Università degli Studi di Catania

Dipartimento di Matematica ed Informatica
Dottorato di Ricerca in Matematica Pura ed Applicata XXVI ciclo

Orazio Puglisi

Authenticating Computation on Groