk}_2 (\mathrm{H}(m||\mathrm{R}) + r}} = h^{\frac{\mathsf{sk}_1}{\mathsf{sk}_2}} = \mathsf{rk}$$

which is the re-signature key that allows to transform any signature of Alice into a signature of Bob. We remark that using this key, Alice is able to sign any message on behalf of Bob. In the same paper, authors show a way to prevent this kind of attack using proofs of knowledge.

Bidirectional Proxy Re-Proof of Knowledge. The following scheme, proposed by Blaze *et al.* in [BBS98], is the only proxy re-proof of knowledge scheme in the literature.

Let Bob's key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$ (resp. Alice's key pair $(\mathsf{pk}_2, \mathsf{sk}_2)$) be such that $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$ (resp. $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$) where g is a generator of $\mathbb{G}$. The re-proof key is $\mathsf{rk} = \mathsf{sk}_1/\mathsf{sk}_2$. Alice and Bob use the proof of knowledge protocol given in Figure 3.4: the prover chooses a random element $r \in \mathbb{Z}_p^*$ and sends $\mathrm{R} = g^r$ to the verifier. The verifier chooses a challenge $c \in \{0,1\}$ and sends it to the prover. If $c = 1$ then the prover sends $z = r/\mathsf{sk}$ to the verifier that checks that $\mathrm{R} = \mathsf{pk}^z$. Else, the prover sends $z = r$ and the verifier checks that $\mathrm{R} = g^z$. The protocol is repeated k times, where k is the security parameter.

The re-proof protocol is given in Figure 3.5: Alice chooses a random element $r \in \mathbb{Z}_p^*$ and sends $\mathrm{R} = g^r$ to the proxy, who forwards it to the verifier. The verifier chooses a challenge $c \in \{0,1\}$ and

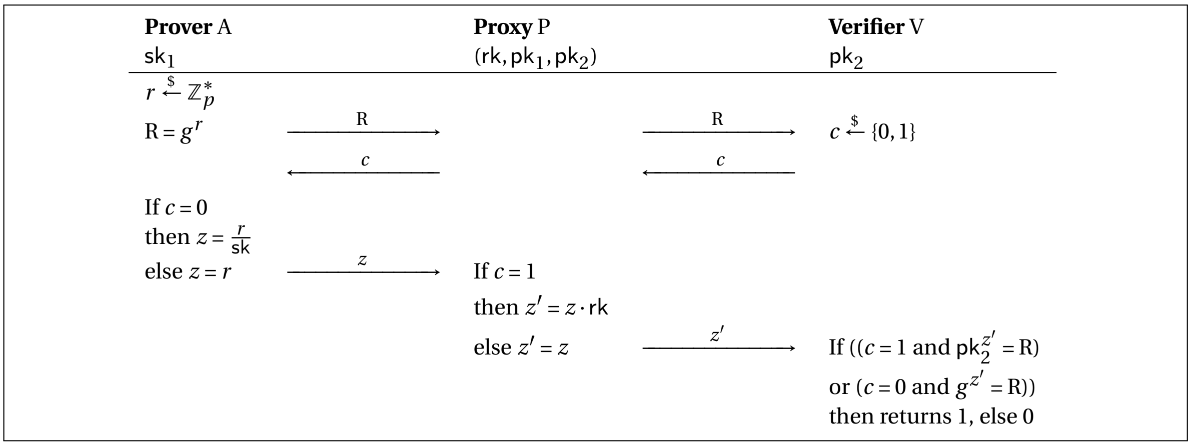

Prover A
sk_1
Proxy P
$(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$
Verifier V
pk_2
$r \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{R} = g^r$
R
R
$c \xleftarrow{\$} \{0,1\}$
c
c
If $c = 0$
then $z = \frac{r}{\mathsf{sk}}$
else $z = r$
z
If $c = 1$
then $z' = z \cdot \mathsf{rk}$
else $z' = z$
z'
If $((c = 1 \text{ and } \mathsf{pk}_2^{z'} = \mathrm{R})$
or $(c = 0 \text{ and } g^{z'} = \mathrm{R}))$
then returns 1, else 0

Figure 3.5: Re-identification protocol of Blaze *et al.* [BBS98].

sends it to the proxy who forwards it to Alice. If $c = 1$ then Alice sends $z = r/\mathsf{sk}_1$ to the proxy, who computes and sends $z' = z \cdot \mathsf{rk} = r/\mathsf{sk}_2$ to the verifier. Else, Alice sends $z = r$ and the proxy sends $z' = z$ to the verifier. If $c = 1$ the verifier checks that $\mathrm{R} = \mathsf{pk}_2^{z'}$. Else, it checks $\mathrm{R} = g^{z'}$. The protocol is repeated k times. We remark that $\mathsf{rk}' = \mathsf{sk}_2/\mathsf{sk}_1$ is the re-proof key that allows to transform Alice's proof into Bob's proof, so this scheme is bidirectional.

Security of this scheme. There does not exist any formal security model for proxy re-proof of knowledge in the literature. Moreover, this scheme has the following security flaws, which makes it unusable in practice:

- if Alice and the verifier collude, then they can recover the secret key of Bob as follows: the verifier sends the challenge $c = 1$ and sends z' to Alice, then she can compute Bob's secret as follows $r/z' = \mathsf{sk}_2$.
- if Alice observes the communication and the verifier sends the challenge $c = 1$, then she can also recover the secret key of Bob in a similar way.

In this chapter, we propose a security model that prevents this kind of attack, and we design the first secure bidirectional proxy re-proof of knowledge.

3.1.5 Related Works

In the late 80's, Okamoto and Ohta introduce the notion of *divertible zero-knowledge interactive proofs* [OO90]. This notion is close to PRP: a zero-knowledge proof between a prover P and a verifier V is *divertible* when a third party W can impersonate V (resp. P) during the protocol such that P (resp. V) cannot distinguish if he interacts with V (resp. P) or W. Then W just *randomizes* but does not transform a proof of a secret to the proof of another one. Thus, W has no *re-key* and cannot be used to delegate the proving capability. PRP can be viewed as an extension of divertible zero-knowledge proofs.

In [CPS14], Canard *et al.* show how the prover can delegate some computations to a proxy in proofs of knowledge of discrete logarithm relations sets. In this work, the proxy performs some computations for the prover without learning any information about his secret, but it knows no additional secret, and it does not transform the proof of the prover's secret knowledge into a proof of another secret knowledge.

We can use a *two-party computation* [KO04] (TPC) to design a protocol that has the same poperties as a proxy re-proof. The delegator shares his secret over the delegate and the proxy. Using TPC, they can compute together the values that allow them to identify as the delegator. However, this generic solution is not efficient, and the delegate does not use the same protocol to prove the knowledge of his secret or to prove the knowledge of the delegator secret.

In distributed and threshold zero-knowledge proofs [KMR12] (TZKP) a secret is shared into n shares, and these shares are distributed to n parties such that a threshold number t of shares allows someone to recover the secret. Then t parties are able to prove the knowledge of the secret together. PRP can be viewed as a particular case of TZKP since the secret of the delegator and the re-proof key allow to compute the delegator's secret key, and the delegate and the proxy interact together to identify as the delegator. However, these schemes do not really *transform* a proof of knowledge into another one for two different public keys. Moreover, unidirectional proxy requires that the re-proof key is computed from the public key of the delegate and the secret of the delegator, and the delegate and the delegator must have a way to prove the knowledge of their secrets alone. The delegate (resp. verifier) uses the same protocol when it interacts directly with the verifier (resp. delegator) and when it interacts with the proxy. Actually, the differences between TZKP and PRP are analogous to the differences between proxy re-encryptions and threshold encryptions, and between proxy re-signatures and threshold signatures.

To the best of our knowledge, there exists neither formal definition nor concrete scheme of unidirectional PRP and non-interactive PRP.

PRP should not be confused with *proxy zero-knowledge proof* defined in [JSMM13]. In this primitive, the proxy helps Alice to perform a proof of knowledge of the secret of the delegator named Bob in order to identify as Bob, but the proxy does not transform the proof of Alice's secret knowledge into a proof of Bob's secret knowledge. Indeed, in PRP, Alice must be able to identify herself to use the proxy, and she uses the same protocol to identify herself (interacting with the verifier) and to identify as Bob (interacting with the proxy). Again, it is the same difference as between proxy encryption and PRE, and between proxy signature and PRS. Moreover, PRP should not be confused with *homomorphic proxy re-authenticators* [DRS17]. This primitive is a kind of proxy re-signature that has the following property: the proxy can evaluate arithmetic functions on the inputs so that the re-signature corresponds to the evaluation of the function. This primitive is not a kind of PRP, and it does not focus on soundness, validity and zero-knowledge properties.

3.1.6 Contributions

In this chapter, we revisit the concept of proxy re-proof of knowledge given in [BBS98]. The first contribution is to give a formal treatment for PRP. We formally define several families for this primitive, namely bidirectional/unidirectional and interactive/non-interactive PRP. We define completness, soundness, validity and zero-knowledge properties for PRP. We give a stronger definition of zero-knowledge than the one given in [BBS98] since we consider collusions between the verifier and the prover. To the best of our knowledge, it is the first time that unidirectionality and non-interactivity are considered for PRP. Our second contribution is to design the four following PRP schemes:

- BIRP, a bidirectional interactive PRP. This scheme does not suffer from the same security weaknesses than the bidirectional interactive PRP proposed in [BBS98].
- UIRP, the first unidirectional interactive PRP.
- BNRP, the first bidirectional non-interactive PRP.
- UNRP, the first unidirectional non-interactive PRP.

3.2 Interactive Proxy Re-Proof

In this section, we formally define interactive proxies re-proof of knowledge and its security properties. We then instantiate this primitive with two schemes: the first one is bidirectional and the second one is unidirectional.

3.2.1 Formal Definition

An interactive proxy re-proof allows a proxy to transform an interactive proof of knowledge of Alice's secret key into another one of Bob's secret key. It contains a setup algorithm Set and three key generation algorithms: Gen_1 (resp. Gen_2) generates Alice's (resp. Bob's) key pair, and RGen generates the proxy re-proof key from the keys of Alice and Bob. It also contains two interactive protocols Proof_1 and Proof_2 that allow Alice and Bob to prove the knowledge of their respective secret keys. Finally, it contains the protocol RProof that allows the proxy, using the re-proof key and interacting with Alice and a verifier, to transform the proof of Alice's secret key knowledge into a proof of Bob's secret key knowledge. More precisely, during the protocol RProof, the proxy runs Proof_1 with Alice as the verifier, and it runs Proof_2 with the verifier as Bob.

We then adapt the properties of interactive proofs of knowledge to proxies re-proof of knowledge:

- The *completness* ensures that if Alice, Bob and the proxy honestly run the protocols, then Alice is able to prove her secret knowledge using the protocol Proof_1, Bob is able to prove his secret knowledge using the protocol Proof_2, and Alice and the proxy are able to *prove Bob's secret knowledge* together using the protocol RProof.

- The *validity* ensures that a user that does not know some secret key is not able to prove the knowledge of this secret key. Note that in the case of the RProof protocol, neither Alice nor the proxy knows Bob's secret key, however, they are able to prove the knowledge of this key because Alice's secret key together with the proxy re-proof key are sufficient to recover Bob's secret key.

- The *zero-knowledge* property ensures that Alice, the proxy and the verifier learns nothing during the protocol RProof. Particulary, Alice leaks nothing about her secret key, and the proxy leaks nothing about the re-proof key. More precisely, the re-proof protocol must be zero-knowledge for:

  - The verifier, even if he has access to the transcript of the full protocol (including the transactions between the prover and the proxy).
  - The proxy colluding with the verifier, particulary, they cannot deduce the secret of the prover.
  - The prover colluding with the proxy, particulary, they cannot deduce the re-proof key of the delegator secret key.

  Note that there exists some primitives in the literature where a proxy *transforms* a proof of knowledge [OO90, KO04], however, they focus on protocols where the proxy does not have any additional secret key. Hence the computations of the proxy could be done by anybody, including the prover and the verifier, which implies that these primitives are by essence collusion resistant. Therefore, we need to define a stronger security model for proxy re-proof of knowledge protocols.

Definition 36 (Proxy re-proof) *A* Proxy Re-Proof *scheme (PRP) is a tuple* $\Pi = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$ *defined as follows:*

$\mathsf{Set}(k)$: *Is an algorithm that returns a setup* set *and two binary relations* $\mathcal{R}_1$ *and* $\mathcal{R}_2$.

$\mathsf{Gen}_1(\mathsf{set}, \mathcal{R}_1)$: *Is an algorithm that returns a public/secret key pair* $(\mathsf{pk}_1, \mathsf{sk}_1) \in \mathcal{R}_1$. *Optionally, it returns an additional public key* pk'_1.

$\mathsf{Gen}_2(\mathsf{set}, \mathcal{R}_2)$: *Is an algorithm that returns a public/secret key pair* $(\mathsf{pk}_2, \mathsf{sk}_2) \in \mathcal{R}_2$. *Optionally, it returns an additional secret key* sk'_2.

$\mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$: *Is an algorithm that returns a re-proof key* rk.

$\mathsf{Proof}_1\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{V}(\mathsf{pk}_1)\rangle$: *Is a two-party protocol between two entities* $\mathsf{P}(\mathsf{sk}_1)$ *(the* prover*) and* $\mathsf{V}(\mathsf{pk}_1)$ *(the* verifier*). At the end of the protocol,* P *returns the bottom symbol* $\perp$ *and* V *returns a bit b.*

$\mathsf{Proof}_2\langle \mathsf{P}(\mathsf{sk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle$**:** *Is a two-party protocol between two entities* $\mathsf{P}(\mathsf{sk}_2)$ *(the* prover*) and* $\mathsf{V}(\mathsf{pk}_2)$ *(the* verifier*). At the end of the protocol,* P *returns the bottom symbol* $\perp$ *and* V *returns a bit* b*.*

$\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle$**:** *Is a three-party protocol between a prover* $\mathsf{P}(\mathsf{sk}_1)$*, a proxy* $\mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$ *and a verifier* $\mathsf{V}(\mathsf{pk}_2)$ *such that* P *runs* Proof_1 *with* Px *as the prover, and* V *runs* Proof_2 *with* Px *as the verifier. Note that* P *and* V *never interact during the protocol.*

We set the two following IP schemes $\mathrm{I}_1 = (\mathsf{Set}_1, \mathsf{Proof}_1)$ *and* $\mathrm{I}_1 = (\mathsf{Set}_2, \mathsf{Proof}_2)$ *where* Set_1 *and* Set_2 *are defined as follows:*

$\mathsf{Set}_1(k)$: *Runs* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$ *and returns* $(\mathsf{set}, \mathscr{R}_1)$.
$\mathsf{Set}_2(k)$: *Runs* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$ *and returns* $(\mathsf{set}, \mathscr{R}_2)$.

I_1 *(resp.* I_2*) is said to be the* delegate proof system *(resp.* delegator proof system*) of* Π*. Moreover, a PRP can have the following properties:*

Completeness: Π *is said to be* complete *when:*

1. I_1 *and* I_2 *are complete.*
2. *For any honest entities* P, Px *and* V *and any* $k \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{c} (\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k); \\ (\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1); \\ (\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2); \\ \mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2); \\ b \leftarrow \mathtt{out}_{\mathsf{V}}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle) \end{array} : b = 1\right] = 1$$

Soundness: Π *is said to be* sound *when* I_1 *and* I_2 *are sound.*

Validity: Π *is said to be* valid*, or is said to be a Proxy Re-Proof of Knowlege (PRPoK), when* I_1 *and* I_2 *are valid.*

Zero-knowledge: Π *is said to be* zero-knowledge *when:*

1. I_1 *and* I_2 *are zero-knowledge.*
2. *For any (possibly dishonest) verifier* V^**, there exists a polynomial time simulator algorithm* Sim_1 *such that for any* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$*, any* $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$*, any* $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$*, any* $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$ *and any* $\alpha \in \{0,1\}^*$*:*

$$\Pr\left[\alpha_* \leftarrow \mathtt{trans}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}^*(\mathsf{pk}_1, \mathsf{pk}_2)\rangle) : \alpha = \alpha_*\right]$$
$$= \Pr\left[\alpha_* \leftarrow \mathsf{Sim}_1(\mathsf{pk}_1, \mathsf{pk}_2) : \alpha = \alpha_*\right]$$

3. *For any (possibly dishonest) proxy* Px^* *and verifier* V^**, there exists a polynomial time algorithm* Sim_2 *such that for any* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$*, any* $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$*, any* $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$*, any* $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$ *and any* $\alpha \in \{0,1\}^*$*:*

$$\Pr\left[\alpha_* \leftarrow \mathtt{view}_{\mathsf{Px}^*, \mathsf{V}^*}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}^*(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}^*(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)\rangle) : \alpha = \alpha_*\right]$$
$$= \Pr\left[\alpha_* \leftarrow \mathsf{Sim}_2(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2) : \alpha = \alpha_*\right]$$

4. *For any (possibly dishonest) prover* P^* *and verifier* V^**, there exists a polynomial time algorithm* Sim_3 *such that for any* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$*, any* $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$*, any* $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$*, any* $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$ *and any* $\alpha \in \{0,1\}^*$*:*

$$\Pr\left[\alpha_* \leftarrow \mathtt{view}_{\mathsf{P}^*, \mathsf{V}^*}(\mathsf{RProof}\langle \mathsf{P}^*(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}^*(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)\rangle) : \alpha = \alpha_*\right]$$
$$= \Pr\left[\alpha_* \leftarrow \mathsf{Sim}_3(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2) : \alpha = \alpha_*\right]$$

Honest-verifier zero-knowledge: *This is a weaker notion of the zero-knowledge property where all entities are honest, i.e., they run correctly the protocols.*

Remark 6 *The validity of the delegate and delegator proof systems is sufficient to achieve the validity of the re-proof system, because the delegator proof system is the same as the re-proof protocol on the verifier point of view. It implies that any entity that is able to prove the delegator secret knowledge knows the delegator secret. In the case of the re-proof protocol, it implies that anybody can efficiently compute the delegator secret* sk_2 *from the secrets known by the prover and the proxy (i.e., the keys* sk_1 *and* rk*).*

We define some additional properties. The first one is the *secret security*. It ensures that no user is able to recover the secret keys of the delegate Alice and the delegator Bob, even if it knows both public keys of Alice and Bob and the corresponding re-proof key. This property implies that the proxy is not able to prove the knowledge of the secret key of Alice or Bob alone.

We remark that zero-knowledge does not imply secret security. Indeed, zero-knowledge ensures that the proxy learns nothing during the protocol, which implies that the protocol transcript is useless to guess the secret keys. However, it implies that the proxy is not able to deduce the secret keys from its re-proof key.

Definition 37 (Secret security) *Let* $\Pi = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$ *be a PRP,* $k \in \mathbb{N}$ *be a security parameter and* $\mathcal{A} \in \textsc{poly}(k)$ *be an algorithm. We define the* secret security advantage *as follows:*

$$\mathsf{Adv}^{\mathsf{SS}}_{\Pi,\mathcal{A}}(k) = \Pr\left[\begin{array}{c} (\mathsf{set},\mathcal{R}_1,\mathcal{R}_2) \leftarrow \mathsf{Set}(k); \\ (\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathcal{R}_1); \\ (\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathcal{R}_2); \\ \mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2); \\ \mathsf{sk}_* \leftarrow \mathcal{A}(\mathsf{pk}_1,\mathsf{pk}'_1,\mathsf{pk}_2,\mathsf{rk}) \end{array} : \exists\, i \in \{1,2\}, \mathsf{sk}_* = \mathsf{sk}_i \right]$$

Π is said to be secret secure *when for any* $\mathcal{A} \in \textsc{poly}(k)$*, there exists a negligible function* ϵ *such that:*

$$\mathsf{Adv}^{\mathsf{SS}}_{\Pi,\mathcal{A}}(k) \leq \epsilon(k)$$

A proxy re-proof is said to be *bidirectional* if a proxy that is able to transform a proof of Alice into a proof of Bob is also able to transform a proof of Bob into a proof of Alice. This property is formally defined as follows.

Definition 38 (Bidirectional PRP) *Let* $\Pi = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$ *be a proxy re-proof.* Π *is said to be* bidirectional *when there exists a polynomial time algorithm* Inv *such that the following equation holds for any* $k \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{c} (\mathsf{set},\mathcal{R}_1,\mathcal{R}_2) \leftarrow \mathsf{Set}(k); \\ (\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathcal{R}_1); \\ (\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathcal{R}_2); \\ \mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2); \\ \mathsf{rk}' \leftarrow \mathsf{RGen}(\mathsf{sk}_2,\mathsf{sk}_1); \\ \mathsf{rk}_* \leftarrow \mathsf{Inv}(\mathsf{rk}) \end{array} : \mathsf{rk}_* = \mathsf{rk}' \right] = 1$$

Finally, a proxy re-proof is said to be *unidirectional* if the delegator Bob is able to compute the re-proof key alone, *i.e.,* using his secret key and the public key of the delegate Alice. We remark that since the secret key of Alice is not used to build the re-proof key known by the proxy, and since her secret key is required to build the key that allows the proxy to delegate the proof capabilities of Alice to Bob, the proxy is not able to transform a proof of Bob into a proof of Alice, then such a scheme is not bidirectional.

Definition 39 (Unidirectional PRP) *A* unidirectional *proxy re-proof scheme is a tuple* $\Pi = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$ *where* Set, Gen_1, Gen_2, Proof_1, Proof_2 *and* RProof *are defined as in Definition 36, and where* RGen *is defined as follows:*

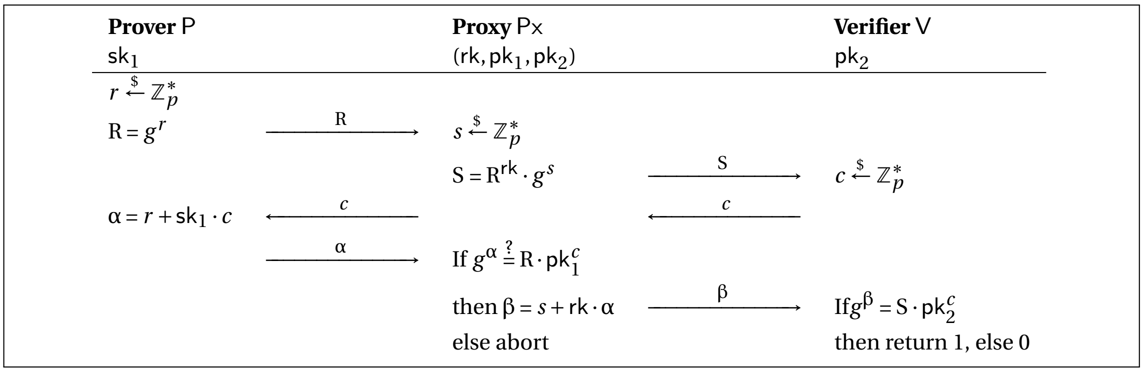

Prover P
sk_1
Proxy Px
$(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$
Verifier V
pk_2
$r \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{R} = g^r$
R
$s \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{S} = \mathrm{R}^{\mathsf{rk}} \cdot g^s$
S
$c \xleftarrow{\$} \mathbb{Z}_p^*$
$\alpha = r + \mathsf{sk}_1 \cdot c$
c
c
α
If $g^\alpha \stackrel{?}{=} \mathrm{R} \cdot \mathsf{pk}_1^c$
then $\beta = s + \mathsf{rk} \cdot \alpha$
β
If$g^\beta = \mathrm{S} \cdot \mathsf{pk}_2^c$
else abort
then return 1, else 0

Figure 3.6: Protocol RProof of IBRP (Definition 40).

$\mathsf{RGen}(\mathsf{pk}_1, \mathsf{pk}_1', \mathsf{sk}_2, \mathsf{sk}_2')$: *Is an algorithm that returns a re-proof key* rk.

Moreover, the following properties are defined similarly to the definition of standard (not unidirectional) PRP: completeness, soundness, validity, zero-knowledge, honest-verifier zero-knowledge, and secret security.

Theorem 11 *Let* $\Pi = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$ *be a proxy re-proof. If* Π *is secret secure, valid and unidirectional, then it is not bidirectional.*

Proof: Let $\Pi = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$ be a proxy re-proof. Assume that Π is valid, unidirectional, and bidirectional. We show how to build a polynomial time algorithm $\mathcal{A}$ that breaks the secret security of Π. $\mathcal{A}$ receives $(\mathsf{pk}_1, \mathsf{pk}_1', \mathsf{pk}_2, \mathsf{rk})$ as input and generates a fresh key set $(\bar{\mathsf{pk}}_2, \bar{\mathsf{sk}}_2, \bar{\mathsf{sk}}_2') \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$. It then generates the re-proof key $\bar{\mathsf{rk}} \leftarrow \mathsf{RGen}(\mathsf{pk}_1, \mathsf{pk}_1', \bar{\mathsf{sk}}_2, \bar{\mathsf{sk}}_2')$ and inverses it using the bilinear property of Π to obtain $\mathsf{rk}_* \leftarrow \mathsf{Inv}(\bar{\mathsf{rk}})$. Using the re-proof key rk_* and the secret key $\bar{\mathsf{sk}}_2$, $\mathcal{A}$ is able to prove the knowledge of the secret corresponding to pk_1 using the protocol $\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_2); \mathsf{Px}(\mathsf{rk}_*, \mathsf{pk}_2, \mathsf{pk}_1); \mathsf{V}(\mathsf{pk}_1)\rangle$ with non-negligible probability. Since the delegate proof system of Π is valid, then there exists a knowledge extractor that returns the secret key sk_1 such that $(\mathsf{pk}_1, \mathsf{sk}_1) \in \mathscr{R}_1$ using the view of the protocol RProof with non-negligible probability, which implies that $\mathcal{A}$ wins its experiment with a non-negligible probability by returning sk_1. □

3.2.2 Bidirectional Interactive Scheme

Our first interactive proxy re-proof scheme is bidirectional. Both delegate and delegator proof protocols are Schnorr's protocol. Thus, this proxy re-proof can easily be implanted in any authentication system already using Schnorr's protocol. We begin this section by giving an informal overview of this scheme.

Let $(\mathbb{G}, p, g)$ be a prime order group. We designate Alice as delegate and Bob as delegator. Alice (resp. Bob) knows sk_1 such that $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$ (resp. sk_2 such that $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$), and the re-key is $\mathsf{rk} = \mathsf{sk}_2/\mathsf{sk}_1$. The scheme is bidirectional since the proxy can compute $\mathsf{rk}^{-1} = \mathsf{sk}_1/\mathsf{sk}_2$. In the re-proof protocol (Figure 3.6), Alice runs a Schnorr protocol as prover with the proxy, and the proxy runs a Schnorr protocol as prover with a verifier. Alice sends her commitment $\mathrm{R} = g^r$, the proxy picks s in $\mathbb{Z}_p^*$ and computes its commitment as follows: $\mathrm{S} = \mathrm{R}^{\mathsf{rk}} \cdot g^s$. Note that since s is randomly chosen, S comes from the uniform distribution on $\mathbb{G}$. The proxy receives the challenge c and forwards it to Alice who responds $\alpha = r + \mathsf{sk}_1 \cdot c$. The proxy computes

$$\beta = s + \mathsf{rk} \cdot \alpha = s + \frac{\mathsf{sk}_2}{\mathsf{sk}_1} \cdot (r + \mathsf{sk}_1 \cdot c) = (r \cdot \frac{\mathsf{sk}_2}{\mathsf{sk}_1} + s) + \mathsf{sk}_2 \cdot c$$

Since $\mathrm{S} = g^{(r \cdot \frac{\mathsf{sk}_2}{\mathsf{sk}_1} + s)}$, then α is the correct response to the challenge c to prove the knowledge of sk_2 using the commitment S.

Definition 40 $\mathsf{IBRP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$ *is a PRP defined as follows:*

$\mathsf{Set}(k)$: *It generates a prime order group setup* $\mathsf{set} = (\mathbb{G}, p, g)$ *and returns* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2)$*, where:*

- $(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}_1 \Leftrightarrow (\mathsf{sk} \in \mathbb{Z}_p^*) \wedge (g^{\mathsf{sk}} = \mathsf{pk})$
- $\mathscr{R}_2 = \mathscr{R}_1$

$\mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$: *It picks* $\mathsf{sk}_1 \xleftarrow{\$} \mathbb{Z}_p^*$*, sets* $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$ *and returns* $(\mathsf{pk}_1, \mathsf{sk}_1)$.

$\mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$: *It picks* $\mathsf{sk}_2 \xleftarrow{\$} \mathbb{Z}_p^*$*, sets* $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$ *and returns* $(\mathsf{pk}_2, \mathsf{sk}_2)$.
$\mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$: *It sets* $\mathsf{rk} = \mathsf{sk}_2/\mathsf{sk}_1$ *and returns it.*
$\mathsf{Proof}_1\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{V}(\mathsf{pk}_1)\rangle$**:** *It is defined as the protocol* Proof *of Schnorr's proof system (Definition 28).*
$\mathsf{Proof}_2\langle \mathsf{P}(\mathsf{sk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle$**:** *It is defined as the protocol* Proof *of Schnorr's proof system (Definition 28).*
$\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle$**:** *It is the protocol given in Figure 3.6.*

In the following, we show that IBRP is bidirectional, complete, sound, valid and honest-verifier zero-knowledge.

Lemma 1 IBRP *is bidirectional.*

Proof: Let the algorithm Inv be such that $\mathsf{Inv}(\mathsf{rk})$ returns $1/\mathsf{rk}$. For any pair of re-proof keys $(\mathsf{rk}, \mathsf{rk}')$ such that $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$ and $\mathsf{rk}' \leftarrow \mathsf{RGen}(\mathsf{sk}_2, \mathsf{sk}_1)$ where RGen is defined as in IBRP, we have $\mathsf{rk} = \mathsf{sk}_2/\mathsf{sk}_1$ and $\mathsf{rk}' = \mathsf{sk}_1/\mathsf{sk}_2$. Thus the algorithm $\mathsf{Inv}(\mathsf{rk})$ returns $1/\mathsf{rk} = 1/(\mathsf{sk}_2/\mathsf{sk}_1) = \mathsf{sk}_1/\mathsf{sk}_2 = \mathsf{rk}'$, wich concludes the proof. □

Lemma 2 IBRP *is complete.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of $\mathsf{IBRP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$. We remark that I_1 (resp. I_2) is defined as $\mathsf{Schnorr}$ (Definition 28), we deduce that I_1 (resp. I_2) is complete (Theorem 6). Let k be an integer and sk_1, pk_1, pk_2 and rk be four keys generated as follows:

- $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$
- $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}_1') \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$
- $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}_2') \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$
- $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$

We deduce that $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$, $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$ and $\mathsf{rk} = \mathsf{sk}_2/\mathsf{sk}_1$. Let a prover P, a proxy Px and a verifier V be three entities that run the protocol $\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle$. We show that V returns 1 with probability 1.

The prover sends $\mathrm{R} = g^r$ to the proxy, the proxy sends $\mathrm{S} = \mathrm{R}^{\mathsf{rk}} \cdot g^s$ to the verifier, and the verifier generates the challenge $c \in \mathbb{Z}_p^*$. Then the prover computes $\alpha = r + \mathsf{sk}_1 \cdot c$ and sends it to the proxy, and the proxy computes $\beta = s + \mathsf{rk} \cdot \alpha$ and sends it to the verifier. We observe that:

$$g^{\beta} = g^{s + \mathsf{rk}\cdot\alpha} = g^s \cdot g^{\mathsf{rk}\cdot r} \cdot g^{\frac{\mathsf{sk}_2}{\mathsf{sk}_1}\cdot \mathsf{sk}_1 \cdot c} = \mathrm{S} \cdot (g^{\mathsf{sk}_2})^c = \mathrm{S} \cdot {\mathsf{pk}_2}^c$$

We deduce that verifier outputs 1, which concludes the proof. □

Lemma 3 IBRP *is sound and valid.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of $\mathsf{IBRP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$. We remark that I_1 (resp. I_2) is defined as $\mathsf{Schnorr}$ (Definition 28), we deduce that I_1 (resp. I_2) is sound and valid (Theorem 6), which concludes the proof. □

Lemma 4 IBRP *is honest-verifier zero-knowledge.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of $\mathsf{IBRP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$. We remark that I_1 (resp. I_2) is defined as Schnorr (Definition 28), we deduce that I_1 (resp. I_2) is honest-verifier zero-knowledge (Theorem 6).

We show how to build a polynomial time algorithm Sim_1 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, and any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, the algorithm $\mathsf{Sim}_1(\mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathtt{trans}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_1, \mathsf{pk}_2)\rangle)$ follow the same probability distribution. We build $\mathsf{Sim}_1(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$ as follows: it picks $(c, \alpha, \beta) \xleftarrow{\$} (\mathbb{Z}_p^*)^3$, and computes $\mathrm{R} = \frac{g^\alpha}{\mathsf{pk}_1^c}$ and $\mathrm{S} = \frac{g^\beta}{\mathsf{pk}_2^c}$. Sim_1 outputs $\tau = ((\mathrm{R}, c, \alpha), (\mathrm{S}, c, \beta))$. The simulator perfectly simulates the protocol and the outputs of Sim_1 follow the same distribution as the real protocol RProof.

We show how to build a polynomial time algorithm Sim_2 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, and any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, the algorithm $\mathsf{Sim}_2(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathtt{view}_{\mathsf{Px},\mathsf{V}}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle)$ follow the same probability distribution. Since I_1 is honest-verifier zero-knowledge, there exists a simulator Sim such that the outputs of $\mathtt{view}_{\mathsf{V}}(\mathsf{Proof}_1\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{V}(\mathsf{pk}_1)\rangle)$ follow the same probability distribution as the outputs of $\mathsf{Sim}(\mathsf{pk}_1)$. We build $\mathsf{Sim}_2(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$ as follows: it runs $(\mathrm{R}, c, \alpha) \leftarrow \mathsf{Sim}(\mathsf{pk}_1)$, picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $\mathrm{S} = \mathrm{R}^{\mathsf{rk}} \cdot g^s$ and $\beta = s + \mathsf{rk} \cdot \alpha$. Sim_2 outputs $\tau = ((\mathrm{R}, \mathrm{S}, c, \alpha, \beta), (\mathrm{S}, c, \beta))$. Knowing rk, the simulator perfectly simulates the proxy behavior and the outputs of Sim_2 follow the same distribution as the real protocol RProof.

We show how to build a polynomial time algorithm Sim_3 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, and any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, the algorithm $\mathsf{Sim}_3(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathtt{view}_{\mathsf{P},\mathsf{V}}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle)$ follow the same probability distribution. We build $\mathsf{Sim}_3(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ as follows: it picks $r \xleftarrow{\$} \mathbb{Z}_p^*$, $c \xleftarrow{\$} \mathbb{Z}_p^*$ and $\beta \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $\mathrm{R} = g^r$, $\alpha = r + \mathsf{sk}_1 \cdot c$ and $\mathrm{S} = \frac{g^\beta}{\mathsf{pk}_2^c}$. It returns the transcript $\tau = ((\mathrm{R}, c, \alpha), (\mathrm{S}, c, \beta))$. It is the transcript of a valid proof since $g^\beta = \mathrm{S} \cdot \mathsf{pk}_2^c$. Moreover β comes from the uniform distribution on $\mathbb{Z}_p^*$, then the outputs of Sim_2 follow the same distribution as the real protocol RProof.

Finally, we have shown that I_1 and I_2 are honest-verifier zero-knowledge and we have shown how to build the algorithms Sim_1, Sim_2 and Sim_3. We deduce that IBRP is honest-verifier zero-knowledge, which concludes the proof. □

Lemma 5 IBRP *is secret secure under the DL assumption.*

Proof: Assume that there exists an adversary $\mathscr{A} \in \mathrm{POLY}(k)$ such that $\epsilon(k) = \mathsf{Adv}^{\mathsf{SS}}_{\mathsf{IBRP},\mathscr{A}}(k)$ is non-negligible. We show how to build $\mathscr{B} \in \mathrm{POLY}(k)$ that breaks DL in $(\mathbb{G}, p, g)$ with the same probability $\epsilon(k)$. $\mathscr{B}$ receives $\mathsf{pk}_0 \in \mathbb{G}$ such that $\mathsf{pk}_0 = g^{\mathsf{sk}_0}$ for unknown $\mathsf{sk}_0 \in \mathbb{Z}_p^*$. It picks $\mathsf{rk}_0 \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $\mathsf{pk}_1 = \mathsf{pk}_0^{\mathsf{rk}_0}$ and $\mathsf{rk}_1 = 1/\mathsf{rk}_0$. It picks $b \xleftarrow{\$} \{0,1\}$ and runs $\mathsf{sk} \leftarrow \mathscr{A}(\mathsf{rk}_b, \mathsf{pk}_b, \mathsf{pk}_{1-b})$. If $g^{\mathsf{sk}} = \mathsf{pk}_0$, then it returns sk, else if $g^{\mathsf{rk}_1 \cdot \mathsf{sk}} = \mathsf{pk}_0$, then it returns $\mathsf{rk}_1 \cdot \mathsf{sk}$. Else it returns $\perp$.

We set $\mathsf{sk}_1 = \mathsf{rk}_0 \cdot \mathsf{sk}_0$, then $\mathsf{rk}_0 = \mathsf{sk}_1/\mathsf{sk}_0$, $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$ and $\mathsf{rk}_1 = \mathsf{sk}_0/\mathsf{sk}_1$. If $\mathscr{A}$ wins his experiment, then it returns sk_0 or sk_1 with non-negligible probability $\epsilon(k)$. If it returns sk_0, then $\mathscr{B}$ wins the experiment since $g^{\mathsf{sk}_0} = \mathsf{pk}_0$. If it returns sk_1, then $\mathscr{B}$ wins the experiment since $g^{\mathsf{rk}_1 \cdot \mathsf{sk}_1} = g^{\mathsf{sk}_0} = \mathsf{pk}_0$. Else $\mathscr{B}$ returns $\perp$ and it does not win the experiment. Finally, we deduce that $\mathscr{B}$ wins its experiment if and only if $\mathscr{A}$ wins its experiment, hence $\Pr[\mathsf{sk}_0 \leftarrow \mathscr{B}(\mathsf{pk}_0)] = \epsilon(k)$, which concludes the proof. □

Theorem 12 IBRP *is bidirectional, complete, sound, valid, honest-verifier zero-knowledge and secret secure under the DL assumption.*

Proof: See Lemma 1, 2, 3, 11 and 5. □

Prover P		**Verifier** V
sk		pk
$s \xleftarrow{\$} \mathbb{Z}_p^*$		
$S = e(g_1, g_2)^s$	$\xrightarrow{S}$	$c \xleftarrow{\$} \mathbb{Z}_p^*$
$\beta = g_2^s \cdot sk^c$	$\xleftarrow{c}$	
	$\xrightarrow{\beta}$	If $e(g_1,\beta) = S \cdot pk^c$
		then returns 1, else return 0

Figure 3.7: Protocol Proof of PKFapi (Definition 41).

Prover P		**Proxy** Px		**Verifier** V
sk_1		(rk, pk_1, pk_2)		pk_2
$r \xleftarrow{\$} \mathbb{Z}_p^*$		$s \xleftarrow{\$} \mathbb{Z}_p^*$		
$R = g_1^r$	$\xrightarrow{R}$	$S = e(R, rk) \cdot e(g_1, g_2)^s$	$\xrightarrow{S}$	$c \xleftarrow{\$} \mathbb{Z}_p^*$
$\alpha = r + sk_1 \cdot c$	$\xleftarrow{c}$		$\xleftarrow{c}$	
	$\xrightarrow{\alpha}$	If $g_1^\alpha \overset{?}{=} R \cdot pk_1^c$		
		then $\beta = g_2^s \cdot rk^\alpha$	$\xrightarrow{\beta}$	If $e(g_1,\beta) = S \cdot pk_2^c$
		else abort		then return 1, else return 0

Figure 3.8: Protocol RProof of IURP (Definition 42).

3.2.3 Unidirectional Interactive Scheme

We show how to construct a unidirectional interactive proxy re-proof scheme. As it is mentioned in the introduction of this chapter, bilinear pairings are the key tool to design unidirectional proxy re-cryptographic schemes. Thus we use bilinear pairings to change our bidirectional scheme into a unidirectional one.

As in the previous scheme, the proof protocol of the delegate is the Schnorr protocol. On the other hand, the delegator proof protocol comes from another proof system that we call PKFapi (for Proof of Knowledge of a FAPI2 instance) and that we define in Definition 41. The detail of this proof protocol is given in Figure 3.7. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three groups of prime order p, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be two generators and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate bilinear pairing. PKFapi allows the delegator to prove the knowledge of the solution sk of a FAPI2 instance $pk = e(g_1, sk)$. It is built using the same methodology as Schnorr's protocol.

Definition 41 PKFapi = (Set, Proof) *is an IP defined as follows:*

Set(k): *Generates the setup* set = $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ *where* $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$ *are three groups of prime order* p*,* $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$*, and* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is a type* 2 *bilinear pairing. This algorithm returns* (set, $\mathcal{R}$)*, where:*

$$(pk, sk) \in \mathcal{R} \Leftrightarrow (sk \in \mathbb{G}_2) \wedge (e(g_1, sk) = pk)$$

Proof⟨P(sk); V(pk)⟩: *Is the protocol given in Figure 3.7.*

We show that PKFapi is complete, sound, valid and honest-verifier zero-knowledge.

Lemma 6 PKFapi *is complete.*

Proof: Let (pk, sk) be such that $(pk, sk) \in \mathcal{R}$, then $(e(g_1, sk) = pk)$. we show that V returns 1 at the end of the protocol Proof⟨P(sk); V(pk)⟩. The prover picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and sends $S = e(g_1, g_2)^s$ to the verifier. The verifier returns a challenge $c \in \mathbb{Z}_p^*$ and the prover returns $\beta = g_2^s \cdot sk^c$. We observe that:

$$e(g_1, \beta) = e(g_1, g_2^s \cdot sk^c) = e(g_1, g_2^s) \cdot e(g_1, sk^c) = S \cdot e(g_1, sk)^c = S \cdot pk^c$$

We deduce that V returns 1, which concludes the proof. □

Lemma 7 PKFapi *is sound and valid.*

Proof: Let k be an integer and pk be such that $\mathsf{pk} \in \mathscr{L}_{\mathscr{R}}$. Let $\mathsf{P}^* \in \textsc{poly}(k)$ be an algorithm and λ be a function such that:

$$\Pr\left[b \leftarrow \mathtt{out}_{\mathsf{V}}\left(\mathsf{Proof}\langle \mathsf{P}^*(\mathsf{in}); \mathsf{V}(\mathsf{pk})\rangle\right) : b = 1\right] = \lambda(k)$$

We build a knowledge extractor $\mathsf{K}(\mathsf{pk})$. This algorithm runs the oracle $\mathsf{P}^*(\mathsf{in})$ and interacts with this oracle as follows:

1. K receives the commitment S.
2. K picks $c_0 \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives β_0.
3. K rewinds the oracle to its state before step 2. It picks $c_1 \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives β_1.

Finally, K returns $\mathsf{sk} = (\beta_1/\beta_0)^{1/(c_1-c_0)}$. If $e(g_1,\beta_0) = \mathsf{S} \cdot \mathsf{pk}^{c_0}$ and $e(g_1,\beta_1) = \mathsf{S} \cdot \mathsf{pk}^{c_1}$ and $c_0 \neq c_1$ then:

$$e(g_1,\mathsf{sk}) = e\left(g_1, \frac{\beta_1}{\beta_0}\right)^{1/(c_1-c_0)} = \frac{e(g_1,\beta_1)}{e(g_1,\beta_0)}^{1/(c_1-c_0)} = \left(\frac{\mathsf{S}\cdot\mathsf{pk}^{c_1}}{\mathsf{S}\cdot\mathsf{pk}^{c_0}}\right)^{1/(c_1-c_0)} = \left(\mathsf{pk}^{(c_1-c_0)}\right)^{1/(c_1-c_0)} = \mathsf{pk}$$

It implies that $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}$.

By hypothesis, we know that $\Pr\left[e(g_1,\beta_0) = \mathsf{S}\cdot\mathsf{pk}^{c_0}\right] \geq \lambda(k)$ and $\Pr\left[e(g_1,\beta_1) = \mathsf{S}\cdot\mathsf{pk}^{c_1}\right] \geq \lambda(k)$. Moreover, since c_0 and c_1 are chosen at random in $\mathbb{Z}_p^*$, we have $\Pr[c_0 = c_1] = 1/(p-1)$, which is negligible. We deduce that:

$$\begin{aligned}\Pr\left[sk \leftarrow \mathsf{K}^{\mathsf{P}^*(\mathsf{in})}(\mathsf{pk}) : (\mathsf{pk},\mathsf{sk}) \in \mathscr{R}\right] &\geq \Pr\left[e(g_1,\beta_0) = \mathsf{S}\cdot\mathsf{pk}^{c_0}\right] \cdot \Pr\left[e(g_1,\beta_1) = \mathsf{S}\cdot\mathsf{pk}^{c_1}\right] - \Pr[c_0 = c_1] \\ &\geq \lambda(k)^2 - \frac{1}{p-1}\end{aligned}$$

which is non-negligible. We deduce that PKFapi is valid. It implies that PKFapi is sound, which concludes the proof. □

Lemma 8 PKFapi *is honest-verifier zero-knowledge.*

Proof: We show how to build a polynomial time algorithm Sim such that for any $k \in \mathbb{N}$, any $(\mathsf{set},\mathscr{R}) \leftarrow \mathsf{Set}(k)$ and any $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}$, the algorithm $\mathsf{Sim}(\mathsf{pk})$ and $\mathtt{view}_{\mathsf{V}}(\mathsf{Proof}\langle \mathsf{P}(\mathsf{sk}); \mathsf{V}(\mathsf{pk})\rangle)$ follow the same probability distribution. Sim picks $\beta \xleftarrow{\$} \mathbb{G}_2$ and $c \xleftarrow{\$} \mathbb{Z}_p^*$, computes $\mathsf{S} = e(g_1,\beta)/\mathsf{pk}^c$ and outputs $\tau = (\mathsf{S}, c, \beta)$. It is the transcript of a valid proof since $e(g_1,\beta) = \mathsf{S}\cdot\mathsf{pk}^c$. Moreover, since β and c comes from the uniform distribution on $\mathbb{Z}_p^*$, then outputs of `Sim` follow the same distribution as the real protocol. □

Theorem 13 PKFapi *is complete, sound, valid and honest-verifier zero-knowledge.*

Proof: See Lemma 6, 7 and 8. □

Our Interactive Unidirectional proxy Re-Proof scheme, called IURP, allows the proxy to transform a proof of a discrete logarithm knowledge into a proof of a pairing inversion knowledge. The re-proof protocol of this scheme is given in Figure 3.8. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three groups of prime order p, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be two generators and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a non-degenerate bilinear pairing. We consider the discrete logarithm problem in the group $\mathbb{G}_1$. The respective secret keys of Alice and Bob are sk_1 and $\mathsf{sk}_2 = g_2^{\mathsf{sk}'_2}$, and $\mathsf{pk}_1 = g_1^{\mathsf{sk}_1}$, $\mathsf{pk}'_1 = g_2^{1/\mathsf{sk}_1}$ and $\mathsf{pk}_2 = e(g_1,\mathsf{sk}_2)$ are the corresponding public keys. The re-key is $\mathsf{rk} = (\mathsf{pk}'_1)^{\mathsf{sk}'_2} = g_2^{(\mathsf{sk}'_2/\mathsf{sk}_1)}$. Note that Bob has to know pk'_1 and sk'_2 to compute the re-key rk. Alice sends her commitment g_1^r, Bob picks s in $\mathbb{Z}_p^*$ and computes his commitment as follows: $\mathsf{S} = e(\mathsf{R},\mathsf{rk})\cdot e(g_1,g_2)^s$. The proxy receives the challenge $c \in \mathbb{Z}_p^*$ and forwards it to Alice who responds $\alpha = r + x \cdot c$. The proxy computes:

$$\beta = g_2^s \cdot \mathsf{rk}^\alpha = g_2^{s+\frac{\mathsf{sk}'_2}{\mathsf{sk}_1}\cdot(r+\mathsf{sk}_1\cdot c)} = g_2^{(r\cdot\frac{\mathsf{sk}'_2}{\mathsf{sk}_1}+s)+\mathsf{sk}'_2\cdot c} = g_2^{(r\cdot\frac{\mathsf{sk}'_2}{\mathsf{sk}_1}+s)}\cdot \mathsf{sk}_2^c$$

Since $\mathsf{S} = e(g_1,g_2)^{(r\cdot\frac{\mathsf{sk}'_2}{\mathsf{sk}_1}+s)}$, it is the correct response to the challenge c to prove the knowledge of sk_2 using the commitment S.

Definition 42 $\mathsf{IURP} = (\mathsf{Set},\mathsf{Gen}_1,\mathsf{Gen}_2,\mathsf{RGen},\ \mathsf{Proof}_1,\mathsf{Proof}_2,\mathsf{RProof})$ *is a unidirectional PRP defined as follows:*

$\mathsf{Set}(k)$: *It generates the setup* $\mathsf{set} = (\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_\mathrm{T},p,g_1,g_2,e)$ *where* $\mathbb{G}_1,\mathbb{G}_2$ *and* $\mathbb{G}_\mathrm{T}$ *are three groups of prime order* p*,* $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$*, and* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$ *is a type* 2 *bilinear pairing. This algorithm returns* $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2)$*, where:*

- $(\mathsf{pk}_1,\mathsf{sk}_1) \in \mathscr{R}_1 \Leftrightarrow (\mathsf{sk}_1 \in \mathbb{Z}_p^*) \wedge (g_1^{\mathsf{sk}_1} = \mathsf{pk}_1)$
- $(\mathsf{pk}_2,\mathsf{sk}_2) \in \mathscr{R}_2 \Leftrightarrow (\mathsf{sk}_2 \in \mathbb{G}_2) \wedge (e(g_1,\mathsf{sk}_2) = \mathsf{pk}_2)$

$\mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$: *It picks* $\mathsf{sk}_1 \xleftarrow{\$} \mathbb{Z}_p^*$*, sets* $\mathsf{pk}_1 = g_1^{\mathsf{sk}_1}$ *and* $\mathsf{pk}'_1 = g_2^{1/\mathsf{sk}_1}$ *and returns* $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}'_1)$.

$\mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$: *It picks* $\mathsf{sk}'_2 \xleftarrow{\$} \mathbb{Z}_p^*$*, sets* $\mathsf{sk}_2 = g_2^{\mathsf{sk}'_2}$ *and* $\mathsf{pk}_2 = e(g_1,\mathsf{sk}_2)$*, and returns* $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}'_2)$.

$\mathsf{RGen}(\mathsf{pk}_1,\mathsf{pk}'_1,\mathsf{sk}_2,\mathsf{sk}'_2)$: *It sets* $\mathsf{rk} = (\mathsf{pk}'_1)^{\mathsf{sk}'_2}$.

$\mathsf{Proof}_1\langle\mathsf{P}(\mathsf{sk}_1);\mathsf{V}(\mathsf{pk}_1)\rangle$: *It is defined as the protocol* Proof *of Schnorr's proof system (Definition 28).*

$\mathsf{Proof}_2\langle\mathsf{P}(\mathsf{sk}_2);\mathsf{V}(\mathsf{pk}_2)\rangle$: *It is defined as the protocol* Proof *of the* PKFapi *proof system (Definition 41).*

$\mathsf{RProof}\langle\mathsf{P}(\mathsf{sk}_1);\mathsf{Px}(\mathsf{rk},\mathsf{pk}_1,\mathsf{pk}_2);\mathsf{V}(\mathsf{pk}_2)\rangle$: *It is the protocol given in Figure 3.8.*

In the following, we prove that IURP is unidirectional, complete, sound, valid, honest-verifier zero-knowledge and secret secure.

Lemma 9 IURP *is complete.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of $\mathsf{IURP} = (\mathsf{Set},\mathsf{Gen}_1,\mathsf{Gen}_2,$ $\mathsf{RGen},\ \mathsf{Proof}_1,\mathsf{Proof}_2,\mathsf{RProof})$. We remark that I_1 is defined as Schnorr (Definition 28) and I_2 is defined as PKFapi, (Definition 41). We deduce that I_1 and I_2 are complete (Theorem 6 and Lemma 6). Let k be an integer and sk_1, pk_1, pk_2 and rk be four keys generated as follows:

- $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k)$
- $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$
- $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$
- $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{pk}_1,\mathsf{pk}'_1,\mathsf{sk}_2,\mathsf{sk}'_2)$.

We deduce that $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$, $\mathsf{pk}_2 = e(g_1,\mathsf{sk}_2)$ and $\mathsf{rk} = (\mathsf{pk}'_1)^{\mathsf{sk}'_2} = g_2^{\mathsf{sk}'_2/\mathsf{sk}_1}$. Let a prover P, a proxy Px and a verifier V be three entities that run the protocol $\mathsf{RProof}\langle\mathsf{P}(\mathsf{sk}_1);\mathsf{Px}(\mathsf{rk},\mathsf{pk}_1,\mathsf{pk}_2);\mathsf{V}(\mathsf{pk}_2)\rangle$. We show that V returns 1 with probability 1.

The prover sends $\mathsf{R} = g_2^r$ to the proxy, the proxy sends $\mathsf{S} = e(\mathsf{R},\mathsf{rk})\cdot e(g_1,g_2)^s$ to the verifier and the verifier generates the challenge c. The prover receives c and computes $\alpha = r + \mathsf{sk}_1 \cdot c$, sends it to the proxy, and the proxy computes $\beta = g_2^s \cdot \mathsf{rk}^\alpha$. Finally, it sends it to the verifier. We observe that:

$$\begin{aligned} e(g_1,\beta) &= e(g_1,g_2^s\cdot\mathsf{rk}^\alpha) = e(g_1,g_2)^s\cdot e(g_1,\mathsf{rk})^\alpha \\ &= e(g_1,g_2)^s\cdot e(g_1,\mathsf{rk})^{r+\mathsf{sk}_1\cdot c} = e(g_1,g_2)^s\cdot e(g_1,\mathsf{rk})^r\cdot e(g_1,\mathsf{rk}^{\mathsf{sk}_1})^c \\ &= e(g_1,g_2)^s\cdot e(g_1^r,\mathsf{rk})\cdot e(g_1,g_2^{\mathsf{sk}'_2})^c = e(g_1,g_2)^s\cdot e(\mathsf{R},\mathsf{rk})\cdot e(g_1,\mathsf{sk}_2)^c \\ &= \mathsf{S}\cdot\mathsf{pk}_2^c \end{aligned}$$

We deduce that the verifier outputs 1, which concludes the proof. □

Lemma 10 IURP *is sound and valid.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of $\mathsf{IURP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$. We remark that I_1 is defined as $\mathsf{Schnorr}$ (Definition 28) and I_2 is defined as PKFapi, (Definition 41). We deduce that I_1 and I_2 are sound and valid (Theorem 6 and Lemma 7), which concludes the proof. □

Lemma 11 IURP *is honest-verifier zero-knowledge.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of $\mathsf{IURP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Proof}_1, \mathsf{Proof}_2, \mathsf{RProof})$. We remark that I_1 is defined as $\mathsf{Schnorr}$ (Definition 28) and I_2 is defined as PKFapi, (Definition 41). We deduce that I_1 and I_2 are honest-verifier zero-knowledge (Theorem 6 and Lemma 8)

We show how to build a polynomial time algorithm Sim_1 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, and any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, the algorithms $\mathsf{Sim}_1(\mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathtt{trans}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_1, \mathsf{pk}_2)\rangle)$ follow the same probability distribution. We build $\mathsf{Sim}_1(\mathsf{pk}_1, \mathsf{pk}_2)$ as follows: it picks $(c, \alpha) \xleftarrow{\$} (\mathbb{Z}_p^*)^2$ and $\beta \xleftarrow{\$} \mathbb{G}_2$, and computes $\mathrm{R} = \frac{g_1^\alpha}{\mathsf{pk}_1^c}$ and $\mathrm{S} = \frac{e(g_1, \beta)}{\mathsf{pk}_2^c}$. Sim_1 outputs $\tau = ((\mathrm{R}, c, \alpha), (\mathrm{S}, c, \beta))$. The simulator perfectly simulates the proxy behavior and the outputs of Sim_1 follow the same distribution as the real protocol RProof.

We show how to build a polynomial time algorithm Sim_2 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, and any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, the algorithms $\mathsf{Sim}_2(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathtt{view}_{\mathsf{Px},\mathsf{V}}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle)$ follow the same probability distribution. Since I_1 is honest-verifier zero-knowledge, there exists a simulator Sim such that the outputs of $\mathtt{view}_{\mathsf{V}}(\mathsf{Proof}_1\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{V}(\mathsf{pk}_1)\rangle)$ follow the same probability distribution as the outputs of $\mathsf{Sim}(\mathsf{pk}_1)$. We build $\mathsf{Sim}_2(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$ as follows: it runs $(\mathrm{R}, \mathrm{S}, c, \alpha, \beta) \leftarrow \mathsf{Sim}(\mathsf{pk}_1)$, picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $\mathrm{S} = e(\mathrm{R}, \mathsf{rk}) \cdot e(g_1, g_2)^s$ and $\beta = g_2^s \cdot \mathsf{rk}^\alpha$. Sim_2 outputs $\tau = ((\mathrm{R}, \mathrm{S}, c, c, \alpha, \beta), (\mathrm{S}, c, \beta))$. Knowing rk, the simulator perfectly simulates the proxy behavior and the outputs of Sim_2 follow the same distribution as the real protocol RProof.

We show how to build a polynomial time algorithm Sim_3 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, and any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, the algorithms $\mathsf{Sim}_3(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathtt{view}_{\mathsf{P},\mathsf{V}}(\mathsf{RProof}\langle \mathsf{P}(\mathsf{sk}_1); \mathsf{Px}(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2); \mathsf{V}(\mathsf{pk}_2)\rangle)$ follow the same probability distribution. We build $\mathsf{Sim}_3(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ as follows: it picks $r \xleftarrow{\$} \mathbb{Z}_p^*$, $c \xleftarrow{\$} \mathbb{Z}_p^*$ and $\beta \xleftarrow{\$} \mathbb{G}_2$, and computes $\mathrm{R} = g_1^r$, $\alpha = r + \mathsf{sk}_1 \cdot c$ and $\mathrm{S} = e(g_1, \beta)/\mathsf{pk}_2^c$. It returns the transcript $\tau = ((\mathrm{R}, c, \alpha), (\mathrm{S}, c, \beta))$. Since $e(g_1, \beta) = \mathrm{S} \cdot \mathsf{pk}_2^c$, it is the transcript of a valid proof. Moreover, since β comes from the uniform distribution on $\mathbb{G}_2$, then the outputs of Sim_2 follow the same distribution as the real protocol RProof.

Finally, we have shown that I_1 and I_2 are honest-verifier zero-knowledge and we have shown how to build the algorithms Sim_1, Sim_2 and Sim_3. We deduce that IURP is honest-verifier zero-knowledge, which concludes the proof. □

Lemma 12 IURP *is secret secure under the DDH assumption.*

Proof: Suppose that there exists an adversary $\mathscr{A}$ that breaks the secret security, *i.e.*, $\mathsf{Adv}^{\mathsf{SS}}_{\mathsf{IBRP},\mathscr{A}}(k)$ is non-negligible. We first show that the probability that $\mathscr{A}$ outputs the delegate's secret key sk_1 is negligible. We then show that the probability that $\mathscr{A}$ outputs the delegator's secret key sk_2 is also negligible, and we conclude that $\mathscr{A}$ has a negligible probability to break the secret security of the scheme, which contradicts our hypothesis. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_\mathrm{T}$ be three groups of prime order p and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$ be a type 2 non-degenerate bilinear pairing:

- We suppose that there exists a polynomial time algorithm $\mathscr{A}(\mathsf{pk}_1, \mathsf{pk}'_1, \mathsf{pk}_2, \mathsf{rk})$ that computes the delegate's secret key sk_1 from his public keys pk_1 and pk'_1, a re-proof key rk and the public key of the corresponding delegator pk_2 with non negligible probability $\epsilon(k)$. Then we show how to construct an algorithm $\mathscr{B}$ that breaks the BDLV assumption in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$, which holds under the DL assumption in $(\mathbb{G}_2, p, g_2)$. $\mathscr{B}$ receives $(\mathsf{pk}_1, \mathsf{pk}'_1)$ as input. This algorithm runs $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$ and $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{pk}_1, \mathsf{pk}'_1, \mathsf{sk}_2, \mathsf{sk}'_2)$. Then $\mathscr{B}$ runs $\mathsf{sk} \leftarrow \mathscr{A}(\mathsf{pk}_1, \mathsf{pk}'_1, \mathsf{pk}_2, \mathsf{rk})$ and outputs sk. Thus if $\mathscr{A}$ wins his experiment then $\mathsf{pk}_1 = g_1^{\mathsf{sk}}$ and $\mathsf{pk}'_1 = g_2^{1/\mathsf{sk}}$, and $\mathscr{B}$ outputs the correct answer. Finally, $\mathscr{B}$ solves its BDLV instance with non-negligible probability $\epsilon(k)$. It implies that $\mathscr{B}$ can be used to break the DL assumption, hence $\mathscr{B}$ can be used to break the DDH assumption.

- We suppose that there exists a polynomial time algorithm $\mathscr{A}(\mathsf{pk}_1, \mathsf{pk}'_1, \mathsf{pk}_2, \mathsf{rk})$ that computes the delegate's secret key sk_2 from his public keys pk_1 and pk'_1, a re-proof key rk and the public key of the corresponding delegator pk_2 with non negligible probability $\epsilon(k)$. Then we show how to construct an algorithm $\mathscr{B}$ that breaks the DCDH assumption in $(\mathbb{G}_2, p, g_2)$, which is equivalent to the DDH assumption. $\mathscr{B}$ receives (X, Y) as input, we set $\mathrm{X} = g_2^x$ and $\mathrm{Y} = g_2^y$. Since e is a type 2 pairing there exists a morphism $\phi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$. Then $\mathscr{B}$ picks $a \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $g_1 = \phi(\mathrm{Y}^a)$ and $\mathsf{pk}_1 = \phi(g_2^a)$. He sets the additional delegate public key $\mathsf{pk}'_1 = \mathrm{Y}$, the re-proof key $\mathsf{rk} = \mathrm{X}$ and the delegator public key $\mathsf{pk}_2 = e(\mathsf{pk}_1, \mathsf{rk})$. Then $\mathscr{B}$ runs $\mathrm{Z} \leftarrow \mathscr{A}(\mathsf{pk}_1, \mathsf{pk}'_1, \mathsf{pk}_2, \mathsf{rk})$ and outputs Z. We set $\mathsf{sk}_1 = 1/y$ and $\mathsf{sk}'_2 = x/y$, we obtain that $\mathrm{Y} = g_2^{1/\mathsf{sk}_1}$ and $\mathrm{X} = g_2^{\mathsf{sk}'_2/\mathsf{sk}_1}$. Moreover, we observe that:

    - $\mathsf{pk}_1 = \phi(g_2^a) = \phi(\mathrm{Y}^{a \cdot \mathsf{sk}_1}) = \phi(\mathrm{Y}^a)^{\mathsf{sk}_1} = g_1^{\mathsf{sk}_1}$
    - $\mathsf{pk}'_1 = \mathrm{Y} = g_2^{1/\mathsf{sk}_1}$
    - $\mathsf{rk} = g_2^{\mathsf{sk}'_2/\mathsf{sk}_1}$
    - $\mathsf{pk}_2 = e(\mathsf{pk}_1, \mathsf{rk}) = e(g_1, g_2)^{x/y} = e(g_1, g_2^{x/y}) = e(g_1, g_2^{\mathsf{sk}'_2})$

  Therefore $\mathscr{B}$ perfectly simulates the SS experiment for $\mathscr{A}$. Moreover, if $\mathscr{A}$ wins the experiment, $\mathrm{Z} = g_2^{\mathsf{sk}'_2} = g_2^{x/y}$ and $\mathscr{B}$ wins its experiment with the same probability as $\mathscr{A}$ breaks the secret security.

We conclude that $\mathscr{A}$ has a non-negligible probability to break the secret security of the scheme under the DDH assumption in $(\mathbb{G}_2, p, g_2)$, which contradicts our hypothesis and concludes the proof.
□

Theorem 14 IURP *is unidirectional, complete, sound, valid, honest-verifier zero-knowledge and secret secure under the DL assumption.*

Proof: IURP is unidirectional by construction. Moreover, it is complete(Lemma 9), valid, sound (Lemma 10), honest-verifier zero-knowledge (Lemma 11) and secret secure (Lemma 12) under the DDH assumption. □

Remark 7 IBRP *and* IURP *are honest-verifier zero-knowledge. As in the Schnorr protocol, it is possible to design two fully zero-knowledge proxy re-proof by forcing the verifier to choose the challenge c in $\{0, 1\}$ instead of $\mathbb{Z}_p^*$. However, these protocols would be less practical than* IBRP *and* IURP *since they would have to be repeated k times (for a chosen security parameter k).*

3.3 Non-Interactive Proxy Re-Proof

In this section, we formally define non-interactive proxies re-proof of knowledge and their security properties. Once again, we instantiate this primitive with two schemes: the first one is bidirectional and the second one is unidirectional.

3.3.1 Formal Definition

Non-interactive proxies re-proof are defined as interactive proxies re-proof, except that the interactive protocols Proof_1 (resp. Proof_2) are replaced by two algorithms Pro_1 and Ver_1 (resp. Pro_2 and Ver_2). The first one allows the prover to generate a non-interactive proof, and the second one allows the verifier to verify this proof. Moreover, the protocol RProof is replaced by an algorithm RPro that transforms a non-interactive proof of the delegate into a non-interactive proof of the delegator.

Definition 43 (Non-interactive proxy re-proof) *A* Non-Interactive proxy Re-Proof *scheme (NIPRP) is a tuple of algorithms* $\Pi = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Pro}_1, \mathsf{Pro}_2, \mathsf{Ver}_1, \mathsf{Ver}_2, \mathsf{RPro})$ *defined as follows:*

$\mathsf{Set}(k)$: *It returns a setup* set *and two binary relations* $\mathscr{R}_1$ *and* $\mathscr{R}_2$*.*
$\mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$: *It returns* $(\mathsf{pk}_1, \mathsf{sk}_1) \in \mathscr{R}_1$*. Optionally, it returns an additional public key* pk'_1*.*
$\mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$: *It returns* $(\mathsf{pk}_2, \mathsf{sk}_2) \in \mathscr{R}_2$*. Optionally, it returns an additional secret key* sk'_2*.*
$\mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$: *It returns a re-proof key* rk*.*
$\mathsf{Pro}_1(\mathsf{sk}_1, \mathsf{pk}_1)$**:** *It returns a proof* π_1*.*
$\mathsf{Pro}_2(\mathsf{sk}_2, \mathsf{pk}_2)$**:** *It returns a proof* π_2*.*
$\mathsf{Ver}_1(\mathsf{pk}_1, \pi_1)$**:** *It returns a bit* b_1*.*
$\mathsf{Ver}_2(\mathsf{pk}_2, \pi_2)$**:** *It returns a bit* b_2*.*
$\mathsf{RPro}(\mathsf{rk}, \pi_1)$**:** *It returns a proof* π_2*.*

We set the two followings NIP schemes $\mathrm{I}_1 = (\mathsf{Set}_1, \mathsf{Pro}_1, \mathsf{Ver}_1)$ *and* $\mathrm{I}_1 = (\mathsf{Set}_2, \mathsf{Pro}_2, \mathsf{Ver}_2)$ *where* Set_1 *and* Set_2 *are defined as follows:*

$\mathsf{Set}_1(k)$: *Runs* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$ *and returns* $(\mathsf{set}, \mathscr{R}_1)$*.*
$\mathsf{Set}_2(k)$: *Runs* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$ *and returns* $(\mathsf{set}, \mathscr{R}_2)$*.*

I_1 *(resp.* I_2*) is said to be the* delegate non-interactive proof system *(resp.* delegator non-interactive proof system*) of* Π*. Moreover, a NIPRP can have the following properties:*

Completeness: Π *is said to be* complete *when:*

1. I_1 *and* I_2 *are complete.*
2. *For any* $k \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{c} (\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k); \\ (\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1); \\ (\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2); \\ \mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2); \pi_1 \leftarrow \mathsf{Pro}_1(\mathsf{sk}_1, \mathsf{pk}_1) \\ \pi_2 \leftarrow \mathsf{RPro}(\mathsf{rk}, \pi_1); b \leftarrow \mathsf{Ver}_2(\mathsf{pk}_2, \pi_2) \end{array} : b = 1\right] = 1$$

Soundness: Π *is said to be* sound *when* I_1 *and* I_2 *are sound.*
(Non-Adaptive) Validity: Π *is said to be (non-adaptively)* valid *when* I_1 *and* I_2 *are (non-adaptively) valid.*
Zero-knowledge: Π *is said to be* zero-knowledge *when:*

1. I_1 *and* I_2 *are zero-knowledge.*
2. *There exists a polynomial time algorithm* Sim_1 *such that for any* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$*, any* $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$*, any* $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$*, any re-key* $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$ *and any* $\alpha \in (\{0,1\}^*)^2$*:*

$$\begin{aligned} &\Pr\left[\alpha_1 \leftarrow \mathsf{Pro}_1(\mathsf{sk}_1, \mathsf{pk}_1); \alpha_2 \leftarrow \mathsf{RPro}(\mathsf{rk}, \alpha_1) : (\alpha_1, \alpha_2) = \alpha_*\right] \\ &\qquad = \Pr\left[\alpha_* \leftarrow \mathsf{Sim}_1(\mathsf{pk}_1, \mathsf{pk}_2) : \alpha = \alpha_*\right] \end{aligned}$$

3. *There exists a polynomial time algorithm* Sim_2 *such that for any* $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, *any* $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$, *any* $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$, *any re-key* $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2)$ *and any* $\alpha \in \{0,1\}^*$*:*

$$\Pr\left[\alpha_* \leftarrow \mathsf{Pro}_1(\mathsf{sk}_1,\mathsf{pk}_1) : \alpha = \alpha_*\right] = \Pr\left[\alpha_* \leftarrow \mathsf{Sim}_2(\mathsf{rk},\mathsf{pk}_1,\mathsf{pk}_2) : \alpha = \alpha_*\right]$$

4. *For any* $k \in \mathbb{N}$ *and any polynomial time algorithm* $\mathscr{A}$*, there exists a polynomial time algorithm* Sim_3 *such that for any* $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, *any* $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$, *any* $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$, *any* $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2)$ *and any* $\alpha \in \{0,1\}^*$*:*

$$\begin{aligned}\Pr&\left[\pi_1 \leftarrow \mathscr{A}(\mathsf{sk}_1,\mathsf{pk}_1,\mathsf{pk}_2); \alpha_* \leftarrow \mathsf{RPro}(\mathsf{rk},\pi_1) : \alpha = \alpha_*\right]\\ &= \Pr\left[\alpha_* \leftarrow \mathsf{Sim}_3(\mathsf{rk},\mathsf{pk}_1,\mathsf{pk}_2) : \alpha = \alpha_*\right]\end{aligned}$$

As in the interactive case, we define three additional properties: secret security, bidirectionality and unidirectionality.

Definition 44 (Secret security) *Let* $\Pi = (\mathsf{Set},\mathsf{Gen}_1,\mathsf{Gen}_2,\mathsf{RGen},\mathsf{Pro}_1,\mathsf{Pro}_2,\mathsf{Ver}_1,\mathsf{Ver}_2,\mathsf{RPro})$ *be a NIPRP,* $k \in \mathbb{N}$ *be a security parameter and* $\mathscr{A} \in \text{POLY}(k)$ *be an algorithm. We define the* secret security advantage *as follows:*

$$\mathsf{Adv}^{\mathsf{SS}}_{\Pi,\mathscr{A}}(k) = \Pr\left[\begin{array}{c}(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k);\\ (\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1);\\ (\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2);\\ \mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2);\\ \mathsf{sk}_* \leftarrow \mathscr{A}(\mathsf{pk}_1,\mathsf{pk}'_1,\mathsf{pk}_2,\mathsf{rk})\end{array} : \exists\, i \in \{1,2\}, \mathsf{sk}_* = \mathsf{sk}_i\right]$$

Π *is said to be* secret secure *when for any* $\mathscr{A} \in \text{POLY}(k)$*, there exists a negligible function* ϵ *such that:*

$$\mathsf{Adv}^{\mathsf{SS}}_{\Pi,\mathscr{A}}(k) \le \epsilon(k)$$

Definition 45 (Bidirectional NIPRP) *Let* $\Pi = (\mathsf{Set},\mathsf{Gen}_1,\mathsf{Gen}_2,\mathsf{RGen},\ \mathsf{Pro}_1,\mathsf{Pro}_2,\mathsf{Ver}_1,\mathsf{Ver}_2,\mathsf{RPro})$ *be a NIPRP.* Π *is said to be* bidirectional *if there exists a polynomial time algorithm* Inv *such that the following equation holds for any and* $k \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{c}(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k);\\ (\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1);\\ (\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2);\\ \mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2);\\ \mathsf{rk}' \leftarrow \mathsf{RGen}(\mathsf{sk}_2,\mathsf{sk}_1);\\ \mathsf{rk}_* \leftarrow \mathsf{Inv}(\mathsf{rk})\end{array} : \mathsf{rk}_* = \mathsf{rk}'\right] = 1$$

Definition 46 (Unidirectional NIPRP) *An* unidirectional *non-interactive proxy re-proof scheme is a tuple* $\Pi = (\mathsf{Set},\mathsf{Gen}_1,\mathsf{Gen}_2,\mathsf{RGen},\ \mathsf{Pro}_1,\mathsf{Pro}_2,\mathsf{Ver}_1,\mathsf{Ver}_2,\mathsf{RPro})$ *where* $\mathsf{Set},\mathsf{Gen}_1,\mathsf{Gen}_2,\mathsf{Pro}_1,\mathsf{Pro}_2,\mathsf{Ver}_1,\mathsf{Ver}_2$ *and* RPro *are defined as in Definition 43, and where* RGen *is defined as follows:*

$\mathsf{RGen}(\mathsf{pk}_1,\mathsf{pk}'_1,\mathsf{sk}_2,\mathsf{sk}'_2)$: *It is an algorithm that returns a re-proof key* rk.

Moreover, the following properties are defined similarly to the definition of standard (i.e., not unidirectional) PRP: completeness, soundness, validity, zero-knowledge, honest-verifier zero-knowledge, and secret security.

3.3.2 Bidirectional Non-interactive Scheme

To transform an interactive proof system into a non-interactive one, the simplest solution is to use, when it is possible, the Fiat-Shamir transformation. The challenges randomly chosen by the verifier are replaced by the hash of the commitments sent by the prover. This solution works in the

Prover P		**Verifier** V
sk		pk
$r \xleftarrow{\$} \mathbb{Z}_p^*; u \xleftarrow{\$} \mathbb{Z}_p^*$		
$\mathrm{R} = g^r; \mathrm{U} = g^u$	$\xrightarrow{(\mathrm{R},\mathrm{U})}$	$c, d \xleftarrow{\$} \mathbb{Z}_p^*$
$z = r + \mathsf{sk} \cdot d$	$\xleftarrow{(c,d)}$	
$\mu = u + r \cdot c$	$\xrightarrow{(z,\mu)}$	If $g^\mu = \mathrm{U} \cdot \mathrm{R}^c$ and $g^z = \mathrm{R} \cdot \mathsf{pk}^d$ returns 1, else return 0

Figure 3.9: Protocol Proof of Schnorr+ (Definition 47).

random oracle model. We then would like to transform our interactive proxies re-proof in a similar way to obtain non-interactive schemes. However, this method cannot be used on our bidirectional interactive proxy re-proof scheme: indeed, the prover and the proxy use the same challenge c during the re-proof protocol, but they use two different commitments R and S, then the proxy cannot use the hash of the delegate's commitment R instead of the hash of its commitment S as the challenge. Therefore we need to design two interactive proof protocols where *the same* commitment is used to compute the challenge. These two non-interactive proof systems, called Schnorr+ and DLright, are built as follows:

Protocol Schnorr+ **(Definition 47):** The protocol of the interactive version of this proof system is given in Figure 3.9. This protocol is the same as Schnorr protocol except that the prover must know the discrete logarithm of his commitment R. Thus, the prover uses *a second* Schnorr protocol to prove it. To do that, the prover sends a second commitment $\mathrm{U} = g^u$ and receives a second challenge d. The prover uses c to prove the knowledge of r using the commitment U by computing $\mu = u + r \cdot c$, then it uses d to prove the knowledge of his secret sk using the commitment R by computing $z = r + \mathsf{sk} \cdot d$, as in the Schnorr protocol. Finally, the verifier uses μ and z to check that $g^\mu = \mathrm{U} \cdot \mathrm{R}^c$ and $g^z = \mathrm{R} \cdot \mathsf{pk}^d$. To transform this protocol into a non-interactive proof system, the prover computes the two challenges c and d by hashing the concatenation of the two commitments R and U.

Protocol DLright **(Definition 48):** The protocol of the interactive version of this proof system is given in Figure 3.10. This protocol is a proof of knowledge of a discrete logarithm in a prime order group $\mathbb{G}$. It is based on Schnorr+, but it requires that the commitment is built in a particular way. More precisely, the prover sends a first *classical* commitment $\mathrm{R} = g^r$ and receives the challenge d. As in Schnorr+, the prover must prove the knowledge of the discrete logarithm of his commitment R. The prover chooses two random values a and b and it commits $\mathrm{A} = \mathrm{R}^a$ and $\mathrm{B} = g^b$. Next the prover chooses two random values t and s in $\mathbb{Z}_p^*$ and it computes $\mathrm{S} = \mathrm{R}^t \cdot g^s$. It sends the commitments A, B and S to the verifier and receives a challenge f. It proves that he knows the two values t and s to the verifier by computing $\alpha = a + t \cdot f$ and $\beta = b + s \cdot f$. The verifier receives α and β and checks that $\mathrm{R}^\alpha \cdot g^\beta = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^f$. Then the prover proves the knowledge of the secret sk as in the Schnorr protocol using the commitment $\mathrm{S} = g^{s+r\cdot t}$ and the challenge d by computing $\theta = (s + r \cdot t) + \mathsf{sk} \cdot d$. The verifier receives θ and checks that $g^\theta = \mathrm{S} \cdot \mathsf{pk}^d$. Note that the prover receives the challenge d before computing the commitment S, which seems contradictory to the methodology of Schnoor-like proof systems. However, the proof remains valid because the prover receives the challende d after he sends the commitment R to the verifier, and the prover must know the value $r \cdot t + s$ (which is the discrete logarithm of S) to compute θ and succeed the proof, which implies that it knows r the discrete logarithm of the commitment R. To transform this protocol into a non-interactive proof system, the prover computes each challenge by hashing the concatenation of all the commitments that he has previously computed.

We formally define the Schnorr+ non-interactive proof system.

Prover P sk		**Verifier** V pk
$a, b, r, s, t, u \xleftarrow{\$} \mathbb{Z}_p^*$		
$\mathrm{R} = g^r ; \mathrm{U} = g^u$	$\xrightarrow{(\mathrm{R},\mathrm{U})}$	
$\mu = u + r \cdot c$	$\xleftarrow{(c,d)}$	$c, d \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{S} = g^{r \cdot t + s}$	$\xrightarrow{\mu}$	
$\mathrm{A} = \mathrm{R}^a ; \mathrm{B} = g^b$	$\xrightarrow{(\mathrm{A},\mathrm{B},\mathrm{S})}$	
$\alpha = a + t \cdot f$	$\xleftarrow{f}$	$f \xleftarrow{\$} \mathbb{Z}_p^*$
$\beta = b + s \cdot f$		
$\theta = s + r \cdot t + \mathsf{sk} \cdot d$	$\xrightarrow{(\alpha,\beta,\theta)}$	If $g^\mu = \mathrm{U} \cdot \mathrm{R}^c$ and $g^\theta = \mathrm{S} \cdot \mathsf{pk}^d$ and $\mathrm{R}^\alpha \cdot g^\beta = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^f$ then return 1, else return 0

Figure 3.10: Protocol Proof of interactive DLright (Definition 48).

Definition 47 $\mathsf{Schnorr+} = (\mathsf{Set}, \mathsf{Pro}, \mathsf{Ver})$ *is a NIP defined as follows:*

$\mathsf{Set}(k)$: *It generates the setup* $\mathsf{set} = (\mathbb{G}, p, g, \mathrm{H})$ *where* $\mathbb{G}$ *is a group of prime order* p*,* $g \in \mathbb{G}$ *and* $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p^*$ *is a hash function. This algorithm returns* $(\mathsf{set}, \mathscr{R})$*, where:*

$$(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R} \Leftrightarrow (\mathsf{sk} \in \mathbb{Z}_p^*) \wedge (g^{\mathsf{sk}} = \mathsf{pk})$$

$\mathsf{Pro}(\mathsf{sk}, \mathsf{pk})$**:** *It picks* $r \xleftarrow{\$} \mathbb{Z}_p^*$ *and* $u \xleftarrow{\$} \mathbb{Z}_p^*$*, computes* $\mathrm{R} = g^r$*,* $\mathrm{U} = g^u$*,* $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$*,* $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$*,* $z = r + \mathsf{sk} \cdot d$ *and* $\mu = u + r \cdot c$*, and outputs* $\pi = (\mathrm{R}, \mathrm{U}, z, \mu)$.

$\mathsf{Ver}(\mathsf{pk}, \pi)$**:** *It computes* $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$ *and* $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$*. If* $g^\mu = \mathrm{U} \cdot \mathrm{R}^c$ *and* $g^z = \mathrm{R} \cdot \mathsf{pk}^d$ *then it outputs* 1*, else* 0.

In the following, we prove that Schnorr+ is complete, sound, (non-adaptively) valid and zero-knowledge.

Lemma 13 Schnorr+ *is complete.*

Proof: Let $(\mathsf{pk}, \mathsf{sk})$ be such that $(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}$, then $\mathsf{pk} = g^{\mathsf{sk}}$. We show that for any $\pi \leftarrow \mathsf{Pro}(\mathsf{sk}, \mathsf{pk})$ and $b \leftarrow \mathsf{Ver}(\mathsf{pk}, \pi)$, then $b = 1$. The algorithm Pro picks $r, u \xleftarrow{\$} \mathbb{Z}_p^*$, sets $\mathrm{R} = g^r$ and $\mathrm{U} = g^y$, computes $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$, $z = r + \mathsf{sk} \cdot d$ and $\mu = u + r \cdot c$, and outputs $\pi = (\mathrm{R}, \mathrm{U}, z, \mu)$. It returns $\pi = (\mathrm{R}, \mathrm{U}, z, \mu)$. We observe that:

$$g^\mu = g^{u + r \cdot c} = g^u \cdot g^{r \cdot c} = \mathrm{U} \cdot \mathrm{R}^c$$
$$g^z = g^{r + \mathsf{sk} \cdot d} = g^r \cdot g^{\mathsf{sk} \cdot d} = \mathrm{R} \cdot \mathsf{pk}^d$$

We deduce that Ver returns 1, which concludes the proof. □

Lemma 14 Schnorr+ *is (non-adaptively) valid and sound.*

Proof: We show that the interactive version of Schnorr+ is valid before to prove that the non-interactive one is (non-adaptively) valid. Let k be an integer and pk be such that $\mathsf{pk} \in \mathscr{L}_{\mathscr{R}}$. Let $\mathsf{P}^* \in \textsc{poly}(k)$ be an algorithm and λ be a function such that:

$$\Pr\left[b \leftarrow \mathsf{out}_{\mathsf{V}}\left(\mathsf{Proof}\langle \mathsf{P}^*(\mathsf{in}); \mathsf{V}(\mathsf{pk})\rangle\right) : b = 1\right] = \lambda(k)$$

We build a knowledge extractor $\mathrm{K}(\mathsf{pk})$. This algorithm runs the oracle $\mathsf{P}^*(\mathsf{in})$ and interacts with it as follows:

1. K receives the commitment (R,U).
2. K picks $(c_0, d_0) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives (z_0, μ_0).
3. K rewinds the oracle to its state before step 2. It picks $(c_1, d_1) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives (z_1, μ_1).

Finally, K returns $\mathsf{sk} = \frac{z_1 - z_0}{d_1 - d_0}$. We show that sk is the secret key corresponding to pk. If $g^{\mu_0} = \mathrm{U}\cdot\mathrm{R}^{c_0}$, $g^{\mu_1} = \mathrm{U}\cdot\mathrm{R}^{c_1}$, $g^{z_0} = \mathrm{R}\cdot\mathsf{pk}^{d_0}$, $g^{z_1} = \mathrm{R}\cdot\mathsf{pk}^{d_1}$ and $d_0 \neq d_1$ then:

$$g^{\mathsf{sk}} = g^{(z_1 - z_0)/(d_1 - d_0)} = \left(\frac{g^{z_1}}{g^{z_0}}\right)^{1/(d_1 - d_0)} = \left(\frac{\mathrm{R}\cdot\mathsf{pk}^{d_1}}{\mathrm{R}\cdot\mathsf{pk}^{d_0}}\right)^{1/(d_1 - d_0)} = \mathsf{pk}$$

It implies that $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}$.

By hypothesis, we know that $\mathsf{Pr}\left[g^{z_0} = \mathrm{R}\cdot\mathsf{pk}^{d_0}\right] \geq \lambda(k)$ and $\mathsf{Pr}\left[g^{z_1} = \mathrm{R}\cdot\mathsf{pk}^{d_1}\right] \geq \lambda(k)$. Moreover, since d_0 and d_1 are chosen at random in $\mathbb{Z}_p^*$, we have $\mathsf{Pr}[d_0 = d_1] = 1/(p-1)$, which is negligible. We deduce that:

$$\begin{aligned}\mathsf{Pr}\left[sk \leftarrow \mathrm{K}^{\mathrm{P}^*(\mathsf{in})}(\mathsf{pk}) : (\mathsf{pk},\mathsf{sk}) \in \mathscr{R}\right] &\geq \mathsf{Pr}\left[g^{z_0} = \mathrm{R}\cdot\mathsf{pk}^{d_0}\right]\cdot\mathsf{Pr}\left[g^{z_1} = \mathrm{R}\cdot\mathsf{pk}^{d_1}\right] - \mathsf{Pr}[d_0 = d_1] \\ &\geq \lambda(k)^2 - \frac{1}{p-1}\end{aligned}$$

We conclude that the interactive version of Schnorr+ is valid. In the non-interactive case, the challenge d comes from a random oracle that takes as input the value (R||U||0). First note that the adversary cannot know d before chosing the commitment (R,U) since the commitment is used as input to the random oracle. On the other hand, the random oracle chooses the challenge d as the verifier in the interactive proof system. We conclude that if the interactive version of Schnorr+ is valid, then the non-interactive one is (non-adaptively) valid. It implies that Schnorr+ is sound, which concludes the proof. □

Lemma 15 Schnorr+ *is zero-knowledge.*

Proof: We show how to build a polynomial time algorithm Sim such that for any $k \in \mathbb{N}$, any $(\mathsf{set},\mathscr{R}) \leftarrow \mathsf{Set}(k)$ and any $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}$, the algorithms $\mathsf{Sim}(\mathsf{pk})$ and $\mathsf{Pro}(\mathsf{sk},\mathsf{pk})$ follow the same probability distribution. Sim picks $z \xleftarrow{\$} \mathbb{Z}_p^*$, $\mu \xleftarrow{\$} \mathbb{Z}_p^*$, $c \xleftarrow{\$} \mathbb{Z}_p^*$ and $d \xleftarrow{\$} \mathbb{Z}_p^*$, and computes $\mathrm{R} = g^z/\mathsf{pk}^d$ and $\mathrm{U} = g^{\mu}/\mathrm{R}^c$. It outputs $\pi = (\mathrm{R},\mathrm{U},z,\mu)$, which is a valid proof since $g^{\mu} = \mathrm{U}\cdot\mathrm{R}^c$ and $g^z = \mathrm{R}\cdot\mathsf{pk}^d$. Moreover, since and z and μ come from the uniform distribution on $\mathbb{Z}_p^*$, then the outputs of `Sim` follow the same distribution as the real protocol. □

Theorem 15 Schnorr+ *is complete, sound, (non-adaptively) valid and zero-knowledge.*

Proof: See Lemma 13, 14 and 15. □

The following definition describes the DLright non-interactive proof system.

Definition 48 $\mathsf{DLright} = (\mathsf{Set},\mathsf{Pro},\mathsf{Ver})$ *is a NIP defined as follows:*

$\mathsf{Set}(k)$: *It generates the setup* $\mathsf{set} = (\mathbb{G}, p, g, \mathrm{H})$ *where* $\mathbb{G}$ *is a group of prime order* p, $g \in \mathbb{G}$ *and and* $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p^*$ *is a hash function. This algorithm returns* $(\mathsf{set},\mathscr{R})$*, where:*

$$(\mathsf{pk},\mathsf{sk}) \in \mathscr{R} \Leftrightarrow (\mathsf{pk} \in \mathbb{Z}_p^*) \wedge (g^{\mathsf{sk}} = \mathsf{pk})$$

$\mathsf{Pro}(\mathsf{sk},\mathsf{pk})$**:** *It picks r, s, t, u, a and b in the uniform distribution on $\mathbb{Z}_p^*$. It computes:*

$$\begin{array}{lll} \mathrm{R} = g^{r} & f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S}) & \mathrm{S} = g^{r\cdot t}\cdot g^{s} \\ c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0) & \beta = b + s\cdot f & \mathrm{B} = g^{b} \\ \mu = u + r\cdot c & \mathrm{U} = g^{u} & \alpha = a + t\cdot f \\ \mathrm{A} = \mathrm{R}^{a} & d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1) & \theta = s + r\cdot t + \mathsf{sk}\cdot d \end{array}$$

It outputs $\pi = (\mathrm{R},\mathrm{U},\mu,\mathrm{S},\mathrm{A},\mathrm{B},\alpha,\beta,\theta)$.

$\mathsf{Ver}(\mathsf{pk},\pi)$**:** *It computes the values* $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$ *and* $f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$. *If the following equations hold, then it outputs* 1, *else* 0*:*

$$g^{\mu} = \mathrm{U}\cdot\mathrm{R}^{c} \qquad \mathrm{R}^{\alpha}\cdot g^{\beta} = \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{S}^{f} \qquad g^{\theta} = \mathrm{S}\cdot\mathsf{pk}^{d}$$

Lemma 16 DLright *is complete.*

Proof: Let $(\mathsf{pk},\mathsf{sk})$ be such that $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}$, then $\mathsf{pk} = g^{\mathsf{sk}}$. We show that for any $\pi \leftarrow \mathsf{Pro}(\mathsf{sk},\mathsf{pk})$ and $b \leftarrow \mathsf{Ver}(\mathsf{pk},\pi)$, then $b = 1$. We set $\pi = (\mathrm{R},\mathrm{U},\mu,\mathrm{S},\mathrm{A},\mathrm{B},\alpha,\beta,\theta)$, we observe that, for $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$ and $f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$:

$$\begin{aligned} g^{\mu} &= g^{u+r\cdot c} = g^{u}\cdot g^{r\cdot c} = \mathrm{U}\cdot\mathrm{R}^{c} \\ \mathrm{R}^{\alpha}\cdot g^{\beta} &= \mathrm{R}^{a}\cdot g^{b}\cdot\mathrm{R}^{t\cdot f}\cdot g^{s\cdot f} = \mathrm{A}\cdot\mathrm{B}\cdot(g^{r\cdot t}\cdot g^{s})^{f} = \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{S}^{f} \\ g^{\theta} &= g^{s+r\cdot t+\mathsf{sk}\cdot d} = g^{r\cdot t}\cdot g^{s}\cdot g^{\mathsf{sk}\cdot d} = \mathrm{S}\cdot\mathsf{pk}^{d} \end{aligned}$$

We deduce that Ver returns 1, which concludes the proof. □

Lemma 17 DLright *is (non-adaptively) valid and sound.*

Proof: We show that the interactive version of DLright is valid before to prove that the non-interactive version is (non-adaptively) valid. Let k be an integer and pk be such that $\mathsf{pk} \in \mathscr{L}_{\mathscr{R}}$. Let $\mathsf{P}^* \in \textsc{poly}(k)$ be an algorithm and λ be a function such that:

$$\mathsf{Pr}\left[b \leftarrow \mathsf{out}_{\mathsf{V}}\left(\mathsf{Proof}\langle\mathsf{P}^*(\mathsf{in});\mathsf{V}(\mathsf{pk})\rangle\right) : b = 1\right] = \lambda(k)$$

We build a knowledge extractor K(pk). This algorithm runs the oracle $\mathsf{P}^*(\mathsf{in})$ and interacts with it as follows:

1. K receives the commitment (R,U).
2. K picks $(c_0,d_0) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives (μ_0) and $(\mathrm{A}_0,\mathrm{B}_0,\mathrm{S}_0)$.
3. K picks $(f_{0,0}) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives $(\alpha_{0,0},\beta_{0,0},\theta_{0,0})$.
4. K rewinds the oracle to its state before step 3. It picks $(f_{0,1}) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives $(\alpha_{0,1},\beta_{0,1},\theta_{0,1})$.
5. K rewinds the oracle to its state before step 2. It picks $(c_1,d_1) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives (μ_1) and $(\mathrm{A}_1,\mathrm{B}_1,\mathrm{S}_1)$.
6. K picks $(f_{1,0}) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives $(\alpha_{1,0},\beta_{1,0},\theta_{1,0})$.
7. K rewinds the oracle to its state before step 6. It picks $(f_{1,1}) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives $(\alpha_{1,1},\beta_{1,1},\theta_{1,1})$.

K sets the following values:

$$\begin{aligned} r &= \frac{\mu_0-\mu_1}{c_0-c_1} \\ s_0 &= r\cdot\frac{\alpha_{0,0}-\alpha_{0,1}}{f_{0,0}-f_{0,1}} + \frac{\beta_{0,0}-\beta_{0,1}}{f_{0,0}-f_{0,1}} \\ s_1 &= r\cdot\frac{\alpha_{1,0}-\alpha_{1,1}}{f_{1,0}-f_{1,1}} + \frac{\beta_{1,0}-\beta_{1,1}}{f_{1,0}-f_{1,1}} \\ \mathsf{sk} &= \frac{\theta_{0,0}-\theta_{1,1}}{d_0-d_1} + \frac{s_1-s_0}{d_0-d_1} \end{aligned}$$

Finally, K returns sk.

Suppose that:

$$
\begin{align}
g^{\mu_0} &= \mathrm{U}\cdot\mathrm{R}^{c_0} \tag{3.1}\\
g^{\mu_1} &= \mathrm{U}\cdot\mathrm{R}^{c_1} \tag{3.2}\\
\mathrm{R}^{\alpha_{0,0}}\cdot g^{\beta_{0,0}} &= \mathrm{A}_0\cdot\mathrm{B}_0\cdot\mathrm{S}_0^{f_{0,0}} \tag{3.3}\\
\mathrm{R}^{\alpha_{0,1}}\cdot g^{\beta_{0,1}} &= \mathrm{A}_0\cdot\mathrm{B}_0\cdot\mathrm{S}_0^{f_{0,1}} \tag{3.4}\\
\mathrm{R}^{\alpha_{1,0}}\cdot g^{\beta_{1,0}} &= \mathrm{A}_1\cdot\mathrm{B}_1\cdot\mathrm{S}_1^{f_{1,0}} \tag{3.5}\\
\mathrm{R}^{\alpha_{1,1}}\cdot g^{\beta_{1,1}} &= \mathrm{A}_1\cdot\mathrm{B}_1\cdot\mathrm{S}_1^{f_{1,1}} \tag{3.6}\\
g^{\theta_{0,0}} &= \mathrm{S}_0\cdot\mathsf{pk}^{d_0} \tag{3.7}\\
g^{\theta_{0,1}} &= \mathrm{S}_0\cdot\mathsf{pk}^{d_0} \tag{3.8}\\
g^{\theta_{1,0}} &= \mathrm{S}_1\cdot\mathsf{pk}^{d_1} \tag{3.9}\\
g^{\theta_{1,1}} &= \mathrm{S}_1\cdot\mathsf{pk}^{d_1} \tag{3.10}
\end{align}
$$

Using equations (3.1) and (3.2), we deduce:

$$g^r = g^{\frac{\mu_0-\mu_1}{c_0-c_1}} = \left(\frac{\mathrm{U}\cdot\mathrm{R}^{c_0}}{\mathrm{U}\cdot\mathrm{R}^{c_1}}\right)^{\frac{1}{c_0-c_1}} = \mathrm{R}$$

Using equations (3.3), (3.4), (3.5) and (3.6), we show that $g^{s_0} = \mathrm{S}_0$ and $g^{s_1} = \mathrm{S}_1$:

$$
\begin{aligned}
g^{s_0} &= g^{r\cdot\frac{\alpha_{0,0}-\alpha_{0,1}}{f_{0,0}-f_{0,1}}+\frac{\beta_{0,0}-\beta_{0,1}}{f_{0,0}-f_{0,1}}} = \mathrm{R}^{\frac{\alpha_{0,0}-\alpha_{0,1}}{f_{0,0}-f_{0,1}}}\cdot g^{\frac{\beta_{0,0}-\beta_{0,1}}{f_{0,0}-f_{0,1}}}\\
&= \left(\mathrm{R}^{\alpha_{0,0}-\alpha_{0,1}}\cdot g^{\beta_{0,0}-\beta_{0,1}}\right)^{\frac{1}{f_{0,0}-f_{0,1}}} = \left(\frac{\mathrm{R}^{\alpha_{0,0}}\cdot g^{\beta_{0,0}}}{\mathrm{R}^{\alpha_{0,1}}\cdot g^{\beta_{0,1}}}\right)^{\frac{1}{f_{0,0}-f_{0,1}}}\\
&= \left(\frac{\mathrm{A}_0\cdot\mathrm{B}_0\cdot\mathrm{S}_0^{f_{0,0}}}{\mathrm{A}_0\cdot\mathrm{B}_0\cdot\mathrm{S}_0^{f_{0,1}}}\right)^{\frac{1}{f_{0,0}-f_{0,1}}} = \mathrm{S}_0\\
g^{s_1} &= g^{r\cdot\frac{\alpha_{1,0}-\alpha_{1,1}}{f_{1,0}-f_{1,1}}+\frac{\beta_{1,0}-\beta_{1,1}}{f_{1,0}-f_{1,1}}} = \mathrm{R}^{\frac{\alpha_{1,0}-\alpha_{1,1}}{f_{1,0}-f_{1,1}}}\cdot g^{\frac{\beta_{1,0}-\beta_{1,1}}{f_{1,0}-f_{1,1}}}\\
&= \left(\mathrm{R}^{\alpha_{1,0}-\alpha_{1,1}}\cdot g^{\beta_{1,0}-\beta_{1,1}}\right)^{\frac{1}{f_{1,0}-f_{1,1}}} = \left(\frac{\mathrm{R}^{\alpha_{1,0}}\cdot g^{\beta_{1,0}}}{\mathrm{R}^{\alpha_{1,1}}\cdot g^{\beta_{1,1}}}\right)^{\frac{1}{f_{1,0}-f_{1,1}}}\\
&= \left(\frac{\mathrm{A}_1\cdot\mathrm{B}_1\cdot\mathrm{S}_1^{f_{1,0}}}{\mathrm{A}_1\cdot\mathrm{B}_1\cdot\mathrm{S}_1^{f_{1,1}}}\right)^{\frac{1}{f_{1,0}-f_{1,1}}} = \mathrm{S}_1
\end{aligned}
$$

Finally, Equations (3.7) and (3.10) implies that:

$$
\begin{aligned}
g^{\mathsf{sk}} &= g^{\frac{\theta_{0,0}-\theta_{1,1}}{d_0-d_1}+\frac{s_1-s_0}{d_0-d_1}} = g^{\frac{\theta_{0,0}-\theta_{1,1}}{d_0-d_1}}\cdot g^{\frac{s_1-s_0}{d_0-d_1}}\\
&= \left(\frac{\mathrm{S}_0\cdot\mathsf{pk}^{d_0}}{\mathrm{S}_1\cdot\mathsf{pk}^{d_1}}\right)^{\frac{1}{d_0-d_1}}\cdot\left(\frac{\mathrm{S}_1}{\mathrm{S}_0}\right)^{\frac{1}{d_0-d_1}} = \left(\frac{\mathrm{S}_0}{\mathrm{S}_1}\right)^{\frac{1}{d_0-d_1}}\cdot\mathsf{pk}\cdot\left(\frac{\mathrm{S}_1}{\mathrm{S}_0}\right)^{\frac{1}{d_0-d_1}}\\
&= \mathsf{pk}
\end{aligned}
$$

Thus, the knowledge extractor K returns sk such that $(\mathsf{pk},\mathsf{sk})\in\mathscr{R}$.

By hypothesis, we know that:

$$\begin{aligned}
\mathrm{P}_1 &= \mathsf{Pr}\left[\left(g^{\mu_0} = \mathrm{U}\cdot\mathrm{R}^{c_0}\right) \wedge \left(\mathrm{R}^{\alpha_{0,0}}\cdot g^{\beta_{0,0}} = \mathrm{A}_0\cdot\mathrm{B}_0\cdot\mathrm{S}_0^{f_{0,0}}\right) \wedge \left(g^{\theta_{0,0}} = \mathrm{S}_0\cdot\mathsf{pk}^{d_0}\right)\right] \geq \lambda(k)\\
\mathrm{P}_2 &= \mathsf{Pr}\left[\left(g^{\mu_0} = \mathrm{U}\cdot\mathrm{R}^{c_0}\right) \wedge \left(\mathrm{R}^{\alpha_{0,1}}\cdot g^{\beta_{0,1}} = \mathrm{A}_0\cdot\mathrm{B}_0\cdot\mathrm{S}_0^{f_{0,1}}\right) \wedge \left(g^{\theta_{0,1}} = \mathrm{S}_0\cdot\mathsf{pk}^{d_0}\right)\right] \geq \lambda(k)\\
\mathrm{P}_3 &= \mathsf{Pr}\left[\left(g^{\mu_1} = \mathrm{U}\cdot\mathrm{R}^{c_1}\right) \wedge \left(\mathrm{R}^{\alpha_{1,0}}\cdot g^{\beta_{1,0}} = \mathrm{A}_1\cdot\mathrm{B}_1\cdot\mathrm{S}_1^{f_{1,0}}\right) \wedge \left(g^{\theta_{1,0}} = \mathrm{S}_1\cdot\mathsf{pk}^{d_1}\right)\right] \geq \lambda(k)\\
\mathrm{P}_4 &= \mathsf{Pr}\left[\left(g^{\mu_1} = \mathrm{U}\cdot\mathrm{R}^{c_1}\right) \wedge \left(\mathrm{R}^{\alpha_{1,1}}\cdot g^{\beta_{1,1}} = \mathrm{A}_1\cdot\mathrm{B}_1\cdot\mathrm{S}_1^{f_{1,1}}\right) \wedge \left(g^{\theta_{1,1}} = \mathrm{S}_1\cdot\mathsf{pk}^{d_1}\right)\right] \geq \lambda(k)
\end{aligned}$$

Moreover, since c_0, c_1, d_0, d_1, f_0 and f_1 are chosen at random in $\mathbb{Z}_p^*$, we have:

$$\begin{aligned}
\mathrm{P}_5 &= \mathsf{Pr}[c_0 = c_1 \vee d_0 = d_1 \vee f_{0,0} = f_{1,1} \vee f_{1,0} = f_{1,1}]\\
&\leq \mathsf{Pr}[c_0 = c_1] + \mathsf{Pr}[d_0 = d_1] + \mathsf{Pr}[f_{0,0} = f_{0,1}] + \mathsf{Pr}[f_{1,0} = f_{1,1}]\\
&\leq \frac{4}{(p-1)}
\end{aligned}$$

which is negligible. We deduce that:

$$\begin{aligned}
\mathsf{Pr}\left[sk \leftarrow \mathsf{K}^{\mathrm{P}^*(\mathsf{in})}(\mathsf{pk}) : (\mathsf{pk},\mathsf{sk}) \in \mathscr{R}\right] &\geq \mathrm{P}_1\cdot\mathrm{P}_2\cdot\mathrm{P}_3\cdot\mathrm{P}_4 - \mathrm{P}_5\\
&\geq \lambda(k)^4 - \frac{4}{p-1}
\end{aligned}$$

We conclude that the interactive version of DLright is valid. In the non-interactive case, the challenge c (resp. d and f) comes from a random oracle that takes as input the value (R||U||0) (reps. (R||U||1) and (R||U||A||B||S)). First note that the adversary cannot know (c,d) (resp. f) before choosing the commitment (R,U) (resp. $(\mathrm{A},\mathrm{B},\mathrm{S})$) since the commitment is used as input to the random oracle. On the other hand, the random oracle chooses the challenges c, d and f as the verifier in the interactive proof system. We conclude that if the interactive version of DLright is valid, then the non-interactive one is (non-adaptively) valid. It implies that DLright is sound, which concludes the proof. □

Lemma 18 DLright *is zero-knowledge.*

Proof: We show how to build a polynomial time algorithm Sim such that for any $k \in \mathbb{N}$, any $(\mathsf{set},\mathscr{R}) \leftarrow \mathsf{Set}(k)$ and any $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}$, the algorithms $\mathsf{Sim}(\mathsf{pk})$ and $\mathsf{Pro}(\mathsf{sk},\mathsf{pk})$ follow the same probability distribution. Sim picks $c,d,f,\mu,\alpha,\beta,\theta \xleftarrow{\$} \mathbb{Z}_p^*$ and $\mathrm{R},\mathrm{A} \xleftarrow{\$} \mathbb{G}$. Then it computes $\mathrm{S} = g^{\theta}/\mathsf{pk}^d$, $\mathrm{U} = g^{\mu}/\mathrm{R}^c$ and $\mathrm{B} = (\mathrm{R}^{\alpha}\cdot g^{\beta})/(\mathrm{A}\cdot\mathrm{S}^f)$. It outputs the proof $\pi = (\mathrm{R},\mathrm{U},c,d,f,\mu,\mathrm{S},\mathrm{A},\mathrm{B},\alpha,\beta,\theta)$ which is a valid proof because:

$$g^{\mu} = \mathrm{U}\cdot\mathrm{R}^c \qquad \mathrm{R}^{\alpha}\cdot g^{\beta} = \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{S}^f \qquad g^{\theta} = \mathrm{S}\cdot\mathsf{pk}^d$$

Moreover, since μ,α,β and θ come from the uniform distribution on $\mathbb{Z}_p^*$, then outputs of `Sim` follow the same distribution as the real protocol. □

Theorem 16 DLright *is complete, sound, (non-adaptively) valid and zero-knowledge.*

Proof: See Lemma 16, 17 and 18. □

Our bidirectional non-interactive proxy re-proof scheme, called NBRP, allows a proxy to transform a (non-interactive) Schnorr+ proof into a (non-interactive) DLright one. The re-prove method

Prover P		**Proxy** Px		**Verifier** V
sk_1		$(\mathsf{rk},\mathsf{pk}_1,\mathsf{pk}_2)$		pk_2
$r \xleftarrow{\$} \mathbb{Z}_p^*;\ u \xleftarrow{\$} \mathbb{Z}_p^*$		$a,b,s \xleftarrow{\$} \mathbb{Z}_p^*$		
$\mathrm{R} = g^r;\ \mathrm{U} = g^u$	$\xrightarrow{(\mathrm{R},\mathrm{U})}$		$\xrightarrow{(\mathrm{R},\mathrm{U})}$	$c,d \xleftarrow{\$} \mathbb{Z}_p^*$
$z = r + \mathsf{sk}\cdot d$	$\xleftarrow{(c,d)}$		$\xleftarrow{(c,d)}$	
$\mu = u + r\cdot c$	$\xrightarrow{(z,\mu)}$	$\mathrm{A} = \mathrm{R}^a$	$\xrightarrow{\mu}$	
		$\mathrm{B} = g^b$		
		$\mathrm{S} = \mathrm{R}^{\mathsf{rk}}\cdot g^s$	$\xrightarrow{(\mathrm{A},\mathrm{B},\mathrm{S})}$	$f \xleftarrow{\$} \mathbb{Z}_p^*$
		$\alpha = a + \mathsf{rk}\cdot f$	$\xleftarrow{f}$	
		$\beta = b + s\cdot f$		
		$\theta = s + z\cdot\mathsf{rk}$	$\xrightarrow{(\alpha,\beta,\theta)}$	If $g^{\mu} = \mathrm{U}\cdot\mathrm{R}^c$
				and $g^{\theta} = \mathrm{S}\cdot\mathsf{pk}_2^d$
				and $\mathrm{R}^{\alpha}\cdot g^{\beta} = \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{S}^f$
				then return 1, else return 0

Figure 3.11: Interactive version of NBRP (Definition 49).

is similar to the IBRP one except that the proxy must prove that it *correctly* constructs the commitment S (as in DLright) from the delegate commitment R. In Figure 3.11, we give the interactive version of the re-proof algorithm of NBRP that transforms an interactive Schnorr+ proof (Figure 3.9) into an interactive DLright one (Figure 3.10). The proxy receives R, U, z and μ from the prover, and receives c and d from the verifier. It computes A, B and β using a chosen number s and the challenge f given by the verifier, as in DLright. Using the re-proof key $\mathsf{sk}_2/\mathsf{sk}_1$, it computes $\mathrm{S} = \mathrm{R}^{\mathsf{rk}}\cdot g^s$, $\alpha = a + \mathsf{rk}\cdot f$ and $\theta = s + z\cdot\mathsf{rk}$. We claim that the verifier cannot distinguish weither it interacts with the proxy or with the delegator because the random numbers r and s *randomize* the commitment S, so the verifier cannot guess weither S is built from the re-proof key rk of not. To transform this protocol into a non-interactive re-proof algorithm, the prover computes each challenge (c, d and f) by hashing the concatenation of all the commitments that were previously computed. More precisely, c and d are generated by hashing R and U, and f is generated by hashing R, U, A, B and S.

Definition 49 $\mathsf{NBRP} = (\mathsf{Set},\mathsf{Gen}_1,\mathsf{Gen}_2,\mathsf{RGen},\ \mathsf{Pro}_1,\mathsf{Pro}_2,\mathsf{Ver}_1,\mathsf{Ver}_2,\mathsf{RPro})$ *is a NIPRP defined as follows:*

$\mathsf{Set}(k)$: *It generates the setup* $\mathsf{set} = (\mathbb{G},p,g,\mathrm{H})$ *where* $\mathbb{G}$ *is a group of prime order* p, $g \in \mathbb{G}$ *and* $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p^*$ *is a hash function. This algorithm returns* $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2)$, *where:*

- $(\mathsf{pk},\mathsf{sk}) \in \mathscr{R}_1 \Leftrightarrow (\mathsf{sk} \in \mathbb{Z}_p^*) \wedge (g^{\mathsf{sk}} = \mathsf{pk})$
- $\mathscr{R}_2 = \mathscr{R}_1$

$\mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$: *It picks* $\mathsf{sk}_1 \xleftarrow{\$} \mathbb{Z}_p^*$, *sets* $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$ *and returns* $(\mathsf{pk}_1,\mathsf{sk}_1)$.

$\mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$: *It picks* $\mathsf{sk}_2 \xleftarrow{\$} \mathbb{Z}_p^*$, *sets* $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$ *and returns* $(\mathsf{pk}_2,\mathsf{sk}_2)$.

$\mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2)$: *It sets* $\mathsf{rk} = \mathsf{sk}_2/\mathsf{sk}_1$ *and returns it.*

$\mathsf{Pro}_1(\mathsf{sk}_1,\mathsf{pk}_1)$**:** *It picks* $r \xleftarrow{\$} \mathbb{Z}_p^*$ *and* $u \xleftarrow{\$} \mathbb{Z}_p^*$, *computes* $\mathrm{R} = g^r$, $\mathrm{U} = g^u$, $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$, $z = r + \mathsf{sk}_1\cdot d$ *and* $\mu = u + r\cdot c$, *and outputs* $\pi = (\mathrm{R},\mathrm{U},z,\mu)$.

$\mathsf{Pro}_2(\mathsf{sk}_2,\mathsf{pk}_2)$**:** *It picks* r,s,t,u,a *and* b *in the uniform distribution on* $\mathbb{Z}_p^*$. *It computes:*

$$\begin{array}{lll}
\mathrm{R} = g^r & f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S}) & \mathrm{S} = g^{r\cdot t}\cdot g^s \\
c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0) & \beta = b + s\cdot f & \mathrm{B} = g^b \\
\mu = u + r\cdot c & \mathrm{U} = g^u & \alpha = a + t\cdot f \\
\mathrm{A} = \mathrm{R}^a & d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1) & \theta = s + r\cdot t + \mathsf{sk}_2\cdot d
\end{array}$$

It outputs $\pi = (\mathrm{R},\mathrm{U},\mu,\mathrm{S},\mathrm{A},\mathrm{B},\alpha,\beta,\theta)$.

$\mathsf{Ver}_1(\mathsf{pk}_1, \pi_1)$**:** *Parses* $\pi_1 = (\mathrm{R}, \mathrm{U}, z, \mu)$, *computes* $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$ *and* $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$. *If* $g^{\mu} = \mathrm{U} \cdot \mathrm{R}^{c}$ *and* $g^{z} = \mathrm{R} \cdot \mathsf{pk}_1^{d}$ *then it outputs* 1, *else* 0.

$\mathsf{Ver}_2(\mathsf{pk}_2, \pi_2)$**:** *Parses* $\pi_2 = (\mathrm{R}, \mathrm{U}, \mu, \mathrm{A}, \mathrm{B}, \mathrm{S}, \alpha, \beta, \theta)$, *computes the values* $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$ *and* $f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$. *If the following equations hold, then it outputs* 1, *else* 0*:*

$$g^{\mu} = \mathrm{U} \cdot \mathrm{R}^{c} \qquad \mathrm{R}^{\alpha} \cdot g^{\beta} = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^{f} \qquad g^{\theta} = \mathrm{S} \cdot \mathsf{pk}_2^{d}$$

$\mathsf{RPro}(\mathsf{rk}, \pi_1)$**:** *Parses* $\pi_1 = (\mathrm{R}, \mathrm{U}, z, \mu)$, *this algorithm picks* s, a *and* b *in the uniform distribution on* $\mathbb{Z}_p^*$ *and computes:*

$$\begin{array}{lll} \mathrm{S} = \mathrm{R}^{\mathsf{rk}} \cdot g^{s} & f = \mathrm{H}(\mathrm{R}, \mathrm{U}, \mathrm{AB}, \mathrm{S}) & \theta = s + z \cdot \mathsf{rk} \\ \mathrm{A} = \mathrm{R}^{a} & \alpha = a + \mathsf{rk} \cdot f & \\ \mathrm{B} = g^{b} & \beta = b + s \cdot f & \end{array}$$

It outputs $\pi_2 = (\mathrm{R}, \mathrm{U}, \mu, \mathrm{S}, \mathrm{A}, \mathrm{B}, \alpha, \beta, \theta)$.

In the following, we prove that NBRP is complete, sound, (non-adaptively) valid, zero-knowledge and secret secure.

Lemma 19 NBRP *is complete.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of $\mathsf{NBRP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Pro}_1, \mathsf{Pro}_2, \mathsf{Ver}_1, \mathsf{Ver}_2, \mathsf{RPro})$. We remark that I_1 is defined as $\mathsf{Schnorr+}$ (Definition 47) and I_2 is defined as $\mathsf{DLright}$ (Definition 48), we deduce that I_1 (resp. I_2) are complete (Theorem 15 and Theorem 16). Let k be an integer and sk_1, pk_1, pk_2 and rk be four keys generated as follows:

- $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$
- $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$
- $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$
- $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$

We deduce that $\mathsf{pk}_1 = g^{\mathsf{sk}_1}$, $\mathsf{pk}_2 = g^{\mathsf{sk}_2}$ and $\mathsf{rk} = \mathsf{sk}_2/\mathsf{sk}_1$. Let the two proofs π_1 and π_2 be generated by $\pi_1 \leftarrow \mathsf{Pro}_1(\mathsf{sk}_1, \mathsf{pk}_1)$ and $\pi_2 \leftarrow \mathsf{RPro}(\mathsf{rk}, \pi_1)$. We show that for any $b \leftarrow \mathsf{Ver}_2(\mathsf{pk}_2, \pi_2)$. We set $\pi = (\mathrm{R}, \mathrm{U}, \mu, \mathrm{S}, \mathrm{A}, \mathrm{B}, \alpha, \beta, \theta)$, we observe that, for $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$ and $f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$:

$$g^{\mu} = g^{u + r \cdot c} = g^{u} \cdot g^{r \cdot c} = \mathrm{U} \cdot \mathrm{R}^{c}$$

$$\mathrm{R}^{\alpha} \cdot g^{\beta} = \mathrm{R}^{a} \cdot g^{b} \cdot \mathrm{R}^{\mathsf{rk} \cdot f} \cdot g^{s \cdot f} = \mathrm{R}^{a} \cdot g^{b} \cdot \left(\mathrm{R}^{\mathsf{rk}} \cdot g^{s}\right)^{f} = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^{f}$$

$$g^{\theta} = g^{s + z \cdot \mathsf{rk}} = g^{s + (r + \mathsf{sk}_1 \cdot d) \cdot \frac{\mathsf{sk}_2}{sk_1}} = g^{s + r \cdot \mathsf{rk} + \mathsf{sk}_2 \cdot d} = g^{s + r \cdot \mathsf{rk}} \cdot g^{\mathsf{sk}_2 \cdot d} = \mathrm{R}^{\mathsf{rk}} \cdot g^{s} \cdot \mathsf{pk}_2^{d} = \mathrm{S} \cdot \mathsf{pk}_2^{d}$$

We deduce that $b = 1$, which concludes the proof. □

Lemma 20 NBRP *is sound and (non-adaptively) valid.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of $\mathsf{NBRP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Pro}_1, \mathsf{Pro}_2, \mathsf{Ver}_1, \mathsf{Ver}_2, \mathsf{RPro})$. We remark that I_1 is defined as $\mathsf{Schnorr+}$ (Definition 47) and I_2 is defined as $\mathsf{DLright}$ (Definition 48), we deduce that I_1 and I_2 are sound and (non-adaptively) valid (Theorem 15 and Theorem 16), which concludes the proof. □

Lemma 21 NBRP *is zero-knowledge.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of NBRP = (Set, Gen_1, Gen_2, RGen, Pro_1, Pro_2, Ver_1, Ver_2, RPro). We remark that I_1 is defined as Schnorr+ (Definition 47) and I_2 is defined as DLright (Definition 48), we deduce that I_1 and I_2 are zero-knowledge (Theorem 15 and Theorem 16).

We show how to build a polynomial time algorithm Sim_1 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, the algorithms $\mathsf{Sim}_1(\mathsf{pk}_1, \mathsf{pk}_2)$ and the two algorithms $\alpha_1 \leftarrow \mathsf{Pro}_1(\mathsf{sk}_1, \mathsf{pk}_1)$ and $\alpha_2 \leftarrow \mathsf{RPro}(\mathsf{rk}, \alpha_1)$ follow the same probability distribution. We build $\mathsf{Sim}_1(\mathsf{pk}_1, \mathsf{pk}_2)$ as follows: this simulator picks $z, c, d, f, \mu, \alpha, \beta, \theta \xleftarrow{\$} \mathbb{Z}_p^*$ and $A \xleftarrow{\$} \mathbb{G}$. It computes $R = g^z/\mathsf{pk}_1^d$, $S = g^\theta/\mathsf{pk}_2^d$, $U = g^\mu/R^c$ and $B = (R^\alpha \cdot g^\beta)/(A \cdot S^f)$. It outputs $((R, U, z, \mu), (R, U, \mu, S, A, B, \alpha, \beta, \theta))$, which is is a valid proof because:

$$g^z = R \cdot \mathsf{pk}_1^d \qquad g^\mu = U \cdot R^c \qquad R^\alpha \cdot g^\beta = A \cdot B \cdot S^f \qquad g^\theta = S \cdot \mathsf{pk}^d$$

Moreover, since μ, α, β, and θ come from the uniform distribution on $\mathbb{Z}_p^*$, then the outputs of Sim_1 follow the same distribution as the real algorithms.

We show how to build a polynomial time algorithm Sim_2 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, and any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, the algorithms $\mathsf{Sim}_2(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{Pro}_1(\mathsf{sk}_1, \mathsf{pk}_1)$ follow the same probability distribution. Since I_1 is zero-knowledge, there exists a simulator Sim such that the outputs of $\mathsf{Pro}_1(\mathsf{sk}_1, \mathsf{pk}_1)$ follow the same probability distribution as the outputs of $\mathsf{Sim}(\mathsf{pk}_1)$. We build $\mathsf{Sim}_2(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$ as follows: it runs $\pi_1 \leftarrow \mathsf{Sim}(\mathsf{pk}_1)$ and returns π_1. Clearly, Sim_2 follows the same distribution as the real algorithm Pro.

We show how to build a polynomial time algorithm Sim_3 such that for any $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{pk}'_1) \leftarrow \mathsf{Gen}_1(\mathsf{set}, \mathscr{R}_1)$, any $(\mathsf{pk}_2, \mathsf{sk}_2, \mathsf{sk}'_2) \leftarrow \mathsf{Gen}_2(\mathsf{set}, \mathscr{R}_2)$, any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1, \mathsf{sk}_2)$, any polynomial time algorithm $\mathscr{A} \in \textsc{poly}(k)$ and any $\pi_1 \leftarrow \mathscr{A}(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ where $\pi_1 = (R, U, z, \mu)$, the algorithms $\mathsf{Sim}_3(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{RPro}(\mathsf{rk}, \pi_1)$ follow the same probability distribution. We set $c = H(R||U||0)$ and $d = H(R||U||1)$. We build $\mathsf{Sim}_3(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ as follows: this simulator picks $f, \alpha, \beta, \theta \xleftarrow{\$} \mathbb{Z}_p^*$ and $A \xleftarrow{\$} \mathbb{G}$. It computes $S = g^\theta/\mathsf{pk}_2^d$ and $B = (R^\alpha \cdot g^\beta)/(A \cdot S^f)$. It outputs $\pi_2 = (R, U, \mu, S, A, B, \alpha, \beta, \theta)$, which is is a valid proof because:

$$g^\mu = U \cdot R^c \qquad R^\alpha \cdot g^\beta = A \cdot B \cdot S^f \qquad g^\theta = S \cdot \mathsf{pk}^d$$

Moreover, since μ, α, β, and θ come from the uniform distribution on $\mathbb{Z}_p^*$, then the outputs of Sim_3 follow the same distribution as the real algorithm RPro. □

Theorem 17 NBRP *is bidirectional, complete, sound, (non-adaptively) valid, zero-knowledge and secret secure under the DL assumption.*

Proof: We already show that NBRP is complet (Lemma 19), sound, (non-adaptively) valid (Lemma 20), and zero-knowledge (Lemma 21). Since the setup and the key generation algorithms Set, Gen_1, Gen_2 and RGen are defined as in IBRP (Definition 40), then we can prove that NBRP is bidirectional and secret secure as we prove that IBRP is bidirectional and secret secure in Theorem 13. □

3.3.3 Unidirectional Non-interactive Scheme

The last scheme we present in this chapter is called NURP and is both unidirectional and non-interactive. It is obtained by merging the (non-interactive) scheme NBRP and the (unidirectional) scheme IURP. More precisely, we use the same method than NBRP except that the delegator proof system, which is DLright in NBRP, is replaced by a proof of knowledge of a pairing inversion called FAPIright (Definition 50). The design of this proof system is similar to the design of DLright. An interactive version of FAPIright is given in Figure 3.12. On the other hand, the delegate proof system is Schnorr+, as in the bidirectional non-interactive scheme NBRP.

Prover P		**Verifier** V
sk		pk
$a, b, r, s, t, u \xleftarrow{\$} \mathbb{Z}_p^*$		
$\mathrm{R} = g_1^r$; $\mathrm{U} = g_1^u$	$\xrightarrow{(\mathrm{R},\mathrm{U})}$	
$\mu = u + r \cdot c$	$\xleftarrow{(c,d)}$	$c, d \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{S} = e(g_1, g_2)^{r \cdot t + s}$	$\xrightarrow{\mu}$	
$\mathrm{A} = e(\mathrm{R}, g_2)^a$		
$\mathrm{B} = e(g_1, g_2)^b$	$\xrightarrow{(\mathrm{A},\mathrm{B},\mathrm{S})}$	
$\alpha = g_2^{a+t \cdot f}$	$\xleftarrow{f}$	$f \xleftarrow{\$} \mathbb{Z}_p^*$
$\beta = g_2^{b+s \cdot f}$		
$\theta = g_2^{s+r \cdot t} \cdot \mathsf{sk}^d$	$\xrightarrow{(\alpha,\beta,\theta)}$	If $g_1^\mu = \mathrm{U} \cdot \mathrm{R}^c$
		and $e(g_1, \theta) = \mathrm{S} \cdot \mathsf{pk}^d$
		and $e(\mathrm{R}, \alpha) \cdot e(g_1, \beta) = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^f$
		then return 1, else 0

Figure 3.12: Protocol Proof of interactive FAPIright (Definition 50).

Definition 50 FAPIright = (Set, Pro, Ver) *is a NIP defined as follows:*

Set(k): *It generates the setup* set $= (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e, \mathrm{H})$ *where* $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$ *are three groups of prime order* p*,* $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$*,* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is a type* 2 *bilinear pairing and* $\mathrm{H} : \{0,1\} \to \mathbb{Z}_p^*$ *is a hash function. This algorithm returns* (set, $\mathscr{R}$) *such that:*

$$(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R} \Leftrightarrow (\mathsf{sk} \in \mathbb{G}_2) \wedge (e(g_1, \mathsf{sk}) = \mathsf{pk})$$

Pro(sk, pk)**:** *It picks r, s, t, u, a and b in the uniform distribution on* $\mathbb{Z}_p^*$ *and computes:*

$$\begin{array}{lll} \mathrm{R} = g_1^r & f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S}) & \mathrm{S} = e(g_1, g_2)^{r \cdot t + s} \\ c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0) & \beta = g_2^{b+s \cdot f} & \mathrm{B} = e(g_1, g_2)^b \\ \mu = u + r \cdot c & \mathrm{U} = g_1^u & \alpha = g_2^{a+t \cdot f} \\ \mathrm{A} = e(\mathrm{R}, g_2)^a & d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1) & \theta = g_2^{s+r \cdot t} \cdot \mathsf{sk}^d \end{array}$$

It outputs $\pi = (\mathrm{R}, \mathrm{U}, \mu, \mathrm{S}, \mathrm{A}, \mathrm{B}, \alpha, \beta, \theta)$.

Ver(pk, π)**:** *It computes the values* $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$*,* $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$ *and* $f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$*. If the following equations hold, then it outputs* 1*, else* 0*:*

$$g_1^\mu = \mathrm{U} \cdot \mathrm{R}^c \qquad e(\mathrm{R}, \alpha) \cdot e(g_1, \beta) = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^f \qquad e(g_1, \theta) = \mathrm{S} \cdot \mathsf{pk}^d$$

In the following, we prove that FAPIright is complete, sound, (non-adaptively) valid and zero-knowledge.

Lemma 22 FAPIright *is complete.*

Proof: Let (pk, sk) be such that (pk, sk) $\in \mathscr{R}$, then pk $= g^{\mathsf{sk}}$. we show that for any $\pi \leftarrow$ Pro(sk, pk) and $b \leftarrow$ Ver(pk, π), then $b = 1$. We set $\pi = (\mathrm{R}, \mathrm{U}, \mu, \mathrm{S}, \mathrm{A}, \mathrm{B}, \alpha, \beta, \theta)$, we observe that, for $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$ and $f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$:

$$\begin{aligned} g_1^\mu = g_1^{u+r \cdot c} = g_1^u \cdot g_1^{r \cdot c} &= \mathrm{U} \cdot \mathrm{R}^c \\ e(\mathrm{R}, \alpha) \cdot e(g_1, \beta) = e(\mathrm{R}, g_2^{a+t \cdot f}) \cdot e(g_1, g_2^{b+s \cdot f}) &= e(\mathrm{R}, g_2)^a \cdot e(\mathrm{R}, g_2)^{t \cdot f} \cdot e(g_1, g_2)^b \cdot e(g_1, g_2)^{s \cdot f} \\ &= \mathrm{A} \cdot \mathrm{B} \cdot \left(e(g_1, g_2)^{r \cdot t} \cdot e(g_1, g_2)^s\right)^f = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^f \\ e(g_1, \theta) = e(g_1, g_2^{s+r \cdot t} \cdot \mathsf{sk}^d) &= e(g_1, g_2)^{s+r \cdot t} \cdot e(g_1, \mathsf{sk})^d = \mathrm{S} \cdot \mathsf{pk}^d \end{aligned}$$

We deduce that Ver returns 1, which concludes the proof. □

Lemma 23 $\mathsf{FAPIright}$ *is (non-adaptively) valid and sound.*

Proof: We show that the interactive version of $\mathsf{FAPIright}$ is valid before to prove that the non-interactive version is (non-adaptively) valid. Let k be an integer and pk be such that $\mathsf{pk} \in \mathscr{L}_{\mathscr{R}}$. Let $\mathsf{P}^* \in \textsc{poly}(k)$ be an algorithm and λ be a function such that:

$$\Pr\left[b \leftarrow \mathsf{out}_{\mathsf{V}}\left(\mathsf{Proof}\langle \mathsf{P}^*(\mathsf{in}); \mathsf{V}(\mathsf{pk})\rangle\right) : b = 1\right] = \lambda(k)$$

We build a knowledge extractor $\mathsf{K}(\mathsf{pk})$. This algorithm runs the oracle $\mathsf{P}^*(\mathsf{in})$ and interacts with it as follows:

1. K receives the commitment (R, U).
2. K picks $(c_0, d_0) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives (μ_0) and $(\mathrm{A}_0, \mathrm{B}_0, \mathrm{S}_0)$.
3. K picks $(f_{0,0}) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives $(\alpha_{0,0}, \beta_{0,0}, \theta_{0,0})$.
4. K rewinds the oracle to its state before step 3. It picks $(f_{0,1}) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives $(\alpha_{0,1}, \beta_{0,1}, \theta_{0,1})$.
5. K rewinds the oracle to its state before step 2. It picks $(c_1, d_1) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives (μ_1) and $(\mathrm{A}_1, \mathrm{B}_1, \mathrm{S}_1)$.
6. K picks $(f_{1,0}) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives $(\alpha_{1,0}, \beta_{1,0}, \theta_{1,0})$.
7. K rewinds the oracle to its state before step 6. It picks $(f_{1,1}) \xleftarrow{\$} \mathbb{Z}_p^*$, sends it to the oracle and receives $(\alpha_{1,1}, \beta_{1,1}, \theta_{1,1})$.

K sets the following values:

$$r = \frac{\mu_0 - \mu_1}{c_0 - c_1}$$
$$s_0 = \left(\frac{\alpha_{0,0}}{\alpha_{0,1}}\right)^{\frac{r}{f_{0,0}-f_{0,1}}} \cdot \left(\frac{\beta_{0,0}}{\beta_{0,1}}\right)^{\frac{1}{f_{0,0}-f_{0,1}}}$$
$$s_1 = \left(\frac{\alpha_{1,0}}{\alpha_{1,1}}\right)^{\frac{r}{f_{1,0}-f_{1,1}}} \cdot \left(\frac{\beta_{1,0}}{\beta_{1,1}}\right)^{\frac{1}{f_{1,0}-f_{1,1}}}$$
$$\mathsf{sk} = \left(\frac{\theta_{0,0} \cdot s_1}{\theta_{1,1} \cdot s_0}\right)^{\frac{1}{d_0-d_1}}$$

Finally, K returns sk.

Assume that:

$$g_1^{\mu_0} = \mathrm{U} \cdot \mathrm{R}^{c_0} \tag{3.11}$$
$$g_1^{\mu_1} = \mathrm{U} \cdot \mathrm{R}^{c_1} \tag{3.12}$$
$$e(\mathrm{R}, \alpha_{0,0}) \cdot e(g_1, \beta_{0,0}) = \mathrm{A}_0 \cdot \mathrm{B}_0 \cdot \mathrm{S}_0^{f_{0,0}} \tag{3.13}$$
$$e(\mathrm{R}, \alpha_{0,1}) \cdot e(g_1, \beta_{0,1}) = \mathrm{A}_0 \cdot \mathrm{B}_0 \cdot \mathrm{S}_0^{f_{0,1}} \tag{3.14}$$
$$e(\mathrm{R}, \alpha_{1,0}) \cdot e(g_1, \beta_{1,0}) = \mathrm{A}_1 \cdot \mathrm{B}_1 \cdot \mathrm{S}_1^{f_{1,0}} \tag{3.15}$$
$$e(\mathrm{R}, \alpha_{1,1}) \cdot e(g_1, \beta_{1,1}) = \mathrm{A}_1 \cdot \mathrm{B}_1 \cdot \mathrm{S}_1^{f_{1,1}} \tag{3.16}$$
$$e(g_1, \theta_{0,0}) = \mathrm{S}_0 \cdot \mathsf{pk}^{d_0} \tag{3.17}$$
$$e(g_1, \theta_{0,1}) = \mathrm{S}_0 \cdot \mathsf{pk}^{d_0} \tag{3.18}$$
$$e(g_1, \theta_{1,0}) = \mathrm{S}_1 \cdot \mathsf{pk}^{d_1} \tag{3.19}$$
$$e(g_1, \theta_{1,1}) = \mathrm{S}_1 \cdot \mathsf{pk}^{d_1} \tag{3.20}$$

Using equations (3.11) and (3.12), we deduce:

$$g_1^r = g_1^{\frac{\mu_0-\mu_1}{c_0-c_1}} = \left(\frac{\mathrm{U} \cdot \mathrm{R}^{c_0}}{\mathrm{U} \cdot \mathrm{R}^{c_1}}\right)^{\frac{1}{c_0-c_1}} = \mathrm{R}$$

Using equations (3.13), (3.14), (3.15) and (3.16), we show that $e(g_1, s_0) = \mathrm{S}_0$ and $e(g_1, s_1) = \mathrm{S}_1$:

$$\begin{aligned}
e(g_1, s_0) &= e\left(g_1, \left(\frac{\alpha_{0,0}}{\alpha_{0,1}}\right)^{\frac{r}{f_{0,0}-f_{0,1}}} \cdot \left(\frac{\beta_{0,0}}{\beta_{0,1}}\right)^{\frac{1}{f_{0,0}-f_{0,1}}}\right) = e\left(g_1, \left(\frac{\alpha_{0,0}}{\alpha_{0,1}}\right)^{\frac{r}{f_{0,0}-f_{0,1}}}\right) \cdot e\left(g_1, \left(\frac{\beta_{0,0}}{\beta_{0,1}}\right)^{\frac{1}{f_{0,0}-f_{0,1}}}\right) \\
&= \left(e\left(g_1^r, \frac{\alpha_{0,0}}{\alpha_{0,1}}\right) \cdot e\left(g_1, \frac{\beta_{0,0}}{\beta_{0,1}}\right)\right)^{\frac{1}{f_{0,0}-f_{0,1}}} = \left(\frac{e\left(\mathrm{R}, \alpha_{0,0}\right) \cdot e\left(g_1, \beta_{0,0}\right)}{e\left(\mathrm{R}, \alpha_{0,1}\right) \cdot e\left(g_1, \beta_{0,1}\right)}\right)^{\frac{1}{f_{0,0}-f_{0,1}}} \\
&= \left(\frac{\mathrm{A}_0 \cdot \mathrm{B}_0 \cdot \mathrm{S}_0^{f_{0,0}}}{\mathrm{A}_0 \cdot \mathrm{B}_0 \cdot \mathrm{S}_0^{f_{0,1}}}\right)^{\frac{1}{f_{0,0}-f_{0,1}}} = \mathrm{S}_0 \\
e(g_1, s_1) &= e\left(g_1, \left(\frac{\alpha_{1,0}}{\alpha_{1,1}}\right)^{\frac{r}{f_{1,0}-f_{1,1}}} \cdot \left(\frac{\beta_{1,0}}{\beta_{1,1}}\right)^{\frac{1}{f_{1,0}-f_{1,1}}}\right) = e\left(g_1, \left(\frac{\alpha_{1,0}}{\alpha_{1,1}}\right)^{\frac{r}{f_{1,0}-f_{1,1}}}\right) \cdot e\left(g_1, \left(\frac{\beta_{1,0}}{\beta_{1,1}}\right)^{\frac{1}{f_{1,0}-f_{1,1}}}\right) \\
&= \left(e\left(g_1^r, \frac{\alpha_{1,0}}{\alpha_{1,1}}\right) \cdot e\left(g_1, \frac{\beta_{1,0}}{\beta_{1,1}}\right)\right)^{\frac{1}{f_{1,0}-f_{1,1}}} = \left(\frac{e\left(\mathrm{R}, \alpha_{1,0}\right) \cdot e\left(g_1, \beta_{1,0}\right)}{e\left(\mathrm{R}, \alpha_{1,1}\right) \cdot e\left(g_1, \beta_{1,1}\right)}\right)^{\frac{1}{f_{1,0}-f_{1,1}}} \\
&= \left(\frac{\mathrm{A}_1 \cdot \mathrm{B}_1 \cdot \mathrm{S}_1^{f_{1,1}}}{\mathrm{A}_1 \cdot \mathrm{B}_1 \cdot \mathrm{S}_1^{f_{1,1}}}\right)^{\frac{1}{f_{1,0}-f_{1,1}}} = \mathrm{S}_1
\end{aligned}$$

Finally, Equations (3.17) and (3.20) implie that:

$$e(g_1, \mathsf{sk}) = e\left(g_1, \left(\frac{\theta_{0,0} \cdot s_1}{\theta_{1,1} \cdot s_0}\right)\right)^{\frac{1}{d_0-d_1}} = \left(\frac{e(g_1, \theta_{0,0}) \cdot e(g_1, s_1)}{e(g_1, \theta_{1,1}) \cdot e(g_1, s_0)}\right)^{\frac{1}{d_0-d_1}} = \left(\frac{\mathrm{S}_0 \cdot \mathsf{pk}^{d_0} \cdot \mathrm{S}_1}{\mathrm{S}_1 \cdot \mathsf{pk}^{d_1} \cdot \mathrm{S}_0}\right)^{\frac{1}{d_0-d_1}} = \mathsf{pk}$$

Thus, the knowledge extractor K returns sk such that $(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}$.

By hypothesis, we know that:

$$\begin{aligned}
\mathrm{P}_1 = \mathsf{Pr}\left[\left(g_1^{\mu_0} = \mathrm{U} \cdot \mathrm{R}^{c_0}\right) \wedge \left(e(\mathrm{R}, \alpha_{0,0}) \cdot e(g_1, \beta_{0,0}) = \mathrm{A}_0 \cdot \mathrm{B}_0 \cdot \mathrm{S}_0^{f_{0,0}}\right) \wedge \left(e(g_1, \theta_{0,0}) = \mathrm{S}_0 \cdot \mathsf{pk}^{d_0}\right)\right] \geq \lambda(k) \\
\mathrm{P}_2 = \mathsf{Pr}\left[\left(g_1^{\mu_0} = \mathrm{U} \cdot \mathrm{R}^{c_0}\right) \wedge \left(e(\mathrm{R}, \alpha_{0,1}) \cdot e(g_1, \beta_{0,1}) = \mathrm{A}_0 \cdot \mathrm{B}_0 \cdot \mathrm{S}_0^{f_{0,1}}\right) \wedge \left(e(g_1, \theta_{0,1}) = \mathrm{S}_0 \cdot \mathsf{pk}^{d_0}\right)\right] \geq \lambda(k) \\
\mathrm{P}_3 = \mathsf{Pr}\left[\left(g_1^{\mu_1} = \mathrm{U} \cdot \mathrm{R}^{c_1}\right) \wedge \left(e(\mathrm{R}, \alpha_{1,0}) \cdot e(g_1, \beta_{1,0}) = \mathrm{A}_1 \cdot \mathrm{B}_1 \cdot \mathrm{S}_1^{f_{1,0}}\right) \wedge \left(e(g_1, \theta_{1,0}) = \mathrm{S}_1 \cdot \mathsf{pk}^{d_1}\right)\right] \geq \lambda(k) \\
\mathrm{P}_4 = \mathsf{Pr}\left[\left(g_1^{\mu_1} = \mathrm{U} \cdot \mathrm{R}^{c_1}\right) \wedge \left(e(\mathrm{R}, \alpha_{1,1}) \cdot e(g_1, \beta_{1,1}) = \mathrm{A}_1 \cdot \mathrm{B}_1 \cdot \mathrm{S}_1^{f_{1,1}}\right) \wedge \left(e(g_1, \theta_{1,1}) = \mathrm{S}_1 \cdot \mathsf{pk}^{d_1}\right)\right] \geq \lambda(k)
\end{aligned}$$

Moreover, since c_0, c_1, d_0, d_1, f_0 and f_1 are chosen at random in $\mathbb{Z}_p^*$, we have:

$$\begin{aligned}
\mathrm{P}_5 &= \mathsf{Pr}[c_0 = c_1 \vee d_0 = d_1 \vee f_{0,0} = f_{1,1} \vee f_{1,0} = f_{1,1}] \\
&\leq \mathsf{Pr}[c_0 = c_1] + \mathsf{Pr}[d_0 = d_1] + \mathsf{Pr}[f_{0,0} = f_{0,1}] + \mathsf{Pr}[f_{1,0} = f_{1,1}] \\
&\leq \frac{4}{(p-1)}
\end{aligned}$$

which is negligible. We deduce that:

$$\begin{aligned}
\mathsf{Pr}\left[sk \leftarrow \mathsf{K}^{\mathrm{P}^*(\mathsf{in})}(\mathsf{pk}) : (\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}\right] &\geq \mathrm{P}_1 \cdot \mathrm{P}_2 \cdot \mathrm{P}_3 \cdot \mathrm{P}_4 - \mathrm{P}_5 \\
&\geq \lambda(k)^4 - \frac{4}{p-1}
\end{aligned}$$

We conclude that the interactive version of $\mathsf{FAPIright}$ is valid. In the non-interactive case, the challenge c (resp. d and f) comes from a random oracle that takes as input the value $(\mathrm{R}||\mathrm{U}||0)$ (reps. $(\mathrm{R}||\mathrm{U}||1)$ and $(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$). First note that the adversary cannot know (c, d) (resp. f) before choosing the commitment (R, U) (resp. $(\mathrm{A}, \mathrm{B}, \mathrm{S})$) since the commitment is used as input to the random oracle. On the other hand, the random oracle chooses the challenges c, d and f as the

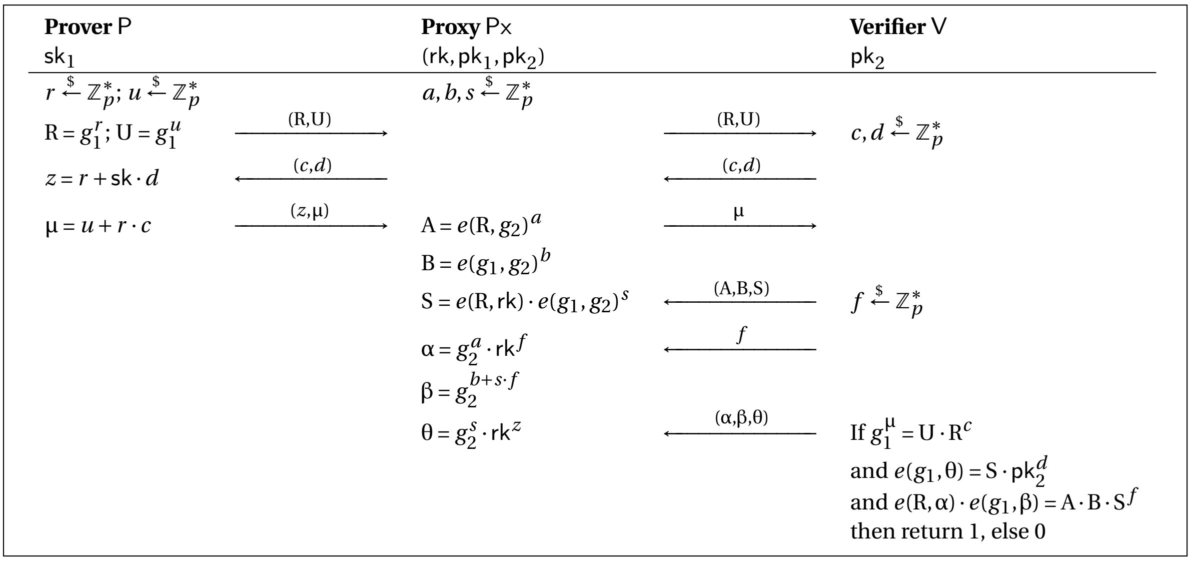

Prover P
sk_1
Proxy Px
$(\mathsf{rk}, \mathsf{pk}_1, \mathsf{pk}_2)$
Verifier V
pk_2
$r \xleftarrow{\$} \mathbb{Z}_p^*;\ u \xleftarrow{\$} \mathbb{Z}_p^*$
$a, b, s \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathrm{R} = g_1^r;\ \mathrm{U} = g_1^u$
$\xrightarrow{(\mathrm{R},\mathrm{U})}$
$\xrightarrow{(\mathrm{R},\mathrm{U})}$
$c, d \xleftarrow{\$} \mathbb{Z}_p^*$
$z = r + \mathsf{sk} \cdot d$
$\xleftarrow{(c,d)}$
$\xleftarrow{(c,d)}$
$\mu = u + r \cdot c$
$\xrightarrow{(z,\mu)}$
$\mathrm{A} = e(\mathrm{R}, g_2)^a$
$\xrightarrow{\mu}$
$\mathrm{B} = e(g_1, g_2)^b$
$\mathrm{S} = e(\mathrm{R}, \mathsf{rk}) \cdot e(g_1, g_2)^s$
$\xleftarrow{(\mathrm{A},\mathrm{B},\mathrm{S})}$
$f \xleftarrow{\$} \mathbb{Z}_p^*$
$\alpha = g_2^a \cdot \mathsf{rk}^f$
$\xleftarrow{f}$
$\beta = g_2^{b+s\cdot f}$
$\theta = g_2^s \cdot \mathsf{rk}^z$
$\xleftarrow{(\alpha,\beta,\theta)}$
If $g_1^\mu = \mathrm{U} \cdot \mathrm{R}^c$
and $e(g_1, \theta) = \mathrm{S} \cdot \mathsf{pk}_2^d$
and $e(\mathrm{R}, \alpha) \cdot e(g_1, \beta) = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^f$
then return 1, else 0

Figure 3.13: Interactive version of NURP (Definition 51).

verifier in the interactive proof system. We conclude that if the interactive version of FAPIright is valid, then the non-interactive one is (non-adaptively) valid. It implies that FAPIright is sound, which concludes the proof. □

Lemma 24 FAPIright *is zero-knowledge.*

Proof: We show how to build a polynomial time algorithm Sim such that for any $k \in \mathbb{N}$, any $(\mathsf{set}, \mathscr{R}) \leftarrow \mathsf{Set}(k)$ and any $(\mathsf{pk}, \mathsf{sk}) \in \mathscr{R}$, the algorithms $\mathsf{Sim}(\mathsf{pk})$ and $\mathsf{Pro}(\mathsf{sk}, \mathsf{pk})$ follow the same probability distribution. Sim picks $c, d, f, \mu, \alpha, \beta, \theta \xleftarrow{\$} \mathbb{Z}_p^*$ and $\mathrm{R}, \mathrm{A} \xleftarrow{\$} \mathbb{G}_1$. Then it computes:

$$\mathrm{S} = \frac{e(g_1,\theta)}{\mathsf{pk}^d} \qquad \mathrm{U} = \frac{g_1^\mu}{\mathrm{R}^c} \qquad \mathrm{B} = \frac{e(\mathrm{R},\alpha) \cdot e(g_1,\beta)}{\mathrm{A} \cdot \mathrm{S}^f}$$

It outputs $\pi = (\mathrm{R}, \mathrm{U}, c, d, f, \mu, \mathrm{S}, \mathrm{A}, \mathrm{B}, \alpha, \beta, \theta)$, which is a valid proof because:

$$g_1^\mu = \mathrm{U} \cdot \mathrm{R}^c \qquad e(\mathrm{R}, \alpha) \cdot e(g_1, \beta) = \mathrm{A} \cdot \mathrm{B} \cdot \mathrm{S}^f \qquad e(g_1, \theta) = \mathrm{S} \cdot \mathsf{pk}^d$$

Moreover, since μ, α, β and θ come from uniform distributions on $\mathbb{Z}_p^*$, then outputs of Sim follow the same distribution as the real algorithm Pro. □

Theorem 18 FAPIright *is complete, sound, (non-adaptively) valid and zero-knowledge.*

Proof: See Lemma 22, 23 and 24. □

The construction of our unidirectional and non-interactive scheme NURP is given in the following definition. Moreover, we give the interactive version of its re-proof algorithm in Figure 3.13.

Definition 51 $\mathsf{NURP} = (\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2, \mathsf{RGen}, \mathsf{Pro}_1, \mathsf{Pro}_2, \mathsf{Ver}_1, \mathsf{Ver}_2, \mathsf{RPro})$ *is a NIPRP defined as follows:*

$\mathsf{Set}(k)$: *It generates the setup* $\mathsf{set} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e, \mathrm{H})$ *where* $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$ *are three groups of prime order* p*,* $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$*,* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *is a type* 2 *bilinear pairing and* $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p^*$*. This algorithm returns* $(\mathsf{set}, \mathscr{R}_1, \mathscr{R}_2)$*, where:*

- $(\mathsf{pk}_1, \mathsf{sk}_1) \in \mathscr{R}_1 \Leftrightarrow (\mathsf{sk}_1 \in \mathbb{Z}_p^*) \wedge (g_1^{\mathsf{sk}_1} = \mathsf{pk}_1)$

- $(\mathsf{pk}_2,\mathsf{sk}_2)\in\mathscr{R}_2 \Leftrightarrow (\mathsf{sk}_2\in\mathbb{G}_2)\wedge(e(g_1,\mathsf{sk}_2)=\mathsf{pk}_2)$

$\mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$: *It picks* $\mathsf{sk}_1 \xleftarrow{\$} \mathbb{Z}_p^*$*, sets* $\mathsf{pk}_1 = g_1^{\mathsf{sk}_1}$ *and* $\mathsf{pk}_1' = g_2^{1/\mathsf{sk}_1}$ *and returns* $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}_1')$*.*

$\mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$: *It picks* $\mathsf{sk}_2' \xleftarrow{\$} \mathbb{Z}_p^*$*, sets* $\mathsf{sk}_2 = g_2^{\mathsf{sk}_2'}$ *and* $\mathsf{pk}_2 = e(g_1,\mathsf{sk}_2)$*, and returns* $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}_2')$*.*

$\mathsf{RGen}(\mathsf{pk}_1,\mathsf{pk}_1',\mathsf{sk}_2,\mathsf{sk}_2')$: *It sets* $\mathsf{rk} = (\mathsf{pk}_1')^{\mathsf{sk}_2'}$*.*

$\mathsf{Pro}_1(\mathsf{sk}_1,\mathsf{pk}_1)$**:** *It picks* $r \xleftarrow{\$} \mathbb{Z}_p^*$ *and* $u \xleftarrow{\$} \mathbb{Z}_p^*$*, computes* $\mathrm{R} = g_1^r$, $\mathrm{U} = g_1^u$, $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$, $z = r + \mathsf{sk}_1\cdot d$ *and* $\mu = u + r\cdot c$*, and outputs* $\pi = (\mathrm{R},\mathrm{U},z,\mu)$*.*

$\mathsf{Pro}_2(\mathsf{sk}_2,\mathsf{pk}_2)$**:** *It picks* r, s, t, u, a *and* b *in the uniform distribution on* $\mathbb{Z}_p^*$ *and computes:*

$$\begin{array}{lll}
\mathrm{R} = g_1^r & f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S}) & \mathrm{S} = e(g_1,g_2)^{r\cdot t+s} \\
c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0) & \beta = g_2^{b+s\cdot f} & \mathrm{B} = e(g_1,g_2)^b \\
\mu = u + r\cdot c & \mathrm{U} = g_1^u & \alpha = g_2^{a+t\cdot f} \\
\mathrm{A} = e(\mathrm{R},g_2)^a & d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1) & \theta = g_2^{s+r\cdot t}\cdot\mathsf{sk}_2^d
\end{array}$$

It outputs $\pi = (\mathrm{R},\mathrm{U},\mu,\mathrm{S},\mathrm{A},\mathrm{B},\alpha,\beta,\theta)$*.*

$\mathsf{Ver}_1(\mathsf{pk}_1,\pi_1)$**:** *Parses* $\pi_1 = (\mathrm{R},\mathrm{U},z,\mu)$*, computes* $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$ *and* $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$*. If* $g_1^{\mu} = \mathrm{U}\cdot\mathrm{R}^c$ *and* $g_1^z = \mathrm{R}\cdot\mathsf{pk}_1^d$ *then it outputs* 1*, else* 0*.*

$\mathsf{Ver}_2(\mathsf{pk}_2,\pi_2)$**:** *Computes the values* $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$*,* $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$ *and* $f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$*. If the following equations hold, then it outputs* 1*, else* 0*:*

$$\begin{array}{lll}
g_1^{\mu} = \mathrm{U}\cdot\mathrm{R}^c & e(\mathrm{R},\alpha)\cdot e(g_1,\beta) = \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{S}^f & e(g_1,\theta) = \mathrm{S}\cdot\mathsf{pk}^d
\end{array}$$

$\mathsf{RPro}(\mathsf{rk},\pi_1)$**:** *Parses* $\pi_1 = (\mathrm{R},\mathrm{U},z,\mu)$*, this algorithm picks* s, a *and* b *in the uniform distribution on* $\mathbb{Z}_p^*$ *and computes:*

$$\begin{array}{lll}
\mathrm{S} = e(\mathrm{R},\mathsf{rk})\cdot e(g_1,g_2)^s & f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S}) & \beta = g_2^{b+s\cdot f} \\
\mathrm{A} = e(\mathrm{R},g_2)^a & \alpha = g_2^a\cdot\mathsf{rk}^f & \theta = g_2^s\cdot\mathsf{rk}^z \\
\mathrm{B} = e(g_1,g_2)^b & &
\end{array}$$

It outputs $\pi_2 = (\mathrm{R},\mathrm{U},\mu,\mathrm{S},\mathrm{A},\mathrm{B},\alpha,\beta,\theta)$*.*

In the following, we prove that NURP is unidirectional, complete, sound, (non-adaptively) valid, zero-knowledge and secret secure.

Lemma 25 NURP *is complete.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of NURP $= (\mathsf{Set},\mathsf{Gen}_1,\mathsf{Gen}_2,$ $\mathsf{RGen},\mathsf{Pro}_1,\mathsf{Pro}_2,\mathsf{Ver}_1,\mathsf{Ver}_2,\mathsf{RPro})$. We remark that I_1 is defined as Schnorr+ (Definition 47) and I_2 is defined as FAPIright (Definition 50), we deduce that I_1 (resp. I_2) are complete (Theorem 15 and Theorem 18). Let k be an integer and sk_1, pk_1, pk_2 and rk be four keys generated as follows:

- $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k)$

- $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}_1') \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$

- $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}_2') \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$

- $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{pk}_1,\mathsf{pk}_1',\mathsf{sk}_2,\mathsf{sk}_2')$

We deduce that $\mathsf{pk}_1 = g_1^{\mathsf{sk}_1}$, $\mathsf{pk}_2 = e(g_1,\mathsf{sk}_2$, $\mathsf{sk}_2 = g_2^{\mathsf{sk}_2'}$ and $\mathsf{rk} = g_2^{\mathsf{sk}_2'/\mathsf{sk}_1}$. Let the two proofs π_1 and π_2 be generated by $\pi_1 \leftarrow \mathsf{Pro}_1(\mathsf{sk}_1,\mathsf{pk}_1)$ and $\pi_2 \leftarrow \mathsf{RPro}(\mathsf{rk},\pi_1)$. We show that for any $b \leftarrow$

$\mathsf{Ver}_2(\mathsf{pk}_2,\pi_2)$. We set $\pi = (\mathrm{R},\mathrm{U},\mu,\mathrm{S},\mathrm{A},\mathrm{B},\alpha,\beta,\theta)$, we observe that, for $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$, $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$ and $f = \mathrm{H}(\mathrm{R}||\mathrm{U}||\mathrm{A}||\mathrm{B}||\mathrm{S})$:

$$\begin{aligned}
g_1^{\mu} &= g_1^{u+r\cdot c} = g_1^{u}\cdot g_1^{r\cdot c} = \mathrm{U}\cdot\mathrm{R}^{c}\\
e(\mathrm{R},\alpha)\cdot e(g_1,\beta) &= e(\mathrm{R},g_2^{a}\cdot\mathsf{rk}^{f})\cdot e(g_1,g_2^{b+s\cdot f}) = e(\mathrm{R},g_2^{a})\cdot e(\mathrm{R},\mathsf{rk})^{f}\cdot e(g_1,g_2)^{b}\cdot e(g_1,g_2^{s})^{f}\\
&= \mathrm{A}\cdot\mathrm{B}\cdot\left(e(\mathrm{R},\mathsf{rk})\cdot e(g_1,g_2)^{s}\right)^{f} = \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{S}^{f}\\
e(g_1,\theta) &= e(g_1,g_2^{s}\cdot\mathsf{rk}^{z}) = e(g_1,g_2^{s}\cdot\mathsf{rk}^{r+\mathsf{sk}_1\cdot d}) = e(g_1,g_2^{s})\cdot e(g_1,\mathsf{rk}^{r})\cdot e(g_1,\mathsf{rk}^{\mathsf{sk}_1\cdot d})\\
&= \left(e(g_1^{r},\mathsf{rk})\cdot e(g_1,g_2)^{s}\right)\cdot e(g_1,g_2^{\frac{\mathsf{sk}_2'}{\mathsf{sk}_1}\cdot\mathsf{sk}_1})^{d} = \left(e(\mathrm{R},\mathsf{rk})\cdot e(g_1,g_2)^{s}\right)\cdot e(g_1,g_2^{\mathsf{sk}_2'})^{d}\\
&= \mathrm{S}\cdot\mathsf{pk}_2^{d}
\end{aligned}$$

We deduce that $b = 1$, which concludes the proof. □

Lemma 26 NURP *is sound and (non-adaptively) valid.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of NURP = (Set, Gen$_1$, Gen$_2$, RGen, Pro$_1$, Pro$_2$, Ver$_1$, Ver$_2$, RPro). We remark that I_1 is defined as Schnorr+ (Definition 47) and I_2 is defined as FAPIright (Definition 50), we deduce that I_1 (resp. I_2) are sound and (non-adaptively) valid (Theorem 15 and Theorem 18), which concludes the proof. □

Lemma 27 NURP *is zero-knowledge.*

Proof: I_1 (resp. I_2) denotes the delegate (resp. delegator) proof system of NURP = (Set, Gen$_1$, Gen$_2$, RGen, Pro$_1$, Pro$_2$, Ver$_1$, Ver$_2$, RPro). We remark that I_1 is defined as Schnorr+ (Definition 47) and I_2 is defined as FAPIright (Definition 50), we deduce that I_1 (resp. I_2) are zero-knowledge (Theorem 15 and Theorem 18).

We show how to build a polynomial time algorithm Sim_1 such that for any $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}_1') \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$, any $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}_2') \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$, any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2)$, the algorithms $\mathsf{Sim}_1(\mathsf{pk}_1,\mathsf{pk}_2)$ and the two algorithms $\alpha_1 \leftarrow \mathsf{Pro}_1(\mathsf{sk}_1,\mathsf{pk}_1)$ and $\alpha_2 \leftarrow \mathsf{RPro}(\mathsf{rk},\alpha_1)$ follow the same probability distribution. We build $\mathsf{Sim}_1(\mathsf{pk}_1,\mathsf{pk}_2)$ as follows: this simulator picks $z,c,d,f,\mu,\alpha,\beta,\theta \xleftarrow{\$} \mathbb{Z}_p^{*}$ and $\mathrm{A} \xleftarrow{\$} \mathbb{G}_{\mathrm{T}}$. It computes:

$$\mathrm{R} = \frac{g_1^{z}}{\mathsf{pk}_1^{d}} \qquad \mathrm{S} = \frac{e(g_1,\theta)}{\mathsf{pk}_2^{d}} \qquad \mathrm{U} = \frac{g_1^{\mu}}{\mathrm{R}^{c}} \qquad \mathrm{B} = \frac{e(\mathrm{R},\alpha)\cdot e(g_1,\beta)}{\mathrm{A}\cdot\mathrm{S}^{f}}$$

It outputs $((\mathrm{R},\mathrm{U},z,\mu),(\mathrm{R},\mathrm{U},\mu,\mathrm{S},\mathrm{A},\mathrm{B},,\alpha,\beta,\theta))$, which are valid proofs because:

$$\begin{aligned}
g_1^{z} &= \mathrm{R}\cdot\mathsf{pk}_1^{d} &\qquad e(\mathrm{R},\alpha)\cdot e(g_1,\beta) &= \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{S}^{f}\\
g_1^{\mu} &= \mathrm{U}\cdot\mathrm{R}^{c} &\qquad e(g_1,\theta) &= \mathrm{S}\cdot\mathsf{pk}^{d}
\end{aligned}$$

Moreover, since μ,α,β and θ come from uniform distributions, then the outputs of Sim_1 follow the same distribution as the real algorithms.

We show how to build a polynomial time algorithm Sim_2 such that for any $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}_1') \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$, any $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}_2') \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$, and any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2)$, the algorithms $\mathsf{Sim}_2(\mathsf{rk},\mathsf{pk}_1,\mathsf{pk}_2)$ and $\mathsf{Pro}_1(\mathsf{sk}_1,\mathsf{pk}_1)$ follow the same probability distribution. Since I_1 is zero-knowledge, there exists a simulator Sim such that the outputs of $\mathsf{Pro}_1(\mathsf{sk}_1,\mathsf{pk}_1)$ follow the same probability distribution as the outputs of $\mathsf{Sim}(\mathsf{pk}_1)$. We build $\mathsf{Sim}_2(\mathsf{rk},\mathsf{pk}_1,\mathsf{pk}_2)$ as follows: it runs $\pi_1 \leftarrow \mathsf{Sim}(\mathsf{pk}_1)$ and returns π_1. Clearly, Sim_2 follows the same distribution as the real algorithm Pro.

We show how to build a polynomial time algorithm Sim_3 such that for any $(\mathsf{set},\mathscr{R}_1,\mathscr{R}_2) \leftarrow \mathsf{Set}(k)$, any $(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{pk}_1') \leftarrow \mathsf{Gen}_1(\mathsf{set},\mathscr{R}_1)$, any $(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}_2') \leftarrow \mathsf{Gen}_2(\mathsf{set},\mathscr{R}_2)$, any re-proof key $\mathsf{rk} \leftarrow \mathsf{RGen}(\mathsf{sk}_1,\mathsf{sk}_2)$, any polynomial time algorithm $\mathscr{A} \in \textsc{poly}(k)$ and any $\pi_1 \leftarrow \mathscr{A}(\mathsf{sk}_1,\mathsf{pk}_1,\mathsf{pk}_2)$

where $\pi_1 = (\mathrm{R}, \mathrm{U}, z, \mu)$, the algorithms $\mathsf{Sim}_3(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ and $\mathsf{RPro}(\mathsf{rk}, \pi_1)$ follow the same probability distribution. We build $\mathsf{Sim}_3(\mathsf{sk}_1, \mathsf{pk}_1, \mathsf{pk}_2)$ as follows: this simulator sets $c = \mathrm{H}(\mathrm{R}||\mathrm{U}||0)$ and $d = \mathrm{H}(\mathrm{R}||\mathrm{U}||1)$, and picks $f, \alpha, \beta, \theta \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ and $\mathrm{A} \overset{\$}{\leftarrow} \mathbb{G}_\mathrm{T}$. It computes:

$$\mathrm{S} = \frac{e(g_1,\theta)}{\mathsf{pk}_2^d} \qquad\qquad \mathrm{B} = \frac{e(\mathrm{R},\alpha)\cdot e(g_1,\beta)}{\mathrm{A}\cdot \mathrm{S}^f}$$

It outputs $\pi_2 = (\mathrm{R}, \mathrm{U}, c, d, \mu, z, \mathrm{S}, \mathrm{A}, \mathrm{B}, f, \alpha, \beta, \theta)$, which is a valid proof because:

$$g_1^\mu = \mathrm{U}\cdot\mathrm{R}^c \qquad\qquad e(\mathrm{R},\alpha)\cdot e(g_1,\beta) = \mathrm{A}\cdot\mathrm{B}\cdot\mathrm{S}^f \qquad\qquad e(g_1,\theta) = \mathrm{S}\cdot\mathsf{pk}^d$$

Moreover, since μ, α, β and θ come from uniform distributions, then the outputs of Sim_2 follow the same distribution as the real algorithms RPro. □

Theorem 19 NURP *is unidirectional, complete, sound, (non-adaptively) valid, zero-knowledge and secret secure under the DL assumption.*

Proof: NURP is unidirectional by construction. We showed that NBRP is complete (Lemma 25), sound, (non-adaptively) valid (Lemma 26), and zero-knowledge (Lemma 27). Since the setup and the key generation algorithms $\mathsf{Set}, \mathsf{Gen}_1, \mathsf{Gen}_2$ and RGen are defined as in IURP (Definition 42), then we can proof that NURP is secret secure as we proove that IURP is secret secure in Theorem 14. □

3.4 Schemes comparison

Table 3.1 compares the properties and the efficiency of the insecure scheme of Blaze *et al.* [BBS98] and our four schemes. The first columns give the properties of each scheme, namely *unidirectional, non-interactive* and *pairing free*. The last columns give the computation cost for each entities: the delegate P_1, the delegator P_2, and the respective verifiers V_1 and V_2. The last column gives the computation cost for the proxy. We only evaluate the number of exponentiations and pairing computations. We denote by e (resp. p) the computation time of an exponentiation (resp. a pairing computation). All our schemes can be used for the applications that we propose in Introduction. However, each of them allows a different compromise between security and efficiency.

Security: Bidirectional schemes require mutual confidence between Alice and Bob. Moreover, if the proxy and the delegator collude then they can deduce the secret key of the delegate. In a practical scenario, the delegate could refuse to reveal his secret key to the trusted re-proof key manager. In unidirectional schemes, the delegator computes the re-proof key with the public key of the delegate, then the secret key of the delegate is protected.

Efficency: The efficiency is evaluated by two different properties: the number of interactions and the number of pairing computations. Thus, our two non-interactive schemes are optimal for the number of interactions (only one interaction), and our two bidirectional schemes are pairing free.

Scheme	Uni.	Non-in.	P. free	P_1	P_2	V_1	V_2	Proxy
Blaze *et al.* [BBS98]	no	no	**yes**	$1e$	$1e$	$1e$	$1e$	0
IBRP	no	no	**yes**	$1e$	$1e$	$2e$	$2e$	$4e$
IURP	**yes**	no	no	$1e$	$1p+3e$	$2e$	$1p+1e$	$2p+5e$
NBRP	no	**yes**	**yes**	$2e$	$5e$	$4e$	$7e$	$4e$
NURP	**yes**	**yes**	no	$2e$	$2p+9e$	$4e$	$3p+4e$	$3p+8e$

Table 3.1: Comparison of our proxy re-proof schemes.

3.5 Conclusion

In this chapter, we give a formal treatment for proxies re-proof. Particularly, we define bidirectional/unidirectional and interactive/non-interactive schemes. We design four schemes with different properites:

- BIRP, a bidirectional interactive scheme.
- UIRP, the first unidirectional interactive scheme.
- BNRP, the first bidirectional non-interactive scheme.
- UNRP, the first unidirectional non-interactive scheme.

Future works will concern the design of multi-hop proxies re-proof: a scheme is said to be multi-hop when several different proxies transform the same proof successively. For example, a first proxy transforms Alice's proof into Bob's proof, and a second proxy re-transforms this proof into a Carol's proof. Our bidirectional interactive scheme BIRP is multi-hop by construction. However, designing a multi-hop unidirectional proxy re-proof, or a multi-hop non-interactive proxy re-proof, seems to be non-trivial, and is an interessant open problem.

Also it should be interesting to investigate the case of proxies re-proof using proofs systems for other langages, such as some NP-complete langage, or some other well know cryptographic assumption.

Finally, we let as open problems the design of non-interactive proxy re-proof protocols that are adaptively valid and in the standard model.

Chapter 4

Verifiable Private Polynomial Evaluation

Contents

Delegating the computation of a polynomial to a server in a verifiable way is challenging. An even more challenging problem is ensuring that this polynomial remains hidden to clients who are able to query such a server. In this chapter, we formally define the notion of *Private Polynomial Evaluation* (PPE). Our main contribution is to design a rigorous security model along with relations between the different security properties for this primitive. We define *polynomial protection* (PP), *proof unforgeability* (UNF), and *indistinguishability against chosen function attack* (IND-CFA), which formalizes the resistance of a private polynomial evaluation scheme against attackers trying to guess which polynomial is used among two polynomials of their choice. As a second contribution, we give a cryptanalysis of two private polynomial evaluation schemes of the literature. Finally, we design a private polynomial evaluation scheme called PIPE and we prove that it is PP, UNF and IND-CFA secure under the decisional Diffie-Hellman assumption in the random oracle model. This work has been conducted in collaboration with Manik Lal Das, Hardik Gajera, David Gérault, Matthieu Giraud and Pascal Lafourcade and has been published in the paper *"Verifiable Private Polynomial Evaluation"* at the ProvSec 2017 conference.

4.1 Introduction

Private Polynomial Evaluation (PPE), introduced by Kate *et al.* in [KZG10], is a cryptographic primitive where a server evalutes some points of a secret polynomial f for a client. The client does not know the polynomial, but he must be convinced that the points (x, y) computed by the server are valid, *i.e.*, $f(x) = y$. In order to achieve this property, the primitive provides a mechanism that allows the server to prove the validity of a point, according to a *verifiable key* generated from the polynomial. Such a scheme is secure when the polynomial remains secret for the client, and when the server cannot prove that a point is valid when it is not.

4.1.1 Functionalities

A Private Polynomial Evaluation (PPE) scheme is composed of the four following algorithms:

System initialization: The algorithm Set initializes the setup of the scheme. The algorithm Gen is run by the company and returns a verification key pk and a server secret key sk from a chosen polynomial f.

Computation: The algorithm Compute is run by the server. It takes as input a value x chosen by the client and the secret key sk, and returns a value y together with a proof π that $y = f(x)$.

Verification: The algorithm Verify is run by the client. It decides whether a proof π is valid according to the verification key pk.

Figure 4.1 illustrates the interaction between the different entities that use a private polynomial evaluation scheme. The company computes the verification key pk and the server secret key sk from the function f using the algorithm Gen, then he sends f and sk to the server and pk to the client. In order to evaluate the point $(x, f(x))$, the client sends x to the server. The server uses the algorithm Compute that outputs $f(x)$ together with the proof π. It sends these two values to the client, who uses the verification algorithm Verify to verify the proof π.

4.1.2 Security Goals

A private polynomial evaluation scheme requires two security properties:

Unforgeability of the proof: The server is not able to convince the client that a point (x, y) is valid according to the polynomial f if $f(x) \neq y$.

Polynomial Protection: The client has no information about the polynomial f except the points evaluated by the server. In their security model [KZG10], Kate *et al.* consider only the attacks where the secret polynomial is randomly chosen. In this model, the scheme is secure when the client is not able to guess the secret polynomial with non-negligible probability.

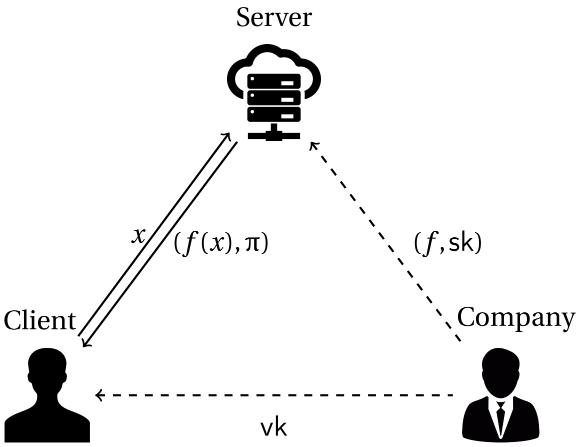


Figure 4.1: Illustration of a private polynomial evaluation scheme.

However, this definition of polynomial protection is too weak for the application we consider in this chapter. Moreover, the two other schemes of the literature [GFLL15, GND16] consider applications where the secret polynomial has small coefficients and degree, so the set of possible secret polynomials is very small. In a realistic scenario, the client knows this set. Thus, the client has non-negligible probability to find the secret polynomial by picking randomly a polynomial in this set. In this chapter, we consider a stronger notion of polynomial protection. We consider a scenario where the client tries to distinguish what is the polynomial used by the server among two polynomials of his choice. The scheme is secure when the client has no significant advantage comparing to an algorithm that tries to guess this polynomial at random. This kind of attack is called *Indistinguishability against Chosen Function Attack* (IND-CFA). Note that in such an attack, we have the following inherent restriction: the client cannot evaluate the points that are different between the two polynomials. Indeed, if the client evaluates a point that is different between the two polynomials, then he can easily deduce what is the polynomial used by the server.

4.1.3 Applications

In their papers, Kate *et al.* [KZG10] and Guo *et al.* [GFLL15] give several applications of private polynomial evaluations. We recall them in this section. Moreover, we show that the security model of Kate *et al.* does not provide a sufficient security level for some applications. Finally, we give a new application that requires IND-CFA security.

Verifiable secret sharing. Kate *et al.* show that private polynomial evaluation schemes can be used as verifiable secret sharing schemes [KZG10]. In a secret sharing scheme, a dealer shares a secret into n shares, such that a threshold number l of shares allows the users to recover the secret. In verifiable secret sharing schemes, the dealer commits the secret such that the users can check whether a share is valid or not. The polynomial of a private polynomial evaluation scheme can be viewed as the secret of a verifiable secret sharing scheme, where each share is a point of the polynomial, and where users can verify that the points are valid according to the secret polynomial using the verification key. Moreover, for a polynomial of degree l, a user that receives more than l points is able to recover the secret polynomial using Lagrange's interpolation.

Zero-knowledge elementary database. In this primitive, introduced by Micali *et al.* in [MRK03], a server sends some data to a client together with a proof that this data comes from a given database such that no information about the other data is leaked. Kate *et al.* show that private polynomial evaluation schemes can be used as zero-knowledge elementary database schemes. All couples (x, y) of index/data in the database are used to build the secret polynomial using Lagrange's interpolation, and a trusted party generates the verification key from this polynomial. Then the database proves that couples (x, y) of index/data it sends are points of the polynomials according to this verification key.

In practice, the database is not randomly generated, and clients should have some information about the database, so this application requires a security model where the polynomial is not randomly chosen. Hence the model of Kate *et al.* is not sufficient for zero-knowledge elementary databases. This application requires the indistinguishability against chosen function attack.

Heals monitoring. In [GFLL15], Guo *et al.* propose an application where a server receives medical data collected by sensors worn by the clients, and provides the clients with an evaluation of their health status. More precisely, the company defines a polynomial f which returns meaningful information, such as potential diseases. Then, it uploads this polynomial to the server, and sells to the end clients the ability to query that function with their own medical data. The function that evaluates the health status of the clients is secret, and the company does not want the clients to be able to recover it.

Note that a scheme that is not secure in the chosen function attack model cannot be used for this application. As it is mentioned in the previous section, Guo *et al.* consider that the function that evaluates the health status is a polynomial with small coefficients and degree, so the set of possible secret polynomials is very small, which does not fit with the model of Kate *et al.* where the polynomial is randomly chosen in a very large set.

Evaluation of secret prediction function. Mathematical models are powerful tools that are used to make predictions about a system's behavior. The idea is to collect a large set of data for a period of time and use it to build a function predicting the evolution of the system in the future. This topic has many applications, for instance, meteorology or economics. It can be used to predict the weather or the behavior of stock exchange.

Consider a company that collects and stores a very large set of data, for example about the state of the soil, such as humidity, acidity, temperature and mineral content. Using it, it computes some function that predicts the state of the soil for next years. The clients are farmers who want to anticipate the state of the soil during the sowing periods to determine how much seeds to buy and when to plant them. The company gives its client access to the prediction function through a cloud server. A paying client can then interact with the server to evaluate the function on his own data. For economic reasons, the company does not want that the clients are able to recover the prediction function. Moreover, the clients do not trust the server: it might be corrupted to produce incorrect results. Hence, the server should provide a proof that its output is correct with regards to the secret prediction function.

The private polynomial evaluation primitive solves this problem because it ensures that: (i) the polynomial f is protected as much as possible, and (ii) the client is able to verify the result given by the server.

Consider a company using a private polynomial evaluation scheme for prediction functions. An attacker wants to guess which prediction function is used by the company. Assume this attacker gains access to some data used to build the prediction function, for instance by corrupting a technician. Thus, the attacker can build several prediction functions by using different mathematical models and the collected data, and try to distinguish which of these functions is used by the company. Intuitively, in a secure private polynomial evaluation scheme, this task should be as hard as if the server only returns $f(x)$, and no additional information for verification. Hence, this application requires indistinguishability against chosen function attack security.

4.1.4 Contributions

- We give a cryptanalysis of two private polynomial evaluation schemes, the first one presented by Guo *et al.* [GFLL15] and the second one presented by Gajera *et al.* [GND16]. Our attack allows an adversary to recover the secret polynomial in a single query.

- We provide a formal definition and security framework for private polynomial evaluation schemes. We define two one-way notions, *Weak Polynomial Protection* (WPP) and *Polynomial Protection* (PP), stating that a client limited to l queries cannot recover the polynomial, where l is the degree of the polynomial. Additionally, we define the IND-CFA security which formalizes the idea that no adversary can guess which of two polynomials of his choice is used. In essence, the proof of a correct computation should not reveal any information about the polynomial. We finally study the relations between these notions.

- We prove that a private polynomial evaluation scheme proposed by Kate *et al.* [KZG10] is secure in our model (*i.e.,* it is IND-CFA secure).

- We design PIPE (for Private IND-CFA Polynomial Evaluation), an efficient IND-CFA secure private polynomial evaluation scheme. This scheme combines the Verifiable Secret Sharing introduced by Feldman [Fel87] and the ElGamal encryption scheme (definition 25) in order to achieve verifiability and IND-CFA security. We also formally prove its security under the DDH assumption in the random oracle model. Finally we compare PIPE with the scheme of Kate *et al.* [KZG10].

4.1.5 Related Works

Verifiable Computation (VC) refers to the cryptographic primitives where an untrusted server can prove the correctness of its output. It was introduced in [GGP10]. The aim of such a primitive is to allow a client with limited computational power to delegate difficult computations. Primitives where everyone can check the correctness of the computation are said to be *publicly verifiable* [PRV12]. This subject has led to a dense literature [PST13, CRR12, FG12, CKKC13, PHGR13]. In 2012, Canetti *et al.* [CRR12] proposed formal security models for VC. Fiore and Gennaro [FG12] propose a scheme for polynomial evaluations and matrix computations. Unlike ours, these works consider that the polynomial used by the server is public.

To the best of our knowledge, only four papers study how to hide the function used by the server [GFLL15, GND16, KZG10, NP99].

Kate *et al.* define a primitive called *commitment to polynomials* (CTP) [KZG10]. In this primitive, a user commits to a hidden polynomial f and reveals some points (x, y) together with a proof that $f(x) = y$. The user can open the commitment *a posteriori* to reveal the polynomial. CTP is close to PPE: the verification key in a PPE scheme can be viewed as a commitment in a CTP scheme, the main difference is that this verification key is computed by a trusted party (the company) and the points are evaluated by an untrusted party (the server). The authors formalize the hardness of guessing the polynomial knowing less than l points. In this model, the polynomial is randomly chosen. Hence they do not consider the case where the adversary tries to distinguish the committed polynomial between two chosen polynomials, as in our IND-CFA model. Moreover, Kate *et al.* design two CTP schemes in [KZG10]. The first one is not IND-CFA since the commitment algorithm is deterministic. We prove that the second scheme is IND-CFA secure in this chapter. Moreover, we show that our scheme PIPE can be used as a CTP scheme, and we compare it to the scheme of Kate *et al.*. We show that our scheme solves an open problem described by Kate *et al.*: designing a scheme that is secure under a weaker assumption than the t-SDH assumption (Definition 13).

Independently of Kate *et al.*, Guo *et al.* [GFLL15] propose a scheme with similar security properties to delegate the computation of a secret health related function on the users' health record. The polynomials are explicitly assumed to have low coefficients and degree, which greatly reduces their randomness. However, the authors give neither security models nor proof. Later, Gajera *et al.* [GND16] show that any user can guess the polynomial using the Lagrange's interpolation on several points. They propose a scheme where the degree l is hidden and claim that it does not suffer from this kind of attack. We show that hiding the degree l is useless and that no scheme can be secure when a user queries more than l points to the server. Moreover, we give a cryptanalysis on these both schemes which requires only one query to the server.

Finally, there has been lots of work done on a similar but slightly different topic, Oblivious Polynomial Evaluation (OPE), introduced by Naor and Pinkas [NP99]. In OPE, there are two parties. One party A holds a polynomial f and another party B holds an element x. The aim of OPE is that the party B receives $f(x)$ in such a way that A learns nothing about x and B learns nothing about f, except $f(x)$. Researchers have studied OPE extensively and shown that it can be used to solve various cryptographic problems, such as set membership, oblivious keyword search, data entanglement, set-intersection and more [FIPR05, FNP04, LP02]. Despite the similarities between OPE and PPE, they are different in nature. In particular, OPE does not consider the verifiability of $f(x)$, whereas it is a crucial point in PPE. Additionally, in a PPE, the requirement that the server does not learn anything about x is relaxed. Since OPE does not consider verifying computation, we feel that it is not fair to compare the performances.

4.2 Cryptanalysis of [GFLL15] and [GND16]

We start by presenting an inherent limitation of PPE schemes, then we explain how to break both schemes presented by Guo *et al.* [GFLL15] and by Gajera *et al.* [GND16].

4.2.1 Inherent Limitation of Private Polynomial Evaluation

In the private polynomial evaluation scheme of Guo *et al.* [GFLL15], the degree l of the polynomial is public. Gajera *et al.* [GND16] use this property to mount the following attack: a client queries $l+1$ points to guess the polynomial using Lagrange's interpolation. To fix this weakness, they propose a scheme where l is secret. However, we show that any client can guess l and f after $l+1$ interactions with the server. To do so, the attacker chooses an input x_0 and sends it to the server. He receives y_0 and computes the polynomial f_0 of degree 0 using Lagrange's interpolation on (x_0, y_0). Next, the attacker chooses a second and a different input x_1 and asks $y_1 = f(x_1)$ to the server. He computes the polynomial f_1 of degree 1 using Lagrange's interpolation on $\{(x_0, y_0), (x_1, y_1)\}$. By repeating this process until the interpolation gives the same polynomial $f_i = f_{i+1}$ for two consecutive iterations, it recovers the degree and the polynomial. Actualy this problem is an inherent limitation of PPE schemes and was already considered in the security model of Kate *et al.* [KZG10]. Thus, to preserve the protection of the polynomial, the server must refuse to evaluate more than l points for each client and we must assume that clients do not collude to collect more than l points. This limitiation implies that the the degree of the polynomial must be quite high in practice.

4.2.2 Cryptanalysis of [GFLL15] and [GND16]

We show how the client can recover the secret polynomial in a single query in both verifiable private polynomial evaluation schemes [GFLL15] and [GND16]. In addition to the protection of f, the scheme [GFLL15] requires that the client's data x is encrypted. The server evaluates the function on the encrypted data, and proves the validity of the result, but it learns nothing about it. More formally, the client uses an encryption algorithm Enc to compute $x' = \mathsf{Enc}(x)$ and sends it to the server which returns y'. Then, the client computes $y = \mathsf{Dec}(y')$ such that $y = f(x)$ where f is the secret polynomial. The encryption scheme is based on the discrete logarithm assumption. The decryption algorithm works in two steps: first the client computes a value h such that $h = g^{f(x)}$ where g is a generator of a multiplicative group of large prime order p, next he computes the discrete logarithm of h in base g using *Pollard's lambda method* [Pol78]. The authors assume that the size of $f(x)$ is reasonable: more formally, they define a set of possible inputs $\mathcal{X}$ and an integer $\mathrm{M} \in \mathbb{N}$ such that $\forall x \in \mathcal{X}, 0 \leq f(x) < \mathrm{M}$. The authors assume that the clients are able to efficiently execute Pollard's lambda algorithm on any $h = g^y$ where $y < \mathrm{M}$. Actually, for practical reasons, since $h = g^{f(x) \mod p}$ and $\log_g(h) = f(x)$, we assume that $0 \leq f(x) < p$ for any input x of reasonable size, *i.e.*, $x \ll p$. Hence, the authors of [GFLL15] consider f as a positive polynomial in $\mathbb{Z}$ with sufficiently small coefficients.

It is easy to evaluate a small M' such that $\mathrm{M}' > \mathrm{M}$ by choosing M' such that Pollard's lambda algorithm on $g^{\mathrm{M}'}$ is computable by a powerful server but is too slow for a practical application. For example, if Pollard's lambda algorithm takes less than one minute for the server but several hours for the client's computer, we can assume that $\mathrm{M}' > \mathrm{M}$, because if $\mathrm{M} > \mathrm{M}'$, then the decryption algorithm is too slow for the client for some $x \in \mathscr{X}$, so the scheme is not practical. On the other hand, any attack that is polynomial in M' is practical using the powerful server. To sum up, our attacker has the following tools:

- $\mathrm{M}' \in \mathbb{N}$ such that $\forall x \in \mathscr{X}$, $0 \leq f(x) < \mathrm{M}'$ and such that algorithms that require $t(\mathrm{M}')$ operations (where t is a polynomial) are *easily computable*.
- A server which returns $y = f(x)$ for any input x. This server can be used at most l times where l is the degree of the polynomial.

Finally, note that the authors assume that $0 \leq f(x)$ for any x and that $\mathscr{X} \subset \mathbb{N}$. We show that any client can guess the secret polynomial during his first interaction with the server.

We first show some useful properties starting with the factor theorem.

Theorem 20 (Factor theorem) *Let $f \in \mathbb{Z}[\mathrm{X}]$ be a polynomial. $f(x)$ has a factor $(x - \alpha)$ if and only if $f(\alpha) = 0$.*

Using this theorem we prove the following two properties:

Property 2 *For any polynomial $f \in \mathbb{Z}[\mathrm{X}]$ and any integers x and y, there exists $\mathrm{P} \in \mathbb{Z}$ such that:*

$$f(x+y) = f(x) + y \cdot \mathrm{P}$$

Proof: For any polynomial $f \in \mathbb{Z}[\mathrm{X}]$ and any integer x, the polynomial $g(y) = f(x+y) - f(x)$ has a root at 0. By the Factor Theorem (Theorem 20) $y - 0$ divides $g(y)$, so y divides $f(x+y) - f(x)$. Hence, there exists $\mathrm{P} \in \mathbb{Z}$ such that $f(x+y) - f(x) = y \cdot \mathrm{P}$, *i.e.*, $f(x+y) = f(x) + y \cdot \mathrm{P}$. □

Note that for any positive integers a and b such that $a < b$, we have $a \mod b = a$. Then, we can deduce the following property from Property 2.

Property 3 *For any polynomial $f \in \mathbb{Z}[\mathrm{X}]$ and any integers x and y such that $0 \leq f(x) < y$ and $0 \leq f(x+y)$, we have:*

$$f(x+y) \mod y = f(x)$$

Proof: From the previous property, we have $f(x+y) = f(x) + y \cdot \mathrm{P}$, where P is an integer. Assume $\mathrm{P} < 0$, we define $\mathrm{P}' = -\mathrm{P} > 0$, then $f(x+y) = f(x) - y \cdot \mathrm{P}' \geq 0$. Hence we have $f(x) \geq y \cdot \mathrm{P}' > f(x) \cdot \mathrm{P}'$.

- If $f(x) > 0$ then we deduce $f(x)/f(x) > \mathrm{P}'$, so $1 > \mathrm{P}'$.
- If $f(x) = 0$ then $0 \geq y \cdot \mathrm{P}' > 0$.

In both cases, we obtain a contradiction. We conclude that $\mathrm{P} \geq 0$. Finally, we deduce $f(x+y) \mod y = f(x) + y \cdot \mathrm{P} \mod y = f(x)$. □

Our attack on [GFLL15] works as follows. The attacker chooses a vector of l integers $(x_1, \ldots, x_l)$ such that, for all $0 < i \leq l$:

$$\sum_{j=1}^{i} x_j \in \mathscr{X}$$

For all $1 \leq i \leq l$, we set:

$$x'_i = \sum_{j=1}^{i} x_j$$

For the sake of clarity, we first show the attack in the particular case where $\{1, \ldots, l\} \subset \mathscr{X}$, then we show the generalized case for any set $\mathscr{X}$ where $|\mathscr{X}| \geq l$.

Simple Case ($\{1,\dots,l\} \subset \mathscr{X}$)

The attacker chooses the vector $(x_1, x_2, \dots, x_l) = (1, 1, \dots, 1)$ and sends $x = l + \mathrm{M}'$ to the server that returns the encryption of $y = f(x)$. Pollard's lambda algorithm complexity [Pol78] on M' is $\mathrm{O}(\mathrm{M}'^{1/2})$. We consider that $l \ll \mathrm{M}'$ (for instance $l \approx 10$ as in [GFLL15]), thus $x < 2 \cdot \mathrm{M}'$, the complexity of the decryption with Pollard's lambda algorithm is $\mathrm{O}(f(2\mathrm{M}')^{1/2}) \approx \mathrm{O}(\mathrm{M}'^{l/2})$. For all $1 \le i \le l$, the attacker computes:

$$\mathrm{M}'_i = l - i + \mathrm{M}'$$
$$y_i = y \mod \mathrm{M}'_i$$

Since for all $a \in \mathscr{X}, \mathrm{M}' > f(a)$, then we have for all $1 \le i \le l$:

$$\mathrm{M}'_i = l - i + \mathrm{M}' \ge \mathrm{M}' > f(a)$$

Using Property 3 and since $i \in \mathscr{X}$, we deduce that:

$$\begin{aligned} y_i &= f(x) \mod \mathrm{M}'_i \\ &= f(l + \mathrm{M}') \mod \mathrm{M}'_i \\ &= f(l - i + i + \mathrm{M}') \mod \mathrm{M}'_i \\ &= f\left(i + \mathrm{M}'_i\right) \mod \mathrm{M}'_i = f(i) \end{aligned}$$

Hence, the attacker obtains $l+1$ points from one single queried point and uses Lagrange's interpolation on $((1, y_1), (2, y_2), \dots, (l, y_l), (x, y))$ to guess f. Then, the attacker can compute f with reasonable computation time.

General Case

In the following we present the attack for any set $\mathscr{X}$ where $|\mathscr{X}| \ge l$. The attacker chooses a vector of l integers $(x_1, \dots, x_l)$ such that, for all $1 \le i \le l, x_i > 0$ and:

$$\sum_{j=1}^{i} x_j \in \mathscr{X}$$

For all $1 \le i \le l$, we set:

$$x'_i = \sum_{j=1}^{i} x_j$$

Then the attacker sends the query:

$$x = \left(\sum_{j=1}^{l} x_j\right) + \mathrm{M}'$$

The attacker receives the encryption of $y = f(x)$.

As in the simple case, the complexity of the decryption with Pollard's lambda algorithm is $\mathrm{O}(f(2\mathrm{M}')^{1/2}) \approx \mathrm{O}(\mathrm{M}'^{l/2})$.

The attacker computes for all $1 \le i \le l$:

$$\mathrm{M}'_i = \left(\sum_{j=i+1}^{l} x_j\right) + \mathrm{M}'$$

For all $1 \le i \le l$ we set $y_i = y \mod \mathrm{M}'_i$. Since for all $a \in \mathscr{X}, \mathrm{M}' > f(a)$, we have for all $1 \le i \le l$ and for all $a \in \mathscr{X}$:

$$\mathrm{M}'_i = \left(\sum_{j=i+1}^{l} x_j\right) + \mathrm{M}' \ge \mathrm{M}' > f(a)$$

Since $x'_i \in \mathscr{X}$, we deduce, using Properties 2 and 3:

$$\begin{aligned}
y_i &= f(x) \mod \mathrm{M}'_i \\
&= f\left(\left(\sum_{j=1}^{l} x_j\right) + \mathrm{M}'\right) \mod \mathrm{M}'_i \\
&= f\left(\left(\sum_{j=i+1}^{l} x_j\right) + \left(\sum_{j=1}^{i} x_j\right) + \mathrm{M}'\right) \mod \mathrm{M}'_i \\
&= f\left(\left(\sum_{j=i+1}^{l} x_j\right) + x'_i + \mathrm{M}'\right) \mod \mathrm{M}'_i \\
&= f\left(x'_i + \mathrm{M}'_i\right) \mod \mathrm{M}'_i \\
&= f(x'_i)
\end{aligned}$$

It is possible to attack the scheme of Gajera *et al.* [GND16] in a similar way. Indeed, as in the scheme in [GFLL15], the client knows a value M such that $\forall x \in \mathscr{X}, f(x) < \mathrm{M}$. A simple countermeasure could be to not allow the client to evaluate inputs that are not in $\mathscr{X}$. Unfortunally, this is not possible in these two schemes since the client encrypts his data x. Hence, the server does not know whether $x \in \mathscr{X}$ or not.

4.3 Formal Definitions

We first formally define the private polynomial evaluation primitive.

4.3.1 Private Polynomial Evaluation

Definition 52 *A* Private Polynomial Evaluation *(PPE) scheme is a tuple of polynomial time algorithms* $(\mathsf{Set}, \mathsf{Gen}, \mathsf{Compute}, \mathsf{Verify})$ *defined as follows:*

$\mathsf{Set}(l,k)$**:** *It returns a ring* F *and a public setup* set.
$\mathsf{Gen}(f)$**:** *It returns a server key* sk *and a verification key* pk *according to the polynomial* $f \in \mathrm{F}[\mathrm{X}]$.
$\mathsf{Compute}(\mathsf{pk}, x, \mathsf{sk}, f)$**:** *It returns y and a proof* π *that* $y = f(x)$.
$\mathsf{Verify}(\mathsf{pk}, x, y, \pi)$**:** *It returns* 1 *if the proof* π *is "accepted", otherwise* 0.

Moreover, such a scheme must have the following property: for any positive integer l and any positive polynomial t, there exists $k_0 \in \mathbb{N}$ such that for any integer $k \geq k_0$ and any $(\mathsf{set}, \mathrm{F}) \leftarrow \mathsf{Set}(l,k)$, *we have* $|\mathrm{F}| \geq t(k)$. *A PPE is said to be* correct *when the following equation holds for any $l \in \mathbb{N}$ and $k \in \mathbb{N}$:*

$$\Pr\left[\begin{array}{c} (\mathsf{set},\mathrm{F}) \leftarrow \mathsf{Set}(l,k); f \xleftarrow{\$} \mathrm{F}[\mathrm{X}]_l; \\ (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(f); x \xleftarrow{\$} \mathrm{F}; \\ (y,\pi) \leftarrow \mathsf{Compute}(\mathsf{pk},x,\mathsf{sk},f); \\ b \leftarrow \mathsf{Verify}(\mathsf{pk},x,y,\pi); \end{array} : (b=1) \wedge (y = f(x))\right] = 1$$

We revisit the formal security model of PPE schemes for two main reasons: (i) Kate *et al.* [KZG10] propose a model where the secret polynomial is randomly chosen. However, they present several practical applications where the polynomial is not actually randomly chosen, and were some information can be infered easily from the context. Their models are clearly not sufficient for analysing the security of this kind of applications. (ii) The schemes presented by Guo *et al.* [GFLL15] and Gajera *et al.* [GND16] consider polynomials that are not randomly chosen. The authors give neither security models nor security proofs. We show previously a practical attack on these two schemes where a client exploits some public information. To avoid such attacks, we need a model where public information does not give significant advantage to an adversary.

Our goal is to design a model where the public parameters and the server's proofs of validity give no advantage to an attacker. Ideally, we would like the attacker to have no more chances to guessing the polynomial than if it only had access to a server reliably returning polynomial evaluations with no proof of validity. Our security model considers an attacker that tries to determine which polynomial is used by a PPE among two polynomials of his choice. This model is inspired by the IND-CPA model used in public key cryptography (Definition 21).

We start be redefining the notion of weak security presented in the literature. We then introduce the notion of chosen function attack and the natural notion of unforgeability.

4.3.2 Polynomial Protection

We introduce the *Polynomial Protection* (PP) security. A PPE is PP secure if no adversary can output a new point (not computed by the server) of the secret polynomial f with a better probability than by guessing it. In this model, the polynomial is randomly chosen and the adversary cannot use the server more than l times, where l is the degree of f. This security model is similar to the *Hiding Model* [KZG10] except that the adversary chooses the points to be evaluated. We define the *Weak Polynomial Protection* (WPP) as the same model as PP except that the adversary has no access to the server.

Definition 53 (PP and WPP security) *Let $\Pi = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Compute}, \mathsf{Verify})$ be a PPE, $k \in \mathbb{N}$ be an integer and and $\mathscr{A}$ be a probabilistic polynomial time adversary. Let the following oracle be:*

$\mathsf{CO}_{\mathsf{PP}}(\cdot)$**:** *On input x, this algorithm runs $(y, \pi) \leftarrow \mathsf{Compute}(\mathsf{pk}, x, \mathsf{sk}, f_b)$. and increments $c \leftarrow c+1$ and $\Sigma \leftarrow \Sigma \cup \{(x,y)\}$. If $c = l+1$, then it returns $\perp$, else it returns (y, π).*

*For any $l \in \mathbb{N}$, the l-*Polynomial Protection *(l-PP) experiment for $\mathscr{A}$ against Π denoted by $\mathsf{Exp}^{l\text{-}PP}_{\Pi,\mathscr{A}}(k)$ is defined as follows.*

```
$\mathsf{Exp}^{l\text{-}\mathbf{PP}}_{\Pi,\mathscr{A}}(k)$:
$(\mathsf{set}, \mathrm{F}) \leftarrow \mathsf{Set}(l, k)$
$f \xleftarrow{\$} \mathrm{F}[\mathrm{X}]_l$
$\Sigma \leftarrow \emptyset$
$c \leftarrow 0$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(f)$
$(x_*, y_*) \leftarrow \mathscr{A}^{\mathsf{CO}_{PP}(\cdot)}(\mathsf{set}, \mathsf{pk}, \mathrm{F}, l)$
If $(x_*, y_*) \notin \Sigma$ and $f(x_*) = y_*$ then return 1
else return 0
```

We define the l-PP advantage of the adversary $\mathscr{A}$ against Π as follows:

$$\mathsf{Adv}^{l\text{-}PP}_{\Pi,\mathscr{A}}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}PP}_{\Pi,\mathscr{A}}(k)\right]$$

We define the l-PP advantage against Π as follows:

$$\mathsf{Adv}^{l\text{-}PP}_{\Pi}(k) = \max_{\mathscr{A} \in \mathrm{POLY}(k)} \left\{\mathsf{Adv}^{l\text{-}PP}_{\Pi,\mathscr{A}}(k)\right\}$$

Π is said to be l-PP secure when the advantage $\mathsf{Adv}^{l\text{-}PP}_{\Pi}(k)$ negligible. Π is said to be PP secure when it is $t(k)$-PP for any plynomial t.

*We define the l-*Weak Polynomial Protection *(l-WPP) experiment as the l-PP experiment except that $\mathscr{A}$ does not have access to the oracle $\mathsf{CO}_{PP}(\cdot)$. In a similar way, we define the WPP advantage and security.*

The following theorem shows that the PP security implies the WPP security

Theorem 21 *For any Π and l, if Π is l-PP secure then Π is l-WPP secure.*

Proof: The only difference between the PP experiment and WPP experiment is that the adversary has no access to the oracle in WPP, so PP security implies the WPP security. □

4.3.3 Chosen Function Attack

We define a model for *indistinguishability against chosen function attack*. In this model, the adversary chooses two polynomials (f_0, f_1) and tries to guess the polynomial f_b used by the server, where $b \in \{0,1\}$. The adversary has access to a server that evaluates and proves the validity of the point $(x, f_b(x))$ only if $f_0(x) = f_1(x)$. This is an inherent limitation: if the adversary can evaluate another point (x, y) such that $f_0(x) \neq f_1(x)$, then it can compare y with $f_0(x)$ and $f_1(x)$ and recover b. In practice, an adversary chooses (f_0, f_1) such that $f_0 \neq f_1$, but with l points (x_i, y_i) such that $f_0(x_i) = f_1(x_i)$. It allows the adversary to maximize its oracle calls in order to increase its chances of success. We remark that schemes [GFLL15] and [GND16] are not IND-CFA secure: clients know a value M and the set of inputs $\mathscr{X}$ such that $\forall x \in \mathscr{X}, f(x) < \mathrm{M}$. An attacker may choose two polynomials f_0 and f_1 such that for a chosen a, $f_0(a) < \mathrm{M}$ and $f_1(a) > \mathrm{M}$. Since $\mathscr{X}$ is public, the attacker returns f_0 if and only if $a \in \mathscr{X}$.

Definition 54 (IND-CFA security) *Let $\Pi = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Compute}, \mathsf{Verify})$ be a PPE, $l \in \mathbb{N}$ and $k \in \mathbb{N}$ be two integers and $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$ be a two-party PPT adversary. Let the following oracle be:*

$\mathsf{CO}_{\mathsf{CFA}}(\cdot)$**:** *On input x, this algorithm runs $(y, \pi) \leftarrow \mathsf{Compute}(\mathsf{pk}, x, \mathsf{sk}, f_b)$. If $f_0(x) \neq f_1(x)$, then it returns $\perp$, else it returns (y, π).*

*The l-*Indistinguishability against Chosen Function Attack *(l-IND-CFA) experiment for $\mathscr{A}$ against Π is defined as follows.*

,

```
$\mathsf{Exp}^{l\text{-IND-CFA}}_{\Pi,\mathscr{A}}(k)$:
$b \xleftarrow{\$} \{0,1\}^*$
$(\mathsf{set}, \mathrm{F}) \leftarrow \mathsf{Set}(l, k)$
$(f_0, f_1, \mathsf{st}) \leftarrow \mathscr{A}_1(\mathsf{set}, \mathrm{F}, l)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(f_b)$
$b_* \leftarrow \mathscr{A}_2^{\mathsf{CO}_{CFA}(\cdot)}(\mathsf{set}, \mathsf{pk}, \mathrm{F}, l, \mathsf{st})$
If $f_0 \notin \mathrm{F}[\mathrm{X}]_l$ or $f_1 \notin \mathrm{F}[\mathrm{X}]_l$ then return $0$
else return $(b = b_*)$
```

We define the l-IND-CFA advantage of the adversary $\mathscr{A}$ against the Π as follows:

$$\mathsf{Adv}^{l\text{-}IND\text{-}CFA}_{\Pi,\mathscr{A}}(k) = \left| \frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}IND\text{-}CFA}_{\Pi,\mathscr{A}}(k)\right] \right|$$

We define the l-IND-CFA advantage against Π as follows:

$$\mathsf{Adv}^{l\text{-}IND\text{-}CFA}_{\Pi}(k) = \max_{\mathscr{A} \in \mathrm{POLY}(k)^2} \left\{ \mathsf{Adv}^{l\text{-}IND\text{-}CFA}_{\Pi,\mathscr{A}}(k) \right\}$$

Π is said to be l-IND-CFA secure if this advantage is negligible for any $\mathscr{A} \in \mathrm{POLY}(k)^2$. Π is said to be IND-CFA secure if it is $t(k)$-IND-CFA for any polynomial t.

In Theorem 22, we prove that IND-CFA security implies WPP security: if there exists an adversary $\mathscr{A}$ against the WPP experiment that is able to *decrypt* a random polynomial from the public values, then we can use it to guess f_b in an IND-CFA experiment for any chosen polynomials (f_0, f_1). However, surprisingly, it is not true for the PP security (Theorem 23). The reason is that the oracle of the IND-CFA experiment has restriction, so it cannot be used to simulate the oracle of the PP experiment in a security reduction.

Theorem 22 *If Π is a l-IND-CFA secure PPE, then it is l-WPP secure.*

Proof: Let l be an integer and Π be a PPE. Suppose that there exists $\mathcal{A} \in \text{POLY}(k)$ such that $\lambda(k) = \mathsf{Adv}^{l\text{-WPP}}_{\Pi,\mathcal{A}}(k)$ is non-negligible. We show how to build $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2) \in \text{POLY}(k)^2$ such that $\mathsf{Adv}^{l\text{-IND-CFA}}_{\mathcal{B},\Pi}(k)$ is non-negligible, then we conclude the proof by contraposition. Algorithm $\mathcal{B}$ works as follows:

$\mathcal{B}_1(\mathsf{set}, \mathrm{F}, l)$**:** $\mathcal{B}_1$ picks $f_0 \xleftarrow{\$} \mathrm{F}[\mathrm{X}]_l$ and $r \xleftarrow{\$} \mathrm{F}^*$, and builds f_1 such that $f_1(x) = f_0(x) + r$. Remark that $\forall x, f_0(x) \neq f_1(x)$. $\mathcal{B}_1$ returns $(f_0, f_1, \perp)$.

$\mathcal{B}_2(\mathsf{set}, \mathsf{pk}, \mathrm{F}, l, \perp)$**:** It runs $(x_*, y_*) \leftarrow \mathcal{A}(\mathsf{set}, \mathsf{pk}, \mathrm{F}, l)$. If $\exists\ b_* \in \{0,1\}$ such that $y_* = f_{b_*}(x_*)$ then it returns b_*, else it returns $b_* \xleftarrow{\$} \{0,1\}$.

We evaluate the probability that $\mathcal{B}$ wins the experiment, *i.e.,* $b_* = b$. First, we remark that if $\mathcal{A}$ wins its experiment, then $b_* = b$ with probability 1. On the other hand, if $\mathcal{A}$ does not win the experiment, we consider two different cases:

1. $\mathcal{A}$ returns (x_*, y_*) such that $f_{1-b}(x_*) = y_*$. The probability that $\mathcal{A}$ returns such a point is at most $1/|\mathrm{F}|$ and is negligible. Indeed, since $\mathcal{A}$ has no information about f_{1-b}, its best strategy to guess a point of f_{1-b} is to randomly pick a point in F^2. In this case, $\mathcal{B}$ wins the experiment with probability 0.

2. $\mathcal{A}$ returns (x_*, y_*) such that $f_{1-b}(x_*) \neq y_*$. The probability that $\mathcal{A}$ returns such a point is at least $1 - (1/|\mathrm{F}|)$. In this case, $\mathcal{B}$ wins the experiment with probability $1/2$.

We set $\epsilon(k) = \Pr[f_{1-b}(x_*) = y_*]$, hence we have $\epsilon(k) \leq \frac{1}{|\mathrm{F}|}$, which is negligible. We have:

$$\begin{aligned}
\Pr[b = b_*] &= \Pr[f_b(x_*) = y_*] \cdot \Pr[b = b_* | f_b(x_*) = y_*] + \Pr[f_b(x_*) \neq y_*] \cdot \Pr[b = b_* | f_b(x_*) \neq y_*] \\
&= \lambda(k) \cdot 1 + (1 - \lambda(k)).\Pr[b = b_* | f_b(x_*) \neq y_*] \\
&= \lambda(k) + (1 - \lambda(k)) \cdot (\Pr[f_{1-b}(x_*) = y_*] \cdot \Pr[b = b_* | f_{1-b}(x_*) = y_*] \\
&\quad + \Pr[f_{1-b}(x_*) \neq y_*] \cdot \Pr[b = b_* | f_b(x_*) \neq y_* \text{ and } f_{1-b}(x_*) \neq y_*]) \\
&= \lambda(k) + (1 - \lambda(k)) \cdot \left(0 + (1 - \epsilon(k)) \cdot \frac{1}{2}\right) \\
&= \lambda(k) + \frac{1}{2} - \frac{\epsilon(k)}{2} - \frac{\lambda(k)}{2} + \frac{\lambda(k) \cdot \epsilon(k)}{2} \\
&= \frac{1}{2} + \frac{\lambda(k)}{2} + \frac{\epsilon(k) \cdot (\lambda(k) - 1)}{2}
\end{aligned}$$

We deduce the advantage of $\mathcal{B}$:

$$\begin{aligned}
\mathsf{Adv}^{l\text{-IND-CFA}}_{\mathcal{B},\Pi}(k) &= \left|\Pr[b = b_*] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2} - \frac{1}{2} + \frac{\lambda(k)}{2} + \frac{\epsilon(k) \cdot (\lambda(k) - 1)}{2}\right| \\
&\geq \frac{\lambda(k)}{2} - \epsilon(k) \cdot \frac{1 - \lambda(k)}{2}
\end{aligned}$$

Since $\lambda(k)$ is non negligible, then $\epsilon(k) \cdot \frac{1-\lambda(k)}{2}$ is negligible. Finally, the advantage $\mathsf{Adv}^{l\text{-IND-CFA}}_{\mathcal{B},\Pi}(k)$ is non-negligible, which concludes the proof. □

Theorem 23 *Let Π be a l-IND-CFA secure PPE, it does not imply that Π is l-PP.*

Proof: Let l be an integer and let $\Pi = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Compute}, \mathsf{Verify})$ be a PPE that is l-PP and that is l-$\mathsf{IND\text{-}CFA}$ secure. Let $\Pi' = (\mathsf{Set}', \mathsf{Gen}', \mathsf{Compute}', \mathsf{Verify}')$ be a tuple of polynomial time algorithms defined as follows:

$\mathsf{Set}'(l)$**:** It returns $(\mathsf{set}, \mathrm{F}) \leftarrow \mathsf{Set}(l, k)$.
$\mathsf{Gen}'(f)$**:** It runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(f)$, picks $\alpha \xleftarrow{\$} \mathrm{F}$ and returns $(\mathsf{pk}', \mathsf{sk}') = ((\mathsf{pk}, \alpha), \mathsf{sk})$.
$\mathsf{Compute}'(\mathsf{pk}', x, \mathsf{sk}', f)$**:** This algorithm runs $(y, \pi) \leftarrow \mathsf{Compute}(\mathsf{pk}, x, \mathsf{sk}, f)$. If $x \neq \alpha$ then sets $\pi' = (\pi, \perp)$ else $\pi' = (\pi, f)$. It returns (y, π').
$\mathsf{Verify}'(\mathsf{pk}', x, y, \pi')$**:** It returns $b \leftarrow \mathsf{Verify}(\mathsf{pk}, x, y, \pi)$.

To prove the theorem, we show that (i) Π' is l-$\mathsf{IND\text{-}CFA}$ secure and (ii) Π' is not l-PP secure.

i) We prove it by contraposition. Assume that there exists $\mathcal{A} \in \text{POLY}(k)^2$ such that the advantage $\lambda(k) = \mathsf{Adv}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi',\mathcal{A}}(k)$ is non-negligible. We show how to build $\mathcal{B} \in \text{POLY}(k)^2$ such that $\mathsf{Adv}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi,\mathcal{B}}(k)$ is non-negligible. The adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ works as follows:

$\mathcal{B}_1(\mathsf{set}, \mathrm{F}, l)$**:** It runs $(f_0, f_1, \tilde{\sigma}) \leftarrow \mathcal{A}_1(\mathsf{set}, \mathrm{F}, l)$ and returns $(f_0, f_1, \tilde{\sigma})$.

$\mathcal{B}_2(\mathsf{set}, \mathsf{pk}, \mathrm{F}, l, \tilde{\sigma})$**:** It picks $\alpha \xleftarrow{\$} \mathrm{F}$.

- If $f_0(\alpha) = f_1(\alpha)$ then $\mathcal{B}_2$ picks $b_* \xleftarrow{\$} \{0,1\}$ and returns b_*.
- If $f_0(\alpha) \neq f_1(\alpha)$ then $\mathcal{B}_2$ runs $b_* \leftarrow \mathcal{A}_2(\mathsf{set}, (\mathsf{pk}, \alpha), \mathrm{F}, l, \tilde{\sigma})$. While $\mathcal{A}_2$ is running, $\mathcal{B}_2$ forwards the queries of $\mathcal{A}_2$ to the oracle $\mathsf{CO}_{\mathsf{CFA}}(\cdot)$. Finally, $\mathcal{B}_2$ returns b_*.

We remark that during the $\mathsf{IND\text{-}CFA}$ experiment, the adversary $\mathcal{A}$ chooses its two polynomials (f_0, f_1) before α is chosen. Since f_0 and f_1 have at most l common points, the probability that $f_0(\alpha) = f_1(\alpha)$ is negligible. We set:

$$\epsilon(k) = \Pr[f_0(\alpha) = f_1(\alpha)] \leq \frac{l}{|\mathrm{F}|}$$

We deduce that $\epsilon(k)$ is negligible. When $f_0(\alpha) \neq f_1(\alpha)$, α is randomly chosen and gives no additional information to $\mathcal{A}$. $\mathcal{A}$ cannot call the oracle $\mathsf{CO}_{\mathsf{CFA}}(\cdot)$ using α as input since $f_0(\alpha) \neq f_1(\alpha)$. Thus α is useless and the CFA experiment is perfectly simulated by $\mathcal{B}$. Below, we evaluate the probability that $\mathcal{B}$ wins the experiment:

$$\begin{aligned}
\Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi,\mathcal{B}}(k)\right] &= \Pr[b = b_*] \\
&= \Pr[f_0(\alpha) = f_1(\alpha)] \cdot \Pr[b = b_* | f_0(\alpha) = f_1(\alpha)] \\
&\quad + \Pr[f_0(\alpha) \neq f_1(\alpha)] \cdot \Pr[b = b_* | f_0(\alpha) \neq f_1(\alpha)] \\
&= \epsilon(k) \cdot \frac{1}{2} + (1 - \epsilon(k)) \cdot \Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi',\mathcal{A}}(k)\right] \\
&= \epsilon(k) \cdot \left(\frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi',\mathcal{A}}(k)\right]\right) + \Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi',\mathcal{A}}(k)\right]
\end{aligned}$$

We then show that the advantage of $\mathcal{B}$ is non-negligible:

$$\begin{aligned}
\mathsf{Adv}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi,\mathcal{B}}(k) &= \left|\Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi,\mathcal{B}}(k)\right] - \frac{1}{2}\right| \\
&= \left|\epsilon(k) \cdot \left(\frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi',\mathcal{A}}(k)\right]\right) + \Pr\left[1 \leftarrow \mathsf{Exp}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi',\mathcal{A}}(k)\right] - \frac{1}{2}\right| \\
&\geq \lambda(k) - \epsilon(k) \cdot \lambda(k)
\end{aligned}$$

It contradicts that Π is l-$\mathsf{IND\text{-}CFA}$.

ii) We construct $\mathcal{A} \in \text{POLY}(k)$ such that $\mathsf{Adv}^{l\text{-}\mathsf{PP}}_{\Pi',\mathcal{A}}(k)$ is non-negligible: $\mathcal{A}$ receives $(\mathsf{set}, (\mathsf{pk}, \alpha), \mathrm{F}, l)$ as input and uses α as input for the oracle $\mathsf{CO}_{\mathsf{CFA}}(.)$ that returns $(f(\alpha), (\pi, f))$. $\mathcal{A}$ chooses $x \in \mathrm{F}$ and returns $(x, f(x))$. Thus, $\mathsf{Adv}^{l\text{-}\mathsf{PP}}_{\Pi',\mathcal{A}}(k) = 1$.

We have (i) Π' is l-IND-CFA secure and (ii) Π' is not l-PP secure. We conclude that l-IND-CFA security it does not imply l-PP security. □

We just prove that l-IND-CFA security does not imply l-PP security, however, we would like to have a simple and sufficient condition under which the IND-CFA security implies the PP security. For this, we define the *proof induced by a PPE* which is the proof algorithm used by the algorithm Compute. We show that if this proof system is zero-knowledge, then the IND-CFA security implies the PP security.

Definition 55 *Let* $\Pi = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Compute}, \mathsf{Verify})$ *be a PPE, the* non-interactive proof inducted by Π*, denoted* $\mathrm{P}_\Pi = (\mathsf{Set}_\Pi, \mathsf{Pro}_\Pi, \mathsf{Ver}_\Pi)$ *is defined as follows.*

$\mathsf{Set}_\Pi(k)$**:** *It runs* $(\mathsf{set}, \mathrm{F}) \leftarrow \mathsf{Set}(l,k)$*, sets* $\mathsf{set}' = (\mathsf{set}, \mathrm{F})$ *and generates the relation* $\mathcal{R}$ *such that:*

$$((f, \mathsf{sk}), (\mathsf{pk}, x, y)) \in \mathcal{R} \Leftrightarrow (f \in \mathrm{F}[\mathrm{X}]_l) \wedge ((x,y) \in \mathrm{F}^2) \wedge (f(x) = y) \wedge ((\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(f))$$

It returns $(\mathsf{set}', \mathcal{R})$.

$\mathsf{Pro}_\Pi((f, \mathsf{sk}), (\mathsf{pk}, x, y))$**:** *It runs* $(y', \pi) \leftarrow \mathsf{Compute}(\mathsf{pk}, x, \mathsf{sk}, f)$ *and returns* π.

$\mathsf{Ver}_\Pi((\mathsf{pk}, x, y), \pi)$**:** *It runs* $b \leftarrow \mathsf{Verify}(\mathsf{pk}, x, y, \pi)$ *and returns it.*

We say that Π *is* Zero-Knowledge *(ZK) if* P_Π *is Zero-Knowledge.*

Theorem 24 *Let* Π *be a ZK and* l*-IND-CFA secure PPE, then* Π *is* l*-PP secure.*

Proof: Let Π be a PPE that is zero-knowledge. $\forall l \in \mathbb{N}$, we assume that there exists $\mathcal{A} \in \text{POLY}(k)$ such that $\lambda(k) = \mathsf{Adv}^{l\text{-PP}}_{\Pi,\mathcal{A}}(k)$ is non negligible and that for any $\mathcal{A} \in \text{POLY}(k)^2$, $\mathsf{Adv}^{l\text{-IND-CFA}}_{\Pi,\mathcal{A}}(k)$ is negligible. We show that there exists $\mathcal{B} \in \text{POLY}(k)^2$ such that $\mathsf{Adv}^{l\text{-IND-CFA}}_{\Pi,\mathcal{B}}(k)$ is non-negligible. We obtain a contradiction, then we deduce that for any zero-knowledge PPE Π such that $\mathsf{Adv}^{l\text{-IND-CFA}}_{\Pi}(k)$ is negligible, then $\mathsf{Adv}^{l\text{-PP}}_{\Pi}(k)$ is negligible.

By hypothesis Π is zero-knowledge, so there exists an algorithm Sim such that the outputs of $\mathsf{proof}_\Pi((\mathsf{set}, \mathsf{pk}, x, y), (f, \mathsf{sk}))$ and $\mathsf{Sim}((\mathsf{set}, \mathsf{pk}, x, y))$ follow the same probability distribution for any $(\mathsf{set}, \mathsf{pk}, x, y), (f, \mathsf{sk})$. We build the following adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$:

$\mathcal{B}_1(\mathsf{set}, \mathrm{F}, l)$**:** It picks $(f_0, f_1) \xleftarrow{\$} \mathrm{F}[\mathrm{X}]_l^2$ and returns $(f_0, f_1, \perp)$.

$\mathcal{B}_2(\mathsf{set}, \mathsf{pk}, \mathrm{F}, l, \perp)$**:** It picks $b' \xleftarrow{\$} \{0,1\}$:

- It runs $(x_*, y_*) \leftarrow \mathcal{A}(\mathsf{set}, \mathsf{pk}, \mathrm{F})$ and simulates the oracle $\mathsf{CO}_{\mathsf{PP}}(\cdot)$ as follows: on input x, $\mathcal{B}_2$ computes $y = f_{b'}(x)$ and runs $\pi \leftarrow \mathsf{Sim}((\mathsf{set}, \mathsf{pk}, x, y))$. It then returns (y, π) to $\mathcal{A}$. Note that $\mathcal{B}$ cannot directly use the oracle $\mathsf{CO}_{\mathsf{CFA}}(\cdot)$ to answer the query because $f_0(x)$ may be different than $f_1(x)$.
- If $f_{b'}(x_*) = y_*$, then $\mathcal{B}_2$ returns $b_* = b'$, else it picks $b_* \xleftarrow{\$} \{0,1\}$ and returns b_*.

We evaluate the probability that $\mathcal{B}$ wins the experiment, *i.e.*, $b_* = b$ where b is the challenge of $\mathcal{B}$:

- if $b' = b$, then the PP experiment is perfectly simulated and $\mathcal{A}$ returns (x_*, y_*) such that $f_{b'}(x_*) = y_*$ with non negligible probability $\lambda(k)$. Then $\mathcal{B}$ wins the experiment with non-negligible advantage. Remark that in this case, the probability that $b_* = b$ is the same as in the proof of Theorem 22:

$$\Pr\left[b_* = b | b' = b\right] = \frac{\lambda(k)}{2} + \epsilon(k) \cdot \frac{1 - \lambda(k)}{2} + \frac{1}{2}$$

where $\epsilon(k) = \Pr[f_{1-b}(x_*) = y_*] \leq 1/|\mathrm{F}|$.

- If $b' \neq b$, then the probability that $\mathscr{A}$ returns (x_*, y_*) such that $f_{b'}(x_*) = y_*$ is negligible: $\mathscr{A}$ knows at most l points of the polynomial $f_{b'}$, then its best strategy to find another point is to pick (x_*, y_*) randomly in $\mathbb{F}^2$. Then $\mathscr{B}$ wins the experiment with negligible advantage. More formally, we have:

$$\begin{aligned}
\Pr\left[b_* = b | b' \neq b\right] &= \Pr[f_b(x_*) = y_*] \cdot \Pr[b = b_* | b' \neq b \text{ and } f_b(x_*) = y_*] \\
&\quad + \Pr[f_b(x_*) \neq y_*] \cdot \Pr[b = b_* | b' \neq b \text{ and } f_b(x_*) \neq y_*] \\
&= 0 + (1 - \epsilon(k)) \cdot \frac{1}{2} = \frac{1}{2} - \frac{\epsilon(k)}{2}
\end{aligned}$$

Finally, we have:

$$\begin{aligned}
\Pr[b_* = b] &= \Pr\left[b' = b\right] \cdot \Pr\left[b_* = b | b' = b\right] + \Pr\left[b' \neq b\right] \cdot \Pr\left[b_* = b | b' \neq b\right] \\
&= \frac{1}{2} \cdot \left(\frac{\lambda(k)}{2} + \epsilon(k) \cdot \frac{1 - \lambda(k)}{2} + \frac{1}{2}\right) + \frac{1}{2}\left(\frac{1}{2} - \frac{\epsilon(k)}{2}\right) \\
&= \frac{1}{2} \cdot \left(\frac{\lambda(k)}{2} + \epsilon(k) \cdot \frac{1 - \lambda(k)}{2} + \frac{1}{2} + \frac{1}{2} - \frac{\epsilon(k)}{2}\right) \\
&= \frac{\lambda(k)}{4} - \epsilon(k) \cdot \frac{\lambda(k)}{4} + \frac{1}{2}
\end{aligned}$$

Finally, we show that $\mathscr{B}$ has a non-negligible advantage:

$$\begin{aligned}
\mathsf{Adv}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\Pi,\mathscr{B}}(k) &= \left|\Pr[b_* = b] - \frac{1}{2}\right| \\
&= \frac{\lambda(k)}{4} - \epsilon(k) \cdot \frac{\lambda(k)}{4} \\
&\geq \frac{\lambda(k)}{4} - \frac{\lambda(k)}{4 \cdot |\mathbb{F}|}
\end{aligned}$$

It contradicts our hypothesis, which concludes the proof. □

In Figure 7.2, we recall all relations between our security properties.

$$\begin{tikzcd}
 & l\text{-IND-CFA} \arrow[r, Rightarrow] \arrow[d, dash] \arrow[dr, Rightarrow, "/" marking] & l\text{-WPP} \\
\text{ZK} \arrow[r, dash] & \text{and} \arrow[r, Rightarrow] & l\text{-PP} \arrow[u, Rightarrow]
\end{tikzcd}$$

Figure 4.2: Security relations.

4.3.4 Unforgeability

Finally, we define the unforgeability. A PPE is unforgeable when it is not possible to produce a valid proof on the point (x, y) when $f(x) \neq y$. The adversary is a dishonest server that chooses a polynomial f and that receives a pair of verification/secret key generated by the key generation algorithm from the polynomial f. The adversary returns a point (x, y) and a proof π, and it succeeds the attack if the proof π is valid and $f(x) \neq y$.

Definition 56 *Let Π be a PPE, $l \in \mathbb{N}$ and $k \in \mathbb{N}$ be two integers and $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$ be a two-party PPT adversary. The* Unforgeability *(UNF) experiment for $\mathscr{A}$ against Π is defined as follows.*

```
$\mathsf{Exp}^{l\text{-}\mathbf{UNF}}_{\Pi,\mathscr{A}}(k)$:
$(\mathsf{set},\mathrm{F}) \leftarrow \mathsf{Set}(l,k)$
$(f,\mathsf{st}) \leftarrow \mathscr{A}_1(\mathsf{set},\mathrm{F})$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(f)$
$(x_*,y_*,\pi_*) \leftarrow \mathscr{A}_2(\mathsf{set},\mathsf{sk},\mathsf{pk},\mathrm{F},f,\mathsf{st})$
If $f(x_*) \neq y_*$ and $\mathsf{Verify}(\mathsf{pk},x_*,y_*,\pi_*)$ then return 1
else return 0
```

We define the advantage of the adversary $\mathscr{A}$ against the l-UNF experiment by:

$$\mathsf{Adv}^{l\text{-}UNF}_{\Pi,\mathscr{A}}(k) = \mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{l\text{-}UNF}_{\Pi,\mathscr{A}}(k)\right]$$

We define the advantage against the l-UNF experiment by:

$$\mathsf{Adv}^{l\text{-}UNF}_{\Pi}(k) = \max_{\mathscr{A}\in\text{POLY}(k)} \left\{\mathsf{Adv}^{l\text{-}UNF}_{\Pi,\mathscr{A}}(k)\right\}$$

A scheme Π is l-UNF secure if this advantage is negligible for any $\mathscr{A} \in \text{POLY}(k)^2$. A scheme Π is unforgeable, or UNF secure, if it is $t(k)$-UNF secure for any polynomial t.

4.3.5 Security Against Collusion Attacks

Until know, we did not take into account the collusion attacks, however, we show this kind of attacks are implicitly prevented in our model. The reason is that the clients have no secret information, so a dishonest party that colludes with a client learns nothing else than the public values. There are two kinds of collusion attack scenarios:

A client colludes with the server: If a client colludes with the server, then it can obviously give him the secret polynomial. This limitation is inherent and cannot be prevented. On the other hand, all keys known by the clients are known to the server, the server has no advantage in colluding with a client. In particular, the collusion does not allow the server to forge fake validity proofs for other clients.

Several clients collude together: All clients have the same verification keys. Thus, a client gains no advantage by colluding with other clients, as long as the total number of known points is less than l after collusion. Obviously the inherent limitation of PPE still holds: if the collusion of clients learn more than l points, then they can guess the polynomial.

4.4 $\mathsf{PolyCommit}_{\mathsf{Ped}}$ Is IND-CFA Secure

In their paper, Kate *et al.* introduce two PPE schemes [KZG10]. The first one, called $\mathsf{PolyCommit}_{\mathsf{DL}}$, is not IND-CFA secure since the algorithm that generates the verification key from the polynomial is deterministic. A client who tries to distiguish what is the polynomial used by the server has just to compute the verification key for the two chosen polynomials. The second scheme, called $\mathsf{PolyCommit}_{\mathsf{Ped}}$, is proven UNF and PP secure. In this section, we recall this scheme, and we show that it is also IND-CFA secure.

Definition 57 $\mathsf{PolyCommit}_{\mathsf{Ped}} = (\mathsf{setup},\mathsf{init},\mathsf{compute},\mathsf{verif})$ *is a* PPE *scheme such that:*

$\mathsf{Set}(l,k)$**:** *It generates two groups $\mathbb{G}$ and $\mathbb{G}_{\mathrm{T}}$ of prime order p and a symmetric bilinear pairing $e\colon \mathbb{G}\times\mathbb{G}\to\mathbb{G}_{\mathrm{T}}$. Moreover, it chooses two generators g and h of $\mathbb{G}$ and picks $\alpha \leftarrow \mathbb{Z}_p^*$. It sets $\mathrm{F} = \mathbb{Z}_p^*$,* $\mathsf{set} = (\mathbb{G},\mathbb{G}_{\mathrm{T}},p,e,g,h,(g^{\alpha},\ldots,g^{\alpha^l}),(h^{\alpha},\ldots,h^{\alpha^l}))$ *and returns* $(\mathsf{set},\mathrm{F})$.

$\mathsf{Gen}(f)$**:** *It picks a random polynomial $r \xleftarrow{\$} \mathrm{F}[\mathrm{X}]_l$. We denote by a_i (resp. r_i) the i^{th} coefficient of f (resp. r):*

$$f(x) = \sum_{i=0}^{l} a_i \cdot x^i$$
$$r(x) = \sum_{i=0}^{l} r_i \cdot x^i$$

It sets $\mathsf{sk} = r$ *and computes:*

$$\mathscr{C} = \prod_{i=0}^{l} (g^{\alpha^i})^{a_i} \cdot (h^{\alpha^i})^{r_i}$$

It sets $\mathsf{pk} = \mathscr{C}$*. Finally, it returns* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Compute}(\mathsf{pk}, x_i, \mathsf{sk}, f)$**:** *Using* $\mathsf{pk} = \mathscr{C}$ *and* $\mathsf{sk} = r$*, this algorithm computes the two following polynomials:*

$$\psi_i(x) = \frac{f(x) - f(x_i)}{x - x_i}$$
$$\hat{\psi}_i(x) = \frac{r(x) - r(x_i)}{x - x_i}$$

Let $(\gamma_0, \ldots, \gamma_l)$ *and* $(\hat{\gamma}_0, \ldots, \hat{\gamma}_l)$ *be two tuples such that:*

$$\psi_i(x) = \sum_{j=0}^{l} \gamma_j \cdot x^j \text{ and } \hat{\psi}_i(x) = \sum_{j=0}^{l} \hat{\gamma}_j \cdot x^j$$

This algorithm computes

$$w_i = \prod_{j=0}^{l} (g^{\alpha^j})^{\gamma_j} \cdot (h^{\alpha^j})^{\hat{\gamma}_j}$$

It sets $\pi = (x_i, r(x_i), w_i)$ *and returns* $(f(x_i), \pi)$.

$\mathsf{Verify}(\mathsf{pk}, x_i, y_i, \pi)$**:** *It parses* $\pi = (x_i, r(x_i), w_i)$ *and* $\mathsf{pk} = \mathscr{C}$*. If the following equation holds:*

$$e(\mathscr{C}, g) = e\left(w_i, \frac{g^\alpha}{g^{x_i}}\right) \cdot e(g^{f(x_i)} \cdot h^{r(x_i)}, g)$$

then it outputs 1*, else it outputs* 0.

We show that $\mathsf{PolyCommit}_{\mathsf{Ped}}$ is l-IND-CFA secure.

Theorem 25 *For any $l \in \mathbb{N}$,* $\mathsf{PolyCommit}_{\mathsf{Ped}}$ *(is inconditionally) l-IND-CFA secure.*

Proof: Let $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2) \in \text{POLY}(k)^2$ be a two party algorithm. We build an algorithm $\mathscr{B} \in \text{POLY}(k)$ that perfectly simulates the experiment $\mathsf{Exp}^{l\text{-IND-CFA}}_{\mathsf{PolyCommit}_{\mathsf{Ped}}, \mathscr{A}}(k)$ for any $l \in \mathbb{N}$, and we show that the probability that $\mathscr{A}$ wins the experiment is 1/2. Our algorithm $\mathscr{B}(k, l)$ works as follows:

- $\mathscr{B}$ picks randomly $b \xleftarrow{\$} \{0, 1\}$.
- $\mathscr{B}$ generates two groups $\mathbb{G}$ and $\mathbb{G}_\mathrm{T}$ of prime order p and a symmetric bilinear pairing $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathrm{T}$. Moreover, it chooses two generators g and h of $\mathbb{G}$ and picks $\alpha \leftarrow \mathbb{Z}_p^*$. It sets $\mathrm{F} = \mathbb{Z}_p^*$ and set $= (\mathbb{G}, \mathbb{G}_\mathrm{T}, p, e, g, h, (g^\alpha, \ldots, g^{\alpha^l}), (h^\alpha, \ldots, h^{\alpha^l}))$.
- $\mathscr{B}$ runs $(f_0, f_1, \mathsf{st}) \leftarrow \mathscr{A}_1(\mathsf{set}, \mathbb{Z}_p, l)$.
- If $f_0(\alpha) \neq f_1(\alpha)$, then $\mathscr{B}$ picks a random element $\mathscr{C} \in \mathbb{G}$. Since $\mathbb{G}$ is a cyclic group, there exists $r_\alpha \in \mathbb{Z}_p$ such that $\mathscr{C} = g^{f_b(\alpha)} \cdot h^{r_\alpha}$ for $b \in \{0, 1\}$. Otherwise, if $f_0(\alpha) = f_1(\alpha)$, $\mathscr{B}$ picks a random $r_\alpha \in \mathbb{Z}_p$ and sets $\mathscr{C} = g^{f_0(\alpha)} \cdot h^{r_\alpha}$. $\mathscr{B}$ sets $\mathsf{pk} = \mathscr{C}$.

- $\mathscr{B}$ runs $b_* \leftarrow \mathscr{A}_2(\mathsf{set}, \mathsf{pk}, \mathbb{Z}_p, l, \mathsf{st})$. To simulate the oracle $\mathsf{CO}_{\mathsf{CFA}}(\cdot)$ on x_i to $\mathscr{A}_2$, the algorithm $\mathscr{B}$ computes $y_i = f_0(x_i)$. If $y_i = f_1(x_i)$, it picks randomly $r_i \in \mathbb{Z}_p$ and sets $w_i = (\mathscr{C} \cdot g^{-y} \cdot h^{-r_i(x_i)})^{\frac{1}{\alpha - x_i}}$. Then, it sets $\pi = (x_i, y, r_i, w_i)$ and returns (y, π) to $\mathscr{A}_2$. Else it returns $\perp$.

r_α denotes the element of $\mathbb{Z}_p^*$ that verifies the following equation: $\mathscr{C} = g^{f_0(\alpha)} \cdot h^{r_\alpha}$. First remark that since $\mathscr{C}$ is chosen in the uniform distribution on $\mathbb{G}$, then r_α comes from the uniform distribution on $\mathbb{Z}_p^*$. Moreover, we remark that there exists at least one ploynomial $r(x) \in \mathbb{Z}_p[x]$ of degree l such that $r(x_i) = r_i$ for $1 \le i \le l$ and $r(\alpha) = r_\alpha$. Since all r_i are randomly generated, we can consider that r was picked in the uniform distribution on $\mathbb{Z}_p^*[X]_l$. Since $r(\alpha) = r_\alpha$, we have $\mathscr{C} = g^{f_b(\alpha)} h^{r(\alpha)}$ and for all $1 \le i \le l$:

$$\begin{aligned} e(w_i, g^{\alpha - x_i}) \cdot e(g^y \cdot h^{r(x_i)}, g) &= e((\mathscr{C} \cdot g^{-y} \cdot h^{-r(x_i)})^{\frac{1}{\alpha - x_i}}, g^{\alpha - x_i}) \cdot e(g^y \cdot h^{r(x_i)}, g) \\ &= e(\mathscr{C} \cdot g^{-y} \cdot h^{-r(x_i)}, g) \cdot e(g^y \cdot h^{r(x_i)}, g) \\ &= e(\mathscr{C} \cdot g^{-y} \cdot h^{-r(x_i)} \cdot g^y \cdot h^{r(x_i)}, g) \\ &= e(\mathscr{C}, g) \end{aligned}$$

We deduce that the simulation does not depend on the chosen b and the experiment l-IND-CFA is perfectly simulated for $\mathscr{A}$. Also, $\mathscr{A}$ cannot do better than the random to guess the value of the chosen b. Hence, for any $\mathscr{A} \in \mathrm{POLY}(k)$[2]:

$$\Pr[1 \leftarrow \mathsf{Exp}^{l\text{-}\mathsf{IND}\text{-}\mathsf{CFA}}_{\mathsf{PolyCommit}_{\mathsf{Ped}}, \mathscr{A}}(k)] = \Pr[b = b_*] = \frac{1}{2}$$

Finally, we have that $\mathsf{Adv}^{l\text{-}\mathsf{IND}\text{-}\mathsf{CFA}}_{\mathsf{PolyCommit}_{\mathsf{Ped}}}(k) = 0$ which concludes the proof.

□

4.5 PIPE: an IND-CFA Secure Verifiable Private Polynomial Evaluation Scheme

In this section we present a new construction of PPE. We recall Feldman's Verifiable Secret Sharing (VSS) scheme and we show how to build a simple PPE scheme from Feldman's VSS that is PP secure but not IND-CFA secure. Then, using the ElGamal encryption scheme, we modificate our PPE scheme in order to design a second PPE scheme, called PIPE, that is IND-CFA secure. We analyse its security and compare it with the scheme of Kate *et al.* [KZG10].

4.5.1 Feldman's Verifiable Secret Sharing

Feldman's VSS [Fel87] is based on Shamir's Secret Sharing [Sha79], where each share is a point (x, y) of a secret polynomial f of degree l. Knowing more than l shares, one can guess the polynomial f and can compute the secret $s = f(0)$. In Feldman's VSS, there is a public value that allows anybody to check the validity of a share. For any point (x, y), anybody can check whether $y = f(x)$ or not. This scheme works as follows. Let $\mathbb{G}$ be a multiplicative group of prime order p where discrete logarithm assumption is hard. Let $f \in \mathbb{Z}_p^*[X]$ be the secret polynomial and $a_i \in \mathrm{F}$ be a coefficient for all $0 \le i \le l$ such that:

$$f(x) = \sum_{i=0}^{l} a_i \cdot x^i$$

Let $g \in \mathbb{G}$ be a generator. For all $i \in \{0, \ldots, l\}$, we set $h_i = g^{a_i}$. Values g and $\{h_i\}_{0 \le i \le l}$ are public, however, the coefficients a_i are hidden under the discret logarithm hypothesis. We remark that $f(x) = y$ if and only if $g^y = \prod_{i=0}^{l} h_i^{x^i}$. Indeed, $(f(x) = y) \Leftrightarrow (g^{f(x)} = g^y)$ and $\prod_{i=0}^{l} h_i^{x^i} = g^{f(x)}$:

$$\prod_{i=0}^{l} h_i^{x^i} = \prod_{i=0}^{l} g^{a_i \cdot x^i} = g^{\sum_{i=0}^{l} a_i \cdot x^i} = g^{f(x)}$$

Then, we can use this equation to check whether (x, y) is a valid share.

4.5.2 PIPE Description

Let $f(x) = \sum_{i=0}^{l} a_i \cdot x^i$ be a polynomial. We show how to use Feldman's VSS to design a PPE scheme that is l-PP secure: using the public values g and $\{h_i\}_{0 \le i \le l}$ such that $\forall i \in \{0, \dots, l\}, h_i = g^{a_i}$, any client can check that the point (x, y) computed by the server is a point of f. However, in practice, the polynomial f is not randomly chosen in a large set.

In the IND-CFA experiment, the attacker knows that $f = f_0$ or $f = f_1$ for two polynomials (f_0, f_1) of its choice. Moreover, since it knows the coefficients $\{a_{0,i}\}_{0 \le i \le l}$ and $\{a_{1,i}\}_{0 \le i \le l}$ of these two polynomials, it can compute the values $\{g^{a_{0,i}}\}_{0 \le i \le l}$ and $\{g^{a_{1,i}}\}_{0 \le i \le l}$, so it can compare it with the public set $\{h_i\}_{0 \le i \le l}$.

In order to construct an IND-CFA scheme, we give an ElGamal key pair $(\mathsf{pk}, \mathsf{sk})$ to the server. We set $\mathsf{pk} = h$ where $h = g^{\mathsf{sk}}$ and we encrypt all the h_i. Then for all $i \in \{0, \dots, l\}$, the clients do not know $h_i = g^{a_i}$ but they know the ElGamal ciphertext (c_i, d_i) such that $c_i = g^{r_i}$ and $d_i = h^{r_i} \cdot h_i$ where r_i is the random coin used during the ElGamal encryption. Since ElGamal is IND-CPA secure, for all $0 \le i \le l$ the attacker cannot distinguish whether the ciphertext (c_i, d_i) encrypts a coefficient of f_0 or of f_1. The attacks on the previous scheme are no longer possible.

Moreover, for any point (x, y) we show how a client can check whether $f(x) = y$ using the values $\{(c_i, d_i)\}_{0 \le i \le l}$. We set $r(x) = \sum_{i=0}^{l} r_i \cdot x^i$. The client computes:

$$c = \prod_{i=0}^{l} c_i^{x^i} = \prod_{i=0}^{l} g^{r_i \cdot x^i} = g^{\sum_{i=0}^{l} r_i \cdot x^i} = g^{r(x)}$$

On the other hand, he computes:

$$d' = \prod_{i=0}^{l} d_i^{x^i} = \left(\prod_{i=0}^{l} h^{r_i \cdot x^i} \right) \cdot \left(\prod_{i=0}^{l} g^{a_i \cdot x^i} \right) = h^{\sum_{i=0}^{l} r_i \cdot x^i} \cdot g^{\sum_{i=0}^{l} a_i \cdot x^i} = h^{r(x)} \cdot g^{f(x)}$$

Finally, $(c, d') = (g^{r(x)}, h^{r(x)} \cdot g^{f(x)})$ is an ElGamal ciphertext of $g^{f(x)}$. Then, to convince the client that $y = f(x)$, the server proves that (c, d') is a ciphertext of g^y using a non-interactive zero-knowledge proof system. On the other words, he proves that $\log_g(h) = \log_c(d'/g^y)$ thanks to the proof system LogEq (Definition 29). This leads us to the following formal definition of our scheme PIPE.

Definition 58 *Let* PIPE = (Set, Gen, Compute, Verify) *be a PPE defined by:*

Set(l, k)**:** *Using the security parameter l, it generates $\mathbb{G}$ a group of prime order p and a generator $g \in \mathbb{G}$. It chooses a hash function* $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p^*$ *and it sets* $\mathrm{F} = \mathbb{Z}_p^*$. *It sets* $\mathsf{set} = (\mathrm{G}, p, g, \mathrm{H})$ *and returns* $(\mathsf{set}, \mathrm{F})$.

Gen(f)**:** *We set $f(x) = \sum_{i=0}^{l} a_i \cdot x^i$. This algorithm picks* $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$ *and computes $h = g^{\mathsf{sk}}$. For all $i \in [\![0, l]\!]$, it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $c_i = g^{r_i}$ and $d_i = \mathsf{pk}^{r_i} \cdot g^{a_i}$. Finally, it sets* $\mathsf{pk} = (\{(c_i, d_i)\}_{0 \le i \le l}, h)$ *and returns* $(\mathsf{pk}, \mathsf{sk})$.

Compute$(\mathsf{pk}, x, \mathsf{sk}, f)$**:** *This algorithm parses* $\mathsf{pk} = (\{(c_i, d_i)\}_{0 \le i \le l}, h)$, *then it picks* $\theta \xleftarrow{\$} \mathbb{Z}_p^*$ *and computes:*

$$c = \prod_{i=0}^{l} c_i^{x^i}$$
$$\pi = (g^\theta, c^\theta, \theta + \mathrm{H}(g^\theta || c^\theta) \cdot \mathsf{sk})$$

Finally, it returns $(f(x), \pi)$.

$\mathsf{Verify}(\mathsf{pk}, x, y, \pi)$**:** *Using* $\mathsf{pk} = (\{(c_i, d_i)\}_{0\le i\le l}, h)$ *and* $\pi = (\mathrm{A}, \mathrm{B}, \omega)$*, this algorithm computes:*

$$c = \prod_{i=0}^{l} c_i^{x^i}$$

$$d = \frac{\left(\prod_{i=0}^{l} d_i^{x^i}\right)}{g^y}$$

If $g^\omega = \mathrm{A} \cdot h^{\mathrm{H}(\mathrm{A}||\mathrm{B})}$ *and* $c^\omega = \mathrm{B} \cdot d^{\mathrm{H}(\mathrm{A}||\mathrm{B})}$*, then the algorithm returns* 1*, else it returns* 0.

4.6 Security Proofs of PIPE

In this section we prove the security of PIPE.

4.6.1 Correctness

We first show that PIPE is correct.

Lemma 28 PIPE *is correct.*

Proof: For any $l \in \mathbb{N}$, any $k \in \mathbb{N}$, any $(\mathsf{set}, \mathbb{Z}_p^*) \leftarrow \mathsf{Set}(l, k)$ such that $\mathsf{set} = (\mathrm{G}, p, g, \mathrm{H})$, any $f \xleftarrow{\$} \mathrm{F}[\mathrm{X}]_l$ denoted by $f(x) = \sum_{i=0}^{l} a_i \cdot x^i$ and any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(f)$, there exists $(c_i, d_i) \in \mathbb{G}^2$ for all i in $[\![0, l]\!]$ such that:

$$\begin{aligned} \mathsf{pk} = (\{(c_i, d_i)\}_{0\le i\le l}, h) \\ h = g^{\mathsf{sk}} \\ \forall i \in [\![0, l]\!], c_i = g^{r_i} \\ \forall i \in [\![0, l]\!], d_i = h^{r_i} \cdot g^{a_i} \end{aligned}$$

We set:

$$c = \prod_{i=0}^{l} c_i^{x^i}$$

$$d = \frac{\left(\prod_{i=0}^{l} d_i^{x^i}\right)}{g^y}$$

For any $x \in \mathbb{Z}_p^*$ and $(y, \pi) \leftarrow \mathsf{Compute}(\mathsf{pk}, x, \mathsf{sk}, f)$, there exists A, B, θ and ω such that:

$$\begin{aligned} \pi = (\mathrm{A}, \mathrm{B}, \omega) \\ \mathrm{A} = g^\theta \\ \mathrm{B} = c^\theta \\ \omega = \theta + \mathrm{H}(\mathrm{A}||\mathrm{B}) \cdot \mathsf{sk} \end{aligned}$$

We have:

$$d = \frac{\prod_{i=0}^{l} d_i^{x^i}}{g^y} = \frac{\prod_{i=0}^{l} h^{r_i \cdot x^i} \cdot g^{a_i \cdot x^i}}{g^y} = \frac{\left(\prod_{i=0}^{l} \left(g^{r_i \cdot x^i}\right)^{\mathsf{sk}}\right) \cdot g^{f(x)}}{g^y} = \frac{c^{\mathsf{sk}} \cdot g^y}{g^y} = c^{\mathsf{sk}}$$

□

On the other hand:

$$g^\omega = g^{\theta + \mathrm{H}(\mathrm{A}||\mathrm{B}) \cdot \mathsf{sk}} = g^\theta \cdot \left(g^{\mathsf{sk}}\right)^{\mathrm{H}(\mathrm{A}||\mathrm{B})} = \mathrm{A} \cdot h^{\mathrm{H}(\mathrm{A}||\mathrm{B})}$$

$$c^\omega = c^{\theta + \mathrm{H}(\mathrm{A}||\mathrm{B}) \cdot \mathsf{sk}} = c^\theta \cdot \left(c^{\mathsf{sk}}\right)^{\mathrm{H}(\mathrm{A}||\mathrm{B})} = \mathrm{B} \cdot d^{\mathrm{H}(\mathrm{A}||\mathrm{B})}$$

We deduce that $\mathsf{Verify}(\mathsf{pk}, x, y, \pi) = 1$, which conclude the proof.

4.6.2 IND-CFA Security

In this section we show that PIPE is IND-CFA secure.

Lemma 29 PIPE *is IND-CFA secure under the DDH assumption in the random oracle model.*

Proof: For any $l \in \mathbb{N}$, we suppose that there exists $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \textsc{poly}(k)^2$ such that the advantage $\mathsf{Adv}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\mathsf{PIPE},\mathcal{A}}(k)$ is non-negligible and we show that there exists an algorithm $\mathcal{B} \in \textsc{poly}(k)$ such that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}_l}_{\mathrm{ElGamal},\mathcal{B}}(k)$ is non-negligible. $\mathcal{B}$ is built as follows:

- $\mathcal{B}$ receives $((\mathrm{G}, p, g), h)$ as input and runs $(f_0, f_1, \mathsf{st}) \leftarrow \mathcal{A}_1((\mathrm{G}, p, g), \mathbb{Z}_p^*, l)$.
- For all $i \in [\![0, l]\!]$, let $(a_{0,i}, a_{1,i})$ be the respective i^{th} coefficients of f_0 and f_1. $\mathcal{B}$ runs the oracle $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{LR}_b(\cdot,\cdot), *)$ on input $(g^{a_{0,i}}, g^{a_{1,i}})$ and obtains the ElGamal ciphertext (c_i, d_i) of $g^{a_{b,i}}$.
- $\mathcal{B}$ runs $b_* \leftarrow \mathcal{A}_2((\mathrm{G}, p, g), (\{(c_i, d_i)\}_{0 \le i \le l}, h), \mathbb{Z}_p^*, l, \mathsf{st})$. To simulate the oracle $\mathsf{CO}_{\mathsf{CFA}}(\cdot)$ on x to $\mathcal{A}_2$, the algorithm $\mathcal{B}$ computes:

$$c = \prod_{i=0}^{l} c_i^{x^i}$$
$$d = \left(\prod_{i=0}^{l} d_i^{x^i}\right) \cdot \frac{1}{g^{f_0(x)}}$$

  In the real experiment, the proof π is computed as in LogEq (Definition 29). Since this protocol is ZK, there exists a polynomial time simulator Sim in the random oracle model such that the outputs of the simulator come from the same distribution that the outputs of the real proof algorithm. Then $\mathcal{B}$ computes $y = f_0(x)$. If $y = f_1(x)$ it uses $\mathsf{Sim}((h, c, d))$ to compute π and returns (y, π) to $\mathcal{A}_2$. Else it returns $\perp$.

- Finally, $\mathcal{B}$ outputs b_*.

We observe that:

1. The l-IND-CFA experiment is perfectly simulated for $\mathcal{A}$.
2. $\mathcal{B}$ wins the IND-CPA$_l$ experiment if and only if $\mathcal{A}$ wins the l-IND-CFA experiment.

Since $\mathsf{Adv}^{l\text{-}\mathsf{IND\text{-}CFA}}_{\mathsf{PIPE},\mathcal{A}}(k)$ is non-negligible, then $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}_l}_{\mathrm{ElGamal},\mathcal{B}}(k)$ is non-negligible. Since ElGamal cryptosystem is IND-CPA secure under the DDH assumption, $\mathcal{B}$ can be used to break the DDH assumption, which contradicts our hypothesis and concludes the proof. □

4.6.3 Zero-Knowledge

In this section we show that PIPE is zero-knowledge.

Lemma 30 PIPE *is unconditionally zero-knowledge in the random oracle model.*

Proof: Let $\mathrm{P}_{\mathsf{PIPE}}$ the proof inducted by PIPE. We show that for any $l, k \in \mathbb{N}$, $(\mathsf{set}, \mathrm{F}) \leftarrow \mathsf{Set}(l, k)$, $f \in \mathrm{F}[\mathrm{X}]_l$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(f)$ such that $\mathsf{pk} = (\{(c_i, d_i)\}_{0 \le i \le l}, h)$, there exists a simulator denoted $\mathsf{Sim}(\mathsf{pk}, x, y)$ that outputs values in the same distribution as the algorithm $\mathsf{Pro}_{\mathsf{PIPE}}((f, \mathsf{sk}), (\mathsf{pk}, x, y))$.

- sim picks $(\omega, \psi) \xleftarrow{\$} (\mathbb{Z}_p^*)^2$ and computes:

$$\mathrm{A} = \frac{g^{\omega}}{h^{\psi}}$$
$$\mathrm{B} = \frac{\left(\prod_{i=0}^{l} c_i^{x^i}\right)^{\omega}}{\left(\left(\prod_{i=0}^{l} d_i^{x^i}\right) \cdot \frac{1}{g^y}\right)^{\psi}}$$

- It adds the pair of input/output $((\mathrm{A}||\mathrm{B}),\psi)$ to the table of the random oracle H and it returns $(\mathrm{A},\mathrm{B},\omega)$.

Since ω and ψ come from the uniform distribution on $\mathbb{Z}_p^*$, then Sim and $\mathsf{Pro}_{\mathsf{PIPE}}$ induct similar distributions. □

4.6.4 Unforgeability

In this section we show that PIPE is unforgeable.

Lemma 31 PIPE *is unconditionally UNF secure in the random oracle model.*

Proof: The proof π is computed as in LogEq (Definition 29). This NIZKP scheme is unconditionally *sound,* then there exists no polynomial time algorithm that forges a valid proof on false statement with non-negligible probability, *i.e.,* a statement (h,c,d) where $\log_g(h)\neq\log_c(d)$. We show that if there exists $\mathscr{A}=(\mathscr{A}_1,\mathscr{A}_2)\in\mathrm{POLY}(k)^2$ such that $\lambda(k)=\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PIPE},\mathscr{A}}(k)$ is non negligible, then there exists $\mathscr{B}\in\mathrm{POLY}(k)$ that forges a valid proof of an statment (h,c,d) where $\log_g(h)\neq\log_c(d)$. It contradicts the soundness of LogEq, which concludes the proof. $\mathscr{B}(\mathrm{G},p,g)$ works as follows:

- It chooses a hash functipon H, sets $\mathsf{set}=(\mathrm{G},p,g,\mathrm{H})$ and $\mathrm{F}=\mathbb{Z}_p^*$, runs $(f,\mathsf{st})\leftarrow\mathscr{A}_1(\mathsf{set},\mathrm{F})$, $(\mathsf{pk},\mathsf{sk})\leftarrow\mathsf{Gen}(f)$ where $\mathsf{pk}=(\{(c_i,d_i)\}_{0\leq i\leq l},h)$ and $(x,y,\pi)\leftarrow\mathscr{A}_2(\mathsf{set},\mathsf{sk},\mathsf{pk},\mathrm{F},f,\mathsf{st})$ where $\pi=(\mathrm{A},\mathrm{B},\omega)$.

- $\mathscr{B}$ computes:

$$c=\prod_{i=0}^{l}c_i^{x^i}$$
$$d=\left(\prod_{i=0}^{l}d_i^{x^i}\right)\cdot\frac{1}{g^y}$$

Then it builds the statement $\mathsf{pk}^*=(h,c,d)$. It returns pk^* together with the proof π.

We observe that since $\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PIPE},\mathscr{A}}(k)$ is non negligible then the probability that $f(x)\neq y$ and $1\leftarrow\mathsf{Verify}(\mathsf{pk},x,y,\pi)$ is non-negligible. Moreover:

i) $f(x)\neq y\Rightarrow d=\left(\prod_{i=0}^{l}d_i^{x^i}\right)\cdot\frac{1}{g^y}=c^{\mathsf{sk}}\cdot g^{f(x)-y}\neq c^{\mathsf{sk}}$. Then $\log_g(h)\neq\log_c(d)$.

ii) $1\leftarrow\mathsf{Verify}(\mathsf{pk},x,y,\pi)\Rightarrow g^{\omega}=\mathrm{A}\cdot h^{\mathrm{H}(\mathrm{A}||\mathrm{B})}$ and $c^{\omega}=\mathrm{B}\cdot d^{\mathrm{H}(\mathrm{A}||\mathrm{B})}$. Then π is a valid proof.

Then $\mathscr{B}$ returns a valid proof of a false statment with non-negligible probability $\lambda(k)$. □

4.6.5 Security of PIPE

Finally, we have the following theorem.

Theorem 26 PIPE *is is ZK, IND-CFA, PP, WPP and UNF secure under the DDH assumption in the random oracle model.*

Proof: Lemma 29 show that PIPE is IND-CFA, and Lemma 30 show that PIPE is ZK. Using Theorem 24, we deduce that PIPE is PP secure, and using Theorem 21, we deduce that PIPE is WPP secure. Finally, Lemma 31 show that PIPE is UNF secure, which concludes the proof. □

4.7 Comparison of PIPE and $\mathsf{PolyCommit}_{\mathsf{Ped}}$

We compare the advantages and the disadvantages of PIPE and $\mathsf{PolyCommit}_{\mathsf{Ped}}$ [KZG10]. Table 4.1 resumes our comparison.

The PIPE verification algorithm is in $\mathcal{O}(k)$ and the $\mathsf{PolyCommit}_{\mathsf{Ped}}$ one is in constant time. However, the $\mathsf{PolyCommit}_{\mathsf{Ped}}$ verification algorithm requires several pairing computations which are significantly costly in terms of computation time whereas PIPE only requires exponentiations and multiplications in a prime order group. Consequently, PIPE is more efficient than $\mathsf{PolyCommit}_{\mathsf{Ped}}$ for polynomial of sufficiently small degree.

The main advantage of $\mathsf{PolyCommit}_{\mathsf{Ped}}$ is the constant size of the verification key, whereas the size of the verification key of PIPE is in $\mathcal{O}(k)$. However, the size of the public setup of the scheme $\mathsf{PolyCommit}_{\mathsf{Ped}}$ is in $\mathcal{O}(k)$ whereas the PIPE one is in constant size. Since the client knows both the verification key and the public setup, $\mathsf{PolyCommit}_{\mathsf{Ped}}$ is advantageous only if each client has access to several polynomials simultaneously.

PIPE is secure under the DDH assumption whereas $\mathsf{PolyCommit}_{\mathsf{Ped}}$ is secure under the t-SDH assumption. Note that finding a scheme that is secure under a weaker assumption than t-SDH was left as an open problem by Kate *et al.* [KZG10]. Finally, note that $\mathsf{PolyCommit}_{\mathsf{Ped}}$ is secure in the standard model whereas PIPE is secure in the random oracle model only. A simple way to obtain a version of PIPE in the standard model is to use the interactive version of LogEq. In return, it requires an interaction between the client and the server during the evaluation algorithm.

	Setup size	Key size	Verif. cost	Pairing	Assumption
PIPE	$\mathcal{O}(1)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$	Pairing free	DDH
$\mathsf{PolyCommit}_{\mathsf{Ped}}$ [KZG10]	$\mathcal{O}(k)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Pairing based	t-SDH

Table 4.1: Comparison of PIPE and $\mathsf{PolyCommit}_{\mathsf{Ped}}$.

4.8 CFA Security for Commitments to Polynomials

Our scheme can be used as a *commitment to polynomials* scheme [KZG10] that is IND-CFA secure. This primitive, introduced by Kate *et al.*, is very close to the private polynomial evaluation primitive. The only difference is that the polynomial is committed by the server, and the server can reveal *a posteriori* the polynomial to the clients by opening the commitment. We give an overview of such a scheme in Figure 4.3. To commit a polynomial f, the committer computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(f)$ and returns the commitment pk to the client corresponding to the encryption of coefficients of the polynomial f. Then, the client sends his data to the committer (x in Figure 4.3) and receives the results with validity proof ($(f(x), \pi)$ in Figure 4.3). To open the commitment, the committer reveals to the client the key sk together with f ($\mathsf{open}(\mathsf{pk}, \mathsf{sk}, f)$ in Figure 4.3), then the client can open all the ElGamal ciphertexts of pk and check they encrypt g^{a_i}, where a_i are the coefficients of f.

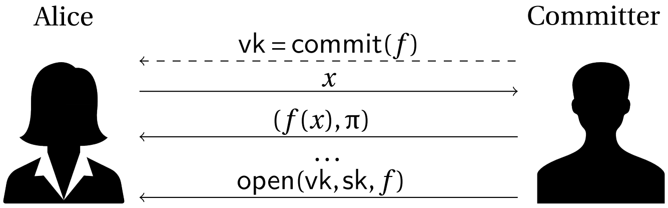

Alice
Committer
$\mathsf{vk} = \mathsf{commit}(f)$
x
$(f(x), \pi)$
...
$\mathsf{open}(\mathsf{vk}, \mathsf{sk}, f)$

Figure 4.3: PIPE scheme used as a commitment to polynomials scheme [KZG10].

4.9 Anonymous Private Polynomial Evaluation

In practice, the company does not allow anybody to interact freely with the computation server. The company distributes authentication keys to the clients, and the server uses a protocol to authenticate the client at the beginning of each interaction. It allows the server to verify that a client does not evaluate more than l points, where l is the degree of the polynomial. However, for a lot of applications, preserving the privacy of the clients is important. Guo *et al.* [GFLL15] propose an anonymous authentication mechanism for their scheme, which is broken and fixed by Gajera *et al.* [GND16].

We remark that anonymous authentication for PPE prevents the server from knowing how much points of the polynomial it gives to each client, leading to security issues. To solve this problem, we suggest that the server uses *l-times anonymous authentication* [TFS04]: this primitive allows a client to anonymously authenticate l times. If a client exceeds this limit, the server can identify him. Using such a scheme, the server can refuse to respond if the client asks more points than allowed, and the privacy of honest clients is preserved.

4.10 Conclusion

In this chapter, we gave a formal definition for a primitive called private polynomial evaluation. This primitive allows a company to delegate computations on a secret polynomial for clients in a verifiable way. In essence, the client sends x and receives y from the server along with a proof of $y = f(x)$, even though he does not know the polynomial f.

We first gave a critical cryptanalysis of two schemes of the literature. Then we proposed a security model including a new security property called indistinguishability against chosen function attack ($\mathsf{IND\text{-}CFA}$). We built a private polynomial evaluation scheme called PIPE which is secure in this model. We showed that the scheme $\mathsf{PolyCommit}_{\mathsf{Ped}}$ proposed in [KZG10] also is $\mathsf{IND\text{-}CFA}$-secure, and we compared it with PIPE.

In the future, we aim at designing a scheme combining the advantages of both PIPE and $\mathsf{PolyCommit}_{\mathsf{Ped}}$, *i.e.,* that is pairing free and that uses constant size verification keys.

Another possible extension is to add practical privacy mechanism to protect the data sent by the clients. In their paper [GFLL15, GND16], Guo *et al.* and Gajera *et al.* investigate this problem and propose schemes where the client data is encrypted such that the server learns nothing about it. Unfortunately, in this chapter we showed these schemes have critical security weaknesses: we propose an attack that allows the client to recover the secret polynomial in a single query. We aim at extending our security model to capt the privacy of the client data and designing a scheme that will be secure in this model.

Chapter 5

Verifiable Ring Signature Revisited

Contents

A *Verifiable Ring Signature* scheme allows their users to sign messages anonymously within a group such that a user can prove *a posteriori* to a verifier whether it is the author of a given signature or not. In this chapter, we revisit this primitive. We improve the proof capabilities of the users, we give a complete security model and we design an efficient and secure scheme called EVeR. Finally, we prove its security under the Decisional Diffie-Hellman assumption in the random oracle model. This work has been conducted in collaboration with Pascal Lafourcade and has been published in the paper "Unlinkable and Strongly Accountable Sanitizable Signatures from Verifiable Ring Signatures" at the CANS 2017 conference.

5.1 Introduction

Ring signature is a well-studied cryptographic primitive introduced by Rivest *et al.* in [RST01], where some users can sign anonymously within a group of users. Such a scheme is said to be *verifiable* [LW03] when any user of the group can prove *a posteriori* he is the signer of a ring signature. However, existing schemes do not allow the user to prove he is not the signer of the signature. In this chapter, we fill this gap by given a new definition of verifiable ring signature where the user can prove whether he is the signer of the signature or not.

In there paper [RST01], Rivest *et al.* give the following application to ring signatures. A member of a national agency, such as a high-ranking White House official, sends an official document to the press on behalf of the agency, but does not want to reveal his identity within the agency. Using the public keys of all the agency members, he uses a ring signature scheme to authenticate the document on behalf of the agency. Now we consider that the agency employs inspectors whose job is to verify the information revealed on behalf of the agency. Using verifiable ring signature, this inspector can ask to the members who signed the document to reveal his identity and to prove that he his the signer. Nonetheless, if the real signer does not reveal his identity, the inspector risks to suspect another member of the agency. Until know, existing verifiable ring signature schemes did not allow this member to prove his innocence. However, using our stronger definition, this member can also prove to the inspecter that he is not the signer of the message.

Moreover, we present another application of this primitive in the next chapter: we show how to build a generic sanitizable signature scheme from any verifiable ring signature scheme.

5.1.1 Functionalities

A verifiable ring signature scheme has the following functionalities:

System initialization: A user generates the public setup. As in ring signatures, each user of the group generates his own public and private key using the key generation algorithm.

Signature and verification: Any user can anonymously sign a message within a group of users using his secret key and the signature algorithm. This signature can be verified by anyone using the set of the public keys of the group users and the verification algorithm.

Proof and judgment: A user can prove that he is (resp. is not) the signer of a given signature using his secret key and the proof algorithm. This proof can be verified by anyone using the user public key and the judge algorithm

5.1.2 Security Goals

Verifiable ring signatures require the same security properties as the ring signatures, *i.e.,* the unforgeability and the anonymity. Furthermore, it requires two additional security properties to assure the soundness of the proofs produced by the users, namely the accountability and the non-seizability.

Unforgeability: A verifiable ring signature scheme is said to be unforgeable when a user who is not in a group cannot forge a valid signature for this group.

Anonymity: A verifiable ring signature scheme is said to be anonymous when nobody can distinguish who is the signer of a given message within the group of users.

Accountability: A verifiable ring signature scheme is said to be accountable when no user can sign a message and prove that he is not the signer.

Non-seizability: A verifiable ring signature scheme is said to be non-seizable when no user can prove that he is the signer of a signature if it is not true, and no user can forge a signature such that somebody else can prove he is the signer instead of the user.

5.1.3 Contributions

We extend this definition of verifiable ring signature to allow any user to prove that he is not the signer of a message. We then give the first formal security model for verifiable ring signatures. Finally, We design an efficient and secure verifiable ring signature scheme called EVeR under the DDH assumption in the random oracle model.

5.1.4 Related Works

Ring signatures [RST01] were introduced by Rivest *et al.* and their security models were defined in [BKM06]. This primitive allows the users to sign anonymously within a group. *Verifiable ring signatures* [LW03] were introduced by Lv and Wang. This primitive is the same as the ring signature, except that a user can prove whether he is the signer of a ring signature. The author of [LW03] gives a scheme that is based on the discrete logarithm problem. Two other schemes were proposed by Wand *et al.* [WMZW11] and by Changlung *et al.* [CYD06]. The first one is based on the Nyberg-Rueppel signature scheme and the second one is a generic construction based on multivariate public key cryptosystems. In these three schemes, a user can prove that he is the signer of a signature, however, he has no way to prove that he is not the signer, and it seems to be nontrivial to add this property to these schemes. *Convertible ring signatures* [LWH05] are very close to verifiable ring signatures: they allow the signer of an anonymous (ring) signature to transform it into a standard signature (*i.e.,* a *desanonymized* signature). It can be used as a verifiable ring signature because the desanonymized signature can be viewed as a proof that the user is the signer of a given message. However, in this chapter we propose a stronger definition of verifiable ring signature where a user also can prove he is not the signer of a message, and this property cannot be achieved using convertible signatures.

A *Revocable-iff-Linked Ring Signature* [ASY06] (also called *List Signature* [CSST06]) is a kind of ring signature that has the following property: if a user signs two messages for the same *event-id*, then it is possible to link these signatures and the user's identity is publicly revealed. It can be used to design a verifiable ring signature scheme in our model: to prove whether he is the signer of a given message, the user signs a second message using the same event-id. If the two signatures are linked, then the judge is convinced that the user is the signer, else he is convinced that the user is not the signer. However, revocable-iff-linked ring signatures require security properties that are too strong for verifiable ring signatures (linkability and traceability) and it would result in less efficient schemes.

5.2 Formal Definitions

In this section, we formally define the Verifiable Ring Signatures (VRS) and their security properties.

5.2.1 Verifiable Ring Signature

A VRS is composed of six algorithms. Init, Gen, Sig and Ver are defined as in the usual ring signature definitions: Gen generates public and private keys, Sig anonymously signs a message according to a set of public keys and Ver verifies the soundness of a signature. On the other hand, the algorithm Proof allows a user to prove whether he is the signer of a message or not, and the algorithm Judge allows anybody to verify the proofs outputted by Proof.

Definition 59 (Verifiable Ring Signature (VRS)) *A* Verifiable Ring Signature *(VRS) scheme is a tuple of six algorithms* $\mathsf{V} = (\mathsf{Init}, \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver}, \mathsf{Proof}, \mathsf{Judge})$ *defined by:*

$\mathsf{Init}(k)$: *It returns a setup value* set.
$\mathsf{Gen}(\mathsf{set})$: *It returns a pair of signer public/private keys* $(\mathsf{pk}, \mathsf{sk})$.
$\mathsf{Sig}(\mathsf{sk}, \mathsf{L}, m)$: *It returns a signature* σ *of the message* m *according to the set of public keys* L.

$\mathsf{Ver}(\mathrm{L}, m, \sigma)$: *It returns a bit b.*

$\mathsf{Proof}(\mathrm{L}, m, \sigma, \mathsf{pk}, \mathsf{sk})$: *It returns a proof* π.

$\mathsf{Judge}(\mathrm{L}, m, \sigma, \mathsf{pk}, \pi)$: *It returns a bit b or the bottom symbol* $\perp$*: if* $b = 1$ *(resp.* 0*) then* π *proves that* σ *was (resp. was not) generated by the signer corresponding to the public key* pk*. It outputs* $\perp$ *when the proof is not well formed.*

Moreover a VRS is said to be correct *when the following equations hold for any* $k \in \mathbb{N}$*, any number of users* $n \in \mathbb{N}$*, any* $l \in [\![1, n]\!]$*, any* $l' \in [\![1, n]\!] \backslash \{l\}$*, and any* $m \in \{0, 1\}^*$*:*

$$\Pr\left[\begin{array}{l} \mathsf{set} \leftarrow \mathit{Init}(k); \forall i \in [\![1, n]\!], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathit{Gen}(\mathsf{set}); \\ \sigma \leftarrow \mathit{Sig}(\mathsf{sk}_l, \{\mathsf{pk}_i\}_{1 \le i \le n}, m); \\ b \leftarrow \mathit{Ver}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma); \end{array} : b = 1\right] = 1$$

$$\Pr\left[\begin{array}{l} \mathsf{set} \leftarrow \mathit{Init}(k); \forall i \in [\![1, n]\!], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathit{Gen}(\mathsf{set}); \\ \sigma \leftarrow \mathit{Sig}(\mathsf{sk}_l, \{\mathsf{pk}_i\}_{1 \le i \le n}, m); \\ \pi \leftarrow \mathit{Proof}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma, \mathsf{pk}_l, \mathsf{sk}_l); \\ b \leftarrow \mathit{Judge}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma, \mathsf{pk}_l, \pi); \end{array} : b = 1\right] = 1$$

$$\Pr\left[\begin{array}{l} \mathsf{set} \leftarrow \mathit{Init}(k); \forall i \in [\![1, n]\!], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathit{Gen}(\mathsf{set}); \\ \sigma \leftarrow \mathit{Sig}(\mathsf{sk}_l, \{\mathsf{pk}_i\}_{1 \le i \le n}, m); \\ \pi \leftarrow \mathit{Proof}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma, \mathsf{pk}_{l'}, \mathsf{sk}_{l'}); \\ b \leftarrow \mathit{Judge}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma, \mathsf{pk}_{l'}, \pi); \end{array} : b = 0\right] = 1$$

5.2.2 Unforgeability

We first adapt the unforgeability property of ring signatures to verifiable ring signatures. Informally, a VRS is unforgeable when no adversary is able to forge a signature without any secret key corresponding to one of the public keys in the group of users. In this model, the adversary has access to a signature oracle $\mathsf{Sig}(\cdot, \cdot, \cdot)$ (that outputs signatures of chosen messages for chosen users in the ring) and a proof oracle $\mathsf{Proof}(\cdot, \cdot, \cdot, \cdot, \cdot)$ (that outputs proofs as the algorithm $\mathsf{V.Proof}$ for chosen signatures and chosen users). The adversary succeeds when it outputs a valid signature that was not already generated by the signature oracle.

Definition 60 (Unforgeability) *Let* Π *be a VRS,* n *and* k *be two integers and* $\mathcal{A} \in \text{POLY}(k)$ *be an algorithm. Let the two following oracles be:*

$\mathsf{Sig}(\cdot, \cdot, \cdot)$**:** *On input* (L, l, m)*, if* $1 \le l \le n$ *then it runs* $\sigma \leftarrow \mathsf{Sig}(m, \mathrm{L}, \mathsf{sk}_l)$ *and returns* σ*, else it returns* $\perp$.

$\mathsf{Proof}(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** *On input* $(\mathrm{L}, m, \sigma, l)$*, if* $1 \le l \le n$ *then it runs* $\pi \leftarrow \mathsf{Proof}(\mathrm{L}, m, \sigma, \mathsf{pk}_l, \mathsf{sk}_l)$ *and returns* π*, else it returns* $\perp$.

We define the n-EUF-CMA experiment as follows, where q_{S} *is the number of calls to the oracle* $\mathsf{Sig}(\cdot, \cdot, \cdot)$ *and* σ_i *is the* i^{th} *signature outputted by the oracle* $\mathsf{Sig}(\cdot, \cdot, \cdot)$*:*

```
$\mathsf{Exp}_{\Pi,\mathcal{A}}^{n\text{-}\mathbf{EUF\text{-}CMA}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$\forall 1 \le i \le n, (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{set})$
$(\mathrm{L}_*, \sigma_*, m_*) \leftarrow \mathcal{A}^{\mathsf{Sig}(\cdot,\cdot,\cdot),\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\{\mathsf{pk}_i\}_{1 \le i \le n})$
if $\mathsf{Ver}(\mathrm{L}_*, \sigma_*, m_*) = 1$ and $\mathrm{L}_* \subseteq \{\mathsf{pk}_i\}_{1 \le i \le n}$ and $\forall\ i \in [\![1, q_{\mathrm{S}}]\!], \sigma_i \neq \sigma_*$
then return 1, else 0
```

We define the n-EUF-CMA advantage of $\mathcal{A}$ *against* Π *as follows:*

$$\mathsf{Adv}_{\Pi,\mathcal{A}}^{n\text{-}EUF\text{-}CMA}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}_{\Pi,\mathcal{A}}^{n\text{-}EUF\text{-}CMA}(k)\right]$$

We define the n-EUF-CMA advantage against Π as follows:

$$\mathsf{Adv}_{\Pi}^{n\text{-}EUF\text{-}CMA}(k) = \max_{\mathscr{A}\in\text{POLY}(k)} \left\{\mathsf{Adv}_{\Pi,\mathscr{A}}^{n\text{-}EUF\text{-}CMA}(k)\right\}$$

Π is said to be n-EUF-CMA secure when the n-EUF-CMA advantage against Π is negligible. Π is said to be unforgeable *when it is $t(k)$-EUF-CMA secure for any polynomial t.*

5.2.3 Anonymity

We adapt the anonymity property of ring signatures to verifiable ring signatures. Informally, a VRS is anonymous when no adversary is able to link a signature to the corresponding user. The adversary has access to the signature oracle and the proof oracle. During a first phase, it chooses two honest users in the ring, and in the second phase, it has access to a challenge oracle denoted $\mathsf{LRSO_b}(d_0, d_1, \cdot, \cdot)$ that outputs signatures of chosen messages using the secret key of one of the two chosen users. The adversary succeeds if he guesses which user is chosen by the challenge oracle. Note that if the adversary uses the proof oracle on the signatures generated by the challenge oracle then he loses the experiment.

Definition 61 (Anonymity) *Let Π be a VRS, n and k be two integers and $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2) \in \text{POLY}(k)^2$ be a two-party algorithm. Let the following oracle be:*

$\mathsf{LRSO_b}(d_0, d_1, \cdot, \cdot)$**:** *On input (m, L), if $\{\mathsf{pk}_{d_0}, \mathsf{pk}_{d_1}\} \subseteq \mathrm{L}$ then this oracle runs $\sigma \leftarrow \mathsf{Sig}(m, \mathrm{L}, \mathsf{sk}_{d_b})$ and returns σ, else it returns $\perp$.*

We define the n-ano experiment as follows, where $\mathsf{Sig}(\cdot,\cdot,\cdot)$ and $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ are defined as in Def. 60 and where q_S (resp. q_P) is the number of calls to the oracle $\mathsf{Sig}(\cdot,\cdot,\cdot)$ (resp. $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$), $(\mathrm{L}_i, m_i, \sigma_i, l_i)$ is the i^{th} query sent to oracle $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ and σ'_j is the j^{th} signature outputted by the oracle $\mathsf{LRSO_b}(d_0, d_1, \cdot, \cdot)$:

```
$\mathsf{Exp}_{\Pi,\mathscr{A}}^{n\text{-}\mathbf{ano}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$\forall 1 \le i \le n, (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{set})$
$(d_0, d_1, \mathsf{st}) \leftarrow \mathscr{A}_1^{\mathsf{Sig}(\cdot,\cdot,\cdot),\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\{\mathsf{pk}_i\}_{1\le i\le n})$
$b \xleftarrow{\$} \{0,1\}$
$b_* \leftarrow \mathscr{A}_2^{\mathsf{Sig}(\cdot,\cdot,\cdot),\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{LRSO_b}(d_0,d_1,\cdot,\cdot)}(\mathsf{st}, \{\mathsf{pk}_i\}_{1\le i\le n})$
if $(b = b_*)$ and $(\forall\ i, j \in [\![1, \max(q_\mathrm{S}, q_\mathrm{P})]\!], (\sigma_i \neq \sigma'_j)$ or $(l_i \neq d_0$ and $l_i \neq d_1))$
then return $1$, else $0$
```

We define the n-ano advantage of $\mathscr{A}$ against Π as follows:

$$\mathsf{Adv}_{\Pi,\mathscr{A}}^{n\text{-}ano}(k) = \left|\frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{Exp}_{\Pi,\mathscr{A}}^{n\text{-}ano}(k)\right]\right|$$

We define the n-ano advantage against Π as follows:

$$\mathsf{Adv}_{\Pi}^{n\text{-}ano}(k) = \max_{\mathscr{A}\in\text{POLY}(k)^2} \left\{\mathsf{Adv}_{\Pi,\mathscr{A}}^{n\text{-}ano}(k)\right\}$$

Π is said to be n-ano secure when the n-ano advantage against Π is negligible. Π is said to be anonymous *when it is $t(k)$-ano secure for any polynomial t.*

5.2.4 Accountability

We consider an adversary that has access to a proof oracle and a signature oracle. A VRS is accountable when no adversary is able to forge a signature σ (that was not outputted by the signature oracle) together with a proof that it is not the signer of σ. Note that the ring used to generate σ must contain at most one public key that does not come from a honest user, thus the adversary knows at most one secret key that corresponds to a public key in the ring.

Definition 62 (Accountability) *Let Π be a VRS, n and k be two integers and $\mathscr{A} \in \text{POLY}(k)$ be an algorithm. We define the n-acc experiment as follows, where* $\mathsf{Sig}(\cdot,\cdot,\cdot)$ *and* $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *are defined as in Def. 60 and where q_S is the number of calls to the oracle* $\mathsf{Sig}(\cdot,\cdot,\cdot)$ *and σ_i is the i^{th} signature outputted by the oracle* $\mathsf{Sig}(\cdot,\cdot,\cdot)$*:*

```
$\mathsf{Exp}^{n\text{-}\mathbf{acc}}_{\Pi,\mathscr{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$\forall 1 \le i \le n, (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{set})$
$(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) \leftarrow \mathscr{A}^{\mathsf{Sig}(\cdot,\cdot,\cdot),\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\{\mathsf{pk}_i\}_{1\le i\le n})$
if $(\mathrm{L} \subseteq \{\mathsf{pk}_i\}_{1\le i\le n} \cup \{\mathsf{pk}_*\})$ and $(\mathsf{Ver}(\mathrm{L}_*, \sigma_*, m_*) = 1)$ and $(\mathsf{Judge}(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) = 0)$
    and $(\forall\ i \in [\![1, q_S]\!], \sigma_i \neq \sigma_*)$
then return 1, else 0
```

We define the n-acc advantage of $\mathscr{A}$ against Π as follows:

$$\mathsf{Adv}^{n\text{-}acc}_{\Pi,\mathscr{A}}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}^{n\text{-}acc}_{\Pi,\mathscr{A}}(k)\right]$$

We define the n-acc advantage against Π as follows:

$$\mathsf{Adv}^{n\text{-}acc}_{\Pi}(k) = \max_{\mathscr{A}\in\text{POLY}(k)} \left\{\mathsf{Adv}^{n\text{-}acc}_{\Pi,\mathscr{A}}(k)\right\}$$

Π is said to be n-acc secure when the n-acc advantage against Π is negligible. Π is said to be accountable *when it is $t(k)$-acc secure for any polynomial t.*

5.2.5 Non-seizability

We distinguish two experiments for this property: the first experiment, denoted non-sei-1, considers an adversary that has access to a proof oracle and a signature oracle. Its goal is to forge a valid signature with a proof that the signer is another user in the ring.

Definition 63 (n-non-sei-1 security) *Let Π be a VRS, n and k be two integers and $\mathscr{A} \in \text{POLY}(k)$ be an algorithm. We define the n-non-sei-1 experiment as follows, where the oracles* $\mathsf{Sig}(\cdot,\cdot,\cdot)$ *are defined as in Def. 60 and* $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *and where q_S is the number of calls to the oracle* $\mathsf{Sig}(\cdot,\cdot,\cdot)$ *and (L_i, l_i, m_i) (resp. σ_i) is the i^{th} query to the oracle* $\mathsf{Sig}(\cdot,\cdot,\cdot)$ *(resp. signature outputted by this oracle):*

```
$\mathsf{Exp}^{n\text{-}\mathbf{non\text{-}sei\text{-}1}}_{\Pi,\mathscr{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$\forall 1 \le i \le n, (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{set})$
$(\mathrm{L}_*, m_*, \sigma_*, l_*, \pi_*) \leftarrow \mathscr{A}^{\mathsf{Sig}(\cdot,\cdot,\cdot),\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\{\mathsf{pk}_i\}_{1\le i\le n})$
if $(\mathsf{Ver}(\mathrm{L}_*, \sigma_*, m_*) = 1)$ and $(\mathsf{Judge}(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}_{l_*}, \pi_*) = 1)$ and
    $(\forall\ i \in [\![1, q_S]\!], \sigma_i \neq \sigma_*)$
then return 1, else 0
```

We define the n-non-sei-1 advantage of $\mathscr{A}$ against Π as follows:

$$\mathsf{Adv}^{n\text{-}non\text{-}sei\text{-}1}_{\Pi,\mathscr{A}}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}^{n\text{-}non\text{-}sei\text{-}1}_{\Pi,\mathscr{A}}(k)\right]$$

We define the n-non-sei-1 advantage against Π as follows:

$$\mathsf{Adv}^{n\text{-}non\text{-}sei\text{-}1}_{\Pi}(k) = \max_{\mathscr{A}\in\text{POLY}(k)} \left\{\mathsf{Adv}^{n\text{-}non\text{-}sei\text{-}1}_{\Pi,\mathscr{A}}(k)\right\}$$

Π is said to be n-non-sei-1 secure when the n-non-sei-1 advantage against Π is negligible. Π is said to be non-sei-1 secure when it is $t(k)$-non-sei-1 secure for any polynomial t.

The second experiment, denoted non-sei-2, considers an adversary that has access to a proof oracle and a signature oracle and that receives the public key of a honest user as input. The goal of the adversary is to forge a signature σ such that the proof algorithm ran by the honest user returns a proof that σ was computed by the honest user (*i.e.*, the judge algorithm returns 1) or a non-valid proof (*i.e.*, the judge algorithm returns ⊥). Moreover, the signature σ must not come from the signature orale.

Definition 64 (Non-seizability) *Let Π be a VRS, n and k be two integers and $\mathcal{A} \in \text{POLY}(k)$ be an algorithm. We define the n-non-sei-2 experiment as follows, where* $\mathsf{Sig}(\cdot,\cdot,\cdot)$ *and* $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *are defined as in Def. 60 and where q_S is the number of calls to the oracle* $\mathsf{Sig}(\cdot,\cdot,\cdot)$ *and σ_i is the i^{th} signature outputted by the oracle* $\mathsf{Sig}(\cdot,\cdot,\cdot)$*:*

```
$\mathsf{Exp}^{n\text{-}\mathbf{non\text{-}sei\text{-}2}}_{\Pi,\mathcal{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{set})$
$(\mathrm{L}_*, m_*, \sigma_*) \leftarrow \mathcal{A}^{\mathsf{Sig}(\cdot,\cdot,\cdot),\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk})$
$\pi \leftarrow \mathsf{Proof}(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}, \mathsf{sk})$
if $(\mathsf{Ver}(\mathrm{L}_*, \sigma_*, m_*) = 1)$ and $(\mathsf{Judge}(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) \neq 0)$ and $(\forall\ i \in [\![1, q_S]\!], \sigma_i \neq \sigma_*)$
then return $1$, else $0$
```

We define the n-non-sei-2 advantage of $\mathcal{A}$ against Π as follows:

$$\mathsf{Adv}^{n\text{-}non\text{-}sei\text{-}2}_{\Pi,\mathcal{A}}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}^{n\text{-}non\text{-}sei\text{-}2}_{\Pi,\mathcal{A}}(k)\right]$$

We define the n-non-sei-2 advantage against Π as follows:

$$\mathsf{Adv}^{n\text{-}non\text{-}sei\text{-}2}_{\Pi}(k) = \max_{\mathcal{A}\in\text{POLY}(k)} \left\{\mathsf{Adv}^{n\text{-}non\text{-}sei\text{-}2}_{\Pi,\mathcal{A}}(k)\right\}$$

Π is said to be n-non-sei-2 secure when the n-non-sei-2 advantage against Π is negligible. Π is said to be non-sei-2 secure when it is $t(k)$-non-sei-2 secure for any polynomial t. Π is non-seizable *when it is both non-sei-1 and non-sei-2 secure.*

5.3 EVeR: an Efficient Verifiable Ring Signature Scheme

We present our verifiable ring signature scheme called EVeR (for *Efficient VErifiable Ring signature*). It works as follows: the signer produces an anonymous commitment from his secret key and the message (*i.e.*, a commitment that leaks no information about the user public key), then he proves that this commitment was produced from a secret key corresponding to one of the public keys of the group members using a zero-knowledge proof system. Note that the same methodology was used to design several ring signature schemes of the literature [ASY06, CL06, HKS10, CSST06]. Moreover, to prove that he is (resp. he is not) the signer of a message, the user proves that the commitment was (resp. was not) produced from the secret corresponding to his public key using a zero-knowledge proof system. Our scheme is based on the DDH assumption and uses a non-interactive zero-knowledge proof of equality of two discrete logarithms out of n elements. In this section, we first show how to build such a proof system, then we give our scheme EVeR.

5.3.1 Proof of Equality of Two Discrete Logarithms Out of n Elements

Let $\mathbb{G}$ be a group of prime order p, $g \in \mathbb{G}$ be a generator and n be an integer, and let the binary relation $\mathcal{R}_n$ be such that for any statement $\mathsf{pk}_* = \{(h_i, y_i, z_i)\}_{1\leq i\leq n}$:

$$(\mathsf{pk}_*, \mathsf{sk}) \in \mathcal{R}_n \Leftrightarrow (\forall i \in [\![1,n]\!], (h_i, y_i, z_i) \in \mathbb{G}^3) \wedge (\exists i \in [\![1,n]\!], (g^{\mathsf{sk}} = y_i) \wedge (h_i^{\mathsf{sk}} = z_i))$$

Firstly, we remark that $\mathscr{R}_1$ is the same relation as in the proof system LogEq (Definition 29). To transform this proof system into a generic one for any relation $\mathscr{R}_n$, we use the Cramer-Damgård-Schoenmakers transformation (Definition 32). The final step is to transform it into a non-interactive proof system. To do that, we use the Fiat-Shamir transformation (Definition 31). We obtain the following proof system.

Definition 65 (LogEq_n **non-interactive proof system)** *Let n be an integer. The LogEq_n non-interactive proof system $\mathsf{LogEq}_n = (\mathsf{L.Set}_n, \mathsf{L.Pro}_n, \mathsf{L.Ver}_n)$ is defined as follows:*

$\mathsf{L.Set}_n(k)$**:** *It generates a prime order group setup $(\mathbb{G}, p, g)$, a hash function $\mathrm{H} : \{0,1\}^* \to \mathbb{Z}_p^*$ and sets $\mathsf{set} = (\mathbb{G}, p, g, \mathrm{H})$, then it generates the binary relation $\mathscr{R}_n$ such that for any statement $\mathsf{pk}_* = \{(h_i, y_i, z_i)\}_{1 \leq i \leq n}$:*

$$(\mathsf{pk}_*, \mathsf{sk}) \in \mathscr{R}_n \Leftrightarrow (\forall i \in [\![1,n]\!], (h_i, y_i, z_i) \in \mathbb{G}^3) \wedge (\exists i \in [\![1,n]\!], (g^{\mathsf{sk}} = y_i) \wedge (h_i^{\mathsf{sk}} = z_i))$$

It returns $(\mathsf{set}, \mathscr{R}_n)$.

$\mathsf{L.Pro}_n(\mathsf{sk}, \mathsf{pk}_*)$. *It parses $\mathsf{pk}_* = \{(h_i, y_i, z_i)\}_{1 \leq i \leq n}$. We denote by j the integer that verifies $\mathsf{sk} = \log_g(y_j)$ $= \log_{h_j}(z_j)$. This algorithm picks $r_j \xleftarrow{\$} \mathbb{Z}_p^*$, and computes $\mathrm{R}_j = g^{r_j}$ and $\mathrm{S}_j = h_j^{r_j}$. For all $i \in [\![1,n]\!] \backslash \{j\}$, it picks $c_i \xleftarrow{\$} \mathbb{Z}_p^*$ and $\gamma_i \xleftarrow{\$} \mathbb{Z}_p^*$, and computes:*

$$\mathrm{R}_i = \frac{g^{\gamma_i}}{y_i^{c_i}} \qquad\qquad \mathrm{S}_i = \frac{h_i^{\gamma_i}}{z_i^{c_i}}$$

It sets $\mathrm{R} = (\mathrm{R}_i)_{1 \leq i \leq n}$ and computes $c = \mathrm{H}(\mathrm{R}||\mathsf{pk}_*)$. It then computes:

$$c_j = \frac{c}{\prod_{i=1; i \neq j}^{n} c_i}$$

It computes $\gamma_j = r_j + c_j \cdot \mathsf{sk}$ and outputs $\pi = \{(\mathrm{R}_i, \mathrm{S}_i, c_i, \gamma_i)\}_{1 \leq i \leq n}$.

$\mathsf{L.Ver}_n(\mathsf{pk}_*, \pi)$. *It parses $\mathsf{pk}_* = \{(h_i, y_i, z_i)\}_{1 \leq i \leq n}$ and $\pi = \{(\mathrm{R}_i, \mathrm{S}_i, c_i, \gamma_i)\}_{1 \leq i \leq n}$. It sets $\mathrm{R} = (\mathrm{R}_i)_{1 \leq i \leq n}$. If:*

$$\mathrm{H}(\mathrm{R}||\mathsf{pk}_*) \neq \prod_{i=1; i \neq j}^{n} c_i$$

then it returns 0. Else if there exists $i \in [\![1,n]\!]$ such that $g_i^{\gamma_i} \neq \mathrm{R}_i \cdot y_i^{c_i}$ or $h_i^{\gamma_i} \neq \mathrm{S}_i \cdot z_i^{c_i}$ then it returns 0, else it returns 1.

Theorem 27 *The non-interactive proof system LogEq_n is a proof of knowledge, moreover it is complete, sound, and zero-knowledge in the random oracle model.*

Proof: LogEq is a sigma protocol, moreover, it is complete, sound, valid and honest verifier zero-knowledge (Theorem 7), so the n-Cramer-Damgård-Schoenmakers Transformation of LogEq is an interactive proof system that is complete, sound, valid and honest-verifier zero-knowledge, and it is a sigma protocol (Theorem 9). We deduce that its Fiat-Shamir transformation is a non-interactive proof system that is complete, sound, valid and zero-knowledge (Theorem 8), which concludes the proof. □

5.3.2 Our Scheme: EVeR

Using LogEq_n, we build our verifiable ring signature scheme called EVeR. Each user U_i has an ElGamal pair of public/secret keys $(\mathsf{pk}_i, \mathsf{sk}_i)$. To sign a message m, a user U_j hashes it in h and computes $z = h^{\mathsf{sk}_j}$, then it proves that $\log_h(z) = \log_g(\mathsf{pk})$ for one public key pk out of the n public keys of the group $\{\mathsf{pk}_i\}_{1 \leq i \leq n}$ using LogEq_n. To prove that it is (resp. it is not) the signer of some signature, the user U_i proves that $\log_h(z) = \log_g(\mathsf{pk}_i)$ (resp. $\log_h(z) \neq \log_g(\mathsf{pk}_i)$) using LogEq_1.

Definition 66 (Efficient VRS (EVeR)) *Let* $\mathsf{LogEq}_n = (L.Set_n, L.Pro_n, L.Ver_n)$ *be the proof defined in Definition 65 for any* $n \in \mathbb{N}$. EVeR = (Init, Gen, Sig, Ver, Proof, Judge) *is a VRS defined by:*

Init(k): *It generates a prime order group setup* $(\mathbb{G}, p, g)$ *and a hash function* $\mathrm{H} : \{0,1\}^* \to \mathbb{G}$. *It returns the setup* $\mathsf{set} = (\mathbb{G}, p, g, \mathrm{H})$.

Gen(set): *It picks* $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$, *computes* $\mathsf{pk} = g^{\mathsf{sk}}$ *and returns* $(\mathsf{pk}, \mathsf{sk})$.

Sig(sk, L, m): *This algorithm picks* $r \xleftarrow{\$} \mathbb{Z}_p^*$ *and computes* $h = \mathrm{H}(m||r||\mathrm{L})$. *Then it computes* $z = h^{\mathsf{sk}}$, *runs* $\mathrm{P} \leftarrow L.Pro_{|\mathrm{L}|}(\mathsf{sk}, \{(h, \mathsf{pk}_l, z)\}_{\mathsf{pk}_l \in \mathrm{L}})$ *and returns* $\sigma = (r, z, \mathrm{P})$.

Ver(L, m, σ): *This algorithm parses* $\sigma = (r, z, \mathrm{P})$ *and computes* $h = \mathrm{H}(m||r||\mathrm{L})$. *Then this algorithm runs* $b \leftarrow L.Ver_{|\mathrm{L}|}(\{(h, \mathsf{pk}_l, z)\}_{\mathsf{pk}_l \in \mathrm{L}}, \mathrm{P})$ *and returns* b.

Proof(L, m, σ, pk, sk): *It parses* $\sigma = (r, z, \mathrm{P})$ *and computes* $h = \mathrm{H}(m||r||\mathrm{L})$. *Then it computes* $\bar{z} = h^{\mathsf{sk}}$, *runs* $\bar{\mathrm{P}} \leftarrow L.Pro_1(\mathsf{sk}, \{(h, \mathsf{pk}, \bar{z})\})$ *and returns* $\pi = (\bar{z}, \bar{\mathrm{P}})$.

Judge(L, m, σ, pk, π): *It parses* $\sigma = (r, z, \mathrm{P})$ *and* $\pi = (\bar{z}, \bar{\mathrm{P}})$. *Then it computes* $h = \mathrm{H}(m||r||\mathrm{L})$ *and runs* $b \leftarrow L.Ver_1(\{(h, \mathsf{pk}, \bar{z})\}, \bar{\mathrm{P}})$. *If* $b \neq 1$ *then it returns* $\perp$, *else, if* $z = \bar{z}$ *then it returns* 1, *else it returns* 0.

5.4 Security Proofs of EVeR

In this section, we show that EVeR is secure in our model by proving that it is correct, unforgeable, anonymous, accountable and non-seizable. In the following, we informally show why EVeR achieves these properties.

Unforgeability: The scheme is unforgeable since for any $\mathsf{pk} \in \mathrm{L}$, nobody can prove that $\log_g(\mathsf{pk}) = \log_h(z)$ without the knowledge of $\mathsf{sk} = \log_g(\mathsf{pk})$.

Anonymity: Break the anonymity of such a signature is equivalent to break the DDH assumption. Indeed, to link a signature $z = h^{\mathsf{sk}}$ with the corresponding public key of Alice $\mathsf{pk} = g^{\mathsf{sk}}$, an attacker must solve the DDH problem on the instance (pk, h, z). Moreover, note that since the value r *randomizes* the signature, it is not possible to link two signatures of the same message produced by Alice.

Accountability: In order to break the accountability, an adversary must forge a valid signature (*i.e.*, must prove that there exists pk_l in the group L such that $\log_g(\mathsf{pk}_l) \neq \log_h(z)$) and prove that he is not the signer (*i.e.*, $\log_g(\mathsf{pk}) \neq \log_h(z)$ where pk is the public key chosen by the adversary). However, since the adversary does not know the secret keys of the other members of the group, he would have to break the DL assumption to win the experiment, which is assumed to be difficult.

Non-seizable: (non-sei-1) no adversary is able to forge a proof that he is the signer of a signature produced by another user since it is equivalent to prove a false statement using a sound proof system. (non-sei-2) the proof algorithm ran by a honest user with the public key pk returns a proof that this user is the signer only if $\log_g(\mathsf{pk}) = \log_h(z)$. Since no adversary is able to compute z such that $\log_g(\mathsf{pk}) = \log_h(z)$ without the corresponding secret key, no adversary is able to break the non-seizability of EvER under the DL assumption.

5.4.1 Correctness

In this section we show the correctness of EVeR.

Lemma 32 EVeR *is correct.*

Proof: Let $k \in \mathbb{N}$, $n \in \mathbb{N}$, $l \in [\![1, n]\!]$, $l' \in [\![1, n]\!] \setminus \{l\}$, and $m \in \{0,1\}^*$ be. For any set generated by Init(k), $(\mathsf{pk}_i, \mathsf{sk}_i)$ generated by Gen(set) for all i in $[\![1, n]\!]$, any σ generated by $\mathsf{Sig}(\mathsf{sk}_l, \{\mathsf{pk}_i\}_{1 \le i \le n}, m)$, any π generated by the algorithm $\mathsf{Proof}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma, \mathsf{pk}_l, \mathsf{sk}_l)$ and any π' generated by the algorithm $\mathsf{Proof}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma, \mathsf{pk}_{l'}, \mathsf{sk}_{l'})$, We show the three following properties.

1. We parse $\sigma = (r, z, \mathrm{P})$. Using $h = \mathrm{H}(m||r||\mathrm{L})$, we have $z = h^{\mathsf{sk}_l}$ and P was generated by the algorithm $\mathsf{L.Pro}_n(\mathsf{sk}_l, \{(h, \mathsf{pk}_i, z)\}_{1 \le i \le n})$. Since the proof LogEq_n is complete, then the algorithm $\mathsf{L.Ver}_n(\{(h, \mathsf{pk}_i, z)\}_{1 \le i \le n}, \mathrm{P})$ always returns 1, which implies that $\mathsf{Ver}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma)$ returns 1 with probability 1.

2. We parse $\sigma = (r, z, \mathrm{P})$ and $\pi = (\bar{z}, \bar{\mathrm{P}})$. Using $h = \mathrm{H}(m||r||\mathrm{L})$, we have $z = h^{\mathsf{sk}_l}$ and P was generated by $\mathsf{L.Pro}_n(\mathsf{sk}, \{(h, \mathsf{pk}_i, z)\}_{1 \le i \le n})$. On the other hand, $\bar{z} = h^{\mathsf{sk}_l}$ and $\bar{\mathrm{P}}$ was generated by the algorithm $\mathsf{L.Pro}_n(\mathsf{sk}_l, \{(h, \mathsf{pk}_l, \bar{z})\})$. We note that $z = \bar{z}$. Since $\mathrm{L}og\mathrm{E}q_n$ is complete, then the algorithm $\mathsf{L.Ver}_n(\{(h, \mathsf{pk}_l, z)\}, \bar{\mathrm{P}})$ always returns 1, which implies that the algorithm $\mathsf{Judge}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma, \mathsf{pk}_l, \pi)$ returns 1 with probability 1.

3. We parse $\sigma = (r, z, \mathrm{P})$ and $\pi' = (\bar{z}, \bar{\mathrm{P}})$. Using $h = \mathrm{H}(m||r||\mathrm{L})$, we have $z = h^{\mathsf{sk}_l}$ and P was generated by $\mathsf{L.Pro}_n(\mathsf{sk}, \{(h, \mathsf{pk}_i, z)\}_{1 \le i \le n})$. On the other hand, $\bar{z} = h^{\mathsf{sk}_{l'}}$ and $\bar{\mathrm{P}}$ was generated by the algorithm $\mathsf{L.Pro}_n(\mathsf{sk}_{l'}, \{(h, \mathsf{pk}_{l'}, \bar{z})\})$. We note that $z \neq \bar{z}$. Since $\mathrm{L}og\mathrm{E}q_n$ is complete, then the algorithm $\mathsf{L.Ver}_n(\{(h, \mathsf{pk}_{l'}, z)\}, \bar{\mathrm{P}})$ always returns 1, which implies that the algorithm $\mathsf{Judge}(\{\mathsf{pk}_i\}_{1 \le i \le n}, m, \sigma, \mathsf{pk}_l, \pi)$ returns 0 with probability 1.

These three properties implies that EVeR is correct. □

5.4.2 Unforgeability

In this section we show that EVeR is unforgeable.

Lemma 33 EVeR *is unforgeable under the DL assumption in the random oracle model.*

Proof: We recall that since for any $n \in \mathbb{N}$, LogEq_n is valid, then for any $\mathsf{pk}_* \in \mathscr{L}_{\mathscr{R}_n}$and any algorithm $\mathscr{A}(\mathsf{pk}_*)$, there exists a knowledge extractor K such that the probability that $\mathsf{K}^{\mathscr{A}(\mathsf{pk}_*)}(k)$ outputs sk such that $(\mathsf{pk}_*, \mathsf{sk}) \in \mathscr{R}_n$ given access to the oracle $\mathscr{A}(\mathsf{pk}_*)$ is equivalent to the probability that $\mathscr{A}(\mathsf{pk}_*)$ outputs a proof π such that $\mathsf{L.Ver}_n(\mathsf{pk}_*, \pi)$ outputs 1. Moreover, since LogEq_n is zero-knowledge there exists a polynomial time algorithm Sim_n (called the *simulator*) such that the outputs of $\mathsf{L.Pro}_n(\mathsf{sk}, \mathsf{pk}_*)$ and the outputs of $\mathsf{Sim}_n(\mathsf{pk}_*)$ follow the same probability distribution.

Assume that there exists an algorithm $\mathscr{A} \in \textsc{poly}(k)$ such that $\lambda(k) = \mathsf{Adv}^{n\text{-}\mathsf{EUF\text{-}CMA}}_{\mathrm{EVeR}, \mathscr{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathscr{B} \in \textsc{poly}(k)$ that solves the DL problem in a prime order group $(\mathbb{G}, p, g)$ with non-negligible probability.

Algorithm $\mathscr{B}(y)$**:** For all $i \in [\![1, n]\!]$, it picks $x_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathsf{pk}_i = y^{x_i}$. $\mathscr{B}$ runs $x' \leftarrow \mathsf{K}^{\mathscr{A}'(\{\mathsf{pk}_i\}_{1 \le i \le n})}(k)$ where $\mathscr{A}'$ is the following algorithm:

> **Algorithm** $\mathscr{A}'(\{\mathsf{pk}_i\}_{1 \le i \le n})$**:** It runs $(\mathrm{L}_*, \sigma_*, m_*) \leftarrow \mathscr{A}(\{\mathsf{pk}_i\}_{1 \le i \le n})$. It simulates the oracles to $\mathscr{A}$ as follows:
>
> **Random oracle** $\mathrm{H}(.)$**:** On the i^{th} input M_i, if there exists $j \in \mathbb{N}$ such that $j < i$ and $\mathrm{M}_j = \mathrm{M}_i$, then it sets $u_j = u_i$. Else it picks $u_i \xleftarrow{\$} \mathbb{Z}_p^*$. Finally, it returns g^{u_i}.
>
> **Oracle** $\mathsf{Sig}(\cdot, \cdot, \cdot)$**:** On the i^{th} input (L_i, l_i, m_i), it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. Using the oracle $\mathrm{H}(.)$, it computes $h_i = \mathrm{H}(m_i||r_i||\mathrm{L}_i)$, then there exists j such that $m_i||r_i||\mathrm{L}_i = \mathrm{M}_j$. It computes $z_i = \mathsf{pk}_{l_i}^{u_j}$ and it runs $\mathrm{P}_i \leftarrow \mathsf{Sim}_{|\mathrm{L}_i|}(\{(h_i, \mathsf{pk}_l, z_i)\}_{\mathsf{pk}_l \in \mathrm{L}_i})$. It returns (r_i, z_i, P_i) to $\mathscr{A}$.
>
> **Oracle** $\mathsf{Proof}(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** On the i^{th} input $(\mathrm{L}'_i, m'_i, \sigma'_i, l'_i)$, it parses $\sigma'_i = (r'_i, z'_i, \mathrm{P}'_i)$. Using the oracle $\mathrm{H}(.)$, it computes $h'_i = \mathrm{H}(m'_i||r'_i||\mathrm{L}'_i)$, then there exists j such that $m'_i||r'_i||\mathrm{L}'_i = \mathrm{M}_j$ It computes $\bar{z}_i = \mathsf{pk}_{l'_i}^{u_j}$ and it runs $\bar{\mathrm{P}}_i \leftarrow \mathsf{Sim}_1(\{(h'_i, \mathsf{pk}_{l'_i}, \bar{z}_i)\})$. It returns $(\bar{z}_i, \bar{\mathrm{P}}_i)$ to $\mathscr{A}$.
>
> Finally, $\mathscr{A}'$ parses $\sigma_* = (r_*, z_*, \mathrm{P}_*)$ and returns P_*.

If there exists $i \in [\![1, n]\!]$ such that $\mathsf{pk}_i = g^{\frac{x'}{x_i}}$, then $\mathscr{B}$ returns $x = \frac{x'}{x_i}$, else it returns $\perp$.

Analysis: First note that the experiment n-EUF-CMA is perfectly simulated for $\mathscr{A}$. Assume that it wins its experiment, then it returns the tuple $(\mathrm{L}_*, \sigma_*, m_*)$ where $\sigma_* = (r_*, z_*, \mathrm{P}_*)$ and $h_* = \mathrm{H}(m_*||r_*||\mathrm{L}_*)$ such that:

$$\mathsf{L.Ver}_{|\mathrm{L}_*|}(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1 \tag{5.1}$$

$$\mathrm{L}_* \subset \{\mathsf{pk}_i\}_{1 \leq i \leq n} \tag{5.2}$$

$$\forall\ i \in [\![1, q_\mathrm{S}]\!], \sigma_i \neq \sigma_* \tag{5.3}$$

where q_S is the number of queries to the oracle $\mathsf{Sig}(\cdot,\cdot,\cdot)$. Moreover, equation (5.3) implies that $\forall\ i \in [\![1, q_\mathrm{S}]\!]$, $\mathrm{P}_i \neq \mathrm{P}_*$, then P_* was not generated by the simulator $\mathsf{Sim}_{|\mathrm{L}_*|}$. We deduce the following equation from (5.1):

$$\mathsf{L.Ver}_{|\mathrm{L}_*|}(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1$$

Thus $\mathscr{A}$ returns a valid proof with non negligible probability $\lambda(k)$ at least:

$$\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1\right] \geq \mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{n\text{-}\mathsf{EUF\text{-}CMA}}_{\mathrm{EVeR},\mathscr{A}}(k)\right] \geq \lambda(k)$$

Since LogEq_n is valid and K is an extractor for LogEq_n, there exists a polynomial t and a negligible function ϵ' such that:

$$\begin{aligned}\lambda'(k) &= \mathsf{Pr}\left[\exists\ \mathsf{pk} \in \mathrm{L}_*, x' = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)\right]\\ &= \mathsf{Pr}\left[(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, x') \in \mathscr{R}_{|\mathrm{L}_*|}\right]\\ &\geq t\left(\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1\right]\right) - \epsilon'(k)\\ &\geq t(\lambda(k)) - \epsilon'(k)\end{aligned}$$

Note that $\lambda'(k)$ is non-negligible because $\lambda(k)$ is non-negligible.

Assume that $\mathscr{A}'$ returns a valid proof, since for all $i \in [\![1, n]\!]$, $\mathsf{pk}_i = y^{x_i}$, and since $\mathrm{L}_* \subset \{\mathsf{pk}_i\}_{1 \leq i \leq n}$, then there exists $j \in [\![1, n]\!]$ such that $\mathsf{pk}_j = g^{x'}$, which implies that discrete logarithm of y is x'/x_j. We deduce that $\mathscr{B}$ returns the discrete logarithm of y with probability at lest $\lambda'(k)$. This function is non-negligible, which concludes the proof. □

5.4.3 Anonymity

In this section we show that EVeR is anonymous.

Lemma 34 EVeR *is anonymous under the DDH assumption in the random oracle model.*

Proof: Since LogEq_n is zero-knowledge, then there exists a polynomial time algorithm Sim_n (called the *simulator*) such that the outputs of $\mathsf{L.Pro}_n(\mathsf{sk}, \mathsf{pk}_*)$ and the outputs of $\mathsf{Sim}_n(\mathsf{pk}_*)$ follow the same probability distribution. Let the n-ano_ψ experiment be the same experiment as n-ano except that the oracle $\mathsf{LRSO_b}$ is called at most ψ times. We prove the two following claims:

Claim 1 If there exists $\mathscr{A} \in \mathrm{POLY}(k)^2$ such that $\mathsf{Adv}^{n\text{-}\mathsf{ano}_1}_{\mathrm{EVeR},\mathscr{A}}(k)$ is non-negligible, then there exists $\mathscr{B} \in \mathrm{POLY}(k)$ that breaks the DDH problem with non-negligible probability. Note that it implies that $\mathsf{Adv}^{n\text{-}\mathsf{ano}_1}_{\mathrm{EVeR}}(k)$ is negligible under the DDH assumption.

Claim 2 Let $\psi \geq 1$ be an integer, assume that $\mathsf{Adv}^{n\text{-}\mathsf{ano}_\psi}_{\mathrm{EVeR}}(k)$ is negligible. If there exists $\mathscr{A} \in \mathrm{POLY}(k)^2$ such that $\mathsf{Adv}^{n\text{-}\mathsf{ano}_{\psi+1}}_{\mathrm{EVeR},\mathscr{A}}(k)$ is non-negligible, then there exists $\mathscr{B} \in \mathrm{POLY}(k)$ that breaks the DDH problem with non-negligible probability. Note that it implies that $\mathsf{Adv}^{n\text{-}\mathsf{ano}_{\psi+1}}_{\mathrm{EVeR}}(k)$ is negligible under the DDH assumption.

By induction, this two claims imply that $\mathsf{Adv}^{n\text{-}\mathsf{ano}_\psi}_{\mathrm{EVeR}}(k)$ is negligible for any n and any $\psi \geq 1$ in $\mathbb{N}$.

Proof of Claim 1: Firstly, we show how to build the algorithm $\mathscr{B}(\mathrm{X}, \mathrm{Y}, \mathrm{Z})$ in the group $(\mathbb{G}, p, g)$. This algorithm picks $d \xleftarrow{\$} [\![1, n]\!]$. For all $i \in [\![1, n]\!]$:

- If $i = d$ then it sets $\mathsf{pk}_i = \mathrm{X}$
- Else, it runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{set})$ where $\mathsf{set} = (\mathbb{G}, p, g, \mathrm{H})$.

$\mathscr{B}$ runs $(d_0, d_1) \leftarrow \mathscr{A}_1(\{\mathsf{pk}_i\}_{1 \le i \le n})$. During the experiment, it simulates the oracle for $\mathscr{A}_1$ as follows:

Random oracle $\mathrm{H}(.)$**:** On the i^{th} input M_i, if there exists $j \in \mathbb{N}$ such that $j < i$ and $\mathrm{M}_j = \mathrm{M}_i$, then it sets $u_j = u_i$. Else it picks $u_i \xleftarrow{\$} \mathbb{Z}_p^*$. Finally, it returns g^{u_i}.

Oracle Sig$(\cdot, \cdot, \cdot)$**:** On the i^{th} input (L_i, l_i, m_i), it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ and it computes $h_i = \mathrm{H}(m_i || r_i || \mathrm{L}_i)$ using the oracle $\mathrm{H}(.)$. Then there exists $j \in \mathbb{N}$ such that $m_i || r_i || \mathrm{L}_i = \mathrm{M}_j$.

- If $l_i = d$ then it computes $z_i = \mathrm{X}^{u_j}$ and it runs $\mathrm{P}_i \leftarrow \mathsf{Sim}_{|\mathrm{L}_i|}(\{(h_i, \mathsf{pk}_l, z_i)\}_{\mathsf{pk}_l \in \mathrm{L}_i})$. It returns $\sigma_i = (r_i, z_i, \mathrm{P}_i)$ to $\mathscr{A}_1$.
- Else it runs and returns $\sigma_i \leftarrow \mathsf{Sig}(m_i, \mathrm{L}_i, \mathsf{sk}_{l_i})$

Oracle Proof$(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** On the i^{th} input $(\mathrm{L}'_i, m'_i, \sigma'_i, l'_i)$, it parses $\sigma'_i = (r'_i, z'_i, \mathrm{P}'_i)$. It computes $h'_i = \mathrm{H}(m'_i || r'_i || \mathrm{L}'_i)$ using the oracle $\mathrm{H}(.)$, then there exists $j \in \mathbb{N}$ such that $m'_i || r'_i || \mathrm{L}'_i = \mathrm{M}_j$ It computes $\bar{z}_i = \mathsf{pk}_{l'_i}^{u_j}$ and it runs $\bar{\mathrm{P}}_i \leftarrow \mathsf{Sim}_1(\{(h'_i, \mathsf{pk}_{l'_i}, \bar{z}_i)\})$. It returns $(\bar{z}_i, \bar{\mathrm{P}}_i)$ to $\mathscr{A}_1$.

$\mathscr{B}$ runs $b_* \leftarrow \mathscr{A}_2(\{\mathsf{pk}_i\}_{1 \le i \le n})$. During the experiment, $\mathscr{B}$ simulates the oracle $\mathsf{Sig}(\cdot, \cdot, \cdot)$ as in the first phase. It simlulates the three other oracles as follows:

Oracle $\mathsf{LRSO_b}(d_0, d_1, \cdot, \cdot)$**:** On input (m'', L''), it picks $r'' \xleftarrow{\$} \mathbb{Z}_p^*$. If there exists $i \in \mathbb{N}$ such that $r_i = r''$ then $\mathscr{B}$ aborts the experiment and returns $b'_* \xleftarrow{\$} \{0, 1\}$, else it runs the simulator $\mathrm{P}'' \leftarrow \mathsf{Sim}_{|\mathrm{L}''|}(\{(\mathrm{Y}, \mathsf{pk}_l, \mathrm{Z})\}_{\mathsf{pk}_l \in \mathrm{L}''})$ and returns $(r'', \mathrm{Z}, \mathrm{P}'')$ to $\mathscr{A}_2$.

Oracle Proof$(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** On the i^{th} input $(\mathrm{L}'_i, m'_i, \sigma'_i, l'_i)$, if $\mathsf{LRSO_b}$ has been already called and $\sigma'_i = \sigma''$ and ($l'_i = d_0$ or $l'_i = d_1$), then it returns $\perp$ to $\mathscr{A}_2$. Else, it process as in first phase.

Random oracle $\mathrm{H}(.)$**:** On the i^{th} input M_i, if $\mathsf{LRSO_b}$ have been already called and $\mathrm{M}_i = (m'' || r'' || \mathrm{L}'')$, then it returns Y to $\mathscr{A}_2$. Else, it process as in first phase.

If $d_0 \neq d$ and $d_1 \neq d$ then $\mathscr{B}$ returns $b'_* \xleftarrow{\$} \{0, 1\}$, else, let b' be the bit that verifies $d_{b'} = d$. If $b' = b_*$ then $\mathscr{B}$ returns $b'_* = 1$, else $b'_* = 0$.

Analysis: Let q_{S} be the number of queries asked to $\mathsf{Sig}(\cdot, \cdot, \cdot)$ and let E be the event *"$\mathscr{B}$ picks b'_* at random"*. We have:

$$
\begin{aligned}
\mathsf{Pr}[\mathrm{E}] &= \mathsf{Pr}\left[(\exists i \in [\![1, q_{\mathrm{S}}]\!], r_i = r'') \vee (d_0 \neq d \wedge d_1 \neq d)\right] \\
&\le \mathsf{Pr}\left[\exists i \in [\![1, q_{\mathrm{S}}]\!], r_i = r''\right] + \mathsf{Pr}\left[d_0 \neq d \wedge d_1 \neq d\right] \\
&\le \sum_{i=1}^{q_{\mathrm{S}}} \mathsf{Pr}\left[r_i = r''\right] + (1 - \mathsf{Pr}\left[d_0 = d \vee d_1 = d\right]) \\
&\le \frac{q_{\mathrm{S}}}{|\mathbb{G}|} + 1 - \mathsf{Pr}\left[d_0 = d\right] - \mathsf{Pr}\left[d_1 = d\right] \\
&\le \frac{q_{\mathrm{S}}}{|\mathbb{G}|} + 1 - \mathsf{Pr}\left[d_0 = d\right] \\
&\le \frac{q_{\mathrm{S}}}{|\mathbb{G}|} + 1 - \frac{1}{n} \\
&\le \frac{q_{\mathrm{S}}}{p} + \frac{n-1}{n}
\end{aligned}
$$

We deduce that:

$$\begin{aligned}\mathsf{Pr}[\neg \mathrm{E}] &\geq 1-\left(\frac{q_{\mathrm{S}}}{p}+\frac{n-1}{n}\right)\\ &\geq \frac{1}{n}-\frac{q_{\mathrm{S}}}{p}\end{aligned}$$

Let α and β be two elements of $\mathbb{Z}_p^*$ such that $\mathrm{X}=g^{\alpha}$ and $\mathrm{Y}=g^{\beta}$. Let b be a bit such that $b=1 \Leftrightarrow \mathrm{Z}=g^{\alpha\cdot\beta}$. If $\mathscr{B}$ picks b'_* at random, then it wins the experiment with probability $1/2$:

$$\mathsf{Pr}[b'_*=b|\mathrm{E}]=\frac{1}{2}$$

If $\mathscr{B}$ does not pick b'_* at random and $\mathrm{Z}=g^{\alpha\cdot\beta}$ the experiment is perfectly simulated for $\mathscr{A}$, then $\mathscr{B}$ wins its experiment with the same probability as $\mathscr{A}$, *i.e.,* there exists a non-negligible function $\lambda_1(k)$ such that:

$$\left|\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}=g^{\alpha\cdot\beta}\right)\right]-\frac{1}{2}\right|=\lambda_1(k)$$

If $\mathscr{B}$ does not pick b'_* at random and $\mathrm{Z}\neq g^{\alpha\cdot\beta}$, then the query to the oracle $\mathsf{LRSO_b}$ gives no information about b to $\mathscr{A}$, so:

$$\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}\neq g^{\alpha\cdot\beta}\right)\right]=\frac{1}{2}$$

We deduce:

$$\begin{aligned}\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\right]&=\mathsf{Pr}\left[\mathrm{Z}=g^{\alpha\cdot\beta}\right]\cdot\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}=g^{\alpha\cdot\beta}\right)\right]+\mathsf{Pr}\left[\mathrm{Z}\neq g^{\alpha\cdot\beta}\right]\cdot\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}\neq g^{\alpha\cdot\beta}\right)\right]\\ &=\frac{1}{2}\cdot\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}=g^{\alpha\cdot\beta}\right)\right]+\frac{1}{2}\cdot\frac{1}{2}\\ &=\frac{1}{2}\cdot\left(\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}=g^{\alpha\cdot\beta}\right)\right]+\frac{1}{2}\right)\end{aligned}$$

We evaluate the probability that $\mathscr{B}$ wins its DDH experiment:

$$\begin{aligned}\mathsf{Pr}[b'_*=b]&=\mathsf{Pr}[\neg\mathrm{E}]\cdot\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\right]+(1-\mathsf{Pr}[\neg\mathrm{E}])\cdot\mathsf{Pr}\left[b'_*=b|\,\mathrm{E}\right]\\ &=\mathsf{Pr}[\neg\mathrm{E}]\cdot\left(\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\right]-\mathsf{Pr}\left[b'_*=b|\mathrm{E}\right]\right)+\mathsf{Pr}\left[b'_*=b|\mathrm{E}\right]\\ &=\mathsf{Pr}[\neg\mathrm{E}]\cdot\left(\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\right]-\frac{1}{2}\right)+\frac{1}{2}\\ &=\mathsf{Pr}[\neg\mathrm{E}]\cdot\left(\frac{1}{2}\cdot\left(\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}=g^{\alpha\cdot\beta}\right)\right]+\frac{1}{2}\right)+\frac{1}{2}\cdot(-1)\right)+\frac{1}{2}\\ &=\frac{\mathsf{Pr}[\neg\mathrm{E}]}{2}\cdot\left(\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}=g^{\alpha\cdot\beta}\right)\right]-\frac{1}{2}\right)+\frac{1}{2}\end{aligned}$$

Finally, we deduce the advantage of $\mathscr{B}$ against the DDH problem:

$$\begin{aligned}\left|\mathsf{Pr}[b'_*=b]-\frac{1}{2}\right|&=\left|\frac{\mathsf{Pr}[\neg\mathrm{E}]}{2}\cdot\left(\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}=g^{\alpha\cdot\beta}\right)\right]-\frac{1}{2}\right)+\frac{1}{2}-\frac{1}{2}\right|\\ &=\frac{\mathsf{Pr}[\neg\mathrm{E}]}{2}\cdot\left|\mathsf{Pr}\left[b'_*=b|\neg\mathrm{E}\wedge\left(\mathrm{Z}=g^{\alpha\cdot\beta}\right)\right]-\frac{1}{2}\right|\\ &=\left(\frac{1}{2\cdot n}-\frac{q_{\mathrm{S}}}{2\cdot p}\right)\cdot\lambda_1(k)\\ &=\frac{\lambda_1(k)}{2\cdot n}-\frac{\lambda_1(k)\cdot q_{\mathrm{S}}}{2\cdot p}\end{aligned}$$

This advantage is non-negligible, which conclude the proof of Claim 1.

Proof of Claim 2: We show how to build the algorithm $\mathscr{B}(\mathrm{X},\mathrm{Y},\mathrm{Z})$. It runs the same reduction as in claim 1, except that the algorithm $\mathscr{B}$ simulates the oracles $\mathsf{LRSO_b}(d_0,d_1,\cdot,\cdot)$ and $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ to $\mathscr{A}_2$ as follows:

Oracle $\mathsf{LRSO_b}(d_0, d_1, \cdot, \cdot)$**:** On the i^{th} input (m''_i, L''_i), if $i = 1$ then this oracle is defined as in the reduction of Claim 1. Else it calls the oracle $\mathsf{Sig}(\cdot, \cdot, \cdot)$ on the input $(m''_i, d, \mathrm{L}''_i)$ and returns the resulted signature σ''_i to $\mathscr{A}_2$.

Oracle Proof$(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** On the i^{th} input $(\mathrm{L}'_i, m'_i, \sigma'_i, l'_i)$, if $\mathsf{LRSO_b}$ has been already called and there exists $j \in \mathbb{N}$ such that $\sigma'_i = \sigma''_j$ and $(l'_i = d_0$ or $l'_i = d_1)$, then it returns $\perp$ to $\mathscr{A}_2$. Else, it processes as in the reduction of Claim 1.

Analysis: Let q_{S} be the number of queries asked to $\mathsf{Sig}(\cdot, \cdot, \cdot)$ and let E be the event *"$\mathscr{B}$ picks b'_* at random"*. As in Claim 1, we have:

$$\Pr[\neg \mathrm{E}] \geq \frac{1}{n} - \frac{q_{\mathrm{S}}}{p}$$

Let α and β be two elements of $\mathbb{Z}_p^*$ such that $\mathrm{X} = g^{\alpha}$ and $\mathrm{Y} = g^{\beta}$. Let b be the solution to the DDH problem, *i.e.,* $b = 1 \Leftrightarrow \mathrm{Z} = g^{\alpha \cdot \beta}$. If $\mathscr{B}$ picks b'_* at random, it wins the experiment with probability 1/2:

$$\Pr[b'_* = b | \mathrm{E}] = \frac{1}{2}$$

If $\mathscr{B}$ does not pick b'_* at random and $\mathrm{Z} = g^{\alpha \cdot \beta}$ the experiment is perfectly simulated for $\mathscr{A}$, then $\mathscr{B}$ wins its experiment with the same probability as $\mathscr{A}$, *i.e.,* there exists a non-negligible function $\lambda_{\psi+1}(k)$ such that:

$$\left| \Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} = g^{\alpha \cdot \beta} \right) \right] - \frac{1}{2} \right| = \lambda_{\psi+1}(k)$$

If $\mathscr{B}$ does not pick b'_* at random and $\mathrm{Z} \neq g^{\alpha \cdot \beta}$, then the first query to the oracle $\mathsf{LRSO_b}$ is useless since it gives no information about b to $\mathscr{A}$, so $\mathscr{B}$ simulates an experiment that is similar to the n-ano_{ψ} one to $\mathscr{A}$, and there exists a negligible function $\epsilon_{\psi}(k)$ such that:

$$\left| \Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} \neq g^{\alpha \cdot \beta} \right) \right] - \frac{1}{2} \right| = \epsilon_{\psi}(k)$$

We deduce:

$$\begin{aligned} \Pr\left[b'_* = b | \neg \mathrm{E} \right] &= \Pr\left[\mathrm{Z} = g^{\alpha \cdot \beta} \right] \cdot \Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} = g^{\alpha \cdot \beta} \right) \right] + \Pr\left[\mathrm{Z} \neq g^{\alpha \cdot \beta} \right] \cdot \Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} \neq g^{\alpha \cdot \beta} \right) \right] \\ &= \frac{1}{2} \cdot \left(\Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} = g^{\alpha \cdot \beta} \right) \right] + \Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} \neq g^{\alpha \cdot \beta} \right) \right] \right) \end{aligned}$$

We compute the probability that $\mathscr{B}$ wins its DDH experiment:

$$\begin{aligned} \Pr[b'_* = b] &= \Pr[\neg \mathrm{E}] \cdot \Pr\left[b'_* = b | \neg \mathrm{E} \right] + (1 - \Pr[\neg \mathrm{E}]) \cdot \Pr\left[b'_* = b | \ \mathrm{E} \right] \\ &= \Pr[\neg \mathrm{E}] \cdot \left(\Pr\left[b'_* = b | \neg \mathrm{E} \right] - \Pr\left[b'_* = b | \mathrm{E} \right] \right) + \Pr\left[b'_* = b | \mathrm{E} \right] \\ &= \Pr[\neg \mathrm{E}] \cdot \left(\Pr\left[b'_* = b | \neg \mathrm{E} \right] - \frac{1}{2} \right) + \frac{1}{2} \\ &= \Pr[\neg \mathrm{E}] \cdot \left(\frac{1}{2} \cdot \left(\Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} = g^{\alpha \cdot \beta} \right) \right] + \Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} \neq g^{\alpha \cdot \beta} \right) \right] \right) - \frac{1}{2} \right) + \frac{1}{2} \\ &= \frac{\Pr[\neg \mathrm{E}]}{2} \cdot \left(\Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} = g^{\alpha \cdot \beta} \right) \right] + \Pr\left[b'_* = b | \neg \mathrm{E} \wedge \left(\mathrm{Z} \neq g^{\alpha \cdot \beta} \right) \right] - 1 \right) + \frac{1}{2} \end{aligned}$$

Finally, we deduce the advantage of $\mathcal{B}$ against the DDH problem:

$$\begin{aligned}\left|\Pr[b'_* = b] - \frac{1}{2}\right| &= \left|\frac{\Pr[\neg \mathrm{E}]}{2}\cdot\left(\Pr\left[b'_* = b|\neg \mathrm{E} \wedge \left(\mathrm{Z} = g^{\alpha\cdot\beta}\right)\right] + \Pr\left[b'_* = b|\neg \mathrm{E}\wedge\left(\mathrm{Z}\neq g^{\alpha\cdot\beta}\right)\right] - 1\right) + \frac{1}{2} - \frac{1}{2}\right| \\ &= \frac{\Pr[\neg \mathrm{E}]}{2}\cdot\left|\left(\Pr\left[b'_* = b|\neg \mathrm{E} \wedge \left(\mathrm{Z} = g^{\alpha\cdot\beta}\right)\right] - \frac{1}{2}\right) + \left(\Pr\left[b'_* = b|\neg \mathrm{E}\wedge\left(\mathrm{Z}\neq g^{\alpha\cdot\beta}\right)\right] - \frac{1}{2}\right)\right| \\ &= \left(\frac{1}{2\cdot n} - \frac{q_{\mathrm{S}}}{2\cdot p}\right)\cdot\left(\lambda_{\psi+1}(k) - \epsilon_{\psi+1}(k)\right) \\ &= \frac{\lambda_{\psi+1}(k)}{2\cdot n} - \left(\frac{q_{\mathrm{S}}}{2\cdot p}\cdot\lambda_{\psi+1}(k) + \epsilon_{\psi+1}(k)\cdot\left(\frac{1}{2\cdot n} - \frac{q_{\mathrm{S}}}{2\cdot p}\right)\right)\end{aligned}$$

This advantage is non-negligible, which concludes the proof of Claim 2 and the proof of the lemma. □

5.4.4 Accountability

In this section, we show that EVeR is accountable.

Lemma 35 EVeR *is accontable under the DL assumption in the random oracle model.*

Proof: For any $n \in \mathbb{N}$, let K (resp. Sim_n) be the knowledge extractor (resp. simulator) of LogEq_n. Assume that there exists an adversary $\mathcal{A} \in \mathrm{POLY}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{n\text{-acc}}_{\mathrm{EVeR},\mathcal{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \mathrm{POLY}(k)$ that solves the DL problem with non-negligible probability.

Algorithm $\mathcal{B}(y)$: For all $i \in [\![1,n]\!]$, it picks $x_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathsf{pk}_i = y^{x_i}$. $\mathcal{B}$ runs $x' \leftarrow \mathsf{K}^{\mathcal{A}'(\{\mathsf{pk}_i\}_{1\le i\le n})}(k)$ where $\mathcal{A}'$ is the following algorithm:

Algorithm $\mathcal{A}'(\{\mathsf{pk}_i\}_{1\le i\le n})$: It runs $(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{1\le i\le n})$. It simulates the oracles to $\mathcal{A}$ as follows:

Random oracle $\mathrm{H}(.)$**:** On the i^{th} input M_i, if there exists $j \in \mathbb{N}$ such that $j < i$ and $\mathrm{M}_j = \mathrm{M}_i$, then it sets $u_j = u_i$. Else it picks $u_i \xleftarrow{\$} \mathbb{Z}_p^*$. Finally, it returns g^{u_i}.

Oracle $\mathsf{Sig}(\cdot,\cdot,\cdot)$**:** On the i^{th} input (L_i, l_i, m_i), it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. Using the oracle $\mathrm{H}(.)$, it computes $h_i = \mathrm{H}(m_i||r_i||\mathrm{L}_i)$, then there exists $j \in \mathbb{N}$ such that $j < i$ and $m_i||r_i||\mathrm{L}_i = \mathrm{M}_j$. It computes $z_i = \mathsf{pk}_{l_i}^{u_j}$ and it runs $\mathrm{P}_i \leftarrow \mathsf{Sim}_{|\mathrm{L}_i|}(\{(h_i, \mathsf{pk}_l, z_i)\}_{\mathsf{pk}_l\in\mathrm{L}_i})$. It returns $\sigma_i = (r_i, z_i, \mathrm{P}_i)$ to $\mathcal{A}$.

Oracle $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$**:** On the i^{th} input $(\mathrm{L}'_i, m'_i, \sigma'_i, l'_i)$, it parses $\sigma'_i = (r'_i, z'_i, \mathrm{P}'_i)$. It computes $h'_i = \mathrm{H}(m'_i||r'_i||\mathrm{L}'_i)$ using the oracle $\mathrm{H}(.)$, then there exists $j \in \mathbb{N}$ such that $m'_i||r'_i||\mathrm{L}'_i = \mathrm{M}_j$. It computes $\bar{z}_i = \mathsf{pk}_{l'_i}^{u_j}$ and it runs $\bar{\mathrm{P}}_i \leftarrow \mathsf{Sim}_1(\{(h'_i, \mathsf{pk}_{l'_i}, \bar{z}_i)\})$. It returns $(\bar{z}_i, \bar{\mathrm{P}}_i)$ to $\mathcal{A}$.

Finally, $\mathcal{A}'$ computes $h_* = \mathrm{H}(m_*||r_*||\mathrm{L}_*)$ using the random oracle $\mathrm{H}(\cdot)$, then it parses $\sigma_* = (r_*, z_*, \mathrm{P}_*)$ and returns P_*.

If there exists $i \in [\![1,n]\!]$ such that $\mathsf{pk}_i = g^{\frac{x'}{x_i}}$, then $\mathcal{B}$ returns $x = \frac{x'}{x_i}$, else it returns $\perp$.

Analysis: We parse $\sigma_* = (r_*, z_*, \mathrm{P}_*)$ and $\pi_* = (\bar{z}_*, \bar{\mathrm{P}}_*)$. Assume that $\mathcal{A}$ wins the experiment, we have:

$$\mathrm{L}_* \subseteq \{\mathsf{pk}_i\}_{1\le i\le n} \cup \{\mathsf{pk}_*\} \tag{5.4}$$

$$\mathsf{Ver}(\mathrm{L}_*, \sigma_*, m_*) = 1 \tag{5.5}$$

$$\mathsf{Judge}(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) = 0 \tag{5.6}$$

$$\forall\ i \in [\![1, q_{\mathrm{S}}]\!], \sigma_i \neq \sigma_* \tag{5.7}$$

where q_{S} is the number of queries to the oracle $\mathsf{Sig}(\cdot,\cdot,\cdot)$. Moreover, equation (5.7) implies that $\forall\ i \in [\![1, q_{\mathrm{S}}]\!]$, $\mathrm{P}_i \neq \mathrm{P}_*$, then P_* was not generated by the simulator $\mathsf{Sim}_{|\mathrm{L}_*|}$. We deduce the following equation from (5.5):

$$\mathsf{L.Ver}_{|\mathrm{L}_*|}(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1$$

Thus $\mathscr{A}$ returns a valid proof with non negligible probability $\lambda(k)$ at least:

$$\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1\right] \geq \mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{n\text{-}\mathsf{acc}}_{\mathrm{EVeR},\mathscr{A}}(k)\right] \geq \lambda(k)$$

Since LogEq_n is valid and K is an extractor for LogEq_n, there exists a polynomial t and a negligible function ϵ' such that:

$$\begin{aligned}
\lambda'(k) &= \mathsf{Pr}\left[\exists\ \mathsf{pk} \in \mathrm{L}_*, x' = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)\right] \\
&= \mathsf{Pr}\left[(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, x') \in \mathscr{R}_{|\mathrm{L}_*|}\right] \\
&\geq t\left(\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1\right]\right) - \epsilon'(k) \\
&\geq t(\lambda(k)) - \epsilon'(k)
\end{aligned} \tag{5.8}$$

Note that $\lambda'(k)$ is non-negligible because $\lambda(k)$ is non-negligible. We deduce the following equations from (5.6):

$$\bar{z}_* \neq z_*$$
$$\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_*, \bar{z}_*)\}, \bar{\mathrm{P}}_*) = 1$$

We deduce the following equation:

$$\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_*, \bar{z}_*)\}, \bar{\mathrm{P}}_*) = 1\right] \geq \mathsf{Adv}^{n\text{-}\mathsf{acc}}_{\mathrm{EVeR},\mathscr{A}}(k) = \lambda(k)$$

Since LogEq_n is sound, there exists a negligible function ϵ' such that:

$$\begin{aligned}
&\mathsf{Pr}\left[(\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_*, \bar{z}_*)\}, \bar{\mathrm{P}}_*) = 1) \wedge (\{(h_*, \mathsf{pk}_*, \bar{z}_*)\} \notin \mathscr{L}_{\mathscr{R}_1})\right] \leq \epsilon'(k) \\
\Rightarrow &\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_*, \bar{z}_*)\}, \bar{\mathrm{P}}_*) = 1\right] \cdot \mathsf{Pr}\left[\{(h_*, \mathsf{pk}_*, \bar{z}_*)\} \notin \mathscr{L}_{\mathscr{R}_1}\right] \leq \epsilon'(k) \\
\Rightarrow &\mathsf{Pr}\left[\{(h_*, \mathsf{pk}_*, \bar{z}_*)\} \notin \mathscr{L}_{\mathscr{R}_1}\right] \leq \frac{\epsilon'(k)}{\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_*, \bar{z}_*)\}, \bar{\mathrm{P}}_*) = 1\right]} \\
\Rightarrow &\mathsf{Pr}\left[\{(h_*, \mathsf{pk}_*, \bar{z}_*)\} \notin \mathscr{L}_{\mathscr{R}_1}\right] \leq \frac{\epsilon'(k)}{\lambda(k)} \\
\Rightarrow &\mathsf{Pr}\left[\log_g(\mathsf{pk}_*) \neq \log_{h_*}(\bar{z}_*)\right] \leq \frac{\epsilon'(k)}{\lambda(k)} \\
\Rightarrow &\mathsf{Pr}\left[\log_g(\mathsf{pk}_*) = \log_{h_*}(z_*)\right] \leq \frac{\epsilon'(k)}{\lambda(k)}
\end{aligned} \tag{5.9}$$

Finally, from (5.4), (5.8) and (5.9) we deduce the probability that $\mathscr{B}$ wins the experiment:

$$\begin{aligned}
&\mathsf{Pr}\left[\exists\ \mathsf{pk} \in \mathrm{L}_*, x' = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)\right] \geq \lambda'(k) \\
\Rightarrow &\mathsf{Pr}\left[\exists\ \mathsf{pk} \in \mathrm{L}_* \backslash \{\mathsf{pk}_*\}, x' = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)\right] + \mathsf{Pr}\left[x' = \log_g(\mathsf{pk}_*) = \log_{h_*}(z_*)\right] \geq \lambda'(k) \\
\Rightarrow &\mathsf{Pr}\left[y = g^x\right] + \mathsf{Pr}\left[x' = \log_g(\mathsf{pk}_*) = \log_{h_*}(z_*)\right] \geq \lambda'(k) \\
\Rightarrow &\mathsf{Pr}\left[y = g^x\right] \geq \lambda'(k) - \mathsf{Pr}\left[\log_g(\mathsf{pk}_*) = \log_{h_*}(z_*)\right] \\
\Rightarrow &\mathsf{Pr}\left[y = g^x\right] \geq \lambda'(k) - \frac{\epsilon'(k)}{\lambda(k)}
\end{aligned}$$

Since $\lambda'(k) - \frac{\epsilon'(k)}{\lambda(k)}$ is non negligible then $\mathscr{B}$ solve the DL problem with non-negligible probability. □

5.4.5 Non-seizability

In this section we show that EVeR is $\mathsf{non\text{-}sei\text{-}1}$ and $\mathsf{non\text{-}sei\text{-}2}$ secure, which implies that it is non-seizable.

Lemma 36 EVeR *is n-non-sei-1 secure for any polynomially bounded n under the DL assumption in the random oracle model.*

Proof: For any $n \in \mathbb{N}$, let K (resp. Sim_n) be the knowledge extractor (resp. simulator) of LogEq_n. Assume that there exists $\mathcal{A} \in \textsc{poly}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{n\text{-}\mathsf{non\text{-}sei\text{-}1}}_{\mathrm{EVeR},\mathcal{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \textsc{poly}(k)$ that solves the DL problem with non-negligible probability.

Algorithm $\mathcal{B}(y)$**:** For all $i \in [\![1,n]\!]$, it picks $x_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathsf{pk}_i = y^{x_i}$. $\mathcal{B}$ runs $x' \leftarrow \mathsf{K}^{\mathcal{A}'(\{\mathsf{pk}_i\}_{1\le i\le n})}(k)$ where $\mathcal{A}'$ is the following algorithm:

Algorithm $\mathcal{A}'(\{\mathsf{pk}_i\}_{1\le i\le n})$**:** It runs $(\mathrm{L}_*, m_*, \sigma_*, l_*, \pi_*) \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{1\le i\le n})$. It simulates the oracles to $\mathcal{A}$ as follows:

Random oracle $\mathrm{H}(.)$**:** On the i^{th} input M_i, if there exists $j \in \mathbb{N}$ such that $j < i$ and $\mathrm{M}_j = \mathrm{M}_i$, then it sets $u_j = u_i$. Else it picks $u_i \xleftarrow{\$} \mathbb{Z}_p^*$. Finally, it returns g^{u_i}.

Oracle $\mathsf{Sig}(\cdot,\cdot,\cdot)$**:** On the i^{th} input (L_i, l_i, m_i), it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. Using the oracle $\mathrm{H}(.)$, it computes $h_i = \mathrm{H}(m_i||r_i||\mathrm{L}_i)$, then there exists $j \in \mathbb{N}$ such that $j < i$ and $m_i||r_i||\mathrm{L}_i = \mathrm{M}_j$. It computes $z_i = \mathsf{pk}_{l_i}^{u_j}$ and it runs $\mathrm{P}_i \leftarrow \mathsf{Sim}_{|\mathrm{L}_i|}(\{(h_i, \mathsf{pk}_l, z_i)\}_{\mathsf{pk}_l \in \mathrm{L}_i})$. It returns $\sigma_i = (r_i, z_i, \mathrm{P}_i)$ to $\mathcal{A}$.

Oracle $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$**:** On the i^{th} input $(\mathrm{L}'_i, m'_i, \sigma'_i, l'_i)$, it parses $\sigma'_i = (r'_i, z'_i, \mathrm{P}'_i)$. It computes $h'_i = \mathrm{H}(m'_i||r'_i||\mathrm{L}'_i)$ using the oracle $\mathrm{H}(.)$, then there exists $j \in \mathbb{N}$ such that $m'_i||r'_i||\mathrm{L}'_i = \mathrm{M}_j$. It computes $\bar{z}_i = \mathsf{pk}_{l'_i}^{u_j}$ and it runs $\bar{\mathrm{P}}_i \leftarrow \mathsf{Sim}_1(\{(h'_i, \mathsf{pk}_{l'_i}, \bar{z}_i)\})$. It returns $(\bar{z}_i, \bar{\mathrm{P}}_i)$ to $\mathcal{A}$.

Finally, $\mathcal{A}'$ computes $h_* = \mathrm{H}(m_*||r_*||\mathrm{L}_*)$ using the random oracle $\mathrm{H}(\cdot)$, then it parses $\sigma_* = (r_*, z_*, \mathrm{P}_*)$ and returns P_*.

Finally $\mathcal{B}$ returns $x = \frac{x'}{x_{l_*}}$.

Analysis: We parse $\sigma_* = (r_*, z_*, \mathrm{P}_*)$ and $\pi_* = (\bar{z}_*, \bar{\mathrm{P}}_*)$. Assume that $\mathcal{A}$ wins the experiment, then we have:

$$\mathsf{Ver}(\mathrm{L}_*, \sigma_*, m_*) = 1 \tag{5.10}$$

$$\mathsf{Judge}(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}_{l_*}, \pi_*) = 1 \tag{5.11}$$

$$\forall\ i \in [\![1, q_{\mathrm{S}}]\!], \sigma_i \neq \sigma_* \tag{5.12}$$

where q_{S} is the number of queries to the oracle $\mathsf{Sig}(\cdot,\cdot,\cdot)$. Moreover, equation (5.12) implies that $\forall\ i \in [\![1, q_{\mathrm{S}}]\!], \mathrm{P}_i \neq \mathrm{P}_*$, then P_* was not generated by the simulator $\mathsf{Sim}_{|\mathrm{L}_*|}$. Equation (5.11) implies:

$$\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_{l_*}, z_*)\}, \pi_*) = 1 \tag{5.13}$$

$$z_* = \bar{z}_* \tag{5.14}$$

Equation (5.10) implies:

$$\mathsf{L.Ver}_{|\mathrm{L}_*|}(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1$$

We deduce that:

$$\Pr\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_{l_*}, z_*)\}, \pi_*) = 1\right] \geq \Pr\left[1 \leftarrow \mathsf{Exp}^{n\text{-}\mathsf{non\text{-}sei\text{-}1}}_{\mathrm{EVeR},\mathcal{A}}(k)\right] \geq \lambda(k)$$

$$\Pr\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1\right] \geq \Pr\left[1 \leftarrow \mathsf{Exp}^{n\text{-}\mathsf{non\text{-}sei\text{-}1}}_{\mathrm{EVeR},\mathcal{A}}(k)\right] \geq \lambda(k)$$

Since LogEq_n is sound, there exists a negligible function ϵ such that:

$$\begin{aligned}
&\mathsf{Pr}\left[(\mathsf{L.Ver}_1(\{(h_*,\mathsf{pk}_{l_*},z_*)\},\bar{\mathrm{P}}_*)=1)\wedge(\{(h_*,\mathsf{pk}_{l_*},z_*)\}\not\in\mathscr{L}_{\mathscr{R}_1})\right]\leq\epsilon(k)\\
\Rightarrow&\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*,\mathsf{pk}_{l_*},z_*)\},\bar{\mathrm{P}}_*)=1\right]\cdot\mathsf{Pr}\left[\{(h_*,\mathsf{pk}_{l_*},z_*)\}\not\in\mathscr{L}_{\mathscr{R}_1}\right]\leq\epsilon(k)\\
\Rightarrow&\mathsf{Pr}\left[\{(h_*,\mathsf{pk}_{l_*},z_*)\}\not\in\mathscr{L}_{\mathscr{R}_1}\right]\leq\frac{\epsilon(k)}{\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*,\mathsf{pk}_{l_*},z_*)\},\bar{\mathrm{P}}_*)=1\right]}\\
\Rightarrow&\mathsf{Pr}\left[\{(h_*,\mathsf{pk}_{l_*},z_*)\}\not\in\mathscr{L}_{\mathscr{R}_1}\right]\leq\frac{\epsilon(k)}{\lambda(k)}\\
\Rightarrow&\mathsf{Pr}\left[\log_g(\mathsf{pk}_{l_*})\neq\log_{h_*}(z_*)\right]\leq\frac{\epsilon(k)}{\lambda(k)}\\
\Rightarrow&\mathsf{Pr}\left[\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(z_*)\right]\geq1-\frac{\epsilon(k)}{\lambda(k)}
\end{aligned}$$

Assume that $\mathscr{A}$ wins the experiment and that $\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)$. Equation (5.14) implies $\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)=\log_{h_*}(z_*)$. We set:

$$\lambda'(k)=\mathsf{Pr}\left[x'=\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)\right]$$

Since LogEq_n is valid and K is an extractor for LogEq_n, there exists a polynomial t and a negligible function ϵ' such that:

$$\begin{aligned}
\lambda'(k)&=\mathsf{Pr}\left[x'=\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)\right]\\
&=\mathsf{Pr}\left[(\{(h_*,\mathsf{pk}_l,z_*)\}_{\mathsf{pk}_l\in\mathrm{L}_*},x')\in\mathscr{R}_{|\mathrm{L}_*|}\right]\\
&\geq t\left(\mathsf{Pr}\left[\mathsf{L.Ver}_1(\{(h_*,\mathsf{pk}_l,z_*)\}_{\mathsf{pk}_l\in\mathrm{L}_*},\mathrm{P}_*)=1\right]\right)-\epsilon'(k)\\
&\geq t(\lambda(k))-\epsilon'(k)
\end{aligned}$$

Note that $x'=\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)\Rightarrow x=\log_g(y)$, indeed, since $x'=\frac{x}{x_{l_*}}$:

$$\begin{aligned}
x'=\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)&\Rightarrow g^{x'}=\mathsf{pk}_{l_*}\\
&\Rightarrow g^{\frac{x'}{x_{l_*}}}=\mathsf{pk}_{l_*}^{\frac{1}{x_{l_*}}}\\
&\Rightarrow g^{x}=\left(y^{x_{l_*}}\right)^{\frac{1}{x_{l_*}}}\\
&\Rightarrow g^{x}=y\\
&\Rightarrow x=\log_g(y)
\end{aligned}$$

Finally, the probability that $\mathscr{B}$ breaks the DL problem is:

$$\begin{aligned}
\mathsf{Pr}\left[x=\log_g(y)\right]&\geq\mathsf{Pr}\left[\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)\right]\cdot\mathsf{Pr}\left[x=\log_g(y)|\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)\right]\\
&\geq\mathsf{Pr}\left[\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)\right]\cdot\mathsf{Pr}\left[x'=\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)|\log_g(\mathsf{pk}_{l_*})=\log_{h_*}(\bar{z}_*)\right]\\
&\geq\left(1-\frac{\epsilon(k)}{\lambda(k)}\right)\cdot\lambda'(k)\\
&\geq\lambda'(k)-\epsilon(k)\left(\frac{\lambda'(k)}{\lambda(k)}\right)
\end{aligned}$$

Since $\lambda'(k)-\epsilon(k)\left(\frac{\lambda'(k)}{\lambda(k)}\right)$ is non negligible then $\mathscr{B}$ solve the DL problem with non-negligible probability. □

Lemma 37 EVeR *is n-non-sei-2 secure for any polynomially bounded n under the DL assumption in the random oracle model.*

Proof: For any $n \in \mathbb{N}$, let K (resp. Sim_n) be the knowledge extractor (resp. simulator) of LogEq_n. Assume that there exists an adversary $\mathscr{A} \in \text{POLY}(k)$ such the advantage $\lambda(k) = \mathsf{Adv}^{n\text{-non-sei-2}}_{\mathrm{EVeR},\mathscr{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathscr{B} \in \text{POLY}(k)$ that solves the DL problem with non-negligible probability.

Algorithm $\mathscr{B}(y)$**:** For all $i \in [\![1,n]\!]$, it picks $x_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathsf{pk}_i = y^{x_i}$. $\mathscr{B}$ runs $x' \leftarrow \mathsf{K}^{\mathscr{A}'(\{\mathsf{pk}_i\}_{1\le i\le n})}(k)$ where $\mathscr{A}'$ is the following algorithm:

Algorithm $\mathscr{A}'(\{\mathsf{pk}_i\}_{1\le i\le n})$**:** It runs $(\mathrm{L}_*, m_*, \sigma_*) \leftarrow \mathscr{A}(\{\mathsf{pk}_i\}_{1\le i\le n})$. It simulates the oracles to $\mathscr{A}$ as follows:

Random oracle $\mathrm{H}(.)$**:** On the i^{th} input M_i, if there exists $j \in \mathbb{N}$ such that $j < i$ and $\mathrm{M}_j = \mathrm{M}_i$, then it sets $u_j = u_i$. Else it picks $u_i \xleftarrow{\$} \mathbb{Z}_p^*$. Finally, it returns g^{u_i}.

Oracle $\mathsf{Sig}(\cdot,\cdot,\cdot)$**:** On the i^{th} input (L_i, l_i, m_i), it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. Using the oracle $\mathrm{H}(.)$, it computes $h_i = \mathrm{H}(m_i||r_i||\mathrm{L}_i)$, then there exists $j \in \mathbb{N}$ such that $j < i$ and $m_i||r_i||\mathrm{L}_i = \mathrm{M}_j$. It computes $z_i = \mathsf{pk}_{l_i}^{u_j}$ and it runs $\mathrm{P}_i \leftarrow \mathsf{Sim}_{|\mathrm{L}_i|}(\{(h_i, \mathsf{pk}_l, z_i)\}_{\mathsf{pk}_l \in \mathrm{L}_i})$. It returns $\sigma_i = (r_i, z_i, \mathrm{P}_i)$ to $\mathscr{A}$.

Oracle $\mathsf{Proof}(\cdot,\cdot,\cdot,\cdot)$**:** On the i^{th} input $(\mathrm{L}'_i, m'_i, \sigma'_i, l'_i)$, it parses $\sigma'_i = (r'_i, z'_i, \mathrm{P}'_i)$. It computes $h'_i = \mathrm{H}(m'_i||r'_i||\mathrm{L}'_i)$ using the oracle $\mathrm{H}(.)$, then there exists $j \in \mathbb{N}$ such that $m'_i||r'_i||\mathrm{L}'_i = \mathrm{M}_j$. It computes $\bar{z}_i = \mathsf{pk}_{l'_i}^{u_j}$ and it runs $\bar{\mathrm{P}}_i \leftarrow \mathsf{Sim}_1(\{(h'_i, \mathsf{pk}_{l'_i}, \bar{z}_i)\})$. It returns $(\bar{z}_i, \bar{\mathrm{P}}_i)$ to $\mathscr{A}$.

Finally, $\mathscr{A}'$ computes $h_* = \mathrm{H}(m_*||r_*||\mathrm{L}_*)$ using the random oracle $\mathrm{H}(\cdot)$, then it parses $\sigma_* = (r_*, z_*, \mathrm{P}_*)$ and returns P_*.

If there exists $i \in [\![1,n]\!]$ such that $\mathsf{pk}_i = g^{\frac{x'}{x_i}}$, then $\mathscr{B}$ returns $x = \frac{x'}{x_i}$, else it returns $\perp$.

Analysis: We parse $\sigma_* = (r_*, z_*, \mathrm{P}_*)$. Assume that $\mathscr{A}$ wins the experiment, then we have, for any $\pi_* \leftarrow \mathsf{Proof}(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}, \mathsf{sk})$ where $\pi_* = (\bar{z}_*, \bar{\mathrm{P}}_*)$:

$$\mathsf{Ver}(\mathrm{L}_*, \sigma_*, m_*) = 1 \tag{5.15}$$

$$\mathsf{Judge}(\mathrm{L}_*, m_*, \sigma_*, \mathsf{pk}, \pi_*) = 1 \tag{5.16}$$

$$\forall\ i \in [\![1, q_{\mathrm{S}}]\!], \sigma_i \neq \sigma_* \tag{5.17}$$

where q_{S} is the number of queries to the oracle $\mathsf{Sig}(\cdot,\cdot,\cdot)$. Moreover, equation (5.17) implies that $\forall\ i \in [\![1, q_{\mathrm{S}}]\!], \mathrm{P}_i \neq \mathrm{P}_*$, then P_* was not generated by the simulator Sim. We deduce the following equation from (5.15):

$$\mathsf{L.Ver}_{|\mathrm{L}_*|}(\{(h_*, \mathsf{pk}_l, z_*)\}_{\mathsf{pk}_l \in \mathrm{L}_*}, \mathrm{P}_*) = 1$$

Thus $\mathscr{A}$ returns a valid proof with non negligible probability $\lambda(k)$. We set:

$$\lambda'(k) = \Pr\left[\exists\ \mathsf{pk}_l \in \mathrm{L}_*, x = \log_g(\mathsf{pk}_l) = \log_{h_*}(z_*)\right] \tag{5.18}$$

Since K is an extractor for LogEq_n, it implies that:

$$\lambda'(k) \geq t(\lambda(k)) - \epsilon(k)$$

We deduce the following equation from (5.16):

$$\bar{z}_* = z_*$$

$$\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}, z_*)\}, \bar{\mathrm{P}}_*) = 1$$

We deduce the following equation:

$$\Pr\left[\mathsf{L.Ver}_1(\{(h_*, \mathsf{pk}_*, z_*)\}, \bar{\mathrm{P}}_*) = 1\right] \geq \mathsf{Adv}^{n\text{-non-sei-2}}_{\mathrm{EVeR},\mathscr{A}}(k) = \lambda(k)$$

Since LogEq_n is sound, there exists a negligible function ϵ' such that:

$$\begin{aligned}
&\Pr\left[(\mathsf{L.Ver}_1(\{(h_*,\mathsf{pk}_*,z_*)\},\bar{\mathsf{P}}_*)=1)\wedge(\{(h_*,\mathsf{pk}_*,z_*)\}\not\in\mathscr{L}_{\mathscr{R}_1})\right]\leq\epsilon'(k)\\
\Rightarrow&\Pr\left[\mathsf{L.Ver}_1(\{(h_*,\mathsf{pk}_*,z_*)\},\bar{\mathsf{P}}_*)=1\right]\cdot\Pr\left[\{(h_*,\mathsf{pk}_*,z_*)\}\not\in\mathscr{L}_{\mathscr{R}_1}\right]\leq\epsilon'(k)\\
\Rightarrow&\Pr\left[\{(h_*,\mathsf{pk}_*,z_*)\}\not\in\mathscr{L}_{\mathscr{R}_1}\right]\leq\frac{\epsilon'(k)}{\Pr\left[\mathsf{L.Ver}_1(\{(h_*,\mathsf{pk}_*,z_*)\},\bar{\mathsf{P}}_*)=1\right]}\\
\Rightarrow&\Pr\left[\{(h_*,\mathsf{pk}_*,z_*)\}\not\in\mathscr{L}_{\mathscr{R}_1}\right]\leq\frac{\epsilon'(k)}{\lambda(k)}\\
\Rightarrow&\Pr\left[\log_g(\mathsf{pk}_*)\neq\log_{h_*}(z_*)\right]\leq\frac{\epsilon'(k)}{\lambda(k)}
\end{aligned}\tag{5.19}$$

We note the following implication:

$$\exists\,\mathsf{pk}_l\in\mathsf{L}_*\backslash\{\mathsf{pk}\},x=\log_g(\mathsf{pk}_l)=\log_{h_*}(z_*)\Rightarrow\log_g(\mathsf{pk}_*)\neq\log_{h_*}(z_*)$$

We deduce that:

$$\Pr\left[\exists\,\mathsf{pk}_l\in\mathsf{L}_*\backslash\{\mathsf{pk}\},x=\log_g(\mathsf{pk}_l)=\log_{h_*}(z_*)\right]\leq\Pr\left[\log_g(\mathsf{pk}_*)\neq\log_{h_*}(z_*)\right]\leq\frac{\epsilon'(k)}{\lambda(k)}\tag{5.20}$$

Finally, from (5.18), (5.19) and (5.20) we deduce the probability that $\mathscr{B}$ wins the experiment:

$$\begin{aligned}
&\Pr\left[\exists\,\mathsf{pk}_l\in\mathsf{L}_*,x=\log_g(\mathsf{pk}_l)=\log_{h_*}(z_*)\right]\geq\lambda'(k)\\
\Rightarrow&\Pr\left[x=\log_g(\mathsf{pk})=\log_{h_*}(z_*)\right]+\Pr\left[\exists\,\mathsf{pk}_l\in\mathsf{L}_*\backslash\{\mathsf{pk}\},x=\log_g(\mathsf{pk}_l)=\log_{h_*}(z_*)\right]\geq\lambda'(k)\\
\Rightarrow&\Pr\left[x=\log_g(\mathsf{pk})=\log_{h_*}(z_*)\right]\geq\lambda'(k)-\Pr\left[\exists\,\mathsf{pk}_l\in\mathsf{L}_*\backslash\{\mathsf{pk}\},x=\log_g(\mathsf{pk}_l)=\log_{h_*}(z_*)\right]\\
\Rightarrow&\Pr\left[y=g^x\right]\geq\lambda'(k)-\Pr\left[\log_g(\mathsf{pk})\neq\log_{h_*}(z_*)\right]\\
\Rightarrow&\Pr\left[y=g^x\right]\geq\lambda'(k)-\frac{\epsilon'(k)}{\lambda(k)}
\end{aligned}$$

Since $\lambda'(k)-\frac{\epsilon'(k)}{\lambda(k)}$ is non negligible then $\mathscr{B}$ solve the DL problem with non-negligible probability. □

5.4.6 Security of EVeR

Finally, we conclude this section with the following theorem.

Theorem 28 EVeR *is correct, unforgeable, anonymous, accountable and non-seizable under the DDH assumption in the random oracle model.*

Proof: The theorem resumes the properties proved in 32, 33, 34, 35, 36 and 37. □

5.5 Algorithms Complexity

In this section, we give the complexity of the algorithms of our scheme EVeR. We give the number of exponentiations in a prime order group for each algorithm. Moreover, we give the size of some values outputted by these algorithms (keys, signatures and proofs). This size is given in the number of group elements. For the sake of clarity, we do not distinguish between elements of a group $\mathbb{G}$ of prime order p where the DDH assumption is hard and elements of $\mathbb{Z}_p^*$.

We give the complexity analysis for a ring of n users, where n is any positive integer. Moreover, we give the complexity analysis in the particular case where the ring contains only two users. We focus on this case because the generic sanitizable signature we present in the next chapter uses verifiable ring signatures for two users.

In Table 5.1, we give the number of exponentiations of each algorithm of the LogEq_n proof system and the size of a proof π_n^{LE} depending to the number n. The first line corresponds to the general case, the two other lines correspond to the case where $n = 1$ and $n = 2$.

In Table 5.2 ande 5.3 , we give the number of exponentiations for each algorithm of EVeR and the size of the secret/public keys $\mathsf{sk}_{\mathrm{EV}}$ and $\mathsf{pk}_{\mathrm{EV}}$, the signature σ_n^{EV} and the size of the proof π_n^{EV}. These values depend on the size n of the ring. The first line corresponds to the generic case, where the values depend on the chosen proof system. The second line corresponds to the case where $n = 2$ and where the proof system is LogEq_2.

LogEq_n	$\mathsf{LEprove}_n$	$\mathsf{LEverif}_n$	π_n^{LE}
n	$2+4\cdot(n-1)$	$4\cdot n$	$4\cdot n$
$n=1$	2	4	4
$n=2$	6	8	8

Table 5.1: Complexity analysis of LogEq (Definition 65).

EVeR	$\mathsf{V.Gen}$	$\mathsf{V.Sig}_n$	$\mathsf{V.Ver}_n$	$\mathsf{V.Proof}$	$\mathsf{V.Judge}$
n (generic)	1	$1+\mathsf{LEprove}_n$	$\mathsf{LEverif}_n$	$1+\mathsf{LEprove}_1$	$\mathsf{LEverif}_1$
$n=2$ (with LogEq_n)	1	7	8	3	4

Table 5.2: Complexity analysis of the algorithms of EVeR (Definition 66).

EVeR	$\mathsf{sk}_{\mathrm{EV}}$	$\mathsf{pk}_{\mathrm{EV}}$	σ_n^{EV}	π_n^{EV}
n (generic)	1	1	$2+\pi_n^{\mathsf{LE}}$	$1+\pi_1^{\mathsf{LE}}$
$n=2$ (with LogEq_n)	1	1	10	5

Table 5.3: Complexity analysis of the elements of EVeR (Definition 66).

5.6 Conclusion

In this chapter, we revisit the notion of verifiable ring signature. We improve its properties of verifiability, we give a security model and we design a simple, efficient and secure scheme named EVeR. In the next chapter, we show that this primitive can be used to design sanitizable signature schemes that are very efficient. In the future, it will be interesting to find other applications to verifiable ring signatures that are secure in our model. Moreover, we will aim to design a scheme that is secure in the standard model.

Chapter 6

Unlinkable Sanitizable Signatures from Verifiable Ring Signature.

Contents

An *Unlinkable Sanitizable Signature* scheme allows a sanitizer to modify some parts of a signed message such that nobody can link the modified signature to the original one. In this chapter, we design an unlinkable sanitizable signature scheme called GUSS (Generic Unlinkable Sanitizable Signature). This scheme is generic, and is based on verifiable ring signatures. We show that GUSS instantiated with EVeR and Schnorr's signature is twice as efficient as the most efficient scheme of the literature. Moreover, we improve the definition of *accountable sanitizable signature.* A sanitizable signature scheme is said to be *accountable* when the signer can prove whether a signature is sanitized or not. We formally define the notion of *strong accountability* where the sanitizer can also prove the origin of a signature. We show that this security property is required in practice. Finally, we prove the security properties of GUSS, including strong accountability. This work has been conducted in collaboration with Pascal Lafourcade and has been published in the paper *"Unlinkable and Strongly Accountable Sanitizable Signatures from Verifiable Ring Signature"* at CANS 2017 conference.

6.1 Introduction

Sanitizable Signatures were introduced by Ateniese *et al.* [ACdMT05], but similar primitives were independently proposed in [SBZ02]. In this primitive, a *signer* allows a proxy (called the *sanitizer*) to modify some parts of a signed message. For example, a magistrate wishes to delegate the power to summon someone to the court to his secretary. He signs the message "*Franz* is summoned to court for an interrogation on *Monday*" [Kaf06] and gives the signature to his secretary, where "Franz" and "Monday" are sanitizable and the other parts are fixed. Hence, in order to summon Joseph K. on Saturday in the name of the magistrate, the secretary can change the signed message into "*Joseph K.* is summoned to the court for an interrogation on *Saturday*".

Ateniese *et al.* in [ACdMT05] proposed some applications of this primitive in privacy of health data, authenticated media streams and reliable routing information.

6.1.1 Application in health Data Protection

We show how this primitive can be used to protect the patients privacy in health data. Consider a hospital where each physician signs a report on the pathology of each of his patient. Then the reports are sent by the hospital to a medical research center which compiles them into its database. However, for the sake of privacy, the hospital cannot reveal the identity of the patients. Using sanitizable signatures, the physician signs the report such that the hospital can change the confidential data about the patient (*i.e.,* his name and his address). Thus, the medical research center uses the sanitized report without learning any private information about the patient. Figure 6.1 illustrates this application: the physician signs the message *"Alice has lupus"* using his secret key sk and the public key of the hospital spk where the sanitizable part of the message is *Alice* (in bold and red) and the fixed part is *has lupus* (in blue). Then, in order to hide the identity of Alice, the hospital sanitizes the signature using its secret key ssk and the physician public key pk such that the message becomes *"Bob has lupus"*. Finally, the research center verifies the signature that authenticates the physician on the message *"Bob has lupus"*.

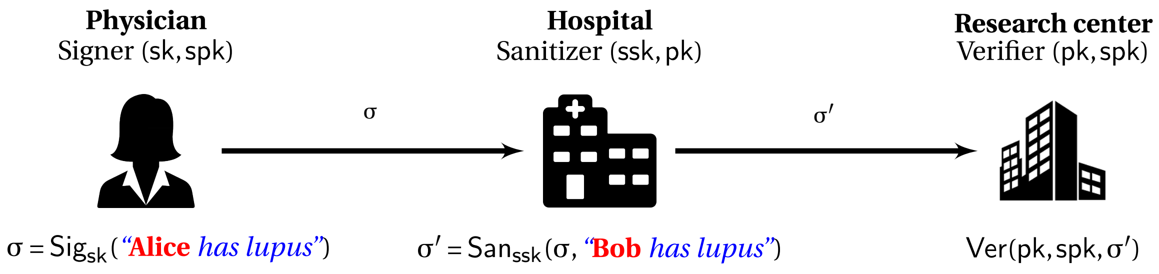

Physician
Signer $(\mathsf{sk}, \mathsf{spk})$
Hospital
Sanitizer $(\mathsf{ssk}, \mathsf{pk})$
Research center
Verifier $(\mathsf{pk}, \mathsf{spk})$
σ
σ'
$\sigma = \mathsf{Sig}_{\mathsf{sk}}($"**Alice** *has lupus*"$)$
$\sigma' = \mathsf{San}_{\mathsf{ssk}}(\sigma,$ "**Bob** *has lupus*"$)$
$\mathsf{Ver}(\mathsf{pk}, \mathsf{spk}, \sigma')$

Figure 6.1: Sanitizable signature for privacy in health data.

Some years later, Brzuska *et al.* [BFLS10] points a non studied property called *unlinkability*: a sanitizable signature scheme is said to be unlinkable when it is not possible to link two sanitized signatures produced from the same signature. To illustrate the relevance of this property, we consider the following scenario. The physician signs a medical report containing the sentence *"Alice has lupus"*, where *Alice* and *lupus* are sanitizable and *has* is fixed. The hospital sanitizes it in σ'_1 with the message *"Bob has lupus"* and sends it to the medical research center, as in the previous paragraph. On the other hand, the hospital would like to send the report to the patient mutual health insurance, but it wants to hide some private information about the pathology of the patient, so it sanitizes the report a second times in σ'_2 with the message *"Alice has a cold"* and sends it to the mutual insurance. We consider a malicious hospital employee who recovers the two signatures σ'_1 and σ'_2. If the scheme is not unlinkable, then the adversary knows that these two signatures was produced from the same signature, so he can try to recover the original message by compiling the information in the two messages. He deduces that the original message is probably *"Alice has lupus"* or *"Bob has a cold"*. Figure 6.2 resumes this attack scenario.

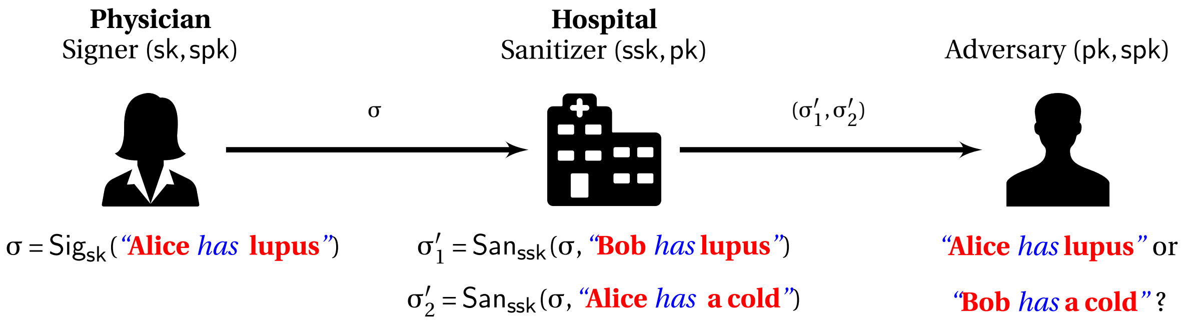

Physician
Signer (sk, spk)
Hospital
Sanitizer (ssk, pk)
Adversary (pk, spk)
σ
(σ'_1, σ'_2)
$\sigma = \mathsf{Sig}_{\mathsf{sk}}($"Alice has lupus"$)$
$\sigma'_1 = \mathsf{San}_{\mathsf{ssk}}(\sigma,$ "Bob has lupus"$)$
$\sigma'_2 = \mathsf{San}_{\mathsf{ssk}}(\sigma,$ "Alice has a cold"$)$
"Alice has lupus" or
"Bob has a cold"?

Figure 6.2: Attack by linkage.

6.1.2 Functionalities

We are going to explain in more details how a sanitizable signature scheme works. A sanitizable signature scheme has the following features:

System initialization: A user generates the public setup. Then, the signer and the sanitizer generate their public/private keys using the key generation algorithms.

Signature and verification: The signer chooses a message and an *admissible* function that decides how the message can be modified by the sanitizer, then he signs this message according to the admissible function using his secret key, the sanitizer public key and the signature algorithm. Anybody can verify the validity of this signature using the verification algorithm with the public keys of both the signer and the sanitizer.

Sanitization: The sanitizer sanitizes the message according to the admissible function, *i.e.*, he produces a new signature of a new *valid* message using the original signature, his secret key, the public key of the signer and the sanitize algorithm. Anybody can verify the validity of this signature using the verification algorithm (the same as for the original signature) with the public keys of both the signer and the sanitizer.

Proof and judgment: The signer or the sanitizer can prove whether he produced a given signature using his secret key and his proof algorithm. This proof can be verified by anyone using the judge algorithms with the public keys of both the signer and the sanitizer.

6.1.3 Security Goals

In their pioneer paper, Ateniese *et al.* [ACdMT05] introduced five security properties for sanitizable signature: *accountability*, *immutability*, *privacy*, *transparency* and *accountability*. They were later formalized by Brzuska *et al.* in [BFF+09]. In [BFLS10], Brzuska *et al.* point a non-studied but relevant property called *unlinkability*.

In this chapter we introduce a new security definition called the *strong accountability*, that improves the notion of accountability defined by Brzuska *et al.* in [BFF+09].

These security properties are informally defined as follows.

Unfogeability: A sanitizable signature scheme is said to be *unforgeable* when no unauthorized user can generate a valid signature.

Immutability: A sanitizable signature scheme is said to be *immutable* when the sanitizer cannot transform a signature from an unauthorized message.

Privacy: A sanitizable signature scheme is said to be *private* when no information about the original message is leaked by a sanitized signature.

Transparency: A sanitizable signature scheme is said to be *transparent* when nobody can guess whether a signature is sanitized or not.

Unlinkability: A sanitizable signature scheme is said to be *unlinkable* when it is not possible to link the sanitized signature to the original one.

Accountability: A sanitizable signature scheme is said to be *accountable* when the signer can prove the origin of a signature (signer or sanitizer) using the proof algorithm such that:

1. The signer cannot forge a signature and a proof that this signature has been forged by the sanitizer.
2. The sanitizer cannot forge a signature such that the proof algorithm blames the signer.

The proof algorithm requires the secret key of the signer.

Strong accountability: To show that the accountability definition is too weak in a practical use, we consider the following scenario. The signer claims that he lost his secret key because of problems with his hard drive. Therefore he cannot prove the origin of a litigious signature. Unfortunately, there is no way to verify whether the signer is lying or not. Indeed, without his secret key, the signer cannot generate the proof for the litigious signature, hence nobody can judge whether the signature is sanitized or not. Depending on whether the signer is lying, there is a risk of accusing the sanitizer wrongly. To solve this problem, we add a second proof algorithm that allows the sanitizer to prove the origin of a signature. A sanitizable signature scheme is said to be *strongly accountable* when it has the two following properties in addition to accountability.

1. The sanitizer cannot sanitize a signature σ and prove that σ is not sanitized.
2. The signer cannot forge a signature such that the sanitizer proof algorithm accuses the sanitizer.

Note that these two properties are built in the same way as the two properties of accountability, except that they consider a dishonest sanitizer instead of a dishonest signer.

6.1.4 Contributions

Our first contribution is to formally define strong accountability. Our second contribution is to propose an efficient and generic unlinkable sanitizable signature scheme called GUSS. It is instantiated by a verifiable ring signature scheme and an unforgeable signature scheme. It is the first sanitizable signature scheme that achieves strong accountability. In the following, we compare the efficiency of our scheme GUSS with the other unlinkable sanitizable signature schemes of the literature.

Brzuska *et al.* [BFLS10]. This scheme is based on group signatures. Our scheme is build on the same idea, but it uses ring signatures instead of group signatures. For small groups, ring signatures are usually much more efficient than group signatures. Since the scheme of Brzuska *et al.* and GUSS uses group/ring signatures for groups of two users, GUSS is much more practicale.

Canard*et al.* [CJL12]. This work focus on sanitizable signatures with several signers and sanitizers. The scheme proposed in [CJL12] achieves unlinkability in the same way as Brzuska *et al.* [BFLS10], *i.e.,* using group signatures. Hence it suffers from the same weaknesses of efficiency.

Fleischhacker *et al.* [FKM$^+$16]. This scheme is based on *signatures with re-randomizable keys*. It is generic, however it uses different tools that must have special properties to be compatible with each other. To the best of our knowledge, it is the most efficient unlinkable scheme of the literature. GUSS instantiated with EVeR and the Schnorr's signature is in average twice as efficient as the best instantiation of this scheme given in [FKM$^+$16].

Lai *et al.* [LZCS16]. Recently, Lai *et al.* proposed an unlinkable sanitizable signature scheme that is secure in the standard model, however it uses pairing and it is much less efficient than the scheme of Fleischhacker *et al.* that is in the random oracle model, thus it is much less efficient than our scheme. In their paper [LZCS16], Lai *et al.* give a detailed comparison of the efficiency of the three schemes of the literature.

6.1.5 Related Works

Sanitizable Signatures were first introduced by Ateniese *et al.* [ACdMT05]. Later, Brzuska *et al.* gave formal security definitions [BFF$^+$09] for *unfogeability*, *immutability*, *privacy*, *transparency* and *accountability*. *Unlinkability* was also introduced and formally defined by Brzuska *et al.* in [BFLS10]. In [BPS13], Brzuska *et al.* introduce an alternative definition of accountability called *non-interactive public accountability* where the proofs of the origin of the signatures are generated by a third party. One year later, the same authors propose a stronger definition of unlinkability [BPS14] and design a scheme that is both strongly unlinkable and non-interactive public accountable. However, non-interactive public accountability is not compatible with transparency. In this chapter, we focus on schemes that are unlinkable, transparent and interactive accountable. To the best of our knowledge, there are only four schemes with these three properties, *i.e.,* [BFLS10, FKM$^+$16, LZCS16, CJL12].

Some works focus on other properties of sanitizable signature schemes that we do not consider here, as schemes with multiple sanitizers [CJL12], or schemes where the capability of the sanitizer is limited [CJ10]. Finally, there exist other primitives that solve related but different problems as homomorphic signatures [JMSW02], redactable signatures [BBD$^+$10] or proxy signatures [FP08]. Differences between these primitives and sanitizable signatures have been detailed in [FKM$^+$16].

6.2 Formal Definitions

In this section, we give the formal definition and the security properties of the sanitizable signatures. Our definition introduces new algorithms that allow the sanitizer to prove the origin of a signature. Moreover, in addition to the usual security models of [BFF$^+$09], we improve the definition of accountability by introducing two new security experiments.

As it is mentioned in Introduction, the sanitizable signature schemes have the following security properties: *unforgeability*, *immutability*, *privacy*, *transparency* and *accountability*. In [BFF$^+$09] authors show that if a scheme has the *immutability*, the *transparency* and the *accountability* properties, then it has the *unforgeability* and the *privacy* properties. Hence we do not need to prove these two properties, so we do not recall their formal definitions in this section.

6.2.1 Sanitizable Signature

A sanitisable signature scheme (SS) contains 10 algorithms denoted Init, SiGen, SaGen, Sig, San, Ver, $\mathsf{SiProof}$, $\mathsf{SaProof}$, $\mathsf{SiJudge}$ and $\mathsf{SaJudge}$. Init outputs the setup values. SiGen and SaGen generate respectively the signer and the sanitizer public/private keys. As in classical signature schemes, the algorithms Sig and Ver allow the users to sign a message and to verify a signature. However, the signatures are computed using a sanitizer public key and an admissible function $\textsc{Adm}$. The algorithm San allows the sanitizer to transform a signature of a message m according to a modification function $\textsc{Mod}$: if $\textsc{Mod}$ is admissible according to the admissible function (*i.e.,* $\textsc{Adm}(\textsc{Mod})) = 1$) this algorithm returns a signature of the message $m' = \textsc{Mod}(m)$.

Concretely, the admissible function $\textsc{Adm}$ allows the sanitizer to modify some parts of the message m. The part of the message that cannot be modified is called *the fixed part* of the message. We define a function $\textsc{Fix}$ that returns the fixed part of a message given an admissible function $\textsc{Adm}$. More formally, we denote by $\textsc{Fix}_{\textsc{Adm}}(m)$ the fixed part of the message m according to $\textsc{Adm}$. Note that for any admissible function $\textsc{Adm}$, any message m and any modification function $\textsc{Mod}$ such that $\textsc{Adm}(\textsc{Mod}) = 1$, we have $\textsc{Fix}_{\textsc{Adm}}(m) = \textsc{Fix}_{\textsc{Adm}}(\textsc{Mod}(m))$ since the fixed part of the message remains the same after modification.

Finally, SiProof allows the signer to prove whether a signature is sanitized or not. Proofs outputted by this algorithm can be verified by anybody using the algorithm SiJudge. The algorithms SaProof and SaJudge have the same functionalities as SiProof and SiJudge, but the proofs are computed from the secret parameters of the sanitizer instead of the signer.

Definition 67 (Sanitizable Signature (SS)) *A* Sanitizable Signature *(SS) scheme is a tuple of* 10 *algorithms $\Pi = (\mathit{Init}, \mathit{SiGen}, \mathit{SaGen}, \mathit{Sig}, \mathit{San}, \mathit{Ver}, \mathit{SiProof}, \mathit{SaProof}, \mathit{SiJudge}, \mathit{SaJudge})$ defined by:*

$\mathsf{Init}(k)$: *It returns a setup value* set.
$\mathsf{SiGen}(\mathsf{set})$: *It returns a pair of signer public/private keys* $(\mathsf{pk}, \mathsf{sk})$.
$\mathsf{SaGen}(\mathsf{set})$: *It returns a pair of sanitizer public/private keys* $(\mathit{spk}, \mathsf{ssk})$.
$\mathsf{Sig}(m, \mathsf{sk}, \mathsf{spk}, \textsc{Adm})$: *This algorithm computes a signature σ from the message m using the secret key* sk*, the sanitizer public key spk and the admissible function* $\textsc{Adm}$*. Note that we assume that* $\textsc{Adm}$ *can be efficiently recovered from any signature as in the definition of Fleischhacker et al. [FKM+ 16].*
$\mathsf{San}(m, \textsc{Mod}, \sigma, \mathsf{pk}, \mathsf{ssk})$: *Let the admissible function* $\textsc{Adm}$ *of the signature* σ*. If* $\textsc{Adm}(\textsc{Mod}) = 1$ *then this algorithm returns a signature* σ' *of the message* $m' = \textsc{Mod}(m)$ *using the signature* σ*, the signer public key* pk *and the sanitizer secret key* ssk*. Else it returns* $\perp$.
$\mathsf{Ver}(m, \sigma, \mathsf{pk}, \mathsf{spk})$: *It returns a bit* b.
$\mathsf{SiProof}(\mathsf{sk}, m, \sigma, \mathsf{spk})$: *It returns a signer proof* π_{si} *for the signature* σ *of* m.
$\mathsf{SaProof}(\mathsf{ssk}, m, \sigma, \mathsf{pk})$: *It returns a sanitizer proof* π_{sa} *for the signature* σ *of* m.
$\mathsf{SiJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{si}})$: *It returns a bit d or the bottom symbol* $\perp$*: if* π_{si} *proves that* σ *comes from the signer corresponding to the public key* pk*, then* $d = 1$*, else if* π_{si} *proves that* σ *comes from the sanitizer corresponding to the public key spk, then* $d = 0$*, else the algorithm outputs* $\perp$.
$\mathsf{SaJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{sa}})$: *It returns a bit d or the bottom symbol* $\perp$*: if* π_{sa} *proves that* σ *comes from the signer corresponding to the public key* pk *then* $d = 1$*, else if* π_{sa} *proves that* σ *comes from the sanitizer corresponding to the public key spk then* $d = 0$*, else the algorithm outputs* $\perp$.

Moreover a SS is said to be correct *when the following equations hold for any* $k \in \mathbb{N}$*, any* $m \in \{0,1\}^*$*, any admissible function* $\textsc{Adm}$ *and any modification function* $\textsc{Mod}$ *such that* $\textsc{Adm}(\textsc{Mod}) = 1$*:*

$$\Pr\left[\begin{array}{l} \mathsf{set} \leftarrow \mathit{Init}(k); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathit{SiGen}(\mathsf{set}); (\mathit{spk}, \mathsf{ssk}) \leftarrow \mathit{SaGen}(\mathsf{set}); \\ \sigma \leftarrow \mathit{Sig}(m, \mathsf{sk}, \mathit{spk}, \textsc{Adm}); \sigma' \leftarrow \mathit{San}(m, \textsc{Mod}, \sigma, \mathsf{pk}, \mathsf{ssk}); \\ b_1 \leftarrow \mathit{Ver}(m, \sigma, \mathsf{pk}, \mathit{spk}); b_2 \leftarrow \mathit{Ver}(\textsc{Mod}(m), \sigma', \mathsf{pk}, \mathit{spk}); \end{array} : (b_1 = 1) \wedge (b_2 = 1)\right] = 1$$

$$\Pr\left[\begin{array}{l} \mathsf{set} \leftarrow \mathit{Init}(k); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathit{SiGen}(\mathsf{set}); (\mathit{spk}, \mathsf{ssk}) \leftarrow \mathit{SaGen}(\mathsf{set}); \\ \sigma \leftarrow \mathit{Sig}(m, \mathsf{sk}, \mathit{spk}, \textsc{Adm}); \\ \pi_{sa} \leftarrow \mathit{SaProof}(\mathsf{ssk}, m, \sigma, \mathsf{pk}); \pi_{si} \leftarrow \mathit{SiProof}(\mathsf{sk}, m, \sigma, \mathit{spk}); \\ d_1 \leftarrow \mathit{SaJudge}(m, \sigma, \mathsf{pk}, \mathit{spk}, \pi_{sa}); d_2 \leftarrow \mathit{SiJudge}(m, \sigma, \mathsf{pk}, \mathit{spk}, \pi_{si}); \end{array} : (d_1 = 1) \wedge (d_2 = 1)\right] = 1$$

$$\Pr\left[\begin{array}{l} \mathsf{set} \leftarrow \mathit{Init}(k); (\mathsf{pk},\mathsf{sk}) \leftarrow \mathit{SiGen}(\mathsf{set}); (\mathit{spk},\mathsf{ssk}) \leftarrow \mathit{SaGen}(\mathsf{set});\\ \sigma \leftarrow \mathit{Sig}(m,\mathsf{sk},\mathit{spk},\textsc{Adm}); \sigma' \leftarrow \mathit{San}(m,\textsc{Mod},\sigma,\mathsf{pk},\mathsf{ssk});\\ \pi_{sa} \leftarrow \mathit{SaProof}(\mathsf{ssk},\textsc{Mod}(m),\sigma',\mathsf{pk});\\ \pi_{si} \leftarrow \mathit{SiProof}(\mathsf{sk},\textsc{Mod}(m),\sigma',\mathit{spk});\\ d_1 \leftarrow \mathit{SaJudge}(\textsc{Mod}(m),\sigma',\mathsf{pk},\mathit{spk},\pi_{sa});\\ d_2 \leftarrow \mathit{SiJudge}(\textsc{Mod}(m),\sigma',\mathsf{pk},\mathit{spk},\pi_{si}); \end{array} : (d_1 = 0) \wedge (d_2 = 0)\right] = 1$$

6.2.2 Immutability

A SS is immutable when no adversary is able to sanitize a signature without the corresponding sanitizer secret key or to sanitize a signature using a modification function that is not admissible (*i.e.*, $\textsc{Adm}(\textsc{Mod}) = 0$). To help him, the adversary has access to a signature oracle $\mathsf{Sig}(.,\mathsf{sk},.,.)$ that signs chosen messages and a proof oracle $\mathsf{SiProof}(\mathsf{sk},.,.,.)$ that computes signer proofs for given signatures. Note that to win the experiment, the adversary must not return a signature that was outputted by the signature oracle.

Definition 68 (Immutability [BFF⁺09]) *Let Π be a SS, k be an integer and $\mathcal{A} \in \textsc{poly}(k)$ be an algorithm. Let the two following oracles be:*

$\mathsf{Sig}(.,\mathsf{sk},.,.)$**:** *On input $(m,\textsc{Adm},\mathit{spk})$, this oracle runs $\sigma \leftarrow \mathit{Sig}(m,\mathsf{sk},\textsc{Adm},\mathit{spk})$ and returns σ.*
$\mathsf{SiProof}(\mathsf{sk},.,.,.)$**:** *On input (m,σ,spk), this oracle runs $\pi_{si} \leftarrow \mathit{SiProof}(\mathsf{sk},m,\sigma,\mathit{spk})$ and returns π_{si}.*

We define the Immut experiment as follows, where q_{Sig} is the number of calls to the oracle $\mathit{Sig}(.,\mathsf{sk},.,.)$, $(m_i,\textsc{Adm}_i,\mathit{spk}_i)$ is the i^{th} query asked to the oracle $\mathit{Sig}(.,\mathsf{sk},.,.)$ and σ_i is the corresponding response:

```
$\mathsf{Exp}^{\mathsf{Immut}}_{\Pi,\mathcal{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{SiGen}(\mathsf{set})$
$(\mathsf{spk}_*,m_*,\sigma_*) \leftarrow \mathcal{A}^{\mathsf{Sig}(.,\mathsf{sk},.,.),\mathsf{SiProof}(\mathsf{sk},.,.,.)}(\mathsf{pk})$
if $(\mathsf{Ver}(m_*,\sigma_*,\mathsf{pk},\mathsf{spk}_*) = 1)$ and $(\forall\ i \in [\![1,q_{Sig}]\!], (\mathit{spk}_* \neq \mathit{spk}_i)$ or
    $(\forall\ \textsc{Mod}$ such that $\textsc{Adm}_i(\textsc{Mod}) = 1, m_* \neq \textsc{Mod}(m_i)))$
then return $1$, else $0$
```

We define the Immut advantage of $\mathcal{A}$ against Π as follows:

$$\mathsf{Adv}^{\mathit{Immut}}_{\Pi,\mathcal{A}}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathit{Immut}}_{\Pi,\mathcal{A}}(k)\right]$$

We define the Immut advantage against Π as follows:

$$\mathsf{Adv}^{\mathit{Immut}}_{\Pi}(k) = \max_{\mathcal{A} \in \textsc{poly}(k)} \left\{\mathsf{Adv}^{\mathit{Immut}}_{\Pi,\mathcal{A}}(k)\right\}$$

Π is said to be Immut secure, or Immutable, *when the Immut advantage against Π is negligible.*

6.2.3 Transparency

The transparency ensures that no adversary is able to distinguish whether a signature is sanitized or not. In addition to the signature oracle and the signer proof oracle, the adversary has access to a sanitize oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$ that sanitizes chosen signatures and a sanitizer proof oracle $\mathsf{SaProof}(\mathsf{ssk},.,.,.)$ that computes sanitizer proofs for given signatures. Moreover the adversary has access to a challenge oracle $\mathsf{Sa/Si}(b,\mathsf{pk},\mathsf{spk},\mathsf{sk},\mathsf{ssk},.,.,.)$ that depends on a randomly chosen bit b: this oracle signs a given message and sanitizes it, if $b = 0$ then it outputs the original signature, otherwise it outputs the sanitized signature. The adversary cannot use the proof oracles on the signatures outputted by the challenge oracle, *i.e.*, $(\mathsf{S}_{\mathsf{Sa/Si}} \cap (\mathsf{S}_{\mathsf{SiProof}} \cup \mathsf{S}_{\mathsf{SaProof}}) = \emptyset)$ where $\mathsf{S}_{\mathsf{Sa/Si}}$

(resp. $\mathsf{S}_{\mathsf{SiProof}}$ and $\mathsf{S}_{\mathsf{SaProof}}$) corresponds to the set of all signatures outputted by the oracle Sa/Si (resp. sent to the oracles SiProof and SaProof). To succeed the experiment, the adversary must guess b.

Definition 69 (Transparency [BFF+09]) *Let Π be a SS, k be an integer and $\mathscr{A} \in \textsc{poly}(k)$ be an algorithm. Let the two following oracles be:*

$\mathsf{San}(.,.,.,.,\mathsf{ssk})$**:** *On input $(m, \textsc{Mod}, \sigma, \mathsf{pk})$, it runs $\sigma' \leftarrow \mathit{San}(m, \textsc{Mod}, \sigma, \mathsf{pk}, \mathsf{ssk})$ and returns σ'.*
$\mathsf{SaProof}(\mathsf{ssk},.,.,.)$**:** *On input (m, σ, pk), this oracle runs $\pi_{sa} \leftarrow \mathit{SaProof}(\mathsf{ssk}, m, \sigma, \mathsf{pk})$ and returns π_{sa}.*
$\mathsf{Sa}/\mathsf{Si}(b, \mathsf{pk}, \mathsf{spk}, \mathsf{sk}, \mathsf{ssk}, ., ., .)$**:** *On input $(m, \textsc{Adm}, \textsc{Mod})$, if $\textsc{Adm}(\textsc{Mod}) = 0$, this oracle returns $\bot$. Else if $b = 0$, this oracle runs $\sigma \leftarrow \mathit{Sig}(\textsc{Mod}(m), \mathsf{sk}, \mathit{spk}, \textsc{Adm})$, else if $b = 1$, this oracle runs $\sigma \leftarrow \mathit{San}(m, \textsc{Mod}, \mathit{Sig}(m, \mathsf{sk}, \mathit{spk}, \textsc{Adm}), \mathsf{pk}, \mathsf{ssk})$. It returns σ.*

We define the Trans experiment as follows, where $\mathit{Sig}(., \mathsf{sk}, ., .)$ and $\mathit{SiProof}(\mathsf{sk}, ., ., .)$ are defined as in Definition 68, and where $\mathsf{S}_{\mathit{Sa}/\mathit{Si}}$ (resp. $\mathsf{S}_{\mathit{SiProof}}$ and $\mathsf{S}_{\mathit{SaProof}}$) corresponds to the set of all signatures outputted by the oracle Sa/Si (resp. sent to the oracles SiProof and SaProof):

```
$\mathsf{Exp}^{\mathbf{Trans}}_{\Pi,\mathscr{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SiGen}(\mathsf{set})$
$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathsf{SaGen}(\mathsf{set})$
$b \overset{\$}{\leftarrow} \{0,1\}$
$b' \leftarrow \mathscr{A}^{\substack{\mathsf{Sig}(.,\mathsf{sk},.,.),\mathsf{San}(.,.,.,.,\mathsf{ssk}),\mathsf{SiProof}(\mathsf{sk},.,.,.) \\ \mathsf{SaProof}(\mathsf{ssk},.,.,.),\mathsf{Sa}/\mathsf{Si}(\mathsf{b},\mathsf{pkk},\mathsf{spk},\mathsf{sk},\mathsf{ssk},.,.,.)}}(\mathsf{pk}, \mathsf{spk})$
if $(b = b')$ and $(\mathsf{S}_{\mathsf{Sa}/\mathsf{Si}} \cap (\mathsf{S}_{\mathsf{SiProof}} \cup \mathsf{S}_{\mathsf{SaProof}}) = \emptyset)$
then return $1$, else $0$
```

We define the Trans advantage of $\mathscr{A}$ against Π as follows:

$$\mathsf{Adv}^{\mathit{Trans}}_{\Pi,\mathscr{A}}(k) = \left| \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathit{Trans}}_{\Pi,\mathscr{A}}(k)\right] - \frac{1}{2} \right|$$

We define the Trans advantage against Π as follows:

$$\mathsf{Adv}^{\mathit{Trans}}_{\Pi}(k) = \max_{\mathscr{A} \in \textsc{poly}(k)} \left\{ \mathsf{Adv}^{\mathit{Trans}}_{\Pi,\mathscr{A}}(k) \right\}$$

Π is said to be Trans secure, or transparent*, when the Trans advantage against Π is negligible.*

6.2.4 Unlinkablility

The unlinkablility property ensures that a sanitized signature cannot be linked to the original one. We consider an adversary that has access to the signature oracle, the sanitize oracle, and both the signer and the sanitizer proof oracles. Moreover, the adversary has access to a challenge oracle $\mathsf{LRSan}(b, \mathsf{pk}, \mathsf{ssk}, ., .)$ that depends on a bit b: this oracle takes as input two signatures σ_0 and σ_1, the two corresponding messages m_0 and m_1 and two modification functions $\textsc{Mod}_0$ and $\textsc{Mod}_1$ chosen by the adversary. If the two signatures have the same admissible function $\textsc{Adm}$, and if $\textsc{Mod}_0$ and $\textsc{Mod}_1$ are admissible according to $\textsc{Adm}$ and if $\textsc{Mod}_0(m_0) = \textsc{Mod}_1(m_1)$ then the challenge oracle sanitizes σ_b using $\textsc{Mod}_b$ and returns it. The goal of the adversary is to guess the bit b.

Definition 70 (Unlinkability [BFF+09]) *Let Π be a SS, k be an integer and $\mathscr{A} \in \textsc{poly}(k)$ be an algorithm. Let the two following oracles be:*

$\mathsf{LRSan}(b, \mathsf{pk}, \mathsf{ssk}, ., .)$**:** *On input $((m_0, \textsc{Mod}_0, \sigma_0)(m_1, \textsc{Mod}_1, \sigma_1))$, if for $i \in \{0,1\}$, $\mathit{Ver}(m_i, \sigma_i, \mathsf{pk}, \mathit{spk}) = 1$ and $\textsc{Adm}_0 = \textsc{Adm}_1$ and $\textsc{Adm}_0(\textsc{Mod}_0) = 1$ and $\textsc{Adm}_1(\textsc{Mod}_1) = 1$ and $\textsc{Mod}_0(m_0) = \textsc{Mod}_1(m_1)$, then this oracle runs $\sigma' \leftarrow \mathit{San}(m_b, \textsc{Mod}_b, \sigma_b, \mathsf{pk}, \mathsf{ssk})$ and returns σ', else it returns 0.*

We define the Unlink experiment as follows, where Sig$(.,\mathsf{sk},.,.)$ *and SiProof*$(\mathsf{sk},.,.,.)$ *are defined as in Definition 68 and San*$(.,.,.,.,\mathsf{ssk})$ *and SaProof*$(\mathsf{ssk},.,.,.)$ *are defined as in Definition 69:*

```
$\mathsf{Exp}^{\mathbf{Unlink}}_{\Pi,\mathscr{A}}(k)$:
$\mathsf{set} \leftarrow \mathit{Init}(k)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathit{SiGen}(\mathsf{set})$
$(\mathit{spk},\mathsf{ssk}) \leftarrow \mathit{SaGen}(\mathsf{set})$
$b \overset{\$}{\leftarrow} \{0,1\}$
$b' \leftarrow \mathscr{A}^{\substack{\mathit{Sig}(.,sk,.,.),\mathit{San}(.,.,.,.,ssk) \\ \mathit{SiProof}(sk,.,.,.),\mathit{SaProof}(ssk,.,.,.),\mathit{LRSan}(b,pkk,spk,.,.)}}(\mathsf{pk},\mathit{spk})$
$\mathit{if}\,(b=b')$ then return $1$, else $0$
```

We define the Unlink advantage of $\mathscr{A}$ *against* Π *as follows:*

$$\mathsf{Adv}^{\mathit{Unlink}}_{\Pi,\mathscr{A}}(k) = \left|\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathit{Unlink}}_{\Pi,\mathscr{A}}(k)\right] - \frac{1}{2}\right|$$

We define the Unlink advantage against Π *as follows:*

$$\mathsf{Adv}^{\mathit{Unlink}}_{\Pi}(k) = \max_{\mathscr{A}\in\mathrm{POLY}(k)}\left\{\mathsf{Adv}^{\mathit{Unlink}}_{\Pi,\mathscr{A}}(k)\right\}$$

Π *is said to be Unlink secure, or* unlinkable*, when the Unlink advantage against* Π *is negligible.*

6.2.5 Accountability

Standard defintion of accountability is splited into two security experiments: the sanitizer accountability and the signer accountability. In the sanitizer accountability experiment, the adversary has access to the signature oracle and the signer proof oracle. Its goal is to forge a signature such that the signer proof algorithm returns a proof that this signature is not sanitized. To succeed the experiment, this signature must not come from the signature oracle.

Definition 71 (Sanitizer Accountability [BFF+09]) *Let* Π *be a SS, k be an integer and* $\mathscr{A} \in \mathrm{POLY}(k)$ *be an algorithm. We define the SaAcc-1 experiment as follows, where the oracles Sig*$(.,\mathsf{sk},.,.)$ *and SiProof*$(\mathsf{sk},.,.,.)$ *are defined as in Definition 68,* q_{Sig} *is the number of calls to the oracle Sig*$(.,\mathsf{sk},.,.)$*, the tuple* $(m_i,\mathrm{ADM}_i,spk_i)$ *is the* i^{th} *query asked to the oracle Sig*$(.,\mathsf{sk},.,.)$ *and* σ_i *is the corresponding response:*

```
$\mathsf{Exp}^{\mathbf{SaAcc\text{-}1}}_{\Pi,\mathscr{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{SiGen}(\mathsf{set})$
$(\mathsf{spk}_*,m_*,\sigma_*) \leftarrow \mathscr{A}^{\mathsf{Sig}(.,\mathsf{sk},.,.),\mathsf{SiProof}(\mathsf{sk},.,.,.)}(\mathsf{pk})$
$\pi^*_{\mathsf{si}} \leftarrow \mathsf{SiProof}(\mathsf{sk},m_*,\sigma_*,\mathsf{spk}_*)$
if $\forall\ i \in [\![1,q_{\mathsf{Sig}}]\!]$, $(\sigma_* \neq \sigma_i)$ and $(\mathsf{Ver}(m_*,\sigma_*,\mathsf{pk},\mathsf{spk}_*)=1)$
    and $(\mathsf{SiJudge}(m_*,\sigma_*,\mathsf{pk},\mathsf{spk}_*,\pi^*_{\mathsf{si}}) \neq 0)$
then return $1$, else$0$
```

We define the SaAcc-1 advantage of $\mathscr{A}$ *against* Π *as follows:*

$$\mathsf{Adv}^{\mathit{SaAcc\text{-}1}}_{\Pi,\mathscr{A}}(k) = \mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathit{SaAcc\text{-}1}}_{\Pi,\mathscr{A}}(k)\right]$$

We define the SaAcc-1 advantage against Π *as follows:*

$$\mathsf{Adv}^{\mathit{SaAcc\text{-}1}}_{\Pi}(k) = \max_{\mathscr{A}\in\mathrm{POLY}(k)}\left\{\mathsf{Adv}^{\mathit{SaAcc\text{-}1}}_{\Pi,\mathscr{A}}(k)\right\}$$

Π *is said to be SaAcc-1 secure, or* sanitizer accountable*, when the SaAcc-1 advantage against* Π *is negligible.*

In the signer accountability experiment, the adversary knows the public key of the sanitizer and has access to the sanitize oracle and the sanitizer proof oracle. Its goal is to forge a signature together with a proof that this signature is sanitized. To succeed the experiment, this signature must not come from the sanitize oracle.

Definition 72 (Signer Accountability [BFF+09]) *Let Π be a SS, k be an integer and $\mathcal{A} \in \text{POLY}(k)$ be an algorithm. We define the SiAcc-1 experiment as follows, where the oracle San$(.,.,.,.,\mathsf{ssk})$ and SaProof$(\mathsf{ssk},.,.,.)$ are defined as in Definition 69 and where q_{San} is the number of calls to the oracle San$(.,.,.,.,\mathsf{ssk})$, $(m_i, \textsc{Mod}_i, \sigma_i, \mathsf{pk}_i)$ is the i^{th} query asked to the oracle San$(.,.,.,.,\mathsf{ssk})$ and σ'_i is the corresponding response:*

```
$\mathsf{Exp}^{\mathbf{SiAcc\text{-}1}}_{\Pi,\mathcal{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$(\mathsf{spk},\mathsf{ssk}) \leftarrow \mathsf{SaGen}(\mathsf{set})$
$(\mathsf{pk}_*, m_*, \sigma_*, \pi^*_{\mathsf{si}}) \leftarrow \mathcal{A}^{\mathsf{San}(.,.,.,.,\mathsf{ssk}),\mathsf{SaProof}(\mathsf{ssk},.,.,.)}(\mathsf{spk})$
if $\forall\ i \in [\![1, q_{\mathsf{San}}]\!]$, $(\sigma_* \neq \sigma'_i)$ and $(\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}) = 1)$
    and $(\mathsf{SiJudge}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}, \pi^*_{\mathsf{si}}) = 0)$
then return 1, else 0
```

We define the SiAcc-1 advantage of $\mathcal{A}$ against Π as follows:

$$\mathsf{Adv}^{\textit{SiAcc-1}}_{\Pi,\mathcal{A}}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}^{\textit{SiAcc-1}}_{\Pi,\mathcal{A}}(k)\right]$$

We define the SiAcc-1 advantage against Π as follows:

$$\mathsf{Adv}^{\textit{SiAcc-1}}_{\Pi}(k) = \max_{\mathcal{A}\in\text{POLY}(k)} \left\{\mathsf{Adv}^{\textit{SiAcc-1}}_{\Pi,\mathcal{A}}(k)\right\}$$

Π is said to be SiAcc-1 secure, or signer accountable, *when the SiAcc-1 advantage against Π is negligible.*

6.2.6 Strong Accountability

Since our definition of sanitizable signature provides a second proof algorithm for the sanitizer, we define two additional security experiments (for signer and sanitizer accountability) to ensure the soundness of the proofs computed by this algorithm. We say that a scheme is strongly accountable when it is signer and sanitizer accountable for both the signer and the sanitizer proof algorithms.

Thus, in our second signer accountability experiment, we consider an adversary that has access to the sanitize oracle and the sanitizer proof oracle. Its goal is to forge a signature such that the sanitizer proof algorithm returns a proof that this signature is sanitized. To win the experiment, this signature must not come from the sanitize oracle.

Definition 73 (Strong Signer Accountability) *Let Π be a SS, k be an integer and $\mathcal{A} \in \text{POLY}(k)$ be an algorithm. We define the SiAcc-2 experiment as follows, where q_{San} is the number of calls to the oracle San$(.,.,.,.,\mathsf{ssk})$, $(m_i, \textsc{Mod}_i, \sigma_i, \mathsf{pk}_i)$ is the i^{th} query asked to the oracle San$(.,.,.,.,\mathsf{ssk})$ and σ'_i is the corresponding response:*

```
$\mathsf{Exp}^{\mathbf{SiAcc\text{-}2}}_{\Pi,\mathcal{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$(\mathsf{spk},\mathsf{ssk}) \leftarrow \mathsf{SaGen}(\mathsf{set})$
$(\mathsf{pk}_*, m_*, \sigma_*) \leftarrow \mathcal{A}^{\mathsf{San}(.,.,.,.,\mathsf{ssk}),\mathsf{SaProof}(\mathsf{ssk},.,.,.)}(\mathsf{spk})$
$\pi^*_{\mathsf{sa}} \leftarrow \mathsf{SaProof}(\mathsf{ssk}, m_*, \sigma_*, \mathsf{pk}_*)$
if $\forall\ i \in [\![1, q_{\mathsf{San}}]\!]$, $(\sigma_* \neq \sigma'_i)$ and $(\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}) = 1)$
    and $(\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}, \pi^*_{\mathsf{sa}}) \neq 1)$
then return 1, else 0
```

We define the SiAcc-2 advantage of $\mathscr{A}$ against Π as follows:

$$\mathsf{Adv}^{\textit{SiAcc-2}}_{\Pi,\mathscr{A}}(k) = \mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\textit{SiAcc-2}}_{\Pi,\mathscr{A}}(k)\right]$$

We define the SiAcc-2 advantage against Π as follows:

$$\mathsf{Adv}^{\textit{SiAcc-2}}_{\Pi}(k) = \max_{\mathscr{A}\in\mathrm{POLY}(k)}\left\{\mathsf{Adv}^{\textit{SiAcc-2}}_{\Pi,\mathscr{A}}(k)\right\}$$

Π is said to be SiAcc-2 secure when the SiAcc-2 advantage against Π is negligible. Π is said to be strong signer accountable *when it is both SiAcc-1 and SiAcc-2 secure.*

Finally, in our second sanitizer accountability experiment, we consider an adversary that knows the public key of the signer and has access to the signer oracle and the signer proof oracle. Its goal is to sanitize a signature with a proof that this signature is not sanitized. To win the experiment, this signature must not come from the signer oracle.

Definition 74 (Strong Sanitizer Accountability) *Let Π be a SS, k be an integer and $\mathscr{A} \in \mathrm{POLY}(k)$ be an algorithm. We define the SaAcc-2 experiment as follows, where $\mathsf{Sig}(.,\mathsf{sk},.,.)$ and $\mathsf{SiProof}(\mathsf{sk},.,.,.)$ are defined as in Definition 68, q_{Sig} is the number of calls to the oracle $\mathsf{Sig}(.,\mathsf{sk},.,.)$, $(m_i, \mathrm{ADM}_i, spk_i)$ is the i^{th} query asked to the oracle $\mathsf{Sig}(.,\mathsf{sk},.,.)$ and σ_i is the corresponding response:*

```
$\mathsf{Exp}^{\mathsf{SaAcc\text{-}2}}_{\Pi,\mathscr{A}}(k)$:
$\mathsf{set} \leftarrow \mathsf{Init}(k)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{SaGen}(\mathsf{set})$
$(\mathsf{spk}_*, m_*, \sigma_*, \pi^*_{\mathsf{sa}}) \leftarrow \mathscr{A}^{\mathsf{Sig}(.,\mathsf{sk},.,.),\mathsf{SiProof}(\mathsf{sk},.,.,.)}(\mathsf{pk})$
if $\forall\ i \in [\![1, q_{\mathsf{Sig}}]\!]$, $(\sigma_* \neq \sigma_i)$ and $(\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*) = 1)$
    and $(\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*, \pi^*_{\mathsf{sa}}) = 1)$
then return 1, else return 0
```

We define the SaAcc-2 advantage of $\mathscr{A}$ against Π as follows:

$$\mathsf{Adv}^{\textit{SaAcc-2}}_{\Pi,\mathscr{A}}(k) = \mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\textit{SaAcc-2}}_{\Pi,\mathscr{A}}(k)\right]$$

We define the SaAcc-2 advantage against Π as follows:

$$\mathsf{Adv}^{\textit{SaAcc-2}}_{\Pi}(k) = \max_{\mathscr{A}\in\mathrm{POLY}(k)}\left\{\mathsf{Adv}^{\textit{SaAcc-2}}_{\Pi,\mathscr{A}}(k)\right\}$$

Π is said to be SaAcc-2 secure when the SaAcc-2 advantage against Π is negligible. Π is said to be strong sanitizer accountable *when it is both SaAcc-1 and SaAcc-2 secure.*

6.3 GUSS: an Unlinkable Sanitizable Signature Scheme

We present our generic sanitizable signature scheme called GUSS. It is instantiated by a digital signature scheme (DS) and a verifiable ring signature scheme (VRS). This scheme works as follows: the signer signs the fixed part of the message using the DS, and signs the full message using the VRS within the group that contains the signer and the sanitizer. Then, to modify the signed message, the sanitizer reuses the digital signature of the fixed part and he signs the full modified message using the VRS. For any signature (sanitized or not), a user verifies its validity by using the verification algorithm of the DS on the signature of the fixed part and the verification algorithm of the VRS on the signature of the full message. The scheme is immutable because the sanitizer cannot modify the fixed part without the signer secret key. It is transparent since the signature of the full message is anonymous in a group containing the signer and the sanitizer. Moreover, it is unlinkable because the sanitized signature contains no information about the original message except its fixed parts. Finally, it is accountable because the signer and the sanitizer can prove the origine of the signature using the proof algorithm of the VRS.

Definition 75 (Generic Unlinkable Sanitizable Signature (GUSS)) *Let* $\mathrm{D} = (D.Init,\ D.Gen,\ D.Sig,\ D.Ver)$ *be a deterministic digital signature scheme and* $\mathrm{V} = (V.Init, V.Gen, V.Sig,\ V.Ver, V.Proof,\ V.Judge)$ *be a verifiable ring signature scheme.* $\mathrm{GUSS} = (Init,\ SiGen,\ SaGen,\ Sig,\ San,\ Ver,\ SiProof,\ SaProof,\ SiJudge,\ SaJudge)$ *instantiated by* (D,V) *is a sanitizable signature scheme defined by:*

$\mathsf{Init}(k)$: *It runs:*

$$\mathsf{set}_d \leftarrow D.Init(k)$$
$$\mathsf{set}_v \leftarrow V.Init(k)$$

Then it returns the setup $\mathsf{set} = (\mathsf{set}_d, \mathsf{set}_v)$.

$\mathsf{SiGen}(\mathsf{set})$: *This algorithm parses* $\mathsf{set} = (\mathsf{set}_d, \mathsf{set}_v)$, *then it runs:*

$$(\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow D.Gen(\mathsf{set}_d)$$
$$(\mathsf{pk}_v, \mathsf{sk}_v) \leftarrow V.Gen(\mathsf{set}_v)$$

It returns $(\mathsf{pk}, \mathsf{sk})$ *where* $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ *and* $\mathsf{sk} = (\mathsf{sk}_d, \mathsf{sk}_v)$.

$\mathsf{SaGen}(\mathsf{set})$: *It parses* $\mathsf{set} = (\mathsf{set}_d, \mathsf{set}_v)$ *and runs* $(spk, \mathsf{ssk}) \leftarrow V.Gen(\mathsf{set}_v)$. *It returns* (spk, ssk).

$\mathsf{Sig}(m, \mathsf{sk}, \mathsf{spk}, \mathrm{ADM})$: *It parses* $\mathsf{sk} = (\mathsf{sk}_d, \mathsf{sk}_v)$ *and computes the fixed message part* $\mathrm{M} \leftarrow \mathrm{FIX}_{\mathrm{ADM}}(m)$. *Then it runs:*

$$\sigma_1 \leftarrow D.Sig(\mathsf{sk}_d, (\mathrm{M}||\mathrm{ADM}||\mathsf{pk}||spk))$$
$$\sigma_2 \leftarrow V.Sig((\sigma_1||m)), \{\mathsf{pk}_v, spk\}, \mathsf{sk}_v)$$

It returns $\sigma = (\sigma_1, \sigma_2, \mathrm{ADM})$.

$\mathsf{San}(m, \mathrm{MOD}, \sigma, \mathsf{pk}, \mathsf{ssk})$: *It parses* $\sigma = (\sigma_1, \sigma_2, \mathrm{ADM})$ *and* $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$, *and computes the modified message* $m' \leftarrow \mathrm{MOD}(m)$. *Then it runs:*

$$\sigma_2' \leftarrow V.Sig((\sigma_1||m'), \{\mathsf{pk}_v, spk\}, \mathsf{ssk})$$

It returns $\sigma' = (\sigma_1, \sigma_2', \mathrm{ADM})$.

$\mathsf{Ver}(m, \sigma, \mathsf{pk}, \mathsf{spk})$: *It parses* $\sigma = (\sigma_1, \sigma_2, \mathrm{ADM})$ *and computes the fixed message part* $\mathrm{M} \leftarrow \mathrm{FIX}_{\mathrm{ADM}}(m)$. *Then it runs:*

$$b_1 \leftarrow D.Ver(\mathsf{pk}_d, (\mathrm{M}||\mathrm{ADM}||\mathsf{pk}||spk), \sigma_1)$$
$$b_2 \leftarrow V.Ver(\{\mathsf{pk}_d, spk\}, (\sigma_1||m), \sigma_2)$$

It returns $b = (b_1 \wedge b_2)$.

$\mathsf{SiProof}(\mathsf{sk}, m, \sigma, \mathsf{spk})$: *It parses* $\sigma = (\sigma_1, \sigma_2, \mathrm{ADM})$ *and* $\mathsf{sk} = (\mathsf{sk}_d, \mathsf{sk}_v)$. *Then it runs:*

$$\pi_{si} \leftarrow V.Proof(\{\mathsf{pk}_v, spk\}, (m||\sigma_1), \sigma_2, \mathsf{pk}_v, \mathsf{sk}_v)$$

It returns π_{si}.

$\mathsf{SaProof}(\mathsf{ssk}, m, \sigma, \mathsf{pk})$: *It parses the signature* $\sigma = (\sigma_1, \sigma_2, \mathrm{ADM})$. *Then it runs:*

$$\pi_{sa} \leftarrow V.Proof(\{\mathsf{pk}_v, spk\}, (m||\sigma_1), \sigma_2, spk, \mathsf{ssk})$$

It returns π_{sa}.

$\mathsf{SiJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{si}})$: *It parses* $\sigma = (\sigma_1, \sigma_2, \mathrm{ADM})$ *and* $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$. *Then it runs:*

$$b \leftarrow V.Judge(\{\mathsf{pk}_v, spk\}, (m||\sigma_1), \sigma_2, \mathsf{pk}_v, \pi_{si})$$

It returns b.

$\mathsf{SaJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{sa}})$: *It parses* $\sigma = (\sigma_1, \sigma_2, \mathrm{ADM})$ *and* $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$. *Then it runs:*

$$b \leftarrow V.Judge(\{\mathsf{pk}_v, spk\}, (m||\sigma_1), \sigma_2, spk, \pi_{sa})$$

It returns $(1 - b)$.

The signer secret key $\mathsf{sk} = (\mathsf{sk}_d, \mathsf{sk}_v)$ contains a secret key sk_d compatible with the DS scheme and a secret key sk_v compatible with the VRS scheme. The signer public key $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ contains the two corresponding public keys. The sanitizer public/secret key pair $(\mathsf{spk}, \mathsf{ssk})$ is generated as in the VRS scheme.

Let m be a message and M be the *fixed part* chosen by the signer according to the admissible function ADM. To sign m, the signer first signs M together with the public key of the sanitizer spk and the admissible function ADM using the DS scheme. We denote this signature by σ_1. The signer then generates σ_2, which is the signature of the full message m together with σ_1 using the VRS scheme for the set of public keys $\mathrm{L} = \{\mathsf{pk}_v, \mathsf{spk}\}$. Informally, he anonymously signs $(\sigma_1||m)$ within a group of two users: the signer and the sanitizer. The final sanitizable signature is $\sigma = (\sigma_1, \sigma_2)$. The verification algorithm works in two steps: it verifies the signature σ_1 and it verifies the anonymous signature σ_2.

To sanitize this signature $\sigma = (\sigma_1, \sigma_2)$, the sanitizer chooses an admissible message m' according to ADM (*i.e.,* m and m' have the same fixed part). Then he anonymously signs m' together with σ_1 using the VRS for the group $\mathrm{L} = \{\mathsf{pk}_v, \mathsf{spk}\}$ using the secret key ssk. We denote by σ'_2 this signature. The final sanitized signature is $\sigma' = (\sigma_1, \sigma'_2)$.

6.4 Security proofs of GUSS

In this section, we analyze the security of GUSS. We first informally explain why our scheme is secure before detailing its formal security proofs.

Immutability: Since it is produced by an unforgeable DS scheme, nobody can forge the signature σ_1 of the fixed part M without the signer secret key. Thus the sanitizer cannot change the fixed part of the signatures. Moreover, since σ_1 signs the public key of the sanitizer in addition to M, the other users cannot forge a signature of an admissible message using σ_1.

Transparency: According to the anonymity of σ_2 and σ'_2, nobody can guess if a signature comes from the signer or the sanitizer, and since both signatures have the same structure, nobody can guess whether a signature is sanitized or not.

Unlinkability: Assume that an adversary knows (i) two signatures σ^0 and σ^1 that have the same fixed part M according to the same function ADM for the same sanitizer and (ii) the sanitized signature $\sigma' = (\sigma'_1, \sigma'_2)$ computed from σ^b for a given admissible message m' and an unknown bit b. To achieve unlinkability, it must be hard to guess b. Since the DS scheme is deterministic, the two signatures $\sigma^0 = (\sigma^0_1, \sigma^0_2)$ and $\sigma^1 = (\sigma^1_1, \sigma^1_2)$ have the same first part (*i.e.,* $\sigma^0_1 = \sigma^1_1$). As it was shown before, the signature σ' has the same first part σ'_1 as the original one, thus $\sigma'_1 = \sigma^0_1 = \sigma^1_1$ and σ'_1 leaks no information about b. On the other hand, the second part of the sanitized signature σ'_2 is computed from the modified message m' and the first part of the original signature. Since $\sigma^0_1 = \sigma^1_1$, we deduce that σ'_2 leaks no information about b. Finally, the best strategy of the adversary is to randomly guess b.

(Strong) Accountability: the signer must be able to prove the provenance of a signature. It is equivalent to break the anonymity of the second parts σ_2 of this signature: if it was created by the signer then it is the original signature, else it was created by the sanitizer and it is a sanitized signature. By definition, the VRS scheme used to generate σ_2 provides a way to prove whether a user is the author of a signature or not. GUSS uses it in its proof algorithm to achieve accountability. Note that since the sanitizer uses the same VRS scheme to sanitize a signature, he can also prove the origin of a given signature to achieve the strong accountability.

6.4.1 Correctness

In this section we show that GUSS is correct.

Lemma 38 *Let* D *be a deterministic digital signature scheme and* V *be a verifiable ring signature scheme. If* D *and* V *are correct, then* GUSS *instanciated by* (D,V) *is correct.*

Proof: For any $k \in \mathbb{N}$, any $m \in \{0,1\}^*$, any admissible function $\textsc{Adm}$ and any modification function $\textsc{Mod}$ such that $\textsc{Adm}(\textsc{Mod}) = 1$, any $\mathsf{set} \leftarrow \mathsf{Init}(k)$, any $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{SiGen}(\mathsf{set})$ and any $(\mathsf{spk},\mathsf{ssk}) \leftarrow \mathsf{SaGen}(\mathsf{set})$ we have the following properties:

1. Using $\mathsf{set} = (\mathsf{set}_d, \mathsf{set}_v)$, $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ and $\mathsf{sk} = (\mathsf{sk}_d, \mathsf{sk}_v)$, we have:

$$\begin{aligned} (\mathsf{pk}_d,\mathsf{sk}_d) &\leftarrow \mathsf{D.Gen}(\mathsf{set}_d) \\ (\mathsf{pk}_v,\mathsf{sk}_v) &\leftarrow \mathsf{V.Gen}(\mathsf{set}_v) \\ (\mathsf{spk},\mathsf{ssk}) &\leftarrow \mathsf{V.Gen}(\mathsf{set}_v) \end{aligned}$$

For any $\sigma \leftarrow \mathsf{Sig}(m,\mathsf{sk},\mathsf{spk},\textsc{Adm})$ and $\sigma' \leftarrow \mathsf{San}(m,\textsc{Mod},\sigma,\mathsf{pk},\mathsf{ssk})$ such that $\sigma = (\sigma_1,\sigma_2,\textsc{Adm})$ and $\sigma = (\sigma'_1,\sigma'_2,\textsc{Adm}')$ we have:

$$\begin{aligned} \mathrm{M} &\leftarrow \textsc{Fix}_{\textsc{Adm}}(m) \\ \sigma_1 &\leftarrow \mathsf{D.Sig}(\mathsf{sk}_d,(\mathrm{M}||\textsc{Adm}||\mathsf{pk}||\mathsf{spk})) \\ \sigma_2 &\leftarrow \mathsf{V.Sig}((\sigma_1||m)),\{\mathsf{pk}_v,\mathsf{spk}\},\mathsf{sk}_v) \\ \sigma'_1 &= \sigma_1 \\ \sigma'_2 &\leftarrow \mathsf{V.Sig}((\sigma_1||m'),\{\mathsf{pk}_v,\mathsf{spk}\},\mathsf{ssk}) \\ \textsc{Adm}' &= \textsc{Adm} \end{aligned}$$

Since D is correct then:

$$\mathsf{D.Ver}(\mathsf{pk}_d,(\mathrm{M}||\textsc{Adm}||\mathsf{pk}||\mathsf{spk}),\sigma_1) = 1$$

Since V is correct then:

$$\begin{aligned} \mathsf{V.Ver}(\{\mathsf{pk}_d,\mathsf{spk}\},(\sigma_1||m),\sigma_2) &= 1 \\ \mathsf{V.Ver}(\{\mathsf{pk}_d,\mathsf{spk}\},(\sigma'_1||\textsc{Mod}(m)),\sigma'_2) &= 1 \end{aligned}$$

We deduce:

$$\begin{aligned} \mathsf{Ver}(m,\sigma,\mathsf{pk},\mathsf{spk}) &= 1 \\ \mathsf{Ver}(\textsc{Mod}(m),\sigma',\mathsf{pk},\mathsf{spk}) &= 1 \end{aligned}$$

2. Using $\mathsf{set} = (\mathsf{set}_d, \mathsf{set}_v)$, $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ and $\mathsf{sk} = (\mathsf{sk}_d, \mathsf{sk}_v)$, we have:

$$\begin{aligned} (\mathsf{pk}_d,\mathsf{sk}_d) &\leftarrow \mathsf{D.Gen}(\mathsf{set}_d) \\ (\mathsf{pk}_v,\mathsf{sk}_v) &\leftarrow \mathsf{V.Gen}(\mathsf{set}_v) \\ (\mathsf{spk},\mathsf{ssk}) &\leftarrow \mathsf{V.Gen}(\mathsf{set}_v) \end{aligned}$$

For any $\sigma \leftarrow \mathsf{Sig}(m,\mathsf{sk},\mathsf{spk},\textsc{Adm})$ such that $\sigma = (\sigma_1,\sigma_2,\textsc{Adm})$ we have:

$$\begin{aligned} \mathrm{M} &\leftarrow \textsc{Fix}_{\textsc{Adm}}(m) \\ \sigma_1 &\leftarrow \mathsf{D.Sig}(\mathsf{sk}_d,(\mathrm{M}||\textsc{Adm}||\mathsf{pk}||\mathsf{spk})) \\ \sigma_2 &\leftarrow \mathsf{V.Sig}((\sigma_1||m)),\{\mathsf{pk}_v,\mathsf{spk}\},\mathsf{sk}_v) \end{aligned}$$

For any $\pi_{\mathsf{si}} \leftarrow \mathsf{SiProof}(\mathsf{sk},m,\sigma,\mathsf{spk})$ and $\pi_{\mathsf{sa}} \leftarrow \mathsf{SaProof}(\mathsf{ssk},m,\sigma,\mathsf{pk})$ we have:

$$\begin{aligned} \pi_{\mathsf{si}} &\leftarrow \mathsf{V.Proof}(\{\mathsf{pk}_v,\mathsf{spk}\},(m||\sigma_1),\sigma_2,\mathsf{pk}_v,\mathsf{sk}_v) \\ \pi_{\mathsf{sa}} &\leftarrow \mathsf{V.Proof}(\{\mathsf{pk}_v,\mathsf{spk}\},(m||\sigma_1),\sigma_2,\mathsf{spk},\mathsf{ssk}) \end{aligned}$$

Since V is correct we have:

$$\mathsf{V.Judge}(\{\mathsf{pk}_v, \mathsf{spk}\}, (m||\sigma_1), \sigma_2, \mathsf{pk}_v, \pi_{\mathsf{si}}) = 1$$
$$\mathsf{V.Judge}(\{\mathsf{pk}_v, \mathsf{spk}\}, (m||\sigma_1), \sigma_2, \mathsf{spk}, \pi_{\mathsf{sa}}) = 0$$

We deduce:

$$\mathsf{SiJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{si}}) = 1$$
$$\mathsf{SaJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{sa}}) = 1$$

3. Using $\mathsf{set} = (\mathsf{set}_d, \mathsf{set}_v)$, $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ and $\mathsf{sk} = (\mathsf{sk}_d, \mathsf{sk}_v)$, we have:

$$(\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{D.Gen}(\mathsf{set}_d)$$
$$(\mathsf{pk}_v, \mathsf{sk}_v) \leftarrow \mathsf{V.Gen}(\mathsf{set}_v)$$
$$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathsf{V.Gen}(\mathsf{set}_v)$$

For any $\sigma \leftarrow \mathsf{Sig}(m, \mathsf{sk}, \mathsf{spk}, \textsc{Adm})$ and $\sigma' \leftarrow \mathsf{San}(m, \textsc{Mod}, \sigma, \mathsf{pk}, \mathsf{ssk})$ such that $\sigma = (\sigma_1, \sigma_2, \textsc{Adm})$ and $\sigma = (\sigma'_1, \sigma'_2, \textsc{Adm}')$ we have:

$$\begin{aligned} \mathrm{M} &\leftarrow \textsc{Fix}_{\textsc{Adm}}(m) \\ \sigma_1 &\leftarrow \mathsf{D.Sig}(\mathsf{sk}_d, (\mathrm{M}||\textsc{Adm}||\mathsf{pk}||\mathsf{spk})) \\ \sigma_2 &\leftarrow \mathsf{V.Sig}((\sigma_1||m)), \{\mathsf{pk}_v, \mathsf{spk}\}, \mathsf{sk}_v) \\ \sigma'_1 &= \sigma_1 \\ \sigma'_2 &\leftarrow \mathsf{V.Sig}((\sigma_1||m'), \{\mathsf{pk}_v, \mathsf{spk}\}, \mathsf{ssk}) \\ \textsc{Adm}' &= \textsc{Adm} \end{aligned}$$

For any $\pi_{\mathsf{si}} \leftarrow \mathsf{SiProof}(\mathsf{sk}, \textsc{Mod}(m), \sigma', \mathsf{spk})$ and $\pi_{\mathsf{sa}} \leftarrow \mathsf{SaProof}(\mathsf{ssk}, \textsc{Mod}(m), \sigma', \mathsf{pk})$ we have:

$$\pi_{\mathsf{si}} \leftarrow \mathsf{V.Proof}(\{\mathsf{pk}_v, \mathsf{spk}\}, (\textsc{Mod}(m)||\sigma'_1), \sigma'_2, \mathsf{pk}_v, \mathsf{sk}_v)$$
$$\pi_{\mathsf{sa}} \leftarrow \mathsf{V.Proof}(\{\mathsf{pk}_v, \mathsf{spk}\}, (\textsc{Mod}(m)||\sigma'_1), \sigma'_2, \mathsf{spk}, \mathsf{ssk})$$

Since V is correct we have:

$$\mathsf{V.Judge}(\{\mathsf{pk}_v, \mathsf{spk}\}, (\textsc{Mod}(m)||\sigma'_1), \sigma'_2, \mathsf{pk}_v, \pi_{\mathsf{si}}) = 0$$
$$\mathsf{V.Judge}(\{\mathsf{pk}_v, \mathsf{spk}\}, (\textsc{Mod}(m)||\sigma'_1), \sigma'_2, \mathsf{spk}, \pi_{\mathsf{sa}}) = 1$$

We deduce:

$$\mathsf{SiJudge}(\textsc{Mod}(m), \sigma', \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{si}}) = 0$$
$$\mathsf{SaJudge}(\textsc{Mod}(m), \sigma', \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{sa}}) = 0$$

These three properties implies that GUSS is correct, which concludes the proof. □

6.4.2 Immutability

In this section, we show that GUSS instanciated by an unforgeable digital signature scheme is immutable.

Lemma 39 *Let* D *be a deterministic digital signature scheme and* V *be a verifiable ring signature scheme. If* D *is EUF-CMA secure, then* GUSS *instanciated by* (D, V) *is immutable.*

Proof: Assume that there exists $\mathscr{A} \in \text{POLY}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{\mathsf{Immut}}_{\mathrm{GUSS},\mathscr{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathscr{B} \in \text{POLY}(k)$ such that $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{D},\mathscr{B}}(k)$ is non-negligible. $\mathscr{B}$ works as follows.

Algorithm $\mathscr{B}(\mathsf{pk}_d)$**:** This algorithm runs $\mathsf{init}_v \leftarrow \mathsf{V.Init}(1^k)$ and $(\mathsf{pk}_v, \mathsf{sk}_v) \leftarrow \mathsf{V.Gen}(\mathsf{init}_v)$. It sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ and runs $(\mathsf{spk}_*, m_*, \sigma_*) \leftarrow \mathscr{A}(\mathsf{pk})$. During the experiment, $\mathscr{B}$ simulates the two oracles $\mathsf{Sig}(.,\mathsf{sk},.,.)$ and $\mathsf{SiProof}(\mathsf{sk},.,.,.)$ to $\mathscr{A}$ as follows:

$\mathsf{Sig}(.,\mathsf{sk},.,.)$**:** On the i^{th} input $(m_i, \mathrm{ADM}_i, \mathsf{spk}_i)$, $\mathscr{B}$ computes the fixed part $\mathrm{M} \leftarrow \mathrm{FIX}_{\mathrm{ADM}_i}(m_i)$ and sends $(\mathrm{M}_i||\mathrm{ADM}_i||\mathsf{pk}||\mathsf{spk}_i)$ to the oracle $\mathsf{D.Sig}(\mathsf{sk}_d,\cdot)$. It receives the signature $\sigma_{i,1}$, then it runs $\sigma_{i,2} \leftarrow \mathsf{V.Sig}((\sigma_{i,1}||m_i)), \{\mathsf{pk}_v, \mathsf{spk}_i\}, \mathsf{sk}_v)$ and returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \mathrm{ADM}_i)$.

$\mathsf{SiProof}(\mathsf{sk},.,.,.)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{spk}'_i)$, it parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \mathrm{ADM}'_i)$. Then it runs $\pi'_{\mathsf{si},i} \leftarrow \mathsf{V.Proof}(\{\mathsf{pk}_v, \mathsf{spk}'_i\}, (m'_i||\sigma'_{i,1}), \sigma'_{i,2}, \mathsf{pk}_v, \mathsf{sk}_v)$ and returns it.

Finally, $\mathscr{B}$ parses $\sigma_* = (\sigma_{1,*}, \sigma_{2,*}, \mathrm{ADM}_*)$, computes $\mathrm{M}_* \leftarrow \mathrm{FIX}_{\mathrm{ADM}_*}(m_*)$ and returns the couple $((\mathrm{M}_*||\mathrm{ADM}_*||\mathsf{pk}||\mathsf{spk}_*), \sigma_{1,*})$.

Analysis: We show that if $\mathscr{A}$ wins its Immut experiment, then $\mathscr{B}$ wins its $\mathsf{EUF\text{-}CMA}$ experiment. Assume that $\mathscr{A}$ wins its experiment, then the following equations hold:

$$\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*) = 1 \tag{6.1}$$

$$\forall\, i \in [\![1, q_{\mathsf{Sig}}]\!], (\mathsf{spk}_* \neq \mathsf{spk}_i) \text{ or } (\mathrm{FIX}_{\mathrm{ADM}_*}(m_*) \neq \mathrm{FIX}_{\mathrm{ADM}_i}(m_i)) \tag{6.2}$$

(6.1) implies the following equation:

$$\mathsf{D.Ver}(\mathsf{pk}_d, (\mathrm{M}_*||\mathrm{ADM}_*||\mathsf{pk}||\mathsf{spk}_*), \sigma_{1,*}) = 1$$

Moreover, (6.2) implies that:

$$\forall\, i \in [\![1, q_{\mathsf{Sig}}]\!], (\mathrm{M}_*||\mathrm{ADM}_*||\mathsf{pk}||\mathsf{spk}_*) \neq (\mathrm{M}_i||\mathrm{ADM}_i||\mathsf{pk}||\mathsf{spk}_i)$$

We deduce that $\mathscr{B}$ never sends the message $(\mathrm{M}_*||\mathrm{ADM}_*||\mathsf{pk}||\mathsf{spk}_*)$ to the oracle $\mathsf{Sig}(.,\mathsf{sk},.,.)$. We remark that the experiment is perfectly simulated for $\mathscr{A}$. Finally if $\mathscr{A}$ wins its Immut experiment, then $\mathscr{B}$ wins its $\mathsf{EUF\text{-}CMA}$ experiment. We deduce:

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{D},\mathscr{B}}(k) \geq \mathsf{Adv}^{\mathsf{Immut}}_{\mathrm{GUSS},\mathscr{A}}(k) = \lambda(k)$$

which concludes the proof. □

6.4.3 Transparency

In this section, we show that GUSS instanciated by a $\mathsf{2\text{-}ano}$ secure verifiable ring signature scheme is transparent.

Lemma 40 *Let* D *be a deterministic digital signature scheme and* V *be a verifiable ring signature scheme. If* V *is* 2*-ano secure, then* GUSS *instanciated by* (D, V) *is transparent.*

Proof: Assume that there exists $\mathscr{A} \in \text{POLY}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{\mathsf{Trans}}_{\mathrm{GUSS},\mathscr{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathscr{B} \in \text{POLY}(k)^2$ such that $\mathsf{Adv}^{\mathsf{2\text{-}ano}}_{\mathrm{V},\mathscr{B}}(k)$ is non-negligible. $\mathscr{B} = (\mathscr{B}_1, \mathscr{B}_2)$ works as follows.

Algorithm $\mathscr{B}_1(\{\mathsf{spk}, \mathsf{pk}_v\})$**:** It returns $(1, 2, \mathsf{st})$.

Algorithm $\mathscr{B}_2(\mathsf{st}, \{\mathsf{spk}, \mathsf{pk}_v\})$**:** It runs $(\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{D.Gen}(\mathsf{D.Init}(1^k))$ and sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$. It runs $b' \leftarrow \mathscr{A}(\mathsf{pk}, \mathsf{spk})$ and returns b'. During the experiment, $\mathscr{B}_2$ simulates the oracles to $\mathscr{A}$ as follows:

$\mathsf{Sig}(.,\mathsf{sk},.,.)$**:** On the i^{th} input $(m_i, \textsc{Adm}_i, \mathsf{spk}_i)$, $\mathscr{B}_2$ first computes the fixed message part $\mathrm{M}_i \leftarrow \textsc{Fix}_{\textsc{Adm}_i}(m_i)$ and runs $\sigma_{i,1} \leftarrow \mathsf{D.Sig}(\mathsf{sk}_d, (\mathrm{M}_i||\textsc{Adm}_i||\mathsf{pk}||\mathsf{spk}_i))$. Then it sends the tuple $(\{\mathsf{pk}_v, \mathsf{spk}_i\}, 1, (m_i||\sigma_{i,1}))$ to the oracle $\mathsf{V.Sig}(.,.,.)$ which returns the signature $\sigma_{i,2}$. $\mathscr{B}_2$ returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \textsc{Adm}_i)$ to $\mathscr{A}$.

$\mathsf{San}(.,.,.,.,\mathsf{ssk})$**:** On the i^{th} input $(m'_i, \textsc{Mod}'_i, \sigma'_i, \mathsf{pk}'_i)$, it parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \textsc{Adm}'_i)$ and $\mathsf{pk}'_i = (\mathsf{pk}'_{d,i}, \mathsf{pk}'_{v,i})$. This algorithm first computes the modified message $\bar{m}'_i \leftarrow \textsc{Mod}'_i(m'_i)$ and it sends $(\{\mathsf{pk}'_{v,i}, \mathsf{spk}\}, 2, (\bar{m}'_i||\sigma'_{i,1}))$ to the oracle $\mathsf{V.Sig}(.,.,.)$ which returns the signature $\bar{\sigma}'_{i,2}$. $\mathscr{B}_2$ returns $\bar{\sigma}'_i = (\sigma'_{i,1}, \bar{\sigma}'_{i,2}, \textsc{Adm}'_i)$ to $\mathscr{A}$.

$\mathsf{SiProof}(\mathsf{sk},.,.,.)$**:** On the i^{th} input $(m''_i, \sigma''_i, \mathsf{spk}''_i)$, $\mathscr{B}_2$ parses $\sigma''_i = (\sigma''_{i,1}, \sigma''_{i,2}, \textsc{Adm}''_i)$. It sends $(\{\mathsf{pk}_v, \mathsf{spk}''_i\}, (m''_i||\sigma''_{i,1}), \sigma''_{i,2}, \mathsf{pk}_v, 1)$ to the oracle $\mathsf{V.Proof}(.,.,.,.,.)$ which returns the proof $\pi''_{\mathsf{si},i}$. Finally, $\mathscr{B}_2$ returns $\pi''_{\mathsf{si},i}$.

$\mathsf{SaProof}(\mathsf{ssk},.,.,.)$**:** On the i^{th} input $(m'''_i, \sigma'''_i, \mathsf{pk}'''_i)$, $\mathscr{B}$ parses $\sigma'''_i = (\sigma'''_{i,1}, \sigma'''_{i,2}, \textsc{Adm}'''_i)$ and $\mathsf{pk}'''_i = (\mathsf{pk}'''_{d,i}, \mathsf{pk}'''_{v,i})$. It sends $(\{\mathsf{pk}'''_{v,i}, \mathsf{spk}\}, (m'''_i||\sigma'''_{i,1}), \sigma'''_{i,2}, \mathsf{spk}, 2)$ to the oracle $\mathsf{V.Proof}(.,.,.,.,.)$ which returns the proof $\pi'''_{\mathsf{sa},i}$. Finally, $\mathscr{B}_2$ returns $\pi'''_{\mathsf{sa},i}$.

$\mathsf{Sa/Si}(b, \mathsf{pk}, \mathsf{spk}, \mathsf{sk}, \mathsf{ssk}, .,.,.)$**:** On the i^{th} input $(m''''_i, \textsc{Adm}''''_i, \textsc{Mod}''''_i)$, if $\textsc{Adm}''''_i(\textsc{Mod}''''_i) = 0$, then $\mathscr{B}_2$ returns $\perp$. Else, $\mathscr{B}_2$ computes $\mathrm{M}''''_i \leftarrow \textsc{Fix}_{\textsc{Adm}''''_i}(m''''_i)$. Then $\mathscr{B}_2$ runs $\sigma''''_{i,1} \leftarrow \mathsf{D.Sig}(\mathsf{sk}_d, (\mathrm{M}''''_i||\textsc{Adm}''''_i||\mathsf{pk}||\mathsf{spk}))$ and it sends $((\textsc{Mod}''''(m''''_i))||\sigma''''_{i,1}), \{\mathsf{pk}_v, \mathsf{skp}\})$ to the oracle $\mathsf{LRSO_b}(1,2,\cdot,\cdot)$ which returns the signature $\sigma''''_{i,2}$. Finally, $\mathscr{B}_2$ returns the signature $\sigma''''_i = (\sigma''''_{i,1}, \sigma''''_{i,2}, \textsc{Adm}''''_i)$ to $\mathscr{A}$.

Analysis: Assume that $\mathscr{A}$ wins its experiment, then $b = b'$ and:

$$\mathsf{S}_{\mathsf{Sa/Si}} \cap (\mathsf{S}_{\mathsf{SiProof}} \cup \mathsf{S}_{\mathsf{SaProof}}) = \emptyset$$

where $\mathsf{S}_{\mathsf{Sa/Si}}$ (resp. $\mathsf{S}_{\mathsf{SiProof}}$ and $\mathsf{S}_{\mathsf{SaProof}}$) corresponds to the set of all signatures outputted by the oracle $\mathsf{Sa/Si}$ (resp. sending to the oracles $\mathsf{SiProof}$ and $\mathsf{SaProof}$). It implies that the messages sent to the oracle $\mathsf{V.Proof}(.,.,.,.,.)$ was not already signed by $\mathsf{LRSO_b}(1,2,\cdot,\cdot)$. More formally, we have:

$$\forall\ i, j \in [\![1, \max(q_{\mathrm{S}}, q_{\mathrm{P}})]\!], (\sigma_i \neq \sigma'_j)$$

where q_{S} (resp. q_{P}) is the number of calls to the oracle $\mathsf{V.Sig}(\cdot,\cdot,\cdot)$ (resp. $\mathsf{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$). Finally, the probability that $\mathscr{B}$ wins its experiment is the same as the probability that $\mathscr{A}$ wins its experiments:

$$\mathsf{Adv}^{\mathsf{2\text{-}ano}}_{\mathrm{V},\mathscr{B}}(k) \geq \mathsf{Adv}^{\mathsf{Trans}}_{\mathrm{GUSS},\mathscr{A}}(k) = \lambda(k)$$

which conclude the proof. □

6.4.4 Unlinkability

We show that GUSS instanciated by an unforgeable digital signature is unlinkable.

Lemma 41 *Let* D *be a deterministic digital signature scheme and* V *be a verifiable ring signature scheme. If* D *is EUF-CMA secure, then* GUSS *instanciated by* (D, V) *is unlinkable.*

Proof: Assume that there exists $\mathscr{A} \in \textsc{poly}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathscr{B} \in \textsc{poly}(k)$ such that $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{D},\mathscr{B}}(k)$ is non-negligible. $\mathscr{B}$ works as follows.

Algorithm $\mathscr{B}(\mathsf{pk}_d)$**:** It runs $(\mathsf{pk}_v, \mathsf{sk}_v) \leftarrow \mathsf{D.Gen}(\mathsf{V.Init}(k))$ and $(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathsf{D.Gen}(\mathsf{V.Init}(k))$, then it sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$. It chooses $b \xleftarrow{\$} \{0,1\}$ and runs $b' \leftarrow \mathscr{A}(\mathsf{pk}, \mathsf{spk})$. During the experiment, $\mathscr{B}$ simulates the oracles to $\mathscr{A}$ as follows:

$\mathsf{Sig}(.,\mathsf{sk},.,.)$**:** On the i^{th} input $(m_i, \textsc{Adm}_i, \mathsf{spk}_i)$, it computes $\mathrm{M} \leftarrow \textsc{Fix}_{\textsc{Adm}_i}(m_i)$ and sends the tuple $(\mathrm{M}_i||\textsc{Adm}_i||\mathsf{pk}||\mathsf{spk}_i)$ to the oracle $\mathrm{D}.\mathsf{Sig}(\mathsf{sk}_d, \cdot)$ and receives the signature $\sigma_{i,1}$. Then it runs $\sigma_2 \leftarrow \mathrm{V}.\mathsf{Sig}((\sigma_{i,1}||m_i)), \{\mathsf{pk}_v, \mathsf{spk}_i\}, \mathsf{sk}_v)$ and returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \textsc{Adm}_i)$ to $\mathscr{A}$.

$\mathsf{SiProof}(\mathsf{sk},.,.,.)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{spk}'_i)$, it parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \textsc{Adm}'_i)$. Then it runs $\pi'_{\mathsf{si},i} \leftarrow \mathrm{V}.\mathsf{Proof}(\{\mathsf{pk}_v, \mathsf{spk}'_i\}, (m'_i||\sigma'_{i,_1}), \sigma'_{i,2}, \mathsf{pk}_v, \mathsf{sk}_v)$ and returns π' to $\mathscr{A}$..

$\mathsf{San}(.,.,.,.,\mathsf{ssk})$**:** On the i^{th} input $(m''_i, \textsc{Mod}''_i, \sigma''_i, \mathsf{pk}''_i)$, the adversary $\mathscr{B}$ runs the algorithm $\bar{\sigma}''_i \leftarrow \mathsf{San}(m''_i, \textsc{Mod}''_i, \sigma''_i, \mathsf{pk}''_i, \mathsf{ssk})$ and returns $\bar{\sigma}''_i$ to $\mathscr{A}$.

$\mathsf{SaProof}(\mathsf{ssk},.,.,.)$**:** On the i^{th} input $(m'''_i, \sigma'''_i, \mathsf{pk}'''_i)$, the adversary $\mathscr{B}$ runs the algorithm $\pi'''_{\mathsf{sa},i} \leftarrow \mathsf{SaProof}(\mathsf{ssk}, m'''_i, \sigma'''_i, \mathsf{pk}'''_i)$ and returns π_{sa} to $\mathscr{A}$.

$\mathsf{LRSan}(b,\mathsf{pk},\mathsf{ssk},.,.)$**:** On the i^{th} input $((m''''_{0,i}, \textsc{Mod}''''_{0,i}, \sigma''''_{0,i})(m''''_{1,i}, \textsc{Mod}''''_{1,i}, \sigma''''_{1,i}))$, if for all j in $\{0,1\}$, $\mathsf{Ver}(m''''_{j,i}, \sigma''''_{j,i}, \mathsf{pk}, \mathsf{spk}) = 1$ and $\textsc{Adm}''''_{0,i} = \textsc{Adm}''''_{1,i}$ and $\textsc{Mod}''''_{0,i}(m''''_{0,i}) = \textsc{Mod}''''_{1,i}(m''''_{1,i})$ and $\textsc{Adm}''''_{j,i}(\textsc{Mod}''''_{j,i}) = 1$, then it runs $\widehat{\sigma}_i \leftarrow \mathsf{San}(m''''_{b,i}, \textsc{Mod}''''_{b,i}, \sigma''''_{b,i}, \mathsf{pk}, \mathsf{ssk})$ and returns $\widehat{\sigma}_i$ to $\mathscr{A}$ where $\widehat{\sigma}_i = (\widehat{\sigma}_{1,b,i}, \widehat{\sigma}_{2,b,i}, \widehat{adm}_{b,i})$. Else it returns $\perp$ to $\mathscr{A}$. Moreover, if for $j \in \{0,1\}$, $\mathsf{Ver}(m''''_{j,i}, \sigma''''_{j,i}, \mathsf{pk}, \mathsf{spk}) = 1$ and $\textsc{Adm}''''_{0,i} = \textsc{Adm}''''_{1,i}$ and $\textsc{Adm}''''_{j,i}(\textsc{Mod}''''_{j,i}) = 1$ and $\textsc{Mod}''''_{0,i}(m''''_{0,i}) = \textsc{Mod}''''_{1,i}(m''''_{1,i})$, and if there exists x such that $\sigma''''_{x,i}$ was not already outputted by the oracle $\mathrm{D}.\mathsf{Sig}(\mathsf{sk}_d, \cdot)$, then $\mathscr{B}$ aborts the experiment for $\mathscr{A}$ and returns $((\textsc{Fix}_{\textsc{Adm}''''_{x,i}}(m''''_{x,i})||\textsc{Adm}''''_{x,i}||\mathsf{pk}||\mathsf{spk}), \sigma''''_{x,i})$ to the challenger.

Finally, if $\mathscr{B}$ has not already aborted the experiment, then it returns $\perp$.

Analysis: We first observe that for any integer $i \in [\![1,q]\!]$ where q is the number of queries to the oracle $\mathsf{LRSan}(b,\mathsf{pk},\mathsf{ssk},.,.)$ and for $j \in \{0,1\}$, if $\mathsf{Ver}(m''''_{j,i}, \sigma''''_{j,i}, \mathsf{pk}, \mathsf{spk}) = 1$ and $\textsc{Adm}''''_{0,i} = \textsc{Adm}''''_{1,i}$ and $\textsc{Adm}''''_{j,i}(\textsc{Mod}''''_{j,i}) = 1$ and $\textsc{Mod}''''_{0,i}(m''''_{0,i}) = \textsc{Mod}''''_{1,i}(m''''_{1,i})$, and $\sigma''''_{j,i}$ was already outputted by the oracle $\mathrm{D}.\mathsf{Sig}(\mathsf{sk}_d, \cdot)$, then we have:

$$\textsc{Fix}_{\textsc{Adm}''''_{0,i}}(m''''_{0,i})||\textsc{Adm}''''_{0,i}||\mathsf{pk}||\mathsf{spk} = \textsc{Fix}_{\textsc{Adm}''''_{1,i}}(m''''_{1,i})||\textsc{Adm}''''_{1,i}||\mathsf{pk}||\mathsf{spk}$$

Since D is deterministic, we deduce that $\widehat{\sigma}_{1,b,i} = \sigma''''_{0,i} = \sigma''''_{1,i}$. On the other hand, the second part of the outputted signature $\widehat{\sigma}_{2,b,i}$ does not depend on b. Finally, $\widehat{\textsc{Adm}}_{b,i} = \textsc{Adm}''''_{0,i} = \textsc{Adm}''''_{1,i}$, then $\widehat{\textsc{Adm}}_{b,i}$ does not depend on b. We deduce that the outputted signature $\widehat{\sigma}_{b,i}$ leaks no information about b. In this case, the best strategy of $\mathscr{A}$ to wins the experiment is to randomly guess the bit b'.

On the other hand, if there exists $i \in [\![1,q]\!]$ and $j \in \{0,1\}$ such that $\mathsf{Ver}(m''''_{j,i}, \sigma''''_{j,i}, \mathsf{pk}, \mathsf{spk}) = 1$ and $\textsc{Adm}''''_{0,i} = \textsc{Adm}''''_{1,i}$ and $\textsc{Adm}''''_{j,i}(\textsc{Mod}''''_{j,i}) = 1$ and $\textsc{Mod}''''_{0,i}(m''''_{0,i}) = \textsc{Mod}''''_{1,i}(m''''_{1,i})$ and there exists x such that $\sigma''''_{x,i}$ was not already outputted by the oracle $\mathrm{D}.\mathsf{Sig}(\mathsf{sk}_d, \cdot)$, then $\mathscr{B}$ returns to the challenger the tuple $((\textsc{Fix}_{\textsc{Adm}''''_{x,i}}(m''''_{x,i})||\textsc{Adm}''''_{x,i}||\mathsf{pk}||\mathsf{spk}), \sigma''''_{x,i})$ and wins its experiment. We denote this event by E. We have:

$$\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{D},\mathscr{B}}(k)\right] \geq \Pr[\mathrm{E}]$$

On the other hand, we have:

$$\begin{aligned}
\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)\right] &= \Pr[\mathrm{E}] \cdot \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)|\mathrm{E}\right] + (1 - \Pr[\mathrm{E}]) \cdot \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)|\neg\mathrm{E}\right] \\
&= \Pr[\mathrm{E}] \cdot \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)|\mathrm{E}\right] + \frac{1}{2} - \frac{1}{2} \cdot \Pr[\mathrm{E}] \\
&= \Pr[\mathrm{E}] \cdot \left(\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)|\mathrm{E}\right] - \frac{1}{2}\right) + \frac{1}{2}
\end{aligned}$$

We deduce:

$$\left(\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)\right] - \frac{1}{2}\right) = \Pr[\mathrm{E}] \cdot \left(\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)|\mathrm{E}\right] - \frac{1}{2}\right)$$

It implies that:

$$\begin{aligned}
\mathsf{Pr}[\mathrm{E}] &= \frac{\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)\right] - \frac{1}{2}}{\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)|\mathrm{E}\right] - \frac{1}{2}} \\
&= \frac{\left|\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)\right] - \frac{1}{2}\right|}{\left|\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)|\mathrm{E}\right] - \frac{1}{2}\right|} \\
&= \frac{\lambda(k)}{\left|\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Unlink}}_{\mathrm{GUSS},\mathscr{A}}(k)|\mathrm{E}\right] - \frac{1}{2}\right|} \\
&\geq \lambda(k)
\end{aligned}$$

Finally, we deduce that:

$$\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{D},\mathscr{B}}(k)\right] \geq \lambda(k)$$

which concludes the proof. □

6.4.5 Accountability

We show that GUSS instanciated by an accountable and non-seizable verifiable ring signature scheme is accountable.

Lemma 42 *Let* D *be a deterministic digital signature scheme and* V *be a verifiable ring signature scheme. If* V *is* 1*-acc secure, then* GUSS *instanciated by* (D, V) *is SiAcc-1 secure.*

Proof: Assume that there exists $\mathscr{A} \in \textsc{poly}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{\mathsf{SiAcc\text{-}1}}_{\mathrm{GUSS},\mathscr{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathscr{B} \in \textsc{poly}(k)$ such that $\mathsf{Adv}^{\mathsf{1\text{-}acc}}_{\mathrm{V},\mathscr{B}}(k)$ is non-negligible.

Algorithm $\mathscr{B}(\mathsf{spk})$: It runs $(\mathsf{pk}_*, m_*, \sigma_*, \pi_{si,*}) \leftarrow \mathscr{A}(\mathsf{spk})$. During the experiment, $\mathscr{B}$ simulates the oracles to $\mathscr{A}$ as follows:

$\mathsf{San}(.,.,.,.,\mathsf{ssk})$**:** On the i^{th} input $(m_i, \textsc{Mod}_i, \sigma_i, \mathsf{pk}_i)$, it parses $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \textsc{Adm}_i)$ and $\mathsf{pk}_i = (\mathsf{pk}_{d,i}, \mathsf{pk}_{v,i})$. Then it computes $\bar{m}_i \leftarrow \textsc{Mod}_i(m_i)$ and it sends $(\{\mathsf{pk}_{v,i}, \mathsf{spk}\}, 1, (\bar{m}_i||\sigma_{i,1}))$ to the oracle $\mathrm{V}.\mathsf{Sig}(.,.,.)$ that returns the signature $\bar{\sigma}_{i,2}$. $\mathscr{B}_2$ returns $\bar{\sigma}_i = (\sigma_{i,1}, \bar{\sigma}_{i,2}, \textsc{Adm}_i)$ to $\mathscr{A}$.

$\mathsf{SaProof}(\mathsf{ssk},.,.,.)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{pk}'_i)$, $\mathscr{B}$ parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \textsc{Adm}'_i)$ and $\mathsf{pk}'_i = (\mathsf{pk}'_{d,i}, \mathsf{pk}'_{v,i})$. It sends $(\{\mathsf{pk}'_{v,i}, \mathsf{spk}\}, (m'_i||\sigma'_{i,1}), \sigma'_{i,2}, \mathsf{spk}, 1)$ to the oracle $\mathrm{V}.\mathsf{Proof}(.,.,.,.,.)$ that returns the proof $\pi'_{\mathsf{sa},i}$. Finally, $\mathscr{B}$ returns $\pi'_{\mathsf{sa},i}$ to $\mathscr{A}$.

Finally, $\mathscr{B}$ parses $\mathsf{pk}_* = (\mathsf{pk}_{d,*}, \mathsf{pk}_{v,*})$ and $\sigma_* = (\sigma_{1,*}, \sigma_{2,*}, \textsc{Adm}_*)$, then it returns the tuple $(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, m_*||\sigma_{1,*}, \sigma_{2,*}, \mathsf{pk}_{v,*}, \pi_{si,*})$.

Analysis: First note that the experiment is perfectly simulated for $\mathscr{A}$. Assume that $\mathscr{A}$ wins its experiment, then:

$$\forall\ i \in [\![1, q_{\mathsf{San}}]\!], (\sigma_* \neq \bar{\sigma}_i) \tag{6.3}$$

$$\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}) = 1 \tag{6.4}$$

$$\mathsf{SiJudge}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}, \pi_{\mathsf{si},*}) = 0 \tag{6.5}$$

where q_{San} is the number of calls to the oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$. First note that:

$$\{\mathsf{spk}, \mathsf{pk}_{v,*}\} \subset \{\mathsf{spk}\} \cup \{\mathsf{pk}_{v,*}\}$$

(6.3) implies that:

$$\forall\ i \in [\![1, q_{\mathsf{S}}]\!], \sigma_{*,2} \neq \bar{\sigma}_{i,2}$$

where q_{S} is the number of queries to $\mathsf{V.Sig}(.,.,.)$. Indeed, if $(\sigma_* \neq \sigma'_i)$ then $\sigma_{1,*} \neq \sigma_{1,i}$ or $\sigma_{2,*} \neq \sigma_{2,i}$ or $\textsc{Adm}_* \neq \textsc{Adm}_i$: if $\textsc{Adm}_* \neq \textsc{Adm}_i$ then $\sigma_{1,*} \neq \sigma_{1,i}$ because $\sigma_{1,*}$ (resp. $\sigma_{1,i}$) is a signature of $\textsc{Adm}_*$ (resp. $\textsc{Adm}_i$). If $\sigma_{1,*} \neq \sigma_{1,i}$ then $\sigma_{2,*} \neq \sigma_{2,i}$ because $\sigma_{2,*}$ (resp. $\sigma_{2,i}$) is a signature of $\sigma_{1,*}$ (resp. $\sigma_{1,i}$). Finally, in all cases $\sigma_{*,2} \neq \bar{\sigma}_{i,2}$.

On the other hand, (6.4) implies that:

$$\mathsf{V.Ver}(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, \sigma_{2,*}, m_*||\sigma_{1,*}) = 1$$

Finally, (6.5) implies that:

$$\mathsf{V.Judge}(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, m_*||\sigma_{1,*}, \sigma_{2,*}, \mathsf{pk}_{v,*}, \pi_{si,*}) = 0$$

We deduce that the probability that $\mathcal{B}$ wins its experiment is the same as the probability that $\mathcal{A}$ wins its experiments:

$$\mathsf{Adv}^{\mathsf{1\text{-}acc}}_{\mathsf{V},\mathcal{B}}(k) \geq \mathsf{Adv}^{\mathsf{SiAcc\text{-}1}}_{\mathsf{GUSS},\mathcal{A}}(k) = \lambda(k)$$

which concludes the proof. □

Lemma 43 *Let* D *be a deterministic digital signature scheme and* V *be a verifiable ring signature scheme. If* V *is* 1*-non-sei-2 secure, then* GUSS *instanciated by* (D, V) *is SaAcc-1 secure.*

Proof: Assume that there exists $\mathcal{A} \in \textsc{poly}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{\mathsf{SaAcc\text{-}1}}_{\mathsf{GUSS},\mathcal{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \textsc{poly}(k)$ such that $\mathsf{Adv}^{\mathsf{1\text{-}non\text{-}sei\text{-}2}}_{\mathsf{V},\mathcal{B}}(k)$ is non-negligible.

Algorithm $\mathcal{B}(\mathsf{pk}_v)$: It generates $(\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{D.Gen}(\mathsf{init}_d)$ and sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$. Then it runs $(\mathsf{spk}_*, m_*, \sigma_*) \leftarrow \mathcal{A}(\mathsf{pk})$. During the experiment, $\mathcal{B}$ simulates the oracles to $\mathcal{A}$ as follows:

$\mathsf{Sig}(.,\mathsf{sk},.,.)$**:** On the i^{th} input $(m_i, \textsc{Adm}_i, \mathsf{spk}_i)$, $\mathcal{B}$ first computes $\mathrm{M}_i \leftarrow \textsc{Fix}_{\textsc{Adm}_i}(m_i)$ and runs $\sigma_{i,1} \leftarrow \mathsf{D.Sig}(\mathsf{sk}_d, (\mathrm{M}_i||\textsc{Adm}_i||\mathsf{pk}||\mathsf{spk}_i))$. Then it sends $(\{\mathsf{pk}_v, \mathsf{spk}_i\}, 1, (m_i||\sigma_{i,1}))$ to the oracle $\mathsf{V.Sig}(.,.,.)$ that returns the signature $\sigma_{i,2}$. $\mathcal{B}$ returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \textsc{Adm}_i)$ to $\mathcal{A}$.

$\mathsf{SiProof}(\mathsf{sk},.,.,.)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{spk}'_i)$, $\mathcal{B}$ parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \textsc{Adm}'_i)$. It sends $(\{\mathsf{pk}_v, \mathsf{spk}'_i\}, (m'_i||\sigma'_{i,1}), \sigma'_{i,2}, \mathsf{pk}_v, 1)$ to the oracle $\mathsf{V.Proof}(.,.,.,.,.)$ that returns the proof $\pi'_{\mathsf{si},i}$. Finally, $\mathcal{B}$ returns $\pi'_{\mathsf{si},i}$.

Finally, $\mathcal{B}$ parses $\sigma_* = (\sigma_{1,*}, \sigma_{2,*}, \textsc{Adm}_*)$ and returns $(\{\mathsf{spk}_*, \mathsf{pk}_v\}, (m_*||\sigma_{1,*}), \sigma_{2,*})$.

Analysis: First note that the experiment is perfectly simulated for $\mathcal{A}$. Assume that $\mathcal{A}$ wins its experiment, then, for any $\pi_{\mathsf{si},*} \leftarrow \mathsf{SiProof}(\mathsf{sk}, m_*, \sigma_*, \mathsf{spk}_*)$:

$$\forall\ i \in [\![1, q_{\mathsf{Sig}}]\!], (\sigma_* \neq \sigma_i) \tag{6.6}$$

$$\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*) = 1 \tag{6.7}$$

$$\mathsf{SiJudge}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*, \pi_{\mathsf{si},*}) \neq 0 \tag{6.8}$$

where q_{San} is the number of calls to the oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$. First note that (6.6) implies that:

$$\forall\ i \in [\![1, q_{\mathsf{S}}]\!], \sigma_{*,2} \neq \sigma_{i,2}$$

where q_{S} is the number of queries to $\mathsf{V.Sig}(.,.,.)$. Indeed, if $(\sigma_* \neq \sigma_i)$, then $\sigma_{1,*} \neq \sigma_{1,i}$ or $\sigma_{2,*} \neq \sigma_{2,i}$ or $\textsc{Adm}_* \neq \textsc{Adm}_i$: if $\textsc{Adm}_* \neq \textsc{Adm}_i$, then $\sigma_{1,*} \neq \sigma_{1,i}$ because $\sigma_{1,*}$ (resp. $\sigma_{1,i}$) is a signature of $\textsc{Adm}_*$ (resp. $\textsc{Adm}_i$). If $\sigma_{1,*} \neq \sigma_{1,i}$, then $\sigma_{2,*} \neq \sigma_{2,i}$ because $\sigma_{2,*}$ (resp. $\sigma_{2,i}$) is a signature of $\sigma_{1,*}$ (resp. $\sigma_{1,i}$). Finally, in all cases $\sigma_{*,2} \neq \sigma_{i,2}$.

On the other hand, (6.7) implies that:

$$\mathsf{V.Ver}(\{\mathsf{spk}_*, \mathsf{pk}_v\}, \sigma_{2,*}, m_* || \sigma_{1,*}) = 1$$

Moreover, (6.8) implies that:

$$\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*, \pi_{\mathsf{si},*}) = 1$$

Indeed, $\pi_{\mathsf{si},*}$ cannot be equal to $\perp$ since it is computed by the proof algorithm from a valid signature. It implies that:

$$\mathsf{V.Judge}(\{\mathsf{spk}_*, \mathsf{pk}_v\}, m_* || \sigma_{1,*}, \sigma_{2,*}, \mathsf{pk}_v, \pi_{si,*}) = 0$$

Finally, note that since $\pi_{\mathsf{si},*} \leftarrow \mathsf{SiProof}(\mathsf{sk}, m_*, \sigma_*, \mathsf{spk}_*)$, then:

$$\pi_{\mathsf{si},*} \leftarrow \mathsf{V.Proof}(\{\mathsf{spk}_*, \mathsf{pk}_v\}, m_* || \sigma_{1,*}, \sigma_{2,*}, \mathsf{pk}_v, \mathsf{sk}_v)$$

We deduce that the probability that $\mathscr{B}$ wins its experiment is the same as the probability that $\mathscr{A}$ wins its experiments:

$$\mathsf{Adv}^{\mathsf{1\text{-}non\text{-}sei\text{-}2}}_{\mathrm{V},\mathscr{B}}(k) \geq \mathsf{Adv}^{\mathsf{SaAcc\text{-}1}}_{\mathrm{GUSS},\mathscr{A}}(k) = \lambda(k)$$

which concludes the proof. □

6.4.6 Strong Accountability

In this section, we show that GUSS instanciated by an accountable and non-seizable verifiable ring signature scheme is strong accountable.

Lemma 44 *Let* D *be a deterministic digital signature scheme and* V *be a verifiable ring signature scheme. If* V *is* 1*-acc secure, then* GUSS *instanciated by* (D, V) *is SaAcc-2 secure.*

Proof: Assume that there exists $\mathscr{A} \in \text{POLY}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{\mathsf{SaAcc\text{-}2}}_{\mathrm{GUSS},\mathscr{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathscr{B} \in \text{POLY}(k)$ such that $\mathsf{Adv}^{\mathsf{1\text{-}acc}}_{\mathrm{V},\mathscr{B}}(k)$ is non-negligible.

Algorithm $\mathscr{B}(\mathsf{pk}_v)$: It generates $(\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{D.Gen}(\mathsf{init}_d)$ and sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$. Then it runs $(\mathsf{spk}_*, m_*, \sigma_*, \pi_{sa,*}) \leftarrow \mathscr{A}(\mathsf{pk})$. During the experiment, $\mathscr{B}$ simulates the oracles to $\mathscr{A}$ as follows:

$\mathsf{Sig}(., \mathsf{sk}, ., .)$**:** On the i^{th} input $(m_i, \text{ADM}_i, \mathsf{spk}_i)$, $\mathscr{B}$ first computes the fixed message part $\mathrm{M}_i \leftarrow \text{FIX}_{\text{ADM}_i}(m_i)$ and runs $\sigma_{i,1} \leftarrow \mathsf{D.Sig}(\mathsf{sk}_d, (\mathrm{M}_i || \text{ADM}_i || \mathsf{pk} || \mathsf{spk}_i))$. Then it sends the tuple $(\{\mathsf{pk}_v, \mathsf{spk}_i\}, 1, (m_i || \sigma_{i,1}))$ to the oracle $\mathsf{V.Sig}(., ., .)$ which returns the signature $\sigma_{i,2}$. It returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \text{ADM}_i)$ to $\mathscr{A}$.

$\mathsf{SiProof}(\mathsf{sk}, ., ., .)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{spk}'_i)$, $\mathscr{B}$ parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \text{ADM}'_i)$. Then it sends $(\{\mathsf{pk}'_{v,i}, \mathsf{spk}\}, (m'_i || \sigma'_{i,1}), \sigma'_{i,2}, \mathsf{spk}, 1)$ to the oracle $\mathsf{V.Proof}(., ., ., ., .)$ that returns the proof $\pi'_{\mathsf{si},i}$. Finally, $\mathscr{B}$ returns $\pi'_{\mathsf{si},i}$ to $\mathscr{A}$.

Finally, $\mathscr{B}$ parses $\sigma_* = (\sigma_{1,*}, \sigma_{2,*}, \text{ADM}_*)$ and returns $(\{\mathsf{spk}_*, \mathsf{pk}_v\}, m_* || \sigma_{1,*}, \sigma_{2,*}, \mathsf{spk}_*, \pi_{sa,*})$.

Analysis: We first note that the experiment is perfectly simulated for $\mathscr{A}$. Assume that $\mathscr{A}$ wins its experiment, then:

$$\forall\ i \in [\![1, q_{\mathsf{Sig}}]\!], (\sigma_* \neq \sigma_i) \tag{6.9}$$

$$\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*) = 1 \tag{6.10}$$

$$\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*, \pi_{\mathsf{sa},*}) = 1 \tag{6.11}$$

where q_{Sig} is the number of calls to the oracle $\mathsf{Sig}(.,\mathsf{sk},.,.)$. We note that:

$$\{\mathsf{spk}_*,\mathsf{pk}_v\} \subset \{\mathsf{spk}_*\} \cup \{\mathsf{pk}_v\}$$

(6.9) implies that:

$$\forall\ i \in [\![1, q_{\mathsf{S}}]\!], \sigma_{*,2} \neq \sigma_{i,2}$$

where q_{S} is the number of queries to $\mathsf{V.Sig}(.,.,.)$. Indeed, if $(\sigma_* \neq \sigma_i)$ then $\sigma_{1,*} \neq \sigma_{1,i}$ or $\sigma_{2,*} \neq \sigma_{2,i}$ or $\textsc{Adm}_* \neq \textsc{Adm}_i$: if $\textsc{Adm}_* \neq \textsc{Adm}_i$ then $\sigma_{1,*} \neq \sigma_{1,i}$ because $\sigma_{1,*}$ (resp. $\sigma_{1,i}$) is a signature of $\textsc{Adm}_*$ (resp. $\textsc{Adm}_i$). If $\sigma_{1,*} \neq \sigma_{1,i}$ then $\sigma_{2,*} \neq \sigma_{2,i}$ because $\sigma_{2,*}$ (resp. $\sigma_{2,i}$) is a signature of $\sigma_{1,*}$ (resp. $\sigma_{1,i}$). Finally, in all cases $\sigma_{*,2} \neq \sigma_{i,2}$.

On the other hand, (6.10) implies that:

$$\mathsf{V.Ver}(\{\mathsf{spk}_*,\mathsf{pk}_v\}, \sigma_{2,*}, m_*||\sigma_{1,*}) = 1$$

Finally, (6.11) implies that:

$$\mathsf{V.Judge}(\{\mathsf{spk}_*,\mathsf{pk}_v\}, m_*||\sigma_{1,*}, \sigma_{2,*}, \mathsf{spk}_*, \pi_{sa,*}) = 0$$

We deduce that the probability that $\mathscr{B}$ wins its experiment is the same as the probability that $\mathscr{A}$ wins its experiments:

$$\mathsf{Adv}^{\mathsf{1\text{-}acc}}_{\mathrm{V},\mathscr{B}}(k) \geq \mathsf{Adv}^{\mathsf{SaAcc\text{-}2}}_{\mathrm{GUSS},\mathscr{A}}(k) = \lambda(k)$$

which concludes the proof. □

Lemma 45 *Let* D *be a deterministic digital signature scheme and* V *be a verifiable ring signature scheme. If* V *is* 1*-non-sei-2 secure, then* GUSS *instanciated by* (D, V) *is SiAcc-2 secure.*

Proof: Assume that there exists $\mathscr{A} \in \textsc{poly}(k)$ such that the advantage $\lambda(k) = \mathsf{Adv}^{\mathsf{SiAcc\text{-}2}}_{\mathrm{GUSS},\mathscr{A}}(k)$ is non-negligible. We show how to build an algorithm $\mathscr{B} \in \textsc{poly}(k)$ such that $\mathsf{Adv}^{\mathsf{1\text{-}non\text{-}sei\text{-}2}}_{\mathrm{V},\mathscr{B}}(k)$ is non-negligible.

Algorithm $\mathscr{B}(\mathsf{spk})$: It runs $(\mathsf{pk}_*, m_*, \sigma_*) \leftarrow \mathscr{A}(\mathsf{spk})$. During the experiment, $\mathscr{B}$ simulates the oracles to $\mathscr{A}$ as follows:

$\mathsf{San}(.,.,.,.,\mathsf{ssk})$**:** On the i^{th} input $(m_i, \textsc{Mod}_i, \sigma_i, \mathsf{pk}_i)$, it parses $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \textsc{Adm}_i)$ and $\mathsf{pk}_i = (\mathsf{pk}_{d,i}, \mathsf{pk}_{v,i})$. Then it computes $\bar{m}_i \leftarrow \textsc{Mod}_i(m_i)$ and it sends $(\{\mathsf{pk}_{v,i}, \mathsf{spk}\}, 1, (\bar{m}_i||\sigma_{i,1}))$ to the oracle $\mathsf{V.Sig}(.,.,.)$ that returns the signature $\bar{\sigma}_{i,2}$. $\mathscr{B}_2$ returns $\bar{\sigma}_i = (\sigma_{i,1}, \bar{\sigma}_{i,2}, \textsc{Adm}_i)$ to $\mathscr{A}$.

$\mathsf{SaProof}(\mathsf{ssk},.,.,.)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{pk}'_i)$, $\mathscr{B}$ parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \textsc{Adm}'_i)$ and $\mathsf{pk}'_i = (\mathsf{pk}'_{d,i}, \mathsf{pk}'_{v,i})$. It sends $(\{\mathsf{pk}'_{v,i}, \mathsf{spk}\}, (m'_i||\sigma'_{i,1}), \sigma'_{i,2}, \mathsf{spk}, 1)$ to the oracle $\mathsf{V.Proof}(.,.,.,.,.)$ that returns the proof $\pi'_{\mathsf{sa},i}$. Finally, $\mathscr{B}$ returns $\pi'_{\mathsf{sa},i}$ to $\mathscr{A}$.

Finally, $\mathscr{B}$ parses $\mathsf{pk}_* = (\mathsf{pk}_{d,*}, \mathsf{pk}_{v,*})$ and $\sigma_* = (\sigma_{1,*}, \sigma_{2,*}, \textsc{Adm}_*)$, then it returns the tuple $(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, (m_*||\sigma_{1,*}), \sigma_{2,*})$.

Analysis: We first note that the experiment is perfectly simulated for $\mathscr{A}$. Assume that $\mathscr{A}$ wins its experiment, then, for any $\pi_{\mathsf{sa},*} \leftarrow \mathsf{SaProof}(\mathsf{ssk}, m_*, \sigma_*, \mathsf{pk}_*)$:

$$\forall\ i \in [\![1, q_{\mathsf{San}}]\!], (\sigma_* \neq \sigma_i) \tag{6.12}$$

$$\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}_{)} = 1 \tag{6.13}$$

$$\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}, \pi_{\mathsf{sa},*}) \neq 1 \tag{6.14}$$

where q_{San} is the number of calls to the oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$. First note that (6.12) implies that:

$$\forall\ i \in [\![1, q_{\mathsf{S}}]\!], \sigma_{*,2} \neq \bar{\sigma}_{i,2}$$

where q_S is the number of queries to $\mathsf{V.Sig}(.,.,.)$. Indeed, if $(\sigma_* \neq \sigma_i)$, then $\sigma_{1,*} \neq \sigma_{1,i}$ or $\sigma_{2,*} \neq \bar{\sigma}_{2,i}$ or $\textsc{Adm}_* \neq \textsc{Adm}_i$: if $\textsc{Adm}_* \neq \textsc{Adm}_i$, then $\sigma_{1,*} \neq \sigma_{1,i}$ because $\sigma_{1,*}$ (resp. $\sigma_{1,i}$) is a signature of $\textsc{Adm}_*$ (resp. $\textsc{Adm}_i$). If $\sigma_{1,*} \neq \sigma_{1,i}$, then $\sigma_{2,*} \neq \bar{\sigma}_{2,i}$ because $\sigma_{2,*}$ (resp. $\bar{\sigma}_{2,i}$) is a signature of $\sigma_{1,*}$ (resp. $\sigma_{1,i}$). Finally, in all cases $\sigma_{*,2} \neq \bar{\sigma}_{i,2}$.

On the other hand, (6.13) implies that:

$$\mathsf{V.Ver}(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, \sigma_{2,*}, m_* || \sigma_{1,*}) = 1$$

Moreover, (6.14) implies that:

$$\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}, \pi_{\mathsf{sa},*}) = 0$$

Indeed, $\pi_{\mathsf{sa},*}$ cannot be equal to $\perp$ since it is computed by the proof algorithm from a valid signature. It implies that:

$$\mathsf{V.Judge}(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, m_* || \sigma_{1,*}, \sigma_{2,*}, \mathsf{spk}, \pi_{sa,*}) = 1$$

Finally, note that since $\pi_{\mathsf{sa},*} \leftarrow \mathsf{SaProof}(\mathsf{ssk}, m_*, \sigma_*, \mathsf{pk}_*)$, then:

$$\pi_{\mathsf{sa},*} \leftarrow \mathsf{V.Proof}(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, m_* || \sigma_{1,*}, \sigma_{2,*}, \mathsf{spk}, \mathsf{ssk})$$

We deduce that the probability that $\mathscr{B}$ wins its experiment is the same as the probability that $\mathscr{A}$ wins its experiments:

$$\mathsf{Adv}^{\mathsf{1\text{-}non\text{-}sei\text{-}2}}_{\mathsf{V},\mathscr{B}}(k) \geq \mathsf{Adv}^{\mathsf{SiAcc\text{-}2}}_{\mathsf{GUSS},\mathscr{A}}(k) = \lambda(k)$$

which concludes the proof. □

6.4.7 Security of GUSS

Theorem 29 *For any correct, deterministic and unforgeable DS scheme* D *and any correct, anonymous, accountable and non-usurpable VRS scheme* V*,* GUSS *instantiated by* (D, V) *is correct, immutable, transparent, strongly accountable and unlinkable.*

Proof: For any deterministic and unforgeable DS scheme D and any anonymous, accountable and non-usurpable VRS scheme V, we have already shown that GUSS instantiated by (D, V) is correct (Lemma 38), immutable (Lemma 39), transparent (Lemma 40), strongly accountable (Lemma 42, 43, 44 and 45) and unlinkable (Lemma 41). □

6.5 Algorithms Complexity and Comparison

In this section, we give the complexity of the algorithms of our scheme GUSS. We give the number of exponentiations in a prime order group for each algorithm. Moreover, we give the size of some values outputted by these algorithms (keys, signatures and proofs). This size is given in the number of group elements. For the sake of clarity, we do not distinguish between elements of a group $\mathbb{G}$ of prime order p where the DDH assumption is hard and elements of $\mathbb{Z}_p^*$. Finally, we compare our scheme with the scheme of Fleischhacker *et al.* [FKM+16]

Efficiency: In Table 6.1, we recall the number of exponentiations of each algorithm of (deterministic) Schnorr's signature (Definition 35), and we give the size in number of group elements of the secret/public keys $\mathsf{sk}_{\mathsf{Sh}}$ and $\mathsf{pk}_{\mathsf{Sh}}$ and the size of a signature σ_{Sh}.

In Table 6.2, we give the number of exponentiations of each algorithm of GUSS. In Table 6.3, we give the size of the secret/public keys of the signer $(\mathsf{sk}, \mathsf{pk})$, and the sanitizer $(\mathsf{ssk}, \mathsf{spk})$, the size of a signature σ and the size of a proof π. The first line corresponds to the generic case, where the values depend on the chosen signature scheme and the chosen verifiable ring signature scheme. The second line corresponds to the case where GUSS is instantiated by Schnorr and EVeR.

Schnorr	D.Gen	D.Sig	D.Ver	$\mathsf{sk}_{\mathsf{Sh}}$	$\mathsf{pk}_{\mathsf{Sh}}$	σ_{Sh}
exp/size	1	1	2	1	1	2

Table 6.1: Complexity analysis of Schnorr (Definition 35).

GUSS	SiGen	SaGen	Sig	San	Ver	SiProof	SiJudge	SaProof	SaJudge
generic	D.Gen + V.Gen	V.Gen	D.Sig + V.Sig$_2$	V.Sig$_2$	D.Ver + V.Ver$_2$	V.Proof	V.Judge	V.Proof	V.Judge
EvER and Schnorr	2	1	8	7	10	3	4	3	4

Table 6.2: Complexity analysis of the algorithms of GUSS (Definition 75).

GUSS	sk	pk	ssk	spk	σ	π
generic	$\mathsf{sk}_{\mathrm{EV}} + \mathsf{sk}_{\mathsf{Sh}}$	$\mathsf{pk}_{\mathrm{EV}} + \mathsf{pk}_{\mathsf{Sh}}$	$\mathsf{sk}_{\mathrm{EV}}$	$\mathsf{pk}_{\mathrm{EV}}$	$\sigma_2^{\mathrm{EV}} + \sigma^{\mathsf{Sc}}$	π_2^{EV}
EvER and Schnorr	2	2	1	1	12	5

Table 6.3: Complexity analysis of the elements size of GUSS (Definition 75).

	SiGen	SaGen	Sig	San	Ver	SiProof	SiJudge	**Total**
Fleischhacker *et al.* [FKM+16]	7	1	15	14	17	23	6	**83**
GUSS	2	1	8	7	10	3	4	**35**

Table 6.4: Comparison of the elements size of GUSS and the scheme of Fleischhacker *et al.* [FKM+16].

	pk	spk	sk	ssk	σ	π	**Total**
Fleischhacker *et al.* [FKM+16]	7	1	14	1	14	4	**41**
GUSS	2	1	2	1	12	5	**23**

Table 6.5: Comparison of the algorithms complexity of GUSS and the scheme of Fleischhacker *et al.* [FKM+16].

Comparison with Fleischhacker *et al.* [FKM+16]: In Table 6.4 and Table 6.5 we compare GUSS and the scheme of Fleischhacker *et al.* which is the most efficient unlinkable scheme of the literature. The first table compares the number of exponentiations of each algorithms of both schemes, namely the key generation algorithm of the signer (SiGen) and the sanitizer (SaGen), the signature algorithm (Sig), the verification algorithm (Ver), the sanitize algorithm (San), the proof algorithm (SiProof) and the judge algorithm (SiJudge). The second table compares the size of the public keys of the signer (pk) and the sanitizer (spk), the size of the secret keys of the signer (sk) and the sanitizer (ssk), the size of a signature (σ) and the size of a proof (π) outputted by SiProof. This size is measured in elements of a group $\mathbb{G}$ of prime order. As in [FKM+16], for the sake of clarity, we do not distinguish between elements of $\mathbb{G}$ and elements of $\mathbb{Z}_p^*$. We consider the best instantiation of the scheme of Fleischhacker *et al.* given in [FKM+16].

6.6 Conclusion

In this chapter, we revisited the notion of unlinkable sanitizable signature. We added a proof algortihm to this primitive that allows the sanitizer to prove the origine of a signature, and we extended the security model of accountability. Finally, we designed a generic unlinkable sanitizable signature scheme, named GUSS, based on verifiable ring signatures, which is twice as efficient as the best scheme of the literature. However, since there does not exist any secure verifiable ring signature scheme in the standard model, GUSS cannot be instantiated in the standard model. In the future, we will aim to design an efficient and secure unlinkable sanitizable signature in the standard model, either by designing a secure verifiable ring signature scheme in the standard model, or by designing a new generic scheme based on the same idea but that uses another kind of group/ring signature.

Chapter 7

How to Delegate Decryptions on a Time Interval

Contents

In this chapter, we focus on the following problem: Alice, who sent encrypted messages to different people, wants to allow Bob to decrypt some of its. More precisely, she would like to allow Bob to decrypt messages that have been sent in a time interval chosen *a posteriori* by Alice, *i.e.,* after she sent the messages. Such a functionality can be used to partially reveal to a judge the content of the encrypted emails sending by Alice during a trial. We formally define a primitive called *A Posteriori Openable Public Key Encryption* (APO-PKE) that have these properties. Moreover we define security models for this primitive. We present a naive scheme where the size of Bob's key (*i.e.,* that opens the interval of ciphertexts) is proportional to the number of ciphertexts, then we design an efficient scheme with keys of constant size. Our construction is generic and can be instantiated with most conventional public key encryption schemes. Finally, we prove the security of our scheme in the random oracle model. This work has been conducted in collaboration with

Pascal Lafourcade and has been published in the paper "A Posteriori Openable Public Key Encryption" at the IFIP SEC 2016 conference.

7.1 Introduction

Email privacy is an important computer security topic. Usually, plaintext messages are sent and stored by the mail server without any protection. There exists many straightforward softwares that allow everyone to encrypt and sign emails using public key cryptography, such as the well known GnuPG[1] tool. Unfortunately, these softwares are rarely used [WT99], consequently encrypted emails may be considered as a suspect behavior. Hence as P. Zimmermann, the designer of PGP, said:“*If privacy is outlawed, only outlaws will have privacy*”.

Our motivation is based on the following scenario, where Alice is suspected during a trial: she could have sent some crucial information to an accomplice by email during a specified time period. To find some clues, the judge Oliver needs to read these emails. The judge uses his authority to obtain from the server all emails sent by Alice (including dates of dispatch and receiver identities). Depending to the security policy used by Alice, we distinguish two cases:

- If the messages are not encrypted then the judge can read emails without relation to the investigation, which is a privacy violation.
- If messages are encrypted with the receivers public keys then the judge can suspect Alice to hide crucial information for the investigation. Moreover, without the receivers’ private keys, Alice has no solution to prove her innocence and cannot reveal her correspondence to the judge.

Neither of these two cases is advantageous to Alice. To solve this problem, Alice needs a mechanism to give to the judge a possibility to open all messages sent during a specified time period. In this chapter, we introduce a new cryptographic primitive called *A Posteriori Openable Public Key Encryption* (APO-PKE) where Alice can generate an interval key for the judge. With this key, he can only read the encrypted messages sent during this specific interval of time, because this key does not allow him to open other encrypted messages stored on the email server. The goal of this chapter is to propose a practical and efficient APO-PKE scheme to solve this problem.

7.1.1 Functionalities

A posteriori openable public key encryptions are public key encryptions with some additional functionalities. Therefore it offers its conventional functionalities: the generation of the public/secret keys of each users, the encryption of a plaintext using a public key, and the decryption of a ciphertext using a secret key. Furthermore, it provides a way to *extract* a special key, called the *interval key*, from two ciphertexts C_i and C_j. This key allows a user, called the *opener*, to decrypts all the messages that have been encrypted between the two ciphertexts C_i and C_j. The functionalities can be classified into three categories:

System initialisation: Each user generates his own public and private key. Moreover, each user initialises his *secret state, i.e.,* a secret value that is updated after each encryption.

Encryption and decryption: Consider two users called Alice and Bob. Using her secret state and Bob’s public key, Alice encrypts a message and updates her secret state. Then Bob decrypts it using his secret key.

Interval key extraction and interval opening: Consider a third user called Oliver. Using his global state and Oliver’s public key, Alice generates an *interval key* from two ciphertexts C_i and C_j she produced previously. Then using this interval key and his secret key, Oliver decrypts all the ciphertexts produced by Alice after C_i and before C_j.

[1] https://www.gnupg.org

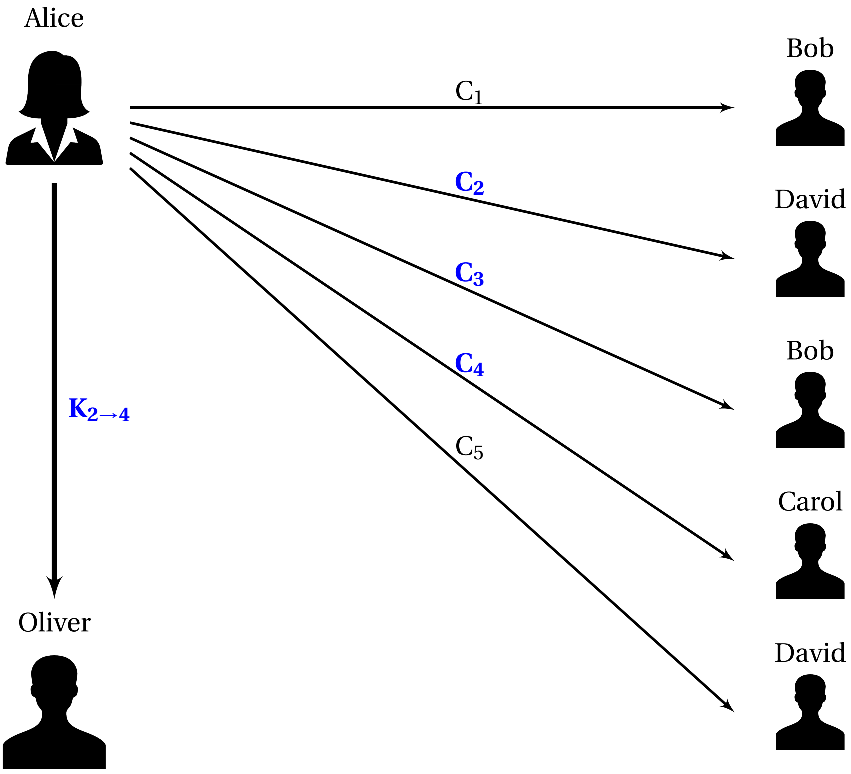


Figure 7.1: A posteriori openable encryption mechanisme overview.

To differentiate the two decryption mechanisms, we say that Bob *decrypts a ciphertext* and Oliver *opens a ciphertexts interval.* Figure 7.1 gives an illustration of the different functionalities of our primitive. Alice has three contacts Bob, Carol and David, and she sends successively five ciphertexts denoted C_1, C_2, C_3, C_4 and C_5 to respectively Bob, David, Bob, Carol and David. Then she generates the interval key $K_{2\to4}$ and she gives it to Oliver. Using this key, Oliver can open C_2, C_3 and C_4 (in bold and blue), but he cannot decrypts C_1 and C_5.

7.1.2 Security Goals

We first extend the IND-CPA security to *a posteriori* openable encryption schemes. However, this primitive allows the users to decrypt messages in two different ways, hence we consider two different kinds of chosen plaintext attack. Moreover, in our application, the judge must be convinced that the messages that he opens and the messages that the users decrypt are the same. Therefore, we define a security model called the *integrity* that formalizes this property.

Chosen plaintext attack: This property assures that messages that are not in the interval remain protected. Consider that the opener and some contacts of Alice collude. An APO-PKE scheme must prevent these adversaries to guess any message sending to a honest user that is not in the interval of time.

Chosen set of plaintext attack: This property assures that the interval key is useless without the opener secret key. The adversaries are several dishonest contacts of Alice that collude and that try to guess the messages encrypted in the time interval. These adversaries know the corresponding interval key generated with the public key of a honest opener.

Integrity: This property ensures that the decryption algorithm and the opening algorithm return the same plaintext for any ciphertext. For example, assume that Alice sends to Bob the message *"Bob, can you whittle my money? I have problems with the tax..."*. During his tax evasion trial, Alice gives an interval key to the judge. The compromising message is in the interval, however, when the judge opens it, he reads *"Bob, I am the most honest person in the world!"*. To achieve the integrity, an APO-PKE must prevent this kind of attack.

7.1.3 A Naive Solution

In Random Coin Decryptable Public key encryption (RCD-PKE) schemes (Definition 23), the random coin used to encrypt a message can be used as an alternative secret key to decrypt it. Based on such encryption schemes, we show how to construct a naïve APO-PKE scheme. Alice uses the RCD-PKE scheme to encrypt each message, and she gives to the Opener n random coins that correspond to the n ciphertexts of the chosen interval. Then the opener can decrypt all these messages. However, this method presents an inherent limitation when the number n is large, and Alice has to store all the random elements used to encrypt all the messages.

To illustrate the impact of this limitation, consider the following scenario. After each transaction, a company uses encrypted mails to send the invoice to its client. A tax inspector would like to check all the transactions carried out by the company over a period of ten years. During this period, the company sent thousands of invoices. However, using our naïve solution, the company sends as much keys as the number of invoices in these ten years to the tax inspector.

The aim of this chapter is to propose a practical scheme where the size of the key, the stored information and the complexity of the key generation algorithm are constant, *i.e.*, they do not increase with the number of ciphertexts.

7.1.4 Contributions

We define a new primitive called *A Posteriori Openable Public Key Encryption* (APO-PKE). It allows an opener to decrypt all messages sent by a user between two given dates. Our scheme is generic since it relies on any IND-CPA secure random coin decryptable public key encryption scheme and hash functions.

The generation of the interval-key and its size are independent to the number of messages in the interval. There is no restriction neither about the total number of ciphertexts nor about the number of ciphertexts in an interval. We provide security models for our primitive and we prove the security of our scheme in the Random Oracle Model (ROM).

In our scheme, the extraction algorithm is used only once per opener (or per set of encrypted mails). We give a restriction in our security model that captures this situation. It is not going against our motivation as long as we consider that two judges having an interval key in two different court cases (for the same set of mails) do not collude. To avoid this drawback, we need to reinitialize the secret values stored by a user after the generation of an interval-key, in order to be able to produce a new interval-key on the next encrypted mails. We leave the construction of a scheme with interval keys of constant size allowing to securly open two distinct intervals as an open problem.

7.1.5 Related Works

Functional encryption [SW05] is a public-key encryption primitive that allows a user to evaluate a function on the plaintext message using a key and a ciphertext. This cryptographic primitive was formalized in [BSW11]. It generalizes many well know cryptographic primitives such identity based encryption [BF01] or attribute based encryption [SW05]. Moreover, some schemes that evaluate an arbitrary function have been proposed in [GKP$^+$13b, GKP$^+$13a]. *A posteriori* openable encryption can be seen as a functional encryption, where all ciphertexts (resp. plaintexts) that are encrypted by one user correspond to a unique large ciphertext (resp. plaintext). Then the interval-keys allow a user to find only some parts of the corresponding plaintext. Our proposal scheme is an efficient solution for this kind of functional encryption.

Deniable encryption [CDNO97, KKK08] is an encryption primitive where there are two plain messages (an original and a hidden one) in the same ciphertext. Using his secret key, the receiver retrieves the original message. Using another shared secret key, the receiver also decrypts the hidden message. It is not possible for the sender to prove whether a ciphertext contains a hidden message or not. In *a posteriori* openable encryption, the judge is only convinced that the plaintext that he decrypts is the same message that the plaintext decrypted by the secret key of the receiver. This notion differs from undeniability since the judge is convinced that a message he decrypts using an interval key has actually been sent and received, but does not deal with messages sent on another channel (including different way to decrypt a message in the same ciphertext).

Some cryptographic primitives deal with time in decryption mechanism or rights delegation. *Timed-Release Encryption* (TRE), proposed in [May93], is a public key encryption where encrypted messages cannot be opened before a *release-time* chosen by the user who encrypted the messages. In this primitive, it is a time server that allows the receiver to decrypt the message *in the future* at a given date. Several TRE with diverse security properties have been proposed [CLQ05, CHKO06, BC05]. More recently, an extension of TRE, called *Time-Specific Encryption* (TSE), has been proposed in [PQ10] and deals with time intervals. Somehow these primitives are close to our because APO-PKE allows somebody to give decryption capabilities *in the future*, after that encrypted messages has been sent. However, TRE and TSE cannot be used to achieve APO-PKE, because TRE ciphertexts are intended to only one user and decryption capabilities cannot be delegated to another party. Moreover, in TRE, the time of decryption must be chosen during the encryption phase (*a priori*), while in our primitive it can be chosen at any time (*a posteriori*).

It is interesting to note that some TRE possess a pre-open mechanism [HYL05] that allows the sender to give decryption capabilities before the pre-specified release-time. In this case, a security requirement (called *binding* property) ensures that the decrypted message from the pre-open mechanism is the message decrypted by the receiver after the release-time [DT07]. For our primitive, we define a similar property, called *integrity*, that assures that the open algorithm and the decryption algorithm returns the same plaintext on the same ciphertext.

Finally, *Key-Insulated Encryption* (KIE) [DKXY02, LQY07, HW10] is a public key encryption primitive where messages are encrypted from a tag corresponding to a time period and a public key. At each time period corresponds a partial secret key computed from a master key and the previous partial secret key. Moreover, the public key is never changed. The motivation of this primitive is to provide secret keys that can be stored in an untrusted device without compromising the master key. Indeed, the leakage of a secret key only compromises the messages received in a specified time interval, and future encryptions remain secure. In the motivation of [DKXY02], the authors give another interesting application of this primitive based on [GPR98]: the secure delegation of decryption rights in a time period. However, this kind of delegation requires pre-defined time period. For example, if the time period corresponds to one month, then the delegation cannot be restricted to the last week of a first month and the first week of the following one. Moreover, the delegator must give a different secret key to each time period, so the decryption keys size is proportional to the number of time periods contained in the interval. Finally, our goal is that the sender of the encrypted messages delegates the decryption capabilities, while KIE focuses on delegations that are provided by the receiver. Thus this primitive cannot solve our problem.

7.2 Formal Definitions

In this section, we formally define our primitive and the related security properties.

7.2.1 A Posteriori Openable Encryption

An *a posteriori* openable encryption scheme is a kind of public key encryption scheme with some additional features: Alice may use some secret data she updates after each encryption, called the *secret state* and denoted sst, to construct a key to *a posteriori* open a sequence of messages that she had encrypted during an interval of time. We do not consider real time but a sequence of n successive ciphertexts $\{c_l\}_{1\leq l\leq n}$. This key allows a user, called the *opener*, to open all the messages between two ciphertexts c_i and c_j, where $1 \leq i < j \leq n$. This key is called the *interval-key*, and is denoted by $\mathrm{K}_{i \to j}$.

This primitive is formally described in Definition 76. In addition to the usual public key encryption algorithms, it contains an algorithm Ini that initializes a new secret state, an algorithm Ext that generates the interval key from the first and the last message of the interval and the public key of the opener, and an algorithm Ope that allows the opener to decrypt the messages in the interval using his secret key and the interval key.

Note that all the key pairs $(\mathsf{pk}, \mathsf{sk})$ come from the same algorithm Gen. However, for the sake of clarity, the keys of the opener are denote by $(\mathsf{pko}, \mathsf{sko})$.

Definition 76 (A Posteriori Openable Public Key Encryption scheme) *An* A Posteriori Openable Public Key Encryption *(APO-PKE) scheme is a tuple of polynomial time algorithms* $(\mathsf{Set}, \mathsf{Gen}, \mathsf{Ini}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ext}, \mathsf{Ope})$ *defined as follows:*

$\mathsf{Set}(k)$**:** *It returns a setup* set *and a messages set* $\mathcal{M}$*.*
$\mathsf{Gen}(\mathsf{set})$**:** *It returns a public/private key pair* $(\mathsf{pk}, \mathsf{sk})$.
$\mathsf{Ini}(\mathsf{set})$**:** *It initializes a global state* sst *and returns it.*
$\mathsf{Enc}^{\mathsf{sst}}_{\mathsf{pk}}(m)$**:** *It returns a ciphertext* c *and updates* sst.
$\mathsf{Dec}_{\mathsf{sk}}(c)$**:** *It returns a plaintext* m*.*
$\mathsf{Ext}^{\mathsf{sst}}_{\mathsf{pko}}(c_i, c_j)$**:** *It generates an interval key* $\mathrm{K}_{i \to j}$*.*
$\mathsf{Ope}_{\mathsf{sko}}(\mathrm{K}_{i \to j}, \{c_l\}_{i \leq l \leq j}, \{\mathsf{pk}_l\}_{i \leq l \leq j})$**:** *Inputs of this algorithm contain a set of ciphertexts* $\{c_l\}_{i \leq l \leq j}$ *and the set of the corresponding public keys* $\{\mathsf{pk}_l\}_{i \leq l \leq j}$*. It returns a set of plaintexts* $\{m_l\}_{i \leq l \leq j}$*.*

Moreover an APO-PKE is said to be correct *when the following equations hold for any* $k, n, w, i, j \in \mathbb{N}$ *such that* $1 \leq i \leq j \leq n$ *and any* n*-uple* $t = (t_1, \ldots, t_n) \in [\![1, w]\!]^n$*:*

$$\Pr\left[\begin{array}{l} (\mathsf{set}, \mathcal{M}) \leftarrow \mathsf{Set}(k); \forall l \in [\![1, n]\!], (\mathsf{pk}_l, \mathsf{sk}_l) \leftarrow \mathsf{Gen}(\mathsf{set}); \\ \mathsf{sst} \leftarrow \mathsf{Gen}(\mathsf{set}); (m_1, \ldots, m_w) \xleftarrow{\$} \mathcal{M}^w; \\ \forall l \in [\![1, w]\!], c_l \leftarrow \mathsf{Enc}^{\mathsf{sst}}_{\mathsf{pk}_{t_l}}(m_l); \\ \forall l \in [\![1, w]\!], m'_l \leftarrow \mathsf{Dec}_{\mathsf{sk}_{t_l}}(c_l); \end{array} : \forall l \in [\![1, w]\!], m'_l = m_l \right] = 1$$

$$\Pr\left[\begin{array}{l} (\mathsf{set}, \mathcal{M}) \leftarrow \mathsf{Set}(k); \forall l \in [\![1, n]\!], (\mathsf{pk}_l, \mathsf{sk}_l) \leftarrow \mathsf{Gen}(\mathsf{set}); \\ (\mathsf{pko}, \mathsf{sko}) \leftarrow \mathsf{Gen}(\mathsf{set}); \mathsf{sst} \leftarrow \mathsf{Gen}(\mathsf{set}); (m_1, \ldots, m_w) \xleftarrow{\$} \mathcal{M}^w; \\ \forall l \in [\![1, w]\!], c_l \leftarrow \mathsf{Enc}^{\mathsf{sst}}_{\mathsf{pk}_{t_l}}(m_l); \mathrm{K}_{i \to j} \leftarrow \mathsf{Ext}^{\mathsf{sst}}_{\mathsf{pko}}(c_i, c_j); \\ \{m'_l\}_{i \leq l \leq j} \leftarrow \mathsf{Ope}_{\mathsf{sko}}(\mathrm{K}_{i \to j}, \{c_l\}_{i \leq l \leq j}, \{\mathsf{pk}_l\}_{i \leq l \leq j}); \end{array} : \forall l \in [\![i, j]\!], m'_l = m_l \right] = 1$$

As in classical public key encryption schemes, we define *verifiable key* APO-PKE. Informally, a key pair $(\mathsf{pk}, \mathsf{sk})$ is valid when any message encrypted by pk will be correctly decrypted by sk. An APO-PKE is said to be *verifiable key* when there is a way to check whether a key pair is valid or not.

Definition 77 (Verifiable Key APO-PKE) *Let* $\mathrm{E} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Ini}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ext}, \mathsf{Ope})$ *be an APO-PKE. We say that a key pair* $(\mathsf{pk}, \mathsf{sk})$ *is* valid *for* E *according to the setup* $(\mathsf{set}, \mathcal{M})$ *when for any message* $m \in \mathcal{M}$ *and any state* sst*:*

$$\Pr\left[c \leftarrow \mathsf{Enc}^{\mathsf{sst}}_{\mathsf{pk}}(m); m' \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c) : m' = m\right] = 1$$

We say that E *is* verifiable-key *(VK) when there exists a* key pair verification algorithm Ver *such that* $1 \leftarrow \mathsf{Ver}(\mathsf{pk}, \mathsf{sk})$ *if and only if* $(\mathsf{pk}, \mathsf{sk})$ *is valid for* E *according to the setup* $(\mathsf{set}, \mathcal{M})$.

7.2.2 IND-CPA Security

This security property concerns the resistance of an *a posteriori* openable encryption scheme against chosen-plaintext attacks. We consider an opener that receives an interval-key and that colludes with dishonest users that receives encrypted messages from Alice. Adversaries choose two messages m_0 and m_1, and Alice encrypts one of these two messages m_b using the public key of a honest user. The opener cannot have a key that opens some interval that contains this ciphertext. The goal of the adversaries is to guess b.

IND-CPA Security vs one-time IND-CPA Security: During the IND-CPA experiment, the adversary has access to an extraction oracle that returns interval keys on intervals that do not contain the challenge ciphertext. Ideally, the adversary should be able to request several interval keys to the oracle. However, as shown previously, the scheme that we propose in this chapter do not achieve this property, because two interval keys allow the judge to open all the messages between the two intervals. Hence we define a weaker property, called one-time IND-CPA (OT-IND-CPA) security where the adversary can ask the extraction oracle only once. We define both IND-CPA and OT-IND-CPA security in the following definition, but we left the design of a (not one-time) IND-CPA secure scheme as an open problem.

Definition 78 (One-Time Indistinguishability against Chosen Plaintext Attack) *Let* $\mathrm{E} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Ini}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ext}, \mathsf{Ope})$ *be an APO-PKE, let* $k \in \mathbb{N}$ *be a security parameter, and let* $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$ *be a couple of polynomial time algorithms. Let the following oracles be:*

$\mathsf{Enc}^{\mathsf{sst}_*}_{\cdot}(\cdot)$: *This oracle takes as input a public key* pk *and a message* m*. On the first call to this oracle, it initializes two counters* $l = 1$ *and* $n = 1$*. On the first call to this oracle in the second phase (i.e., the first call of* $\mathscr{A}_2$*), it increments the counter* l*. Only in the first phase (i.e., while* $\mathscr{A}_1$ *is running), it increments the counter* n*. In any case, it runs* $c_l \leftarrow \mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pk}}(m)$ *and it increments the counter* l*. Finally, it returns* c_l

$\mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)$: *This oracle can be used only one time during the experiment. It takes as input a public key* pko *and two ciphertexts* c' *and* c''*. In the second phase, if there exists* $c_i = c'$ *and* $c_j = c''$ *such that* $i \leq n \leq j$ *then the oracle returns* $\perp$*. Else it runs* $\mathrm{K}_{i\to j} \leftarrow \mathsf{Ext}^{\mathsf{sst}_*}_{\mathsf{pko}}(c', c'')$ *and returns* $\mathrm{K}_{i\to j}$*.*

We define the One-Time Indistinguishability against Chosen Plaintext Attack *(OT-IND-CPA) experiment for the adversary* $\mathscr{A}$ *against* E *as follows:*

```
$\mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\mathrm{E},\mathscr{A}}(k)$:
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{set}, \mathscr{M}) \leftarrow \mathsf{Set}(k)$
$(\mathsf{pk}_*, \mathsf{sk}_*) \leftarrow \mathsf{Gen}(\mathsf{set})$
$\mathsf{sst}_* \leftarrow \mathsf{Ini}(\mathsf{set})$
$(m_0, m_1, \mathsf{st}) \leftarrow \mathscr{A}_1^{\mathsf{Enc}^{\mathsf{sst}_*}_{\cdot}(\cdot), \mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)}(\mathsf{set}, \mathsf{pk}_*)$
$c_* \leftarrow \mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pk}_*}(m_b)$
$b' \leftarrow \mathscr{A}_2^{\mathsf{Enc}^{\mathsf{sst}_*}_{\cdot}(\cdot), \mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)}(\mathsf{st}, \mathsf{pk}_*, c_*)$
Return $(b = b')$
```

We define the IND-CPA experiment as the OT-IND-CPA experiment except that the adversary can ask the oracle $\mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)$ *several times. We define the OT-IND-CPA advantage of* $\mathscr{A}$ *against* E *as follows:*

$$\mathsf{Adv}^{\mathit{OT\text{-}IND\text{-}CPA}}_{\mathrm{E},\mathscr{A}}(k) = \left| \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathit{OT\text{-}IND\text{-}CPA}}_{\mathrm{E},\mathscr{A}}(k)\right] - \frac{1}{2} \right|$$

We define the OT-IND-CPA advantage against E *as follows:*

$$\mathsf{Adv}^{\mathit{OT\text{-}IND\text{-}CPA}}_{\mathrm{E}}(k) = \max_{\mathscr{A} \in \mathrm{POLY}(k)^2} \left\{ \mathsf{Adv}^{\mathit{OT\text{-}IND\text{-}CPA}}_{\mathrm{E},\mathscr{A}}(k) \right\}$$

We define the IND-CPA advantages in a similar way. E *is said to be OT-IND-CPA (resp. IND-CPA) secure when* $\mathsf{Adv}_{\mathrm{E}}^{\textit{OT-IND-CPA}}(k)$ *(resp.* $\mathsf{Adv}_{\mathrm{E}}^{\textit{IND-CPA}}(k)$*) is negligible.*

7.2.3 IND-CSPA Security

An interval of ciphertexts coupled with the corresponding interval key can be seen as a unique ciphertext that encrypts a sequence of plaintexts. Thus, we define an indistinguishability based security model where several dishonest users, who collude, choose two sequences of plaintexts, and where the challenger encrypts successively each message of one of the two sequences. The adversaries receive the interval key of an honest opener who opens the interval that contains all these ciphertexts. The adversaries try to guess what is the sequence of plaintexts used by the challenger. They can use some oracles to create new honest users, to encrypt messages, and to extract interval keys for the honest opener.

Definition 79 (Indistinguishability against Chosen Set of Plaintexts Attack) *Let* $\mathrm{E} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Ini}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ext}, \mathsf{Ope})$ *be an APO-PKE, let* $k \in \mathbb{N}$ *be a security parameter, let* $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$ *be a couple of polynomial time algorithms, and let* $\mu \in \mathbb{N}$ *be an integer. Let the following oracles be:*

- $\mathsf{Gen}(\mathsf{set})$**:** *On the first call to this oracle, it creates an empty list of public key* $\mathscr{K}$*. In any case, it runs* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{set})$ *and adds* pk *to* $\mathscr{K}$*. Then it returns* pk*. This oracle can be called only* μ *times during the experiment.*
- $\mathsf{Enc}_{\cdot}^{\mathsf{sst}_*}(\cdot)$**:** *This oracle takes as input a public key* pk *and a message* m*. On the first call to this oracle, it initializes a counter* $n = 1$*. Only in the first phase (i.e., while* $\mathscr{A}_1$ *is running), it increments the counter* n*. In any case, it runs* $c_l \leftarrow \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{sst}_*}(m)$ *and it returns* c_l
- $\mathsf{Ext}_{\mathsf{pko}_*}^{\mathsf{sst}_*}(\cdot,\cdot)$**:** *This oracle takes as input two ciphertexts* c' *and* c''*. It runs* $\mathrm{K}_{i \to j} \leftarrow \mathsf{Ext}_{\mathsf{pko}}^{\mathsf{sst}_*}(c', c'')$ *and returns* $\mathrm{K}_{i \to j}$*.*

We define the μ-Indistinguishability against Chosen Set of Plaintexts Attack *(*μ*-IND-CSPA) experiment for the adversary* $\mathscr{A}$ *against* E *as follows:*

```
$\mathsf{Exp}_{\mathrm{E},\mathscr{A}}^{\mu\text{-}\mathbf{IND\text{-}CSPA}}(k)$:
$(b,d) \xleftarrow{\$} \{0,1\}^2$
$(\mathsf{set}, \mathscr{M}) \leftarrow \mathsf{Set}(k)$
$(\mathsf{pko}_*, \mathsf{sko}_*) \leftarrow \mathsf{Gen}(\mathsf{set})$
$\mathsf{sst}_* \leftarrow \mathsf{Ini}(\mathsf{set})$
$(q, \{m_{0,l}\}_{n<l\leq n+q}, \{m_{1,l}\}_{n<l\leq n+q}, \{\mathsf{pk}_l\}_{n<l\leq n+q}, st) \leftarrow \mathscr{A}_1^{\mathsf{Gen}(\mathsf{set}),\mathsf{Enc}_{\cdot}^{\mathsf{sst}_*}(\cdot),\mathsf{Ext}_{\mathsf{pko}_*}^{\mathsf{sst}_*}(\cdot,\cdot)}(\mathsf{set}, \mathsf{pko}_*)$
For all $l \in [\![n+1, n+q]\!]$:
    If $\mathsf{pk}_l = \mathsf{pko}_*$ or $\mathsf{pk}_l \in \mathscr{K}$, then $c_l^* \leftarrow \mathsf{Enc}_{\mathsf{pk}_l}^{\mathsf{sst}_*}(m_{b,l})$
    else, $c_l^* \leftarrow \mathsf{Enc}_{\mathsf{pk}_l}^{\mathsf{sst}_*}(m_{d,l})$
$\mathrm{K}_{(n+1)\to(n+q)} \leftarrow \mathsf{Ext}_{\mathsf{pko}_*}^{\mathsf{sst}_*}(c_{n+1}, c_{n+q})$
$b' \leftarrow \mathscr{A}_2^{\mathsf{Gen}(\mathsf{set}),\mathsf{Enc}_{\cdot}^{\mathsf{sst}_*}(\cdot),\mathsf{Ext}_{\mathsf{pko}_*}^{\mathsf{sst}_*}(\cdot,\cdot)}(\mathsf{st}, \mathsf{pko}_*, \{c_l^*\}_{n<l\leq n+q}, \mathrm{K}_{(n+1)\to(n+q)})$
Return $(b = b')$
```

We define the μ*-IND-CSPA advantage of* $\mathscr{A}$ *against* E *as follows:*

$$\mathsf{Adv}_{\mathrm{E},\mathscr{A}}^{\mu\text{-}\textit{IND-CSPA}}(k) = \left| \Pr\left[1 \leftarrow \mathsf{Exp}_{\mathrm{E},\mathscr{A}}^{\mu\text{-}\textit{IND-CSPA}}(k)\right] - \frac{1}{2} \right|$$

We define the μ*-IND-CSPA advantage against* E *as follows:*

$$\mathsf{Adv}_{\mathrm{E}}^{\mu\text{-}\textit{IND-CSPA}}(k) = \max_{\mathscr{A} \in \mathrm{POLY}(k)^2} \left\{ \mathsf{Adv}_{\mathrm{E},\mathscr{A}}^{\mu\text{-}\textit{IND-CSPA}}(k) \right\}$$

E *is said to be IND-CSPA secure when* $\mathsf{Adv}_{\mathrm{E}}^{t(k)\text{-}\textit{IND-CSPA}}(k)$ *is negligible for any polynomial* t*.*

Note that the adversary can choose the public key used to encrypt the messages of the sequence of plaintexts. Moreover, it can send the public keys of some dishonest users, *i.e.,* public keys that were not generated by the oracle $\mathsf{Gen}(\mathsf{set})$. However, the challenger does not really encrypt the message of the chosen sequence for the public keys of the dishonest users. More formally, the challenger picks two random bits b and d. The bit b determines the chosen sequence $\{m_{b,l}\}_{n<l\leq n+q}$. For all $l \in [\![n+1, n+q]\!]$, if pk_l corresponds to a honest user, then the challenger computes $c_l^* = \mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pk}_l}(m_{b,l})$. Else it computes $c_l^* = \mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pk}_l}(m_{d,l})$. In this case, c_l^* leaks no information about b.

7.2.4 Integrity

The last security property that we define is the *integrity.* It ensures that the decryption algorithm and the open algorithm return the same plaintext when there are used to decrypt the same ciphertext. We consider an adversary that generates several ciphertexts for different keys and an interval key. It wins its attack if one of these ciphertexts give two different plaintexts depending to the algorithm that the challenger uses to decrypt it.

Definition 80 (Integrity) *Let* $\mathrm{E} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Ini}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ext}, \mathsf{Ope})$ *be a verifiable key APO-PKE, let* $k \in \mathbb{N}$ *be a security parameter and let* $\mathcal{A}$ *be a polynomial time algorithm. We denote by* Ver *the key pair verification algorithm of* E*. We define the* integrity *experiment for the adversary* $\mathcal{A}$ *against* E *as follows:*

```
$\mathsf{Exp}^{\mathsf{Integrity}}_{\mathrm{E},\mathcal{A}}(k)$:
$(\mathsf{set}, \mathcal{M}) \leftarrow \mathsf{Set}(k)$
$(\mathsf{pko}_*, \mathsf{sko}_*) \leftarrow \mathsf{Gen}(\mathsf{set})$
$(\mathrm{N}, \{c_l\}_{1\leq l\leq \mathrm{N}}, \{\mathsf{pk}_l\}_{1\leq l\leq \mathrm{N}}, x, \mathsf{sk}_x, i, j, \mathrm{K}_{i\to j}) \leftarrow \mathcal{A}(\mathsf{set}, \mathsf{pk}_*)$
$\{m_l\}_{i\leq l\leq j} \leftarrow \mathsf{Ope}_{\mathsf{sko}_*}(\mathrm{K}_{i\to j}, \{c_l\}_{i\leq l\leq j}, \{\mathsf{pk}_l\}_{i\leq l\leq j})$
if $m_l \neq \mathsf{Dec}_{\mathsf{sk}_x}(c_x)$ and $1 \leftarrow \mathsf{Ver}(\mathsf{pk}_x, \mathsf{sk}_x)$,
then return $1$, else $0$.
```

We define the integrity advantage of $\mathcal{A}$ *against* E *as follows:*

$$\mathsf{Adv}^{\mathit{Integrity}}_{\mathrm{E},\mathcal{A}}(k) = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathit{Integrity}}_{\mathrm{E},\mathcal{A}}(k)\right]$$

We define the integrity advantage against E *as follows:*

$$\mathsf{Adv}^{\mathit{Integrity}}_{\mathrm{E}}(k) = \max_{\mathcal{A}\in\mathrm{POLY}(k)} \left\{\mathsf{Adv}^{\mathit{Integrity}}_{\mathrm{E},\mathcal{A}}(k)\right\}$$

E *is said to have the integrity property when* $\mathsf{Adv}^{\mathit{Integrity}}_{\mathrm{E}}(k)$ *is negligible.*

7.3 GAPO: a Generic A Posteriori Openable Encryption Scheme

In this section, we show how to build an efficient and generic *a posteriori* openable encryption scheme, were the interval key is in constant size (*i.e.,* it does not depend on the size of the interval). We first present the idea of our construction, then we formally describe our scheme.

7.3.1 Informal Overview

Our scheme uses a generic random coin decryptable public key encryption scheme (Definition 23). To encrypt the i^{th} message m_i, Alice shares it in two parts $\tilde{m}_i$ and $\hat{m}_i$ such that $m_i = \tilde{m}_i \oplus \hat{m}_i$. Then she encrypts $\tilde{m}_i$ in $\tilde{c}_i$ and $\hat{m}_i$ in $\hat{c}_i$ using the public key of Bob and two random coins. She sends $c_i = (\tilde{c}_i, \hat{c}_i)$ to Bob. To recover m_i, Bob decrypts $\tilde{c}_i$ and $\hat{c}_i$ to find $\tilde{m}_i$ and $\hat{m}_i$ and computes $m_i = \tilde{m}_i \oplus \hat{m}_i$.

Consider that Alice successively generates 6 ciphertexts denoted $c_i = (\widetilde{c}_i, \widehat{c}_i)$ for all $i \in [\![1,6]\!]$ using the method given in the previous paragraph. She wants to give a key that allows Oliver to decrypt the messages c_2, c_3 and c_4.

First, Alice gives to Oliver the *random coin* that was used to encrypt $\widehat{c}_2$. This *random coin* allows Oliver to decrypt $\widehat{c}_2$. Moreover, using this coin, Oliver extracts from $\widehat{c}_2$ the *random coin* that allows him to decrypt $\widehat{c}_3$. Then Oliver repeats this operation until decrypting $\widehat{c}_6$. Finally, Oliver knows $\widehat{m}_2$, $\widehat{m}_3$, $\widehat{m}_4$, $\widehat{m}_5$ and $\widehat{m}_6$.

Secondly, Alice gives the *random coin* that allows him to decrypt $\widetilde{c}_4$ to Oliver. Moreover, using this coin, Oliver extracts from $\widetilde{c}_4$ the *random coin* that allows him to decrypt $\widetilde{c}_3$. Then Oliver repeats this operation until decrypting $\widetilde{c}_1$. Finally, Oliver knows $\widetilde{m}_4$, $\widetilde{m}_3$, $\widetilde{m}_2$ and $\widetilde{m}_1$.

Figure 7.2 illustrates these two steps. Finally, Oliver computes $m_i = \widetilde{m}_i \oplus \widehat{m}_i$ for all $i \in [\![2,4]\!]$, however, he learns nothing about m_1 since he does not know $\widehat{m}_1$, and he learns nothing about m_5 and m_6 since he does not know $\widetilde{m}_5$ and $\widetilde{m}_6$.

Note that to prevent attacks where Bob and Oliver collude, the secret key of Bob must not allow him to extract the keys that open $\widetilde{c}_{i-1}$ and $\widehat{c}_{i+1}$ when he decrypts $c_i = (\widetilde{c}_i, \widehat{c}_i)$. In practice, the keys that Alice gives to Oliver are the random coins of $\widehat{c}_2$ and $\widetilde{c}_4$. Note that using this method, Alice gives two keys to Oliver regardless of the number of ciphertexts in the interval.

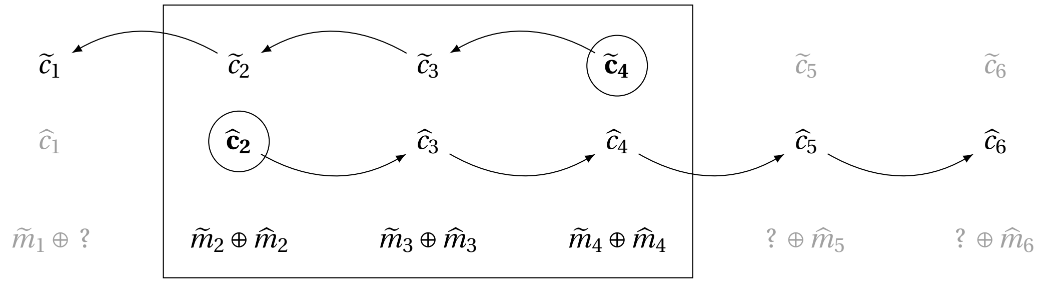


Figure 7.2: Opening mechanism for the interval $[2,4]$

7.3.2 GAPO Description

We give a generic a posteriori openable encryption scheme based on the idea given in the previous section.

Scheme 1 (GAPO) *Let* $\mathsf{RE} = (\mathsf{RSet}, \mathsf{RGen}, \mathsf{REnc}, \mathsf{RDec})$ *be a random coin decryptable (Definition 23) and verifiable key (Definition 24) public key encryption scheme, and let* RCDec *be the random coin decryption algorithm of* RE. *GAPO* $= (\mathsf{Set}, \mathsf{Gen}, \mathsf{Ini}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ext}, \mathsf{Ope})$ *is an a posteriori openable encryption scheme defined as follows:*

$\mathsf{Set}(k)$**:** *It runs* $(\mathsf{set}', \mathcal{M}', \mathcal{C}) \leftarrow \mathsf{RSet}(k)$ *and choses three hash functions* F, G *and* H *defined as follows:*

- $\mathsf{F} : \{0,1\}^* \rightarrow \{0,1\}^k$
- $\mathsf{G} : \{0,1\}^* \rightarrow \mathcal{C}$
- $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^{2k}$

Let $\mathcal{M} = \{\mathsf{LB}_{(|m|-k)}(m)\}_{m \in \mathcal{M}'}$ *be a set of messages where the function* $\mathsf{LB}_x(m)$ *returns the x first left bits of the bit-string m. This algorithm returns the setup* $\mathsf{set} = (\mathsf{set}', \mathrm{F}, \mathrm{G}, \mathrm{H})$ *and the set of messages* $\mathcal{M}$.

$\mathsf{Gen}(\mathsf{set})$**:** *It runs* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{RGen}(\mathsf{set}')$ *and returns* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Ini}(1^k)$**:** *It picks* $\widehat{\sigma} \xleftarrow{\$} \{0,1\}^k$, $\widetilde{\sigma} \xleftarrow{\$} \{0,1\}^k$ *and* $s \xleftarrow{\$} \{0,1\}^k$, *and it returns the secret state* $\mathsf{sst} = (s, \widehat{\sigma}, \widetilde{\sigma})$.

$\mathsf{Enc}_{\mathsf{pk}}^{\mathsf{sst}}(m)$**:** *This algorithm parses* $\mathsf{sst} = (s, \widehat{\sigma}, \widetilde{\sigma})$. *It picks* $\widehat{m} \leftarrow \{0,1\}^{|m|}$ *and computes* $\widetilde{m} = \widehat{m} \oplus m$. *It picks* $\widehat{\sigma}' \xleftarrow{\$} \{0,1\}^k$ *and* $\widetilde{\sigma}' \xleftarrow{\$} \{0,1\}^k$, *and it computes* $\widehat{\alpha} = (\widehat{m} || (\widehat{\sigma}' \oplus \mathsf{F}(\widehat{\sigma})))$ *and* $\widetilde{\alpha} = (\widetilde{m} || (\widetilde{\sigma} \oplus \mathsf{F}(\widetilde{\sigma}')))$. *Then it computes* $\widehat{c} \leftarrow \mathsf{REnc}_{\mathsf{pk}}(\widehat{\alpha}, \mathsf{G}(\widehat{\sigma}))$, $\widetilde{c} \leftarrow \mathsf{REnc}_{\mathsf{pk}}(\widetilde{\alpha}, \mathsf{G}(\widetilde{\sigma}'))$ *and* $d = (\widehat{\sigma} || \widetilde{\sigma}') \oplus \mathrm{H}(s || \widehat{c} || \widetilde{c})$. *Finally it updates the state* $\mathsf{sst} = (s, \widehat{\sigma}', \widetilde{\sigma}')$ *and returns* $c = (\widehat{c}, \widetilde{c}, d)$.

$\mathsf{Dec}_{\mathsf{sk}}(c)$**:** *This algorithm parses* $c = (\widehat{c}, \widetilde{c}, d)$*, then it runs* $(\widehat{m}||\widehat{x}) \leftarrow \mathsf{RDec}_{\mathsf{sk}}(\widehat{c})$ *and* $(\widetilde{m}||\widetilde{x}) \leftarrow \mathsf{RDec}_{\mathsf{sk}}(\widetilde{c})$*. It computes* $m = \widehat{m} \oplus \widetilde{m}$ *and returns it.*

$\mathsf{Ext}^{\mathsf{sst}}_{\mathsf{pko}}(c_i, c_j)$**:** *This algorithm parses* $\mathsf{sst} = (s, \widehat{\sigma}, \widetilde{\sigma})$*,* $c_i = (\widehat{c}_i, \widetilde{c}_i, d_i)$ *and* $c_j = (\widehat{c}_j, \widetilde{c}_j, d_j)$*. Then it computes* $(\widehat{\sigma}_{i-1}||\widetilde{\sigma}_i) = d_i \oplus \mathrm{H}(s||\widehat{c}_i||\widetilde{c}_i)$ *and* $(\widehat{\sigma}_{j-1}||\widetilde{\sigma}_j) = d_j \oplus \mathrm{H}(s||\widehat{c}_j||\widetilde{c}_j)$*. It picks* $r \xleftarrow{\$} \mathscr{C}$ *and returns* $\mathrm{K}_{i\to j} \leftarrow \mathsf{REnc}_{\mathsf{pko}}((\widehat{\sigma}_{i-1}||\widetilde{\sigma}_j), r)$*.*

$\mathsf{Ope}_{\mathsf{sko}}(\mathrm{K}_{i\to j}, \{c_l\}_{i\leq l\leq j}, \{\mathsf{pk}_l\}_{i\leq l\leq j})$**:** *This algorithm parses* $c_l = (\widehat{c}_l, \widetilde{c}_l, d_l)$ *for all* $l \in [\![i, j]\!]$*. Then it runs* $(\widehat{\sigma}_{i-1}||\widetilde{\sigma}_j) \leftarrow \mathsf{RDec}_{\mathsf{sko}}(\mathrm{K}_{i\to j})$*.*

- *For all* l *in* $\{i, i+1, \dots, j\}$*, it computes* $\widehat{r}_l = \mathsf{G}(\widehat{\sigma}_{l-1})$ *and runs* $(\widehat{m}_l||\widehat{x}_l) \leftarrow \mathsf{RCDec}_{\widehat{r}_l}(\widehat{c}_l, \mathsf{pk}_l)$*. It computes* $\widehat{\sigma}_l = \widehat{x}_l \oplus \mathsf{F}(\widehat{\sigma}_{l-1})$*. If* $\mathsf{REnc}_{\mathsf{pk}_l}((\widehat{m}_l||\widehat{x}_l), \mathsf{G}(\widehat{\sigma}_{l-1})) \neq \widehat{c}_l$ *then it returns* $\perp$*.*
- *For all* l *in* $\{j, j-1, \dots, i\}$*, it computes* $\widetilde{r}_l = \mathsf{G}(\widetilde{\sigma}_l)$ *and runs* $(\widetilde{m}_l||\widetilde{x}_{l-1}) \leftarrow \mathsf{RCDec}_{\widetilde{r}_l}(\widetilde{c}_l, \mathsf{pk}_l)$*. It computes* $\widetilde{\sigma}_{l-1} = \widetilde{x}_{l-1} \oplus \mathsf{F}(\widetilde{\sigma}_l)$*. If* $\mathsf{REnc}_{\mathsf{pk}_l}((\widetilde{m}_l||\widetilde{x}_{l-1}), \mathsf{G}(\widetilde{\sigma}_l)) \neq \widetilde{c}_l$ *then it returns* $\perp$*. It computes* $m_l = \widehat{m}_l \oplus \widetilde{m}_l$*.*

Finally, it returns $\{m_l\}_{i\leq l\leq j}$*.*

The encryption algorithm shares the l^{th} plaintext m_l in two parts $\widehat{m}_l$ and $\widetilde{m}_l$ such that $m = \widehat{m}_l \oplus \widehat{m}_l$. It generates two random values $\widehat{\sigma}_l$ and $\widetilde{\sigma}_l$, and it recovers the two values $\widehat{\sigma}_{l-1}$ and $\widetilde{\sigma}_{l-1}$ stored in the state sst (they were generated during the previous encryption). Using the random coin decryptable encryption scheme, it encrypts $\widehat{m}_l||(\widehat{\sigma}_l \oplus \mathsf{F}(\widehat{\sigma}_{l-1}))$ in $\widehat{c}_l$ with the random coin $\mathsf{G}(\widehat{\sigma}_{l-1})$, and $\widetilde{m}_l||(\widetilde{\sigma}_{l-1} \oplus \mathsf{F}(\widetilde{\sigma}_l))$ in $\widetilde{c}_l$ with the random coin $\mathsf{G}(\widetilde{\sigma}_l)$. Finally it masks the random elements $\widehat{\sigma}_{l-1}$ and $\widetilde{\sigma}_l$ with $\mathrm{H}(s||\widehat{c}||\widetilde{c})$ in d_l. The ciphertext is the triplet $c_l = (\widehat{c}_l, \widetilde{c}_l, d_l)$.

Knowing the secret key it is possible to recover $\widehat{m}_l$ and $\widetilde{m}_l$ by decrypting $\widehat{c}_l$ and $\widetilde{c}_l$, and to recover the plaintext m_l by computing $m_l = \widehat{m}_l \oplus \widehat{m}_l$.

The interval key generated from c_i and c_j is the encryption of $\widehat{\sigma}_{i-1}$ and $\widetilde{\sigma}_j$. These two values are hidden in the third part of the ciphertexts c_i and c_j, *i.e.*, d_i and d_j. These values can be recovered using the secret s stored in sst by computing $d_i \oplus \mathrm{H}(s||\widehat{c}_i||\widetilde{c}_i)$ and $d_j \oplus \mathrm{H}(s||\widehat{c}_j||\widetilde{c}_j)$.

Since the encryption scheme is random coin decryptable, the opener uses the random coin $\mathsf{G}(\widehat{\sigma}_{i-1})$ to open $\widehat{c}_i$. It obtains $\widehat{m}_i$ and $(\widehat{\sigma}_i \oplus \mathsf{F}(\widehat{\sigma}_{i-1}))$. It recovers $\widehat{\sigma}_i$ by computing $(\widehat{\sigma}_i \oplus \mathsf{F}(\widehat{\sigma}_{i-1})) \oplus \mathsf{F}(\widehat{\sigma}_{i-1})$. Then it opens $\widehat{c}_{i+1}$ using $\mathsf{G}(\widehat{\sigma}_i)$. Using recursively this algorithm, it is possible to decrypt all the ciphertexts in $\{\widehat{c}_l\}_{i\leq l\leq n}$, where n is the number of ciphertexts.

Using a similar method, the opener uses $\widetilde{\sigma}_j$ to decrypt all the ciphertexts in $\{\widetilde{c}_l\}_{1\leq l\leq j}$. Finally, the opener knows $\widehat{m}_l$ and $\widetilde{m}_l$ for all $l \in [\![i, j]\!]$, then it recovers the set of messages $\{m_l\}_{i\leq l\leq j}$.

7.4 Security Proofs of GAPO

In this section, we analyze the security of GAPO. We first informally explain why our scheme is secure before detailing the security proofs. We prove that GAPO is correct, OT-IND-CPA secure, IND-CSPA secure, and has the integrity property. Note that GAPO is not IND-CPA secure: if an opener has two interval keys for two disjoint intervals, then it opens all messages between the two intervals.

OT-IND-CPA security: The interval key allows the opener to decrypt all the messages in the interval. Moreover, this key also allows the opener to decrypt the first part $\widehat{c}$ of each ciphertext produced after the interval, and the second part $\widetilde{c}$ of each ciphertext produced before the interval. The random coins used to encrypt the other parts of these ciphertexts remain secret. So an adversary who has an interval key has no information about the messages that are not in the interval. If a message was sent before the interval, then the first part of the ciphertext $\widehat{c}$ was encrypted by an IND-CPA encryption scheme, so the adversary has no information about $\widehat{m}$, which implies that he has no information about m. On the other hand, if some message was sent after the interval, then the second part of the ciphertext $\widetilde{c}$ was encrypted by an IND-CPA encryption scheme, so the adversary has no information about $\widetilde{m}$, which implies that he has no information about m.

IND-CSPA security: The interval keys are encrypted using the public key of the opener, so without the secret key of the opener, the interval keys are useless. It implies that without the secret key of the opener, an adversary has no information about the bit-strings σ used to generate the random coins, because these bit-strings are always hashed or hiden by a one-time pad. Moreover, the messages in the interval are encrypted by an IND-CPA encryption scheme, so adversaries cannot guess what set of messages is encrypted in the IND-CSPA experiment.

Integrity: Since the public key encryption scheme is correct and verifiable key, then messages encrypted with the public key of a user are correctly decrypted with the corresponding secret key. Moreover, when he opens an interval of ciphertexts, the opener verifies that each message he opens is correctly encrypted: using the public key and the random coin, he re-encrypts the message. Since the encryption algorithm is deterministic, then he should obtain the same ciphertext. If so, then he is convinced that the recipient decrypts the same plaintext, else he aborts the algorithm.

7.4.1 Correctness

First, we show that GAPO is correct.

Theorem 30 *For any public key encryption scheme* RE *that is correct, random coin decryptable and verifiable key, GAPO instanciated with* RE *is correct.*

Proof: Let $\mathsf{RE} = (\mathsf{RSet}, \mathsf{RGen}, \mathsf{REnc}, \mathsf{RDec})$ be a random coin decryptable and verifiable key public key encryption scheme, and let RCDec be the random coin decryption algorithm of RE. Let $\mathsf{GAPO} = (\mathsf{Set}, \mathsf{Gen}, \mathsf{Ini}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Ext}, \mathsf{Ope})$ instanciated with RE. Let $(k, n, w, i, j) \in \mathbb{N}^5$ be five integers such that $1 \leq i \leq j \leq n$ and let $t = (t_1, \ldots, t_n) \in [\![1, w]\!]^n$ be a n-uple.

- For any $(\mathsf{set}, \mathscr{M}) \leftarrow \mathsf{Set}(k)$ such that $\mathsf{set} = (\mathsf{set}', \mathsf{F}, \mathsf{G}, \mathsf{H})$, any $(\mathsf{pk}_l, \mathsf{sk}_l) \leftarrow \mathsf{Gen}(\mathsf{set})$ for $l \in [\![1, n]\!]$, any $\mathsf{sst} \leftarrow \mathsf{Gen}(\mathsf{set})$ such that $\mathsf{sst} = (s, \widehat{\sigma}_0, \widehat{\sigma}_0)$, at each encryption $\mathsf{Enc}^{\mathsf{sst}}_{\mathsf{pk}_{t_l}}(m_l)$ for all $l \in [\![1, w]\!]$, we have:

$$\begin{aligned}
\widehat{\sigma}_l \xleftarrow{\$} \{0,1\}^k; \widetilde{\sigma}_l &\xleftarrow{\$} \{0,1\}^k \\
\widehat{m} \xleftarrow{\$} \{0,1\}^{|m|}; \widetilde{m} &= \widehat{m} \oplus m \\
\widehat{x}_l &= (\widehat{m}_l || (\widehat{\sigma}_l \oplus \mathsf{F}(\widehat{\sigma}_{l-1}))) \\
\widetilde{x}_l &= (\widetilde{m}_l || (\widetilde{\sigma}_{l-1} \oplus \mathsf{F}(\widetilde{\sigma}_l))) \\
\widehat{c}_l &= \mathsf{REnc}_{\mathsf{pk}_l}(\widehat{x}_l, \mathsf{G}(\widehat{\sigma}_{l-1})) \\
\widetilde{c}_l &= \mathsf{REnc}_{\mathsf{pk}_l}(\widetilde{x}_l, \mathsf{G}(\widetilde{\sigma}_l)) \\
c_l &= (\widehat{c}_l, \widetilde{c}_l, d_l)
\end{aligned}$$

  At each decryption $m'_l \leftarrow \mathsf{Dec}_{\mathsf{sk}_{t_l}}(c_l)$ for all $l \in [\![1, w]\!]$, we have:

$$\begin{aligned}
(\widehat{m}_l || \widehat{x}_l) &\leftarrow \mathsf{RDec}_{\mathsf{sk}_l}(\widehat{c}_l) \\
(\widetilde{m}_l || \widetilde{x}_l) &\leftarrow \mathsf{RDec}_{\mathsf{sk}_l}(\widetilde{c}_l)
\end{aligned}$$

  because RE is correct. Finally, $m_l = \widehat{m}_l \oplus \widetilde{m}_l$.

- for any $(\mathsf{set}, \mathscr{M}) \leftarrow \mathsf{Set}(k)$ such that $\mathsf{set} = (\mathsf{set}', \mathsf{F}, \mathsf{G}, \mathsf{H})$, any $(\mathsf{pk}_l, \mathsf{sk}_l) \leftarrow \mathsf{Gen}(\mathsf{set})$ for $l \in [\![1, n]\!]$, any $\mathsf{sst} \leftarrow \mathsf{Gen}(\mathsf{set})$ such that $\mathsf{sst} = (s, \widehat{\sigma}_0, \widehat{\sigma}_0)$, at each encryption $\mathsf{Enc}^{\mathsf{sst}}_{\mathsf{pk}_{t_l}}(m_l)$ for all $l \in [\![1, w]\!]$, we have:

$$\widehat{\sigma}_l \xleftarrow{\$} \{0,1\}^k; \widetilde{\sigma}_l \xleftarrow{\$} \{0,1\}^k$$
$$\widehat{m} \xleftarrow{\$} \{0,1\}^{|m|}; \widetilde{m} = \widehat{m} \oplus m$$
$$\widehat{x}_l = (\widehat{m}_l || (\widehat{\sigma}_l \oplus \mathsf{F}(\widehat{\sigma}_{l-1})))$$
$$\widetilde{x}_l = (\widetilde{m}_l || (\widetilde{\sigma}_{l-1} \oplus \mathsf{F}(\widetilde{\sigma}_l)))$$
$$\widehat{c}_l \leftarrow \mathsf{REnc}_{\mathsf{pk}_l}(\widehat{x}_l, \mathsf{G}(\widehat{\sigma}_{l-1}))$$
$$\widetilde{c}_l \leftarrow \mathsf{REnc}_{\mathsf{pk}_l}(\widetilde{x}_l, \mathsf{G}(\widetilde{\sigma}_l))$$
$$c_l = (\widehat{c}_l, \widetilde{c}_l, d_l)$$

After these encryptions, it holds that sst $= (s, \widehat{\sigma}_w, \widehat{\sigma}_w)$. For any $\mathrm{K}_{i\to j} \leftarrow \mathsf{Ext}^{\mathsf{sst}}_{\mathsf{pko}}(c_i, c_j)$:

$$\mathrm{K}_{i\to j} \leftarrow \mathsf{REnc}_{\mathsf{pko}}((\widehat{\sigma}_{i-1} || \widetilde{\sigma}_j), r)$$

For some random coin r. Moreover, we observe that:

$$(\widehat{\sigma}_{i-1} || \widetilde{\sigma}_j) \leftarrow \mathsf{RDec}_{\mathsf{sko}}(\mathrm{K}_{i\to j})$$

because RE is correct. For any $\{m'_l\}_{i\le l\le j} \leftarrow \mathsf{Ope}_{\mathsf{sko}}(\mathrm{K}_{i\to j}, \{c_l\}_{i\le l\le j}, \{\mathsf{pk}_l\}_{i\le l\le j})$, we have for all $l \in [\![i,j]\!]$:

$$\widehat{r}_l = \mathsf{G}(\widehat{\sigma}_{l-1}); \widetilde{r}_l = \mathsf{G}(\widetilde{\sigma}_l)$$
$$(\widehat{m}_l || \widehat{x}_l) \leftarrow \mathsf{RCDec}_{\widehat{r}_l}(\widehat{c}_l, \mathsf{pk}_l)$$
$$(\widetilde{m}_l || \widetilde{x}_{l-1}) \leftarrow \mathsf{RCDec}_{\widetilde{r}_l}(\widetilde{c}_l, \mathsf{pk}_l)$$

because RE is random coin decryptable. Finally, for all $l \in [\![i,j]\!]$, $m'_l = \widehat{m}_l \oplus \widetilde{m}_l = m_l$

These two properties conclude the proof. □

7.4.2 IND-CPA Security

Before proving that GAPO is OT-IND-CPA secure, we need to prove the two following technical lemmas.

Lemma 46 *Let $k \in \mathbb{N}$ be a security parameter, let F be a hash function, and let $\mathscr{A}$ be a pair of polynomial time algorithms. We define the following experiment for any $q \in \mathbb{N}$:*

```
$\mathsf{Exp}^{q\text{-}\mathsf{lstring}}_{\mathrm{F},\mathscr{A}}(k)$:
$\forall\ i \in [\![0,q]\!], \sigma_i \xleftarrow{\$} \{0,1\}^k$
$\{\theta_i\}_{1\le i\le n} \leftarrow \mathscr{A}((\sigma_i \oplus \mathsf{F}(\sigma_{i-1}))_{1\le i\le q})$
If $\exists\ i \le n$ such that $\sigma_q = \theta_i$ then return 1
Else return 0
```

Then $\Pr\left[1 \leftarrow \mathsf{Exp}^{q\text{-}lstring}_{\mathrm{F},\mathscr{A}}(k)\right]$ is negligible for any $q \in \mathbb{N}$ and any $\mathscr{A} \in \mathrm{POLY}(k)$ in the random oracle model.

Proof: We proof this lemma by induction.

- Let $\mathscr{A} \in \mathrm{POLY}(k)$ be an adversary that runs the 1-lstring experiment. Since $q = 1$, the adversary receives $\sigma_1 \oplus \mathsf{F}(\sigma_0)$ and try to guess σ_1 . Note that the adversary does not know σ_0. Since F is a random oracle, $\mathsf{F}(\sigma_0)$ is random from the adversary point of view. $\mathsf{F}(\sigma_0)$ acts as a one time pad on σ_1, so the best strategy of $\mathscr{A}$ is to picks each θ_i at random. The adversary returns a set of n values $\{\theta_i\}_{1\le i\le n}$, so the probability that he wins the experiment is lower than $n/(2^k - n)$, which is negligible.

- Assume that there exists a negligible function ϵ such that $\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{q\text{-lstring}}_{\mathrm{F},\mathscr{A}}(k)\right] \leq \epsilon(k)$ for any $\mathscr{A} \in \textsc{poly}(k)$, we show that $\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-lstring}}_{\mathrm{F},\mathscr{A}}(k)\right]$ is negligible for any $\mathscr{A} \in \textsc{poly}(k)$. We prove it by contraposition: assume that there exists an adversary $\mathscr{A} \in \textsc{poly}(k)$ such that $\mathsf{Pr}\left[\mathsf{Exp}^{(q+1)\text{-lstring}}_{\mathrm{F},\mathscr{A}}(k) = 1\right] = \lambda(k)$ is non-negligible, we show how to build $\mathscr{B} \in \textsc{poly}(k)$ such that $\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-lstring}}_{\mathrm{F},\mathscr{A}}(k)\right]$ is non negligible. $\mathscr{B}$ receives the tuple $(\sigma_i \oplus \mathsf{F}(\sigma_{i-1}))_{1\leq i\leq q}$. We set σ'_i for all $i \in [\![2, q+1]\!]$ such that $\sigma'_i = \sigma_{i-1}$, then $(\sigma_i \oplus \mathsf{F}(\sigma_{i-1}))_{1\leq i\leq q} = (\sigma'_i \oplus \mathsf{F}(\sigma'_{i-1}))_{2\leq i\leq q+1}$. The adversary $\mathscr{B}$ picks $\sigma^* \xleftarrow{\$} \{0,1\}^k$ and runs:

$$\{\theta_i\}_{1\leq i\leq n} \leftarrow \mathscr{A}((\sigma^*, (\sigma'_2 \oplus \mathsf{F}(\sigma_1)), \ldots, (\sigma'_{q+1} \oplus \mathsf{F}(\sigma'_q))))$$

  If there exists $\sigma'_0 \in \{0,1\}^k$ such that $\sigma^* = (\sigma'_1 \oplus \mathsf{F}(\sigma'_0))$, then $\mathscr{B}$ perfectly simulates the experiment for $\mathscr{A}$. We calculate the following probability:

$$\mathsf{Pr}\left[\exists \sigma'_0 \in \{0,1\}^k, \sigma^* = (\sigma'_1 \oplus \mathsf{F}(\sigma'_0))\right] = 1 - \left(\frac{k-1}{k}\right)^k \geq \frac{1}{2}$$

  Moreover, if $\mathscr{A}$ wins the experiment, then there exists $i \in \mathbb{N}$ such that $\sigma_q = \sigma'_{q+1} = \theta_i$, which implies that $\mathscr{B}$ wins its experiment. We deduce:

$$\begin{aligned}
&\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-lstring}}_{\mathrm{F},\mathscr{A}}(k)\right] \\
&= \mathsf{Pr}\left[\exists i \in \mathbb{N}, \sigma_q = \theta_i\right] \\
&\geq \mathsf{Pr}\left[\exists \sigma'_0 \in \{0,1\}^k, \sigma^* = (\sigma'_1 \oplus \mathsf{F}(\sigma'_0))\right] \cdot \mathsf{Pr}\left[\exists i \in \mathbb{N}, \sigma'_{q+1} = \theta_i | \exists \sigma'_0 \in \{0,1\}^k, \sigma^* = (\sigma'_1 \oplus \mathsf{F}(\sigma'_0))\right] \\
&\geq \frac{1}{2} \cdot \mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-lstring}}_{\mathrm{F},\mathscr{A}}(k)\right] \\
&\geq \frac{1}{2} \cdot \lambda(k)
\end{aligned}$$

  We deduce that $\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-lstring}}_{\mathrm{F},\mathscr{A}}(k)\right]$ is non negligible.

Finally, we conclude that $\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{q\text{-lstring}}_{\mathrm{F},\mathscr{A}}(k)\right]$ is negligible for any $q \in \mathbb{N}$ and any $\mathscr{A} \in \textsc{poly}(k)$ in the random oracle model. □

Lemma 47 *Let $k \in \mathbb{N}$ be a security parameter, let F be a hash function, and let $\mathscr{A}$ be a pair of polynomial time algorithms. We define the following experiment for any $q \in \mathbb{N}$:*

```
$\mathsf{Exp}^{q\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k)$:
$\forall\ i \in [\![0, q]\!], \sigma_i \xleftarrow{\$} \{0,1\}^k$
$\{\theta_i\}_{1\leq i\leq n} \leftarrow \mathscr{A}((\sigma_{i-1} \oplus \mathsf{F}(\sigma_i))_{1\leq i\leq q})$
If $\exists\ i \leq n$ such that $\sigma_0 = \theta_i$ then return $1$
Else return $0$
```

Then $\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{q\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k)\right]$ is negligible for any $q \in \mathbb{N}$ and any $\mathscr{A} \in \textsc{poly}(k)$ in the random oracle model.

Proof: We proof this lemma by induction.

- Let $\mathscr{A} \in \textsc{poly}(k)$ be an adversary that runs the 1-rstring experiment. Since $q = 1$, the adversary receives $\sigma_0 \oplus \mathsf{F}(\sigma_1)$ and tries to guess σ_0 . Note that the adversary does not know σ_1. Since F is a random oracle, $\mathsf{F}(\sigma_1)$ is random from the adversary point of view. $\mathsf{F}(\sigma_1)$ acts as a one time pad on σ_0, so the best strategy of $\mathscr{A}$ is to picks each θ_i at random. The adversary returns a set of n values $\{\theta_i\}_{1\leq i\leq n}$, so the probability that he wins the experiment is lower than $n/(2^k - n)$, which is negligible.

- Assume that there exists a negligible function ϵ such that $\Pr\left[1 \leftarrow \mathsf{Exp}^{q\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k)\right] \leq \epsilon(k)$ for any $\mathscr{A} \in \text{POLY}(k)$, we show that $\Pr\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k)\right]$ is negligible for any $\mathscr{A} \in \text{POLY}(k)$. We prove it by contraposition: assume that there exists an adversary $\mathscr{A} \in \text{POLY}(k)$ such that $\Pr\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k)\right] = \lambda(k)$ is non-negligible, we show how to build $\mathscr{B} \in \text{POLY}(k)$ such that $\Pr\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k) = \right]$ is non negligible. $\mathscr{B}$ receives the tuple $(\sigma_{i-1} \oplus \mathsf{F}(\sigma_i))_{1\leq i\leq q}$. The adversary $\mathscr{B}$ picks $\sigma^* \xleftarrow{\$} \{0,1\}^k$ and runs:

$$\{\theta_i\}_{1\leq i\leq n} \leftarrow \mathscr{A}(((\sigma_0 \oplus \mathsf{F}(\sigma_1)),\ldots,(\sigma_{q-1} \oplus \mathsf{F}(\sigma_q),\sigma^*)))$$

If there exists $\sigma_{q+1} \in \{0,1\}^k$ such that $\sigma^* = (\sigma_q \oplus \mathsf{F}(\sigma_{q+1}))$, then $\mathscr{B}$ perfectly simulates the experiment for $\mathscr{A}$. We bound the following probability:

$$\Pr\left[\exists \sigma_{q+1} \in \{0,1\}^k, \sigma^* = (\sigma_q \oplus \mathsf{F}(\sigma_{q+1}))\right] = 1 - \left(\frac{k-1}{k}\right)^k \geq \frac{1}{2}$$

Moreover, if $\mathscr{A}$ wins the experiment, then there exists $i \in \mathbb{N}$ such that $\sigma_0 = \theta_i$, which implies that $\mathscr{B}$ wins its experiment. We deduce:

$$\begin{aligned}&\Pr\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k)\right]\\ &= \Pr\left[\exists i \in \mathbb{N}, \sigma_0 = \theta_i\right]\\ &\geq \Pr\left[\exists \sigma_{q+1} \in \{0,1\}^k, \sigma^* = \sigma_q \oplus \mathsf{F}(\sigma_{q+1})\right] \cdot \Pr\left[\exists i \in \mathbb{N}, \sigma_0 = \theta_i | \exists \sigma_{q+1} \in \{0,1\}^k, \sigma^* = \sigma_q \oplus \mathsf{F}(\sigma_{q+1})\right]\\ &\geq \frac{1}{2}\cdot\lambda(k)\end{aligned}$$

We deduce that $\Pr\left[1 \leftarrow \mathsf{Exp}^{(q+1)\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k)\right]$ is non negligible.

Finally, we conclude that $\Pr\left[1 \leftarrow \mathsf{Exp}^{q\text{-rstring}}_{\mathrm{F},\mathscr{A}}(k)\right]$ is negligible for any $q \in \mathbb{N}$ and any $\mathscr{A} \in \text{POLY}(k)$ in the random oracle model. □

In the following, we show that the OT-IND-CPA advantage of GAPO is negligible when the adversary uses the oracle $\mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)$ before it receives the challenge (Lemma 48) and after it receives the challenge (Lemma 49). We deduce that GAPO is OT-IND-CPA secure (Theorem 31) from Lemma 48 and Lemma 49.

Lemma 48 *Let* $\mathsf{Exp}^{\mathit{OT\text{-}IND\text{-}CPA'}}_{\mathrm{E},\mathscr{A}}(k)$ *be the same experiment as* $\mathsf{Exp}^{\mathit{OT\text{-}IND\text{-}CPA}}_{\mathrm{E},\mathscr{A}}(k)$ *except that* $\mathscr{A}_1$ *does not have access to the oracle* $\mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)$. *If* RE *is IND-CPA then for all* $\mathscr{A} \in \text{POLY}(k)^2$ *there exists a negligible function* $\epsilon(k)$ *such that:*

$$\left|\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathit{OT\text{-}IND\text{-}CPA'}}_{\mathit{GAPO},\mathscr{A}}(k)\right] - \frac{1}{2}\right| = \epsilon(k)$$

Proof: We assume there exists a polynomial time adversary $\mathscr{A}$ and a non negligible function λ such that:

$$\left|\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA'}}_{\mathsf{GAPO},\mathscr{A}}(k)\right] - \frac{1}{2}\right| = \lambda(k)$$

We show how to construct an adversary $\mathscr{B} = (\mathscr{B}_1,\mathscr{B}_2) \in \text{POLY}(k)^2$ such that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathrm{RE},\mathscr{B}}(k)$ is non negligible:

Algorithm $\mathscr{B}_1(\mathsf{set},\mathsf{pk}_*)$**:** It runs $\mathsf{sst}_* \leftarrow \mathsf{Ini}(\mathsf{set})$ and parses $\mathsf{sst}_* = (s,\widetilde{\sigma}_0,\widetilde{\sigma}_0)$. It initializes two counters $n = 1$ and $l = 1$, and three empty lists F_{LIST}, G_{LIST} and H_{LIST}. It then runs $(\mathsf{st},m_0,m_1) \leftarrow \mathscr{A}_1(\mathsf{set},\mathsf{pk}_*)$. While $\mathscr{A}_1$ is running, $\mathscr{B}_1$ simulates the oracles as follows:

Random oracle $\mathsf{F}(.)$**:** $\mathscr{B}_1$ receives the input x. If there exists $(x',\mathrm{X}') \in \mathsf{F}_{\text{LIST}}$ such that $x = x'$ then it returns X'. Else it picks $\mathrm{X} \xleftarrow{\$} \{0,1\}^k$, adds (x,X) to the list F_{LIST} and returns X.

Random oracle $\mathsf{G}(.)$**:** $\mathscr{B}_1$ receives the input y. If there exists $(y', \mathrm{Y}') \in \mathrm{G}_{\mathrm{LIST}}$ such that $y = y'$ then it returns Y'. Else it picks $\mathrm{Y} \xleftarrow{\$} \mathscr{C}$, adds (y, Y) to the list $\mathrm{G}_{\mathrm{LIST}}$ and returns Y.

Random oracle $\mathrm{H}(.)$**:** $\mathscr{B}_1$ receives the input z. If z is queried by $\mathscr{A}_1$ and there exists a bit string α such that $z = s||\alpha$, then $\mathscr{B}_1$ aborts the experiment. Else, if there exists $(z', \mathrm{Z}') \in \mathrm{H}_{\mathrm{LIST}}$ such that $z = z'$ then it returns Z'. Else it picks $\mathrm{Z} \xleftarrow{\$} \{0,1\}^{2\cdot k}$, adds (z, Z) to the list $\mathrm{H}_{\mathrm{LIST}}$ and returns Z.

Oracle $\mathsf{Enc}^{\mathsf{sst}_*}_{\cdot}(\cdot)$**:** Using the input (pk, m) and the random oracles $\mathsf{F}(.)$, $\mathsf{G}(.)$ and $\mathrm{H}(.)$, $\mathscr{B}_1$ runs the algorithm $\mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pk}}(m)$ to compute c_l. It increments l and n and returns c_l.

Oracle $\mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)$**:** Using the input $(\mathsf{pko}, c'_i, c'_j)$ and the oracle $\mathrm{H}(.)$, $\mathscr{B}_2$ runs the algorithm $\mathrm{K}_{i\to j} \leftarrow \mathsf{Ext}^{\mathsf{sst}_*}_{\mathsf{pko}}(c'_i, c'_j)$ and returns $\mathrm{K}_{i\to j}$.

$\mathscr{B}_1$ picks $\widehat{\sigma}_n, \widetilde{\sigma}_n \xleftarrow{\$} \{0,1\}^k$ and $\widehat{m} \xleftarrow{\$} \{0,1\}^{|m_0|}$. It computes $\widetilde{m}_0 = m_0 \oplus \widehat{m}$ and $\widetilde{m}_1 = m_1 \oplus \widehat{m}$. Using $\mathsf{sst}_* = (s, \widehat{\sigma}_{n-1}, \widetilde{\sigma}_{n-1})$ and the random oracles $\mathsf{F}(.)$ and $\mathsf{G}(.)$, it sets $\mathrm{M}_0 = \widetilde{m}_0||(\widetilde{\sigma}_{n-1} \oplus \mathsf{F}(\widetilde{\sigma}_n))$ and $\mathrm{M}_1 = \widetilde{m}_1||(\widetilde{\sigma}_{n-1} \oplus \mathsf{F}(\widetilde{\sigma}_n))$. It runs $\widehat{c}_n \leftarrow \mathsf{REnc}_{\mathsf{pk}_*}(\widehat{m}||(\widehat{\sigma}_n \oplus \mathsf{F}(\widehat{\sigma}_{n-1})), \mathsf{G}(\widehat{\sigma}_{n-1}))$ and returns $(\mathsf{st}, \mathrm{M}_0, \mathrm{M}_1)$ to the challenger.

Algorithm $\mathscr{B}_2(\mathsf{st}, \widetilde{c}_n)$**:** If there does not exist $(z_*, \mathrm{Z}_*) \in \mathrm{H}_{\mathrm{LIST}}$ such that $z_* = (s||\widehat{c}_n||\widetilde{c}_n)$, then $\mathscr{B}_2$ picks $\mathrm{Z}_* \xleftarrow{\$} \{0,1\}^{2\cdot k}$, sets $z_* = (s||\widehat{c}_n||\widetilde{c}_n)$ and adds (z_*, Z_*) to the list $\mathrm{H}_{\mathrm{LIST}}$. It then computes $d_n = (\widehat{\sigma}_{n-1}||\widetilde{\sigma}_n) \oplus \mathrm{Z}$ and $c_n = (\widehat{c}_n, \widetilde{c}_n, d_n)$, and it increments l. $\mathscr{B}_2$ runs $b' \leftarrow \mathscr{A}_2(\mathsf{st}, \mathsf{pk}_*, c_n)$ While $\mathscr{A}_2$ is running, $\mathscr{B}_2$ simulates the oracles as follows:

Random oracle $\mathsf{F}(.)$**:** $\mathscr{B}_2$ simulates this oracle as $\mathscr{B}_1$ did.

Random oracle $\mathsf{G}(.)$**:** $\mathscr{B}_2$ receives the input y. If $y = \widetilde{\sigma}_n$ then $\mathscr{B}_2$ returns $b_* \xleftarrow{\$} \{0,1\}$ and aborts the experiment. Else, if there exists $(y', \mathrm{Y}') \in \mathrm{G}_{\mathrm{LIST}}$ such that $y = y'$ then it returns Y'. Else it picks $\mathrm{Y} \xleftarrow{\$} \mathscr{C}$, adds (y, Y) to the list $\mathrm{G}_{\mathrm{LIST}}$ and returns Y.

Random oracle $\mathrm{H}(.)$**:** $\mathscr{B}_2$ receives the input z. If z is queried by $\mathscr{A}_2$ and there exists a bit string α such that $z = s||\alpha$, then $\mathscr{B}_2$ returns $b_* \xleftarrow{\$} \{0,1\}$ and aborts the experiment. Else, if there exists $(z', \mathrm{Z}') \in \mathrm{H}_{\mathrm{LIST}}$ such that $z = z'$ then it returns Z'. Else it picks $\mathrm{Z} \xleftarrow{\$} \{0,1\}^{2\cdot k}$, adds (z, Z) to the list $\mathrm{H}_{\mathrm{LIST}}$ and returns Z.

Oracle $\mathsf{Enc}^{\mathsf{sst}_*}_{\cdot}(\cdot)$**:** Using the input (pk, m) and the random oracles $\mathsf{F}(.)$, $\mathsf{G}(.)$ and $\mathrm{H}(.)$, $\mathscr{B}_1$ runs the algorithm $\mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pk}}(m)$ to compute c_l. It increments l and returns c_l.

Finally, $\mathscr{B}_2$ returns $b_* = b'$.

Analysis: We remark that if $\mathscr{B}$ does not abort the experiment, then this experiment is perfectly simulated for $\mathscr{A}$. In this case, $\mathscr{B}$ wins its experiment if and only if $\mathscr{A}$ wins its experiment. Let E_1 and E_2 be the two following events:

- E_1 = "*$\mathscr{B}_2$ aborts during the simulation of the oracle* $\mathrm{H}(.)$"
- E_2 = "*$\mathscr{B}_2$ aborts during the simulation of the oracle* $\mathrm{G}(.)$"

The probability that $\mathscr{A}_2$ sends the query $z = s||\alpha$ to the oracle $\mathrm{H}(.)$ is lower than the probability that it guesses s. Since $\mathscr{A}$ does not know any information about s, this probability is lower than the probability that it guesses s at random. Let q_{H} be the number of queries to $\mathrm{H}(.)$, we deduce that:

$$\Pr[\mathrm{E}_1] \leq \frac{q_{\mathrm{H}}}{2^k - q_{\mathrm{H}}}$$

We observe that $\mathrm{E}_2 \Rightarrow \neg\mathrm{E}_1$: if $\mathscr{B}_2$ aborts on $\mathrm{H}(.)$ then it does not abort on $\mathrm{G}(.)$. $\neg\mathrm{E}_1$ implies that all d_i for all $i < l$ are random elements from $\mathscr{A}$ point of view. In this case, the values $\{\widetilde{\sigma}_{i-1} \oplus \mathsf{F}(\widetilde{\sigma}_i)\}_{n<i\leq l}$ are the only one information about $\widetilde{\sigma}_n$ that $\mathscr{A}$ knows. Lemma 46 claims that for any integer n and

l, and for any polynomialy bounded number of queries to the random oracle G(.), the probability $\epsilon(k)$ to guess $\widetilde{\sigma}_n$ using $\{\widetilde{\sigma}_{i-1} \oplus \mathsf{F}(\widetilde{\sigma}_i)\}_{n<i\leq l}$ is negligible . We deduce that:

$$\mathsf{Pr}[\mathrm{E}_2] \leq \epsilon(k)$$

We set E = "$\mathscr{B}$ does not abort". We set $\epsilon'(k) = \mathsf{Pr}[\mathrm{E}]$. Note that E = (E$_1$ or E$_2$). We deduce:

$$\epsilon'(k) = \mathsf{Pr}\,[\mathrm{E}] \leq \mathsf{Pr}\,[\mathrm{E}_1] + \mathsf{Pr}\,[\mathrm{E}_2] \leq \frac{q_{\mathrm{H}}}{2^k - q_{\mathrm{H}}} + \epsilon(k)$$

We deduce that $\epsilon'(k)$ is negligible. From previous results, we observe:

$$\begin{aligned}
\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathrm{RE},\mathscr{B}}(k)\right] &= \mathsf{Pr}[b = b_*] \\
&= \mathsf{Pr}[\mathrm{E}]\cdot\mathsf{Pr}[b = b_*|\mathrm{E}] + \mathsf{Pr}[\neg\mathrm{E}]\cdot\mathsf{Pr}[b = b_*|\neg\mathrm{E}] \\
&= \mathsf{Pr}[\mathrm{E}]\cdot\mathsf{Pr}[b = b_*|\mathrm{E}] + (1 - \mathsf{Pr}[\mathrm{E}])\cdot\mathsf{Pr}[b = b * |\neg\mathrm{E}] \\
&= \epsilon'(k)\cdot\frac{1}{2} + \left(1 - \epsilon'(k)\right)\cdot\mathsf{Pr}[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)] \\
&= \epsilon'(k)\cdot\left(\frac{1}{2} - \mathsf{Pr}[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)]\right) + \mathsf{Pr}[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)]
\end{aligned}$$

Finally:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathrm{RE},\mathscr{B}}(k) &= \left|\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathrm{RE},\mathscr{B}}(k)\right] - \frac{1}{2}\right| \\
&= \left|\epsilon'(k)\cdot\left(\frac{1}{2} - \mathsf{Pr}[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)]\right) + \mathsf{Pr}[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)] - \frac{1}{2}\right| \\
&\geq \lambda(k) - \epsilon'(k)\cdot\lambda(k)
\end{aligned}$$

This advantage is non-negligible, which contradicts our hypothesis and concludes the proof. □

Lemma 49 *Let* $\mathsf{Exp}^{\mathit{OT\text{-}IND\text{-}CPA}''}_{\mathrm{E},\mathscr{A}}(k)$ *be the same experiment as* $\mathsf{Exp}^{\mathit{OT\text{-}IND\text{-}CPA}}_{\mathrm{E},\mathscr{A}}(k)$ *except that* $\mathscr{A}_2$ *does not access to the oracle* $\mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)$. *If* RE *is IND-CPA then for all* $\mathscr{A} \in \textsc{poly}(k)^2$ *there exists a negligible function* $\epsilon(k)$ *such that:*

$$\left|\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathit{OT\text{-}IND\text{-}CPA}''}_{\mathit{GAPO},\mathscr{A}}(k)\right] - \frac{1}{2}\right| = \epsilon(k)$$

Proof: We assume there exists a polynomial time adversary $\mathscr{A}$ and a non negligible function λ such that:

$$\left|\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}''}_{\mathsf{GAPO},\mathscr{A}}(k)\right] - \frac{1}{2}\right| = \lambda(k)$$

We show how to construct an adversary $\mathscr{B} = (\mathscr{B}_1, \mathscr{B}_2) \in \textsc{poly}(k)^2$ such that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathrm{RE},\mathscr{B}}(k)$ is non negligible:

Algorithm $\mathscr{B}_1(\mathsf{set}, \mathsf{pk}_*)$**:** It runs $\mathsf{sst}_* \leftarrow \mathsf{Ini}(\mathsf{set})$ and parses $\mathsf{sst}_* = (s, \widehat{\sigma}_0, \widetilde{\sigma}_0)$. It initializes two counters $n = 1$ and $l = 1$, and three empty lists $\mathsf{F}_{\mathrm{LIST}}$, $\mathrm{G}_{\mathrm{LIST}}$ and $\mathrm{H}_{\mathrm{LIST}}$. Then it runs $(\mathsf{st}, m_0, m_1) \leftarrow \mathscr{A}_1(\mathsf{set}, \mathsf{pk}_*)$. While $\mathscr{A}_1$ is running, $\mathscr{B}_1$ simulates the oracles as follows:

- **Random oracle** $\mathsf{F}(.)$**:** $\mathscr{B}_1$ receives the input x. If there exists $(x', \mathrm{X}') \in \mathsf{F}_{\mathrm{LIST}}$ such that $x = x'$ then it returns X$'$. Else it picks $\mathrm{X} \xleftarrow{\$} \{0,1\}^k$, adds (x, X) to the list $\mathsf{F}_{\mathrm{LIST}}$ and returns X.
- **Random oracle** $\mathsf{G}(.)$**:** $\mathscr{B}_1$ receives the input y. If there exists $(y', \mathrm{Y}') \in \mathrm{G}_{\mathrm{LIST}}$ such that $y = y'$ then it returns Y$'$. Else it picks $\mathrm{Y} \xleftarrow{\$} \mathscr{C}$, adds (y, Y) to the list $\mathrm{G}_{\mathrm{LIST}}$ and returns Y.
- **Random oracle** H(.)**:** $\mathscr{B}_1$ receives the input z. If z is queried by $\mathscr{A}_1$ and there exists a bit string α such that $z = s||\alpha$, then $\mathscr{B}_1$ aborts the experiment. Else, if there exists $(z', \mathrm{Z}') \in \mathrm{H}_{\mathrm{LIST}}$ such that $z = z'$ then it returns Z$'$. Else it picks $\mathrm{Z} \xleftarrow{\$} \{0,1\}^{2\cdot k}$, adds (z, Z) to the list $\mathrm{H}_{\mathrm{LIST}}$ and returns Z.

Oracle $\mathsf{Enc}^{\mathsf{sst}_*}_{\cdot}(\cdot)$**:** Using the input (pk, m) and the random oracles $\mathsf{F}(.)$, $\mathsf{G}(.)$ and $\mathrm{H}(.)$, $\mathscr{B}_1$ runs the algorithm $\mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pk}}(m)$ to compute c_l. It increments l and n and returns c_l.

$\mathscr{B}_1$ picks $\widehat{\sigma}_n, \widetilde{\sigma}_n \xleftarrow{\$} \{0,1\}^k$ and $\widehat{m} \xleftarrow{\$} \{0,1\}^{|m_0|}$. It computes $\widetilde{m}_0 = m_0 \oplus \widehat{m}$ and $\widetilde{m}_1 = m_1 \oplus \widehat{m}$. Using $\mathsf{sst}_* = (s, \widehat{\sigma}_{n-1}, \widetilde{\sigma}_{n-1})$ and the random oracles $\mathsf{F}(.)$ and $\mathsf{G}(.)$, it sets $\mathrm{M}_0 = \widehat{m}_0 || (\widehat{\sigma}_n \oplus \mathsf{F}(\widehat{\sigma}_{n-1}))$ and $\mathrm{M}_1 = \widetilde{m}_1 || (\widehat{\sigma}_n \oplus \mathsf{F}(\widehat{\sigma}_{n-1}))$. It runs $\widetilde{c}_n \leftarrow \mathsf{REnc}_{\mathsf{pk}_*}(\widehat{m} || (\widetilde{\sigma}_{n-1} \oplus \mathsf{F}(\widetilde{\sigma}_n)), \mathsf{G}(\widetilde{\sigma}_n))$ and returns $(\mathsf{st}, \mathrm{M}_0, \mathrm{M}_1)$ to the challenger.

Algorithm $\mathscr{B}_2(\mathsf{st}, \widehat{c}_n)$**:** If there does not exist $(z_*, \mathrm{Z}_*) \in \mathrm{H}_{\mathrm{LIST}}$ such that $z_* = (s || \widehat{c}_n || \widetilde{c}_n)$, then $\mathscr{B}_2$ picks $\mathrm{Z}_* \xleftarrow{\$} \{0,1\}^{2 \cdot k}$, sets $z_* = (s || \widehat{c}_n || \widetilde{c}_n)$ and adds (z_*, Z_*) to the list $\mathrm{H}_{\mathrm{LIST}}$. It then computes $d_n = (\widehat{\sigma}_{n-1} || \widetilde{\sigma}_n) \oplus \mathrm{Z}$ and $c_n = (\widehat{c}_n, \widetilde{c}_n, d_n)$, and it increments l. $\mathscr{B}_2$ then runs $b' \leftarrow \mathscr{A}_1(\mathsf{st}, \mathsf{pk}_*, c_n)$ While $\mathscr{A}_1$ is running, $\mathscr{B}_2$ simulates the oracles as follows:

Random oracle $\mathsf{F}(.)$**:** $\mathscr{B}_2$ simulates this oracle as $\mathscr{B}_1$ did.

Random oracle $\mathsf{G}(.)$**:** $\mathscr{B}_2$ receives the input y. If $y = \widehat{\sigma}_{n-1}$ then $\mathscr{B}_2$ picks $b_* \xleftarrow{\$} \{0,1\}$ returns b_* and aborts the experiment. Else, if there exists $(y', \mathrm{Y}') \in \mathrm{G}_{\mathrm{LIST}}$ such that $y = y'$ then it returns Y'. Else it picks $\mathrm{Y} \xleftarrow{\$} \mathscr{C}$, adds (y, Y) to the list $\mathrm{G}_{\mathrm{LIST}}$ and returns Y.

Random oracle $\mathrm{H}(.)$**:** $\mathscr{B}_2$ receives the input z. If z is queried by $\mathscr{A}_2$ and there exists a bit string α such that $z = s || \alpha$, then $\mathscr{B}_2$ picks $b_* \xleftarrow{\$} \{0,1\}$ returns b_* and aborts the experiment. Else, if there exists $(z', \mathrm{Z}') \in \mathrm{H}_{\mathrm{LIST}}$ such that $z = z'$ then it returns Z'. Else it picks $\mathrm{Z} \xleftarrow{\$} \{0,1\}^{2 \cdot k}$, adds (z, Z) to the list $\mathrm{H}_{\mathrm{LIST}}$ and returns Z.

Oracle $\mathsf{Enc}^{\mathsf{sst}_*}_{\cdot}(\cdot)$**:** Using the input (pk, m) and the random oracles $\mathsf{F}(.)$, $\mathsf{G}(.)$ and $\mathrm{H}(.)$, $\mathscr{B}_1$ runs the algorithm $\mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pk}}(m)$ to compute c_l. It increments l and returns c_l.

Oracle $\mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot, \cdot)$**:** Using the input $(\mathsf{pko}, c'_{i'}, c'_{j'})$ and the oracle $\mathrm{H}(.)$, if there exists $i, j \in \mathbb{N}$ such that $i \leq n \leq j$ and $c_i = c'_{i'}$ and $c_j = c'_{j'}$, then $\mathscr{B}_2$ returns $\perp$, else $\mathscr{B}_2$ runs the algorithm $\mathrm{K}_{i' \to j'} \leftarrow \mathsf{Ext}^{\mathsf{sst}_*}_{\mathsf{pko}}(c'_{i'}, c'_{j'})$ and returns $\mathrm{K}_{i' \to j'}$.

Finally, $\mathscr{B}_2$ returns $b_* = b'$.

Analysis: We remark that if $\mathscr{B}$ does not abort the experiment, then this experiment is perfectly simulated for $\mathscr{A}$. In this case, $\mathscr{B}$ wins its experiment if and only if $\mathscr{A}$ wins its experiment. Let E_1 and E_2 be the two following events:

- $\mathrm{E}_1 =$ "*$\mathscr{B}_2$ aborts during the simulation of the oracle* $\mathrm{H}(.)$ "
- $\mathrm{E}_2 =$ "*$\mathscr{B}_2$ aborts during the simulation of the oracle* $\mathsf{G}(.)$ "

The probability that $\mathscr{A}_2$ sends the query $z = s || \alpha$ to the oracle $\mathrm{H}(.)$ is lower than the probability that it guesses s. Since $\mathscr{A}$ does not know any information about s, this probability is lower than the probability that it guesses s at random. Let q_{H} be the number of queries to $\mathrm{H}(.)$, we deduce that:

$$\mathsf{Pr}[\mathrm{E}_1] \leq \frac{q_{\mathrm{H}}}{2^k - q_{\mathrm{H}}}$$

We observe that $\mathrm{E}_2 \Rightarrow \neg \mathrm{E}_1$: if $\mathscr{B}_2$ aborts on $\mathrm{H}(.)$ then it does not abort on $\mathsf{G}(.)$. $\neg \mathrm{E}_1$ implies that all d_i for all $i < l$ are random elements from $\mathscr{A}$ point of view. In this case, the values $\{\widehat{\sigma}_i \oplus \mathsf{F}(\widehat{\sigma}_{i-1})\}_{1 \leq i < n}$ are the only one information about $\widehat{\sigma}_{n-1}$ that $\mathscr{A}$ knows. Lemma 47 claims that for any integer n and l, and for any polynomialy bounded number of queries to the random oracle $\mathsf{G}(.)$, the probability $\epsilon(k)$ to guess $\widehat{\sigma}_{n-1}$ using $\{\widehat{\sigma}_i \oplus \mathsf{F}(\widehat{\sigma}_{i-1})\}_{1 \leq i < n}$ is negligible . We deduce that:

$$\mathsf{Pr}[\mathrm{E}_2] \leq \epsilon(k)$$

We set $\mathrm{E} =$ "$\mathscr{B}$ does not abort". We set $\epsilon'(k) = \mathsf{Pr}[\mathrm{E}]$. We have $\mathrm{E} = (\mathrm{E}_1$ or $\mathrm{E}_2)$. We deduce:

$$\epsilon'(k) = \mathsf{Pr}\,[\mathrm{E}] \leq \mathsf{Pr}\,[\mathrm{E}_1] + \mathsf{Pr}\,[\mathrm{E}_2] \leq \frac{q_{\mathrm{H}}}{2^k - q_{\mathrm{H}}} + \epsilon(k)$$

We deduce that $\epsilon'(k)$ is negligible. From previous results, we deduce:

$$\begin{aligned}
\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{RE},\mathscr{B}}(k)\right] &= \Pr[b = b_*] \\
&= \Pr[\mathrm{E}] \cdot \Pr[b = b_* | \mathrm{E}] + \Pr[\neg \mathrm{E}] \cdot \Pr[b = b_* | \neg \mathrm{E}] \\
&= \Pr[\mathrm{E}] \cdot \Pr[b = b_* | \mathrm{E}] + (1 - \Pr[\mathrm{E}]) \cdot \Pr[b = b * | \neg \mathrm{E}] \\
&= \epsilon'(k) \cdot \frac{1}{2} + \left(1 - \epsilon'(k)\right) \cdot \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}''}_{\mathsf{GAPO},\mathscr{A}}(k)] \\
&= \epsilon'(k) \cdot \left(\frac{1}{2} - \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}''}_{\mathsf{GAPO},\mathscr{A}}(k)]\right) + \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}''}_{\mathsf{GAPO},\mathscr{A}}(k)]
\end{aligned}$$

Finally:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{RE},\mathscr{B}}(k) &= \left| \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{RE},\mathscr{B}}(k)\right] - \frac{1}{2} \right| \\
&= \left| \epsilon'(k) \cdot \left(\frac{1}{2} - \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}''}_{\mathsf{GAPO},\mathscr{A}}(k)]\right) + \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}''}_{\mathsf{GAPO},\mathscr{A}}(k)] - \frac{1}{2} \right| \\
&\geq \lambda(k) - \epsilon'(k) \cdot \lambda(k)
\end{aligned}$$

This advantage is non-negligible, which contradicts our hypothesis and concludes the proof.

□

Theorem 31 *Let* RE *be an IND-CPA secure RCD-PKE, then GAPO instantiated by* RE *is OT-IND-CPA secure in the random oracle model.*

Proof: This theorem is a direct consequence of Lemma 48 and 49: Let $\mathscr{A}$ be an adversary against APO-PKE. During the experiment $\mathsf{Exp}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\mathsf{APO\text{-}PKE},\mathscr{A}}(k)$, if $\mathscr{A}$ chooses to call oracle $\mathscr{O}^{\mathsf{CPA}}_{\mathsf{ext}}$ in the first phase then his advantage is negligible. However, if $\mathscr{A}$ chooses to call oracle $\mathscr{O}^{\mathsf{CPA}}_{\mathsf{ext}}$ in the second phase then his advantage is also negligible. Thus, we can conclude that $\mathsf{Adv}^{\mathsf{OT\text{-}IND\text{-}CPA}}_{\mathsf{APO\text{-}PKE},\mathscr{A}}(k)$ is negligible.

□

7.4.3 IND-CSPA Security

In this section we show that GAPO is IND-CSPA secure.

Lemma 50 *Let k be a security parameter, and μ be a positive integer. Let* $\mathsf{RE} = (\mathsf{RSet}, \mathsf{RGen}, \mathsf{REnc}, \mathsf{RDec})$ *be a random coin decryptable public key encryption that is IND-CPA$_{t(k)}$ secure for any polynomial t. We instanciate GAPO by* RE. *For any algorithm $\mathscr{A}$, let the experiment* $\mathsf{Exp}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)$ *(resp. the advantage* $\mathsf{Adv}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)$ *and the advantage* $\mathsf{Adv}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}_{\mathsf{GAPO}}(k)$*) be the same expermiment as* $\mathsf{Exp}^{\mu\text{-}\mathsf{IND\text{-}CSPA}}_{\mathsf{GAPO},\mathscr{A}}(k)$ *(resp. the same advantages as* $\mathsf{Adv}^{\mu\text{-}\mathsf{IND\text{-}CSPA}}_{\mathsf{GAPO},\mathscr{A}}(k)$ *and* $\mathsf{Adv}^{\mu\text{-}\mathsf{IND\text{-}CSPA}}_{\mathsf{GAPO}}(k)$*) except that:*

- $\mathrm{K}_{(n+1)\to(n+q)}$ *is computed as follows: the experiment picks* $\gamma \overset{\$}{\leftarrow} \mathscr{M}$ *and* $r \overset{\$}{\leftarrow} \mathscr{C}$*, and runs* $\mathrm{K}_{(n+1)\to(n+q)} \leftarrow \mathsf{REnc}_{\mathsf{pko}}(\gamma, r)$.
- $\mathsf{Ext}^{\mathsf{sst}_*}_{\mathsf{pko}_*}(\cdot,\cdot)$ *is replaced by the following oracle:*

  $\mathsf{R\text{-}Ext}^{\mathsf{sst}_*}_{\mathsf{pko}_*}(\cdot,\cdot)$**:** *This oracle takes as input a public key* pko *and two ciphertexts c' and c''. It picks* $\gamma \overset{\$}{\leftarrow} \mathscr{M}$ *and* $r \overset{\$}{\leftarrow} \mathscr{C}$*, and runs* $\mathrm{K} \leftarrow \mathsf{REnc}_{\mathsf{pko}}(\gamma, r)$*. It returns* K.

If $\mathsf{Adv}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}_{\mathsf{GAPO}}(k)$ *is negligible then* $\mathsf{Adv}^{\mu\text{-}\mathsf{IND\text{-}CSPA}}_{\mathsf{GAPO}}(k)$ *is negligible.*

Proof: Let k be a security parameter, and μ be a positive integer. Assume that there exists an algorithm $\mathscr{A} \in \text{POLY}(k)^2$, a negligible function ϵ and a non-negligible function λ such that:

$$\mathsf{Adv}_{\mathsf{GAPO}}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}(k) \leq \epsilon(k)$$
$$\mathsf{Adv}_{\mathsf{GAPO},\mathscr{A}}^{\mu\text{-}\mathsf{IND\text{-}CSPA}}(k) \geq \lambda(k)$$

We show how to build an algorithm $\mathscr{B} \in \text{POLY}(k)$ such that $\mathsf{Adv}_{\mathsf{RE},\mathscr{B}}^{\mathsf{IND\text{-}CPA}_{t(k)}}(k)$ is non-negligible for a polynomial t.

Algorithm $\mathscr{B}(\mathsf{set},\mathsf{pko}_*)$**:** It initializes three hash function F, G and G and sets $\mathsf{set}_* = (\mathsf{set},\mathrm{F},\mathrm{G},\mathrm{H})$. It runs $\mathsf{sst}_* \leftarrow \mathsf{Ini}(\mathsf{set})$ and runs:

$$(q,\{m_{0,l}\}_{n<l\leq n+q},\{m_{1,l}\}_{n<l\leq n+q},\{\mathsf{pk}_l\}_{n<l\leq n+q},\mathsf{st}) \leftarrow \mathscr{A}_1(\mathsf{set}_*,\mathsf{pko}_*)$$

While $\mathscr{A}_1$ is running, $\mathscr{B}$ simulates the following oracles to $\mathscr{A}_1$.

Oracle $\mathsf{Gen}(\mathsf{set})$**:** On the first call to this oracle, it creates an empty list of public key $\mathscr{K}$. In any case, it runs $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{set}_*)$ and adds pk to $\mathscr{K}$. Then it returns pk. This oracle can be called only μ times during the experiment.

Oracle $\mathsf{Enc}_{\mathsf{pko}_*}^{\mathsf{sst}_*}(\cdot)$**:** This oracle takes as input a public key pk and a message m. On the first call to this oracle, it initializes a counter $n := 1$. Only in the first phase (*i.e.,* while $\mathscr{A}_1$ is running), it increments the counter n. In any case, it runs $c_l \leftarrow \mathsf{Enc}_{\mathsf{pk}}^{\mathsf{sst}_*}(m)$ and it returns c_l

Oracle $\mathsf{Ext}_{\cdot}^{\mathsf{sst}_*}(\cdot,\cdot)$**:** At the l^{th} call, this oracle takes as input two ciphertexts c' and c''. This algorithm parses $\mathsf{sst}_* = (s,\widehat{\sigma},\widetilde{\sigma})$, $c' = (\widehat{c}_i,\widetilde{c}_i,d_i)$ and $c'' = (\widehat{c}_j,\widetilde{c}_j,d_j)$. Then it computes $(\widehat{\sigma}_{i-1}||\widetilde{\sigma}_i) = d_i \oplus \mathrm{H}(s||\widehat{c}_i||\widetilde{c}_i)$ and $(\widehat{\sigma}_{j-1}||\widetilde{\sigma}_j) = d_j \oplus \mathrm{H}(s||\widehat{c}_j||\widetilde{c}_j)$. It sets $\mathrm{M}_{l,0} = (\widehat{\sigma}_{i-1}||\widetilde{\sigma}_j)$ and picks $\mathrm{M}_{l,1} \xleftarrow{\$} \mathscr{M}$. $\mathscr{B}$ then sends $(\mathrm{M}_{l,0},\mathrm{M}_{l,1})$ to the encryption oracle $\mathsf{Enc}_{\mathsf{pko}_*}(\mathsf{LR}_b(\cdot,\cdot),*)$ and receives K_l. It returns K_l to $\mathscr{A}_1$.

$\mathscr{B}$ then picks $(b',d) \xleftarrow{\$} \{0,1\}^2$. For all $l \in [\![n+1,n+q]\!]$:

- If $\mathsf{pk}_l = \mathsf{pko}_*$ or $\mathsf{pk}_l \in \mathscr{K}$, then it runs $c_l^* \leftarrow \mathsf{Enc}_{\mathsf{pk}_l}^{\mathsf{sst}_*}(m_{b,l})$.
- Else, it runs , $c_l^* \leftarrow \mathsf{Enc}_{\mathsf{pk}_l}^{\mathsf{sst}_*}(m_{d,l})$.

$\mathscr{B}$ parses $\mathsf{sst}_* = (s,\widehat{\sigma}_{n+q},\widetilde{\sigma}_{n+q})$, $c_{n+1} = (\widehat{c}_{n+1},\widetilde{c}_{n+1},d_{n+1})$ and $c_{n+q} = (\widehat{c}_{n+q},\widetilde{c}_{n+q},d_{n+q})$, then it computes:

$$(\widehat{\sigma}_n||\widetilde{\sigma}_{n+1}) = d_{n+1} \oplus \mathrm{H}(s||\widehat{c}_{n+1}||\widetilde{c}_{n+1})$$

$$(\widehat{\sigma}_{n+q-1}||\widetilde{\sigma}_{n+q}) = d_{n+q} \oplus \mathrm{H}(s||\widehat{c}_{n+q}||\widetilde{c}_{n+q})$$

It sets $\mathrm{M}_{*,0} = (\widehat{\sigma}_n||\widetilde{\sigma}_{n+q})$ and picks $\mathrm{M}_{l,1} \xleftarrow{\$} \mathscr{M}$. $\mathscr{B}$ sends $(\mathrm{M}_{l,0},\mathrm{M}_{l,1})$ to the encryption oracle $\mathsf{Enc}_{\mathsf{pko}_*}(\mathsf{LR}_b(\cdot,\cdot),*)$ and receives $\mathrm{K}_{(n+1)\to(n+q)}$. Finally, $\mathscr{B}$ runs:

$$b' \leftarrow \mathscr{A}_2(\mathsf{st},\mathsf{pko}_*,\{c_l^*\}_{n<l\leq n+q},\mathrm{K}_{(n+1)\to(n+q)})$$

While $\mathscr{A}_2$ is running, $\mathscr{B}$ simulates the oracles as for $\mathscr{A}_1$. If $(b'' = b')$, then $\mathscr{B}$ returns $b_* = 0$, else it returns $b_* = 1$.

Analysis: $\mathscr{B}$ wins the experiment if and only if $b_* = b$. Let q_{ext} be the number of calls to the oracle

$\mathsf{Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)$:

$$\begin{aligned}
\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}_{q_{\mathrm{ext}}+1}}_{\mathsf{RE},\mathscr{B}}(k)\right] &= \mathsf{Pr}[b_* = b] \\
&= \mathsf{Pr}[b=0]\cdot\mathsf{Pr}[b_* = b|b=0] + \mathsf{Pr}[b=1]\cdot\mathsf{Pr}[b_* = b|b=1] \\
&= \frac{1}{2}\cdot(\mathsf{Pr}[b_* = b|b=0] + \mathsf{Pr}[b_* = b|b=1]) \\
&= \frac{1}{2}\cdot(\mathsf{Pr}[b_* = 0|b=0] + \mathsf{Pr}[b_* = 1|b=1]) \\
&= \frac{1}{2}\cdot\left(\mathsf{Pr}[b'' = b'|b=0] + \mathsf{Pr}[b'' = b'|b=1]\right) \\
&= \frac{1}{2}\cdot\left(\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mu\text{-}\mathsf{IND\text{-}CSPA}}_{\mathsf{GAPO},\mathscr{A}}(k)\right] + \left[1 \leftarrow \mathsf{Exp}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)\right]\right)
\end{aligned}$$

We deduce the advantage of $\mathscr{B}$:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{IND\text{-}CPA}_{q_{\mathrm{ext}}+1}}_{\mathsf{RE},\mathscr{B}}(k) &= \left|\frac{1}{2} - \frac{1}{2}\cdot\left(\mathsf{Pr}\left[1 \leftarrow \mathsf{Exp}^{\mu\text{-}\mathsf{IND\text{-}CSPA}}_{\mathsf{GAPO},\mathscr{A}}(k)\right] + \left[1 \leftarrow \mathsf{Exp}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)\right]\right)\right| \\
&\geq \frac{\lambda(k) - \epsilon(k)}{2}
\end{aligned}$$

This advantage is non-negligible, which concludes the proof. □

Lemma 51 *Let k be a security parameter, and μ be a positive integer. Let* $\mathsf{RE} = (\mathsf{RSet}, \mathsf{RGen}, \mathsf{REnc}, \mathsf{RDec})$ *be a random coin decryptable public key encryption. We instanciate GAPO by* RE. *If* RE *is IND-CPA*$_{(t(k),\mu+1)}$ *secure for any polynomial t, then* $\mathsf{Adv}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}_{\mathsf{GAPO}}(k)$ *is negligible in the random oracle model.*

Proof: Assume that there exists $\mathscr{A} \in \text{POLY}(k)^2$ such that $\lambda(k) = \mathsf{Adv}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}_{\mathsf{GAPO},\mathscr{A}}(k)$ is negligible. We show how to build $\mathscr{B} \in \text{POLY}(k)$ such that there exists a polynomial t such that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}_{(t(k),\mu+1)}}_{\mathsf{RE},\mathscr{B}}(k)$ is negligible.

Algorithm $\mathscr{B}(\mathsf{set}, \mathsf{pk}_0, \ldots \mathsf{pk}_\mu)$**:** $\mathscr{B}$ initializes three random oracles F, G and H. It sets $\mathsf{set}' = (\mathsf{set}, \mathrm{F}, \mathrm{G}, \mathrm{H})$ and $\mathsf{pko}_* = \mathsf{pk}_0$. It runs $\mathsf{sst}_* \leftarrow \mathsf{Ini}(\mathsf{set}')$. It initializes an empty set $\mathscr{S} := \emptyset$, a counter $\theta := 0$ and three empty lists $\mathrm{F}_{\mathrm{LIST}}$, $\mathrm{G}_{\mathrm{LIST}}$ and $\mathrm{H}_{\mathrm{LIST}}$. It runs:

$$(q, \{m_{0,l}\}_{n<l\leq n+q}, \{m_{1,l}\}_{n<l\leq n+q}, \{\mathsf{pk}'_l\}_{n<l\leq n+q}, \mathsf{st}) \leftarrow \mathscr{A}_1(\mathsf{set}, \mathsf{pko}_*)$$

It simulates the oracles to $\mathscr{A}_1$ as follows:

Random oracle $\mathsf{F}(.)$**:** $\mathscr{B}$ receives the input x. If there exists $(x', \mathrm{X}') \in \mathrm{F}_{\mathrm{LIST}}$ such that $x = x'$ then it returns X'. Else it picks $\mathrm{X} \xleftarrow{\$} \{0,1\}^k$, adds (x, X) to the list $\mathsf{F}_{\mathrm{LIST}}$ and returns X. If the oracle has been called by $\mathscr{A}_1$, then $\mathscr{B}$ updates $\mathscr{S} := \mathscr{S} \cup \{x\}$.

Random oracle $\mathsf{G}(.)$**:** $\mathscr{B}$ receives the input y. If there exists $(y', \mathrm{Y}') \in \mathrm{G}_{\mathrm{LIST}}$ such that $y = y'$ then it returns Y'. Else it picks $\mathrm{Y} \xleftarrow{\$} \mathscr{C}$, adds (y, Y) to the list $\mathrm{G}_{\mathrm{LIST}}$ and returns Y. If the oracle has been called by $\mathscr{A}_1$, then $\mathscr{B}$ updates $\mathscr{S} := \mathscr{S} \cup \{y\}$.

Random oracle $\mathrm{H}(.)$**:** $\mathscr{B}$ receives the input z. If there exists $(z', \mathrm{Z}') \in \mathrm{H}_{\mathrm{LIST}}$ such that $z = z'$ then it returns Z'. Else it picks $\mathrm{Z} \xleftarrow{\$} \{0,1\}^{2\cdot k}$, adds (z, Z) to the list $\mathrm{H}_{\mathrm{LIST}}$ and returns Z. If the oracle has been called by $\mathscr{A}_1$, then $\mathscr{B}$ updates $\mathscr{S} := \mathscr{S} \cup \{z\}$.

Oracle $\mathsf{Gen}(\mathsf{set})$**:** At the l^{th} call, it returns pk_l.

Oracle $\mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pko}_*}(\cdot)$**:** This oracle takes as input a public key pk and a message m. It parses $\mathsf{sst}_* = (s, \widehat{\sigma}_\theta, \widetilde{\sigma}_\theta)$. It picks $\widehat{m} \leftarrow \{0,1\}^{|m|}$ and computes $\widetilde{m} = \widehat{m} \oplus m$. It picks $\widehat{\sigma}_{\theta+1} \xleftarrow{\$} \{0,1\}^k$

and $\tilde{\sigma}_{\theta+1} \xleftarrow{\$} \{0,1\}^k$. Using the random oracles F, G and H, it computes:

$$\begin{aligned} \hat{x} &= (\hat{m}||(\hat{\sigma}_{theta+1} \oplus \mathsf{F}(\hat{\sigma}_\theta))) \\ \tilde{x} &= (\tilde{m}||(\tilde{\sigma}_\theta \oplus \mathsf{F}(\tilde{\sigma}_{theta+1}))) \\ \hat{c} &\leftarrow \mathsf{REnc}_{\mathsf{pk}}(\hat{x}, \mathsf{G}(\hat{\sigma}_\theta)) \\ \tilde{c} &\leftarrow \mathsf{REnc}_{\mathsf{pk}}(\tilde{x}, \mathsf{G}(\tilde{\sigma}_{\theta+1})) \\ d &= (\hat{\sigma}_\theta||\tilde{\sigma}_{\theta+1}) \oplus \mathsf{H}(s||\hat{c}||\tilde{c}) \end{aligned}$$

It updates $\mathsf{sst}_* = (s, \hat{\sigma}_{\theta+1}, \tilde{\sigma}_{\theta+1})$. It returns $c = (\hat{c}, \tilde{c}, d)$.

Oracle $\mathsf{R{-}Ext}^{\mathsf{sst}_*}_{\cdot}(\cdot,\cdot)$**:** It takes as input a public key pko and two ciphertexts c' and c''. It picks $\gamma \xleftarrow{\$} \mathcal{M}$ and $r \xleftarrow{\$} \mathcal{C}$, and runs $\mathrm{K} \leftarrow \mathsf{REnc}_{\mathsf{pko}_*}(\gamma, r)$. It returns K.

$\mathcal{B}$ picks $d \xleftarrow{\$} \{0,1\}$. For all $l \in [\![n+1, n+q]\!]$, it parses $\mathsf{sst}_* = (s, \hat{\sigma}_\theta, \tilde{\sigma}_\theta)$. It picks $\hat{m}_l \leftarrow \{0,1\}^{|m_l|}$ and computes $\tilde{m}_l = \hat{m}_l \oplus m_l$. It picks $\hat{\sigma}_{\theta+1} \xleftarrow{\$} \{0,1\}^k$ and $\tilde{\sigma}_{\theta+1} \xleftarrow{\$} \{0,1\}^k$:

- If $\mathsf{pk}'_l \in \{\mathsf{pk}_{l'}\}_{0 \le l' \le \mu}$, then it computes, using the random oracles F, G and H:

$$\begin{aligned} \hat{x}_l &= (\hat{m}_l||(\hat{\sigma}_{\theta+1} \oplus \mathsf{F}(\hat{\sigma}_\theta))) \\ \tilde{x}_l &= (\tilde{m}_l||(\tilde{\sigma}_\theta \oplus \mathsf{F}(\tilde{\sigma}_{\theta+1}))) \end{aligned}$$

  It sends $\hat{x}_l$ to the oracle $\mathsf{Enc}_{\mathsf{pk}'_l}(\mathsf{LR}_b(\cdot,\cdot), *)$ and receives $\hat{c}_l$, then it sends $\tilde{x}_l$ to the oracle $\mathsf{Enc}_{\mathsf{pk}'_l}(\mathsf{LR}_b(\cdot,\cdot), *)$ and receives $\tilde{c}_l$. It computes $d_l = (\hat{\sigma}_\theta||\tilde{\sigma}_{\theta+1}) \oplus \mathsf{H}(s||\hat{c}_l||\tilde{c}_l)$. It sets $c^*_l = (\hat{c}_l, \tilde{c}_l, d_l)$ and updates $\theta := \theta + 1$.
- Else, it uses the oracle $\mathsf{Enc}^{\mathsf{sst}_*}_{\mathsf{pko}_*}(\cdot)$ on (pk'_l, m_l) and obtains c^*_l.

$\mathcal{B}$ picks $\gamma \xleftarrow{\$} \mathcal{M}$ and $r \xleftarrow{\$} \mathcal{C}$, and runs $\mathrm{K}_{(n+1)\to(n+q)} \leftarrow \mathsf{REnc}_{\mathsf{pko}_*}(\gamma, r)$. $\mathcal{B}$ then runs:

$$b' \leftarrow \mathcal{A}_2(\mathsf{st}, \mathsf{pko}_*, \{c^*_l\}_{n<l\le n+q}, \mathrm{K}_{(n+1)\to(n+q)})$$

While $\mathcal{A}_2$ is running, $\mathcal{B}$ simulates the oracles as for $\mathcal{A}_1$. If there exists $l \in [\![0,\theta]\!]$, $\beta \in \{0,1\}^*$ and $\alpha \in \mathcal{S}$ such that $\alpha = \hat{\sigma}_l$ or $\alpha = \tilde{\sigma}_l$ or $\alpha = s||\beta$, then $\mathcal{B}$ picks $b_* \xleftarrow{\$} \{0,1\}$. Else it sets $b_* = b'$. If $(b'' = b')$ then $\mathcal{B}$ returns $b_* = 0$, else it returns $b_* = 1$.

Analysis: Let the following event be:

$$\mathrm{E} = \text{"}\exists l \in [\![0,\theta]\!] \text{ and } \beta \in \{0,1\}^* \text{ and } \alpha \in \mathcal{S} \text{ such that } \alpha = \hat{\sigma}_l \text{ or } \alpha = \tilde{\sigma}_l \text{ or } \alpha = s||\beta\text{"}$$

We set $\epsilon(k) = \Pr[\mathrm{E}]$. Let q_{RO} be the number of calls to the three random oracles.

The probability that $\mathcal{A}_2$ sends the query $\alpha = s||\beta$ to the random oracles is lower than the probability that it guesses s. Since $\mathcal{A}$ does not know any information about s, this probability is lower than the probability that it guesses s at random in q_{RO} queries. We deduce that:

$$\Pr[\exists \beta \in \{0,1\}^* \text{ and } \alpha \in \mathcal{S} \text{ such that } \alpha = s||\beta] \le \frac{q_{\mathsf{RO}}}{2^k - q_{\mathsf{RO}}}$$

For any $l \in [\![0,\theta]\!]$, since $\mathcal{A}$ does not know any information about $\hat{\sigma}_l$ (resp. $\tilde{\sigma}_l$) then the probability that $\mathcal{A}_2$ sends the query $\hat{\sigma}_l$ (resp. $\tilde{\sigma}_l$) to the random oracles is lower than the probability that it guesses $\hat{\sigma}_l$ (resp. $\tilde{\sigma}_l$) at random in q_{RO} queries. We deduce that:

$$\Pr[\exists \alpha \in \mathcal{S} \text{ such that } \alpha = \hat{\sigma}_l] \le \frac{q_{\mathsf{RO}}}{2^k - q_{\mathsf{RO}}}$$

$$\Pr[\exists \alpha \in \mathcal{S} \text{ such that } \alpha = \tilde{\sigma}_l] \le \frac{q_{\mathsf{RO}}}{2^k - q_{\mathsf{RO}}}$$

Using these equations, we evaluate $\epsilon(k)$.

$$\begin{aligned}
\epsilon(k) &= \Pr[\mathrm{E}] \\
&\leq \Pr[\exists\beta \in \{0,1\}^* \text{ and } \alpha \in \mathscr{S} \text{ such that } \alpha = s||\beta] \\
&\quad + \Pr[\exists l \in [\![0,\theta]\!] \text{ and } \alpha \in \mathscr{S} \text{ such that } \alpha = \widehat{\sigma}_l] \\
&\quad + \Pr[\exists l \in [\![0,\theta]\!] \text{ and } \alpha \in \mathscr{S} \text{ such that } \alpha = \widetilde{\sigma}_l] \\
&\leq \Pr[\exists\beta \in \{0,1\}^* \text{ and } \alpha \in \mathscr{S} \text{ such that } \alpha = s||\beta] \\
&\quad + \sum_{l=0}^{\theta} \Pr[\exists\alpha \in \mathscr{S} \text{ such that } \alpha = \widehat{\sigma}_l] + \sum_{l=0}^{\theta} \Pr[\exists\alpha \in \mathscr{S} \text{ such that } \alpha = \widetilde{\sigma}_l] \\
&\leq \theta \cdot \frac{q_{\mathsf{RO}}}{2^k - q_{\mathsf{RO}}} + \theta \cdot \frac{q_{\mathsf{RO}}}{2^k - q_{\mathsf{RO}}} + \frac{q_{\mathsf{RO}}}{2^k - q_{\mathsf{RO}}} \\
&\leq \frac{(2\cdot\theta + 1)\cdot q_{\mathsf{RO}}}{2^k - q_{\mathsf{RO}}}
\end{aligned}$$

It implies that $\epsilon(k)$ is negligible. We evaluate the probability that $\mathscr{B}$ wins the experiment:

$$\begin{aligned}
\Pr\left[1 \leftarrow \mathsf{Exp}_{\mathsf{RE},\mathscr{B}}^{\mathsf{IND\text{-}CPA}_{(q,\mu+1)}}(k)\right] &= \Pr[b_* = b] \\
&= \Pr[\mathrm{E}] \cdot \Pr[b_* = b|\mathrm{E}] + \Pr[\neg\mathrm{E}] \cdot \Pr[b_* = b|\neg\mathrm{E}] \\
&= \Pr[\mathrm{E}] \cdot \Pr[b_* = b|\mathrm{E}] + (1 - \Pr[\mathrm{E}]) \cdot \Pr[b_* = b|\neg\mathrm{E}] \\
&= \Pr[b_* = b|\neg\mathrm{E}] + \Pr[\mathrm{E}] \cdot (\Pr[b_* = b|\mathrm{E}] - \Pr[b_* = b|\neg\mathrm{E}]) \\
&= \Pr\left[1 \leftarrow \mathsf{Exp}_{\mathsf{GAPO},\mathscr{A}}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}(k)\right] + \epsilon(k) \cdot \left(\frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{Exp}_{\mathsf{GAPO},\mathscr{A}}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}(k)\right]\right)
\end{aligned}$$

We deduce the advantage of $\mathscr{B}$:

$$\begin{aligned}
\mathsf{Adv}_{\mathsf{RE},\mathscr{B}}^{\mathsf{IND\text{-}CPA}_{(q,\mu+1)}}(k) &= \left|\frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{Exp}_{\mathsf{RE},\mathscr{B}}^{\mathsf{IND\text{-}CPA}_{(q,\mu+1)}}(k)\right]\right| \\
&= \left|\frac{1}{2} - \Pr[b_* = b]\right| \\
&\geq \lambda(k) - \epsilon(k) \cdot \lambda(k)
\end{aligned}$$

This advantage is non-negligible, which concules the proof. □

Finally, we have the following theorem.

Theorem 32 *Let k be a security parameter, and let* RE *be a random coin decryptable and IND-CPA$_{t(k),t(k)}$ secure public key encryption scheme for any polynomials t and t'. GAPO instantiated by* RE *is IND-CSPA secure in the random oracle model.*

Proof: Let k be a security parameter, and μ be a positive integer. Let RE be a random coin decryptable and $\mathsf{IND\text{-}CPA}_{t(k),t(k)}$ secure public key encryption scheme for any polynomials t and t'. We instanciate GAPO by RE. Lemma 51 show that $\mathsf{Adv}_{\mathsf{GAPO}}^{\mu\text{-}\mathsf{IND\text{-}CSPA}'}(k)$ is negligible in the random oracle model. We deduce that GAPO instantiated by RE is IND-CSPA secure in the random oracle model using Lemma 50. □

7.4.4 Integrity

We show that GAPO satisfies the integrity property.

Theorem 33 *Let* RE *be a RCD and VK PKE. GAPO instantiated by* RE *satisfies the integrity property.*

Proof: Assume that $\mathsf{Adv}^{\mathsf{Integrity}}_{\mathsf{GAPO}}(k) \neq 0$ then there exists an adversary that breaks the integrity of GAPO with non-null probability. It implies that there exists a setup set and a tuple

$$(\mathrm{N}, \{c_l\}_{1 \leq l \leq \mathrm{N}}, \{\mathsf{pk}_l\}_{1 \leq l \leq \mathrm{N}}, x, \mathsf{sk}_x, i, j, \mathrm{K}_{i \to j})$$

such that there exists a tuple $(m_i, \ldots, m_j)$ such that:

$$\begin{aligned} \{m_l\}_{i \leq l \leq j} = \mathsf{Ope}_{\mathsf{sk}_*}(\mathrm{K}^{\mathsf{pk}_*}_{i \to j}, \{c_l\}_{i \leq l \leq j}, \{\mathsf{pk}_l\}_{i \leq l \leq j}) \\ \mathsf{Ver}(\mathsf{pk}_x, \mathsf{sk}_x) = 1 \\ m_x \neq \mathsf{Dec}_{\mathsf{sk}_x}(\mathrm{C}_x) \end{aligned}$$

We parse $c_l = (\widehat{c}_x || \widetilde{c}_x || d_x)$. There exists $\widehat{\mathrm{R}}$, $\widetilde{\mathrm{R}}$, $\widetilde{m}_x$, $\widehat{m}_x$, $\widetilde{\alpha}$ and $\widehat{\alpha}$ such that:

$$\begin{aligned} m_x = \widetilde{m}_x \oplus \widehat{m}_x \\ (\widehat{m}_x || \widehat{\alpha}) \leftarrow \mathsf{RCDec}_{\widehat{\mathrm{R}}}(\widehat{c}_x, \mathsf{pk}_x) \\ (\widetilde{m}_x || \widetilde{\alpha}) \leftarrow \mathsf{RCDec}_{\widetilde{\mathrm{R}}_x}(\widetilde{c}_x, \mathsf{pk}_x) \\ \widehat{c}_x \leftarrow \mathsf{REnc}_{\mathsf{pk}_x}((\widehat{m}_x || \widehat{\alpha}), \widehat{\mathrm{R}}) \\ \widetilde{c}_x \leftarrow \mathsf{REnc}_{\mathsf{pk}_x}((\widetilde{m}_x || \widetilde{\alpha}), \widetilde{\mathrm{R}}) \end{aligned}$$

Thus, for any $m'_x \leftarrow \mathsf{Dec}_{\mathsf{sk}_x}(c_x)$, we have:

$$\begin{aligned} (\widehat{m}_x || \widehat{\sigma}_x) \leftarrow \mathsf{Dec}_{\mathsf{sk}_x}(\widehat{c}_x) \\ (\widetilde{m}_x || \widetilde{\sigma}_x) \leftarrow \mathsf{Dec}_{\mathsf{sk}_x}(\widetilde{c}_x) \end{aligned}$$

because RE is correct and $(\mathsf{pk}_x, \mathsf{sk}_x)$ is a valid key pair. Finally, $m'_x = \widehat{m}_x \oplus \widetilde{m}_x = m_x$, which contradicts that $m_x \neq \mathsf{Dec}_{\mathsf{sk}_x}(\mathrm{C}_x)$. We conclude that $\mathsf{Adv}^{\mathsf{Integrity}}_{\mathsf{GAPO}}(k) = 0$, thus GAPO instantiated with any verifiable key public key encryption satisfies the integrity property.

□

7.5 Conclusion

We introduce the notion of RCD-PKE. Based on this notion, we propose an *a posteriori* openable PKE (APO-PKE) scheme. Our scheme allows a user to prove his innocence by showing to a judge the content of his encrypted communication with several PKE during a period of time. Our construction preserves the privacy of the others communications, meaning that the judge cannot learn any information concerning the other encrypted messages. Moreover the receivers of the encrypted messages cannot collude in order to learn more information that is contained in the received messages. Our construction is proven secure in the Random Oracle Model and is generic because it only requires RCD-PKE and hash functions.

In the future, we aim at proving that is not possible to have a secure construction that supports several generations of interval key with constant size interval-key and stored data (state). Another future work is to design a security model for chosen-ciphertext security of APO-PKE and to provide a generic construction that achieves this higher security. Finally, it may be interesting to design such a scheme in the standard model.

Chapter 8

Conclusion

In this thesis, we presented several new cryptographic protocols. These protocols have innovative mechanisms for secure delegation of rights. They have practical applications in the protection of sensitive dematerialized data and in the protection of the user's privacy. Each studied primitive and each studied security property was formally modelized. Using these models, we rigorously proved the security properties of our protocols.

Our first contribution was to define security models and protocols for proxy re-proof primitives. This primitive allows a server to transform a proof of the delegator secret knowledge into a proof of the delegate secret knowledge. We showed that this primitive allows a company to manage the access rights of its employees to different services in a practical, simple, efficient and secure way.

We first showed that the security of the only proxy re-proof protocol of the literature is inadequate for practical use. We then formally defined four proxy re-proof families: bidirectional interactive, unidirectional interactive, bidirectional non-interactive, and unidirectional non-interactive. Each of these families gives a different trade-off in terms of efficiency, communication cost and security. We defined five security properties for these primitives: correctness, soundness, validity, zero-knowledge and secret security. Finally, each of these families was instantiated with a concrete protocol.

Our second contribution was about verifiable private polynomial evaluations. We first gave a cryptanalysis of two protocols of the literature. We then showed that the security models of this primitive were not realistic for some applications. Particularly, these models assumed that the polynomial was chosen randomly, which is not always true. We gave an application for this primitive to private evaluations of prediction functions where the security models of literature were insufficient. We defined a security model for verifiable private polynomial evaluation that are more realistic for such applications. In these models, an opponent must distinguish which polynomial is used by the server among two polynomials of his choice. Finally, we designed a scheme that is secure in these models.

We then defined a security model for verifiable ring signatures. In this primitive, any user can sign messages anonymously within a group. Moreover, this user can prove whether he is the signer of a given message. We added the following property to this primitive: a user can also prove that he is not the signer of a given message. We then defined formal security models for verifiable ring signatures. Finally, we designed a scheme based on the decisional Diffie-Hellman assumption.

Our fourth contribution was the design of a sanitizable signature scheme. This scheme is generic and uses verifiable ring signatures. This scheme is unlinkable, which means that it is not possible to link the sanitized signature to the original one. Our scheme is the most efficient of the literature in terms of computation time.

Finally, we defined a primitive called a posteriori openable public key encryption. This primitive allows the user who encrypts the messages to generate an interval key, which allows a delegate to open all messages that was encrypted during a chosen interval of time. We showed that this primitive can be used during a trial to reveal some of the emails sent by the accused without compromising other emails.

Throughout this thesis, we proposed several problems that have not been solved yet. At the end of each chapter, we highlighted these open problems.

First, some protocols proposed in this thesis are in the random oracle model: the non-interactive proxy re-proof protocols, the verifiable private polynomial evaluation protocol, the verifiable ring signature scheme, and the a posteriori openable public key encryption scheme. This security model is not optimal, so we would like, in the future, to design equivalent protocols in the standard model.

The proxies re-proof provide new challenges in the field of proxy re-cryptography. In this thesis, we focused on proofs of knowledge based on number-theory based algorithmic problems. In the future, it would be interesting to design proxies re-proof from proofs of knowledge based on other algorithmic problems. For example, we know how to build proofs of knowledge for any NP problem, hence it would be interesting to show if it is possible to design proxies re-proof that transform a proof of knowledge for some NP problem into another proof for another NP problem.

In the chapter about verifiable private polynomial evaluation, we showed two schemes that are secure in the chosen function attack model. In the first scheme, the size of the verification key and the complexity of the verification algorithm are constant, however, this scheme uses bilinear pairings, which require time-consuming operations. Moreover, this scheme is based on a non-standard assumption. The second scheme does not use pairings and is based on the decisional Diffie-Hellman assumption, however, the size of the verification key and the complexity of the verification algorithm are linear in the degree of the polynomial. We let as an open problem the design of a scheme based on a standard assumption, without pairing and where the size of the keys and the computational complexity are constant.

Another open problem is the design of a scheme that supports the multiple generation of interval keys for a posteriori openable public key encryption. The scheme that we proposed in this thesis allows the user to generate only one key interval. Indeed, if the user generates two interval keys for two disjoint intervals, the opener will be able to decrypt all the messages sent between these two intervals. In the future, we would like to design a scheme that remains secure even if the user generates several interval keys.

Finally, we would like to investigate the design of anonymous proxy re-signature protocols. In this thesis we showed how to build a zero-knowledge proof of knowledge of the discrete logarithm of an element among a set of n elements. We showed how to use such a proof to design ring signature schemes. It would be interesting to design proxy re-proof protocols for this kind of proof of knowledge. These proxies re-proof protocols would be used to transform a proof of knowledge of one element among a set of n elements into a proof of knowledge of one element among another set of m elements for two integers n and m possibly different. Such a proxy re-proof protocol could be used to design ring proxy re-signature protocols: the proxy would transform an anonymous signature within a users group into another anonymous signature within another group. This new primitive could be used in the cases where a user group wants to delegate its ability to sign to another group.

In conclusion, the works presented in this manuscript solve some concrete problems of secure delegation of rights in public key cryptography primitives. Nevertheless, some protocols can still be improved in terms of functionality, security and efficiency. Thereby, this thesis leaves some open problems and future works perspectives in the field of cryptography applied to the delegation of rights.

Bibliography

[ABR98] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHIES: an encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a, 1998. 22

[ACdMT05] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In *ESORICS 2005*. LNCS. Springer, 2005. 116, 117, 119

[AES01] Advanced encryption standard (aes). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, 2001. 2

[AFGH05] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS 2005*. The Internet Society, 2005. 32

[AH05] Giuseppe Ateniese and Susan Hohenberger. Proxy re-signatures: New definitions, algorithms, and applications. In *ACM CCS 05*. ACM Press, 2005. 32, 35, 36

[ASY06] Man Ho Au, Willy Susilo, and Siu-Ming Yiu. Event-oriented k-times revocable-iff-linked group signatures. In *ACISP 06*. LNCS. Springer, 2006. 10, 95, 99

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT 2004*, LNCS. Springer, 2004. 17

[BBD$^+$10] Christina Brzuska, Heike Busch, Oezguer Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In *ACNS 2010*. LNCS. Springer, 2010. 119

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT 2000*, LNCS. Springer, 2000. 21

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO 2007*, LNCS. Springer, 2007. 3

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT'98*, LNCS. Springer, 1998. ix, 32, 35, 36, 37, 38, 67

[BC05] Ian F. Blake and Aldar C-F. Chan. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing. ICDS, IEEE Computer Society Press, 2005. 145

[BDG$^+$17] Xavier Bultel, Manik Lal Das, Hardik Gajera, David Gérault, Matthieu Giraud, and Pascal Lafourcade. Verifiable private polynomial evaluation. In *ProvSec 17*. LNCS. Springer, 2017. 6

[BDZ03] Feng Bao, Robert H. Deng, and HuaFei Zhu. Variations of diffie-hellman problem. In *ICICS 03*, 2003. 17

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO 2001*, LNCS. Springer, 2001. 144

[BFF$^+$09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *PKC 2009*, LNCS. Springer, 2009. 117, 119, 121, 122, 123, 124

[BFLS10] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In *PKC 2010*, LNCS. Springer, 2010. 116, 117, 118, 119

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In CRYPTO 1992 [CRY93]. 23

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *TCC 2006*, LNCS. Springer, 2006. 95

[BL16] Xavier Bultel and Pascal Lafourcade. A posteriori openable public key encryption. In *ICT Systems Security and Privacy Protection*. LNCS. Springer, 2016. 6, 7

[BL17a] Xavier Bultel and Pascal Lafourcade. Unlinkable and strongly accountable sanitizable signatures from verifiable ring signatures. In *CANS 2017*. LNCS. Springer, 2017. 6, 7

[BL17b] Xavier Bultel and Pascal Lafourcade. Zero-knowledge proxy re-identification revisited. Cryptology ePrint Archive, Report 2017/112, 2017. `https://eprint.iacr.org/2017/112`. 6

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, Vol. 17(4), 2004. 35

[Bon98] Dan Boneh. The decision diffie-hellman problem. In *ANTS-III*. LNCS. Springer, 1998. 17

[BPS13] Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. Non-interactive public accountability for sanitizable signatures. In *EuroPKI*, 2013. 119

[BPS14] Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In *EuroPKI 2013*. LNCS. Springer, 2014. 119

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *ASIACRYPT 2012*. LNCS. Springer, 2012. 24, 26

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*. ACM Press, 1993. 19

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC 2011*, LNCS. Springer, 2011. 144

[CDNO97] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *CRYPTO'97*. LNCS. Springer, 1997. 144

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO'94*, LNCS. Springer, 1994. 27

[CHKO06] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Timed-release and key-insulated public key encryption. In *FC 2006*, LNCS. Springer, 2006. 145

[CJ10] Sébastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In *CT-RSA 10*. LNCS. Springer, 2010. 119

[CJL12] Sébastien Canard, Amandine Jambert, and Roch Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AFRICACRYPT 12*. LNCS. Springer, 2012. 119

[CKKC13] S. G. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In *TCC 2013*. LNCS. Springer, 2013. 73

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO 2006*. LNCS. Springer, 2006. 99

[CLQ05] J. Cathalo, B. Libert, and J.-J. Quisquater. Efficient and non-interactive timed-release encryption. In *ICICS 05*. LNCS. Springer, 2005. 145

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In CRYPTO 1992 [CRY93]. 25

[CP08] Sherman S. M. Chow and Raphael C.-W. Phan. Proxy re-signatures in the standard model. In *ISC 2008*, LNCS. Springer, 2008. 32

[CPS14] Sébastien Canard, David Pointcheval, and Olivier Sanders. Efficient delegation of zero-knowledge proofs of knowledge in a pairing-friendly setting. In *PKC 2014*, LNCS. Springer, 2014. 37

[CRR12] Ran Canetti, Ben Riva, and Guy N. Rothblum. Two protocols for delegation of computation. In ICITS 2012 [ICI12]. 73

[CRY93] *CRYPTO'92*, LNCS. Springer, 1993. 168, 169

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, Vol. 33(1), 2003. 22

[CSST06] Sébastien Canard, Berry Schoenmakers, Martijn Stam, and Jacques Traoré. List signature schemes. *Discrete Applied Mathematics*, vol. 154(2), 2006. 10, 95, 99

[CYD06] Z. Changlun, L. Yun, and H. Dequan. A new verifiable ring signature scheme based on nyberg-rueppel scheme. In *Signal Processing 2006*, 2006. 95

[Dan15] Quynh H. Dang. Secure hash standard. In *Federal Inf. Process. Stds. (NIST FIPS)*. NIST Pubs, 2015. 19

[DES77] Data encryption standard. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce, 1977. 2

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, Vol. 22(6), 1976. 2, 16, 17, 28

[DKXY02] Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In *EUROCRYPT 2002*. LNCS. Springer, 2002. 145

[DRS17] David Derler, Sebastian Ramacher, and Daniel Slamanig. Homomorphic proxy re-authenticators and applications to verifiable multi-user data aggregation. Cryptology ePrint Archive, Report 2017/086, 2017. `http://eprint.iacr.org/2017/086.pdf`. 38

[DT07] Alexander W. Dent and Qiang Tang. Revisiting the security model for timed-release encryption with pre-open capability. In *ISC 2007*, LNCS. Springer, 2007. 145

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, Vol. 31, 1985. 22

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*. IEEE Computer Society Press, 1987. 73, 86

[FG12] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *ACM CCS 12*. ACM Press, 2012. 73

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC 2005*. LNCS. Springer, 2005. 74

[FKI06] Jun Furukawa, Kaoru Kurosawa, and Hideki Imai. An efficient compiler from sigma-protocol to 2-move deniable zero-knowledge. In *ICALP 2006, Part II*, LNCS. Springer, 2006. 25

[FKM$^+$16] N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *PKC 2016*. LNCS. Springer, 2016. xi, 119, 120, 137, 138

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT 04*. LNCS. Springer, 2004. 74

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, Vol. 26(1), 2013. 22

[FP08] Georg Fuchsbauer and David Pointcheval. Anonymous proxy signatures. In *SCN 08*. LNCS. Springer, 2008. 119

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, LNCS. Springer, 1987. 26

[GA07] Matthew Green and Giuseppe Ateniese. Identity-based proxy re-encryption. In *ACNS 07*, LNCS. Springer, 2007. 32

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*. ACM Press, 2009. 3

[GFLL15] Linke Guo, Yuguang Fang, Ming Li, and Pan Li. Verifiable privacy-preserving monitoring for cloud-assisted mhealth systems. In *INFOCOM*. IEEE, 2015. vi, 69, 71, 72, 73, 74, 75, 76, 77, 79, 91, 92

[GGP10] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO 2010*. LNCS. Springer, 2010. 73

[GH08] David Galindoa and Javier Herranz. On the security of public key cryptosystems with a double decryption mechanism. *Information Processing Letters*, 108(5), November 2008. 21

[GHV08] Steven D. Galbraith, Florian Hess, and Frederik Vercauteren. Aspects of pairing inversion. In *Information Theory, IEEE Transactions*, volume Vol. 54. IEEE, 2008. 18

[GKP$^+$13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich:. How to run turing machines on encrypted data. In *CRYPTO 2013*. LNCS. Springer, 2013. 144

[GKP$^+$13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich:. Reusable garbled circuits and succinct functional encryption. In *45th ACM STOC*. ACM Press, 2013. 144

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, Vol. 28(2), 1984. 20

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, Vol. 17(2), 1988. 28

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, Vol. 18(1), 1989. 4, 23, 24

[GND16] Hardik Gajera, Shruti Naik, and Manik Lal Das. On the security of "verifiable privacy-preserving monitoring for cloud-assisted mhealth systems". In *ICISS*. LNCS. Springer, 2016. vi, 69, 71, 72, 73, 74, 77, 79, 91, 92

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, UK, 2001. 13

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004. 13

[GPR98] Oded Goldreich, Birgit Pfitzmann, and Ronald L. Rivest. Self-delegation with controlled propagation - or - what if you lose your laptop. In *CRYPTO'98*, LNCS. Springer, 1998. 145

[HKS10] Fumitaka Hoshino, Tetsutaro Kobayashi, and Koutarou Suzuki. Anonymizable signature and its construction from pairings. In *Pairing 2010*. LNCS. Springer, 2010. 99

[HW10] Goichiro Hanaoka and Jian Weng. Generic constructions of parallel key-insulated encryption. In *SCN'10*, 2010. 145

[HYL05] Y. H. Hwang, D. H. Yum, and P. J. Lee. Timed-release encryption with pre-open capability and its application to certified e-mail system. In *ISC 2005*. LNCS. Springer, 2005. 145

[ICI12] *ICITS 12*, LNCS. Springer, 2012. 169, 171

[ID03] Anca Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *NDSS 2003*. The Internet Society, 2003. 32, 35

[JMSW02] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA 02*. LNCS. Springer, 2002. 119

[Jou04] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, Vol. 17(4), 2004. 17

[JSMM13] Hoda Jannati1, Mahmoud Salmasizadeh, Amir Javad Mohajeri, and Amir Moradi. Introducing proxy zero-knowledge proof and utilization in anonymous credential systems. In *SCN 2013*. LNCS. Springer, 2013. 38

[Kaf06] Franz Kafka. Der prozess. In *Der Prozess*. Anaconda Verlag, 2006. 116

[KKK08] Marek Klonowski, Przemysław Kubiak, and Mirosław Kutyłowski. Practical deniable encryption. In *SOFSEM 2008*. LNCS. Springer, 2008. 144

[KMR12] Marcel Keller, Gert Læssøe Mikkelsen, and Andy Rupp. Efficient threshold zero-knowledge with applications to user-centric protocols. In ICITS 2012 [ICI12]. 37

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO 2004*, LNCS. Springer, 2004. 37, 39

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*. LNCS. Springer, 2010. ix, 70, 71, 73, 74, 77, 78, 84, 86, 90, 91, 92

[LHC10] Song Luo, Jian-bin Hu, and Zhong Chen. Ciphertext policy attribute-based proxy re-encryption. In *ICICS 10*, LNCS. Springer, 2010. 32

[LP02] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, Vol. 15(3), 2002. 74

[LQY07] Benoît Libert, Jean-Jacques Quisquater, and Moti Yung. Parallel key-insulated public key encryption without random oracles. In *PKC 2007*, LNCS. Springer, 2007. 145

[LV08a] Benoît Libert and Damien Vergnaud. Multi-use unidirectional proxy re-signatures. In *ACM CCS 08*. ACM Press, 2008. 32

[LV08b] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *PKC 2008*, LNCS. Springer, 2008. 32

[LW03] Jiqiang Lv and Xinmei Wang. Verifiable ring signature. In *CANS'03*. DMS Proceedings, 2003. 94, 95

[LWH05] Kuo-Chang Lee Lee, Hsiang-An Wen, and Tzonelih Hwang. Convertible ring signature. *IEEE Proceedings - Communications*, Vol. 152(4), Aug 2005. 95

[LZCS16] Russell W. F. Lai, Tao Zhang, Sherman S. M. Chow, and Dominique Schröder. Efficient sanitizable signatures without random oracles. In *ESORICS 2016*. LNCS. Springer, 2016. 119

[May93] T. May. Time-release crypto. Manuscript, 1993. 145

[MRK03] Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *44th FOCS*. IEEE Computer Society Press, 2003. 71

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC '99*. ACM, 1999. 73, 74

[OO90] Tatsuaki Okamoto and Kazuo Ohta. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In *EUROCRYPT'89*, LNCS. Springer, 1990. 37, 39

[PHGR13] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE S&P*. IEEE, 2013. 73

[Pol78] John M. Pollard. A monte carlo method for index computation (mod p). In *Mathematics of Computation*, volume Vol. 32. LNCS. Springer, 1978. 74, 76

[PQ10] Kenneth G. Paterson and Elizabeth A. Quaglia. Time-specific encryption. In *SCN'10*, 2010. 145

[PRV12] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC 2012*. LNCS. Springer, 2012. 73

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *EUROCRYPT'96*, LNCS. Springer, 1996. 29

[PST13] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *TCC 2013*. LNCS. Springer, 2013. 73

[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*, Vol. 21(2), 1978. 2

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT 2001*, LNCS. Springer, 2001. 94, 95

[SBZ02] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In *ICISC 2001*. LNCS. Springer, 2002. 116

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, LNCS. Springer, 1990. 25, 29

[SFZ$^+$10] Jun Shao, Min Feng, Bin Zhu, Zhenfu Cao, and Peng Liu. The security model of unidirectional proxy re-signature with private re-signature key. In *ACISP 10*, LNCS. Springer, 2010. 32

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, Vol. 22(11), 1979. 26, 86

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/2004/332`. 22

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In *EUROCRYPT 2005*, LNCS. Springer, 2005. 144

[TFS04] Isamu Teranishi, Jun Furukawa, and Kazue Sako. k-times anonymous authentication (extended abstract). LNCS. Springer, 2004. 92

[WMZW11] Shangping Wang, Rui Ma, Yaling Zhang, and Xiaofeng Wang. Ring signature scheme based on multivariate public key cryptosystems. *Computers and Mathematics with Applications*, vol. 62(10), 2011. 95

[WT99] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *USENIX Security Symposium*. USENIX Association, 1999. 142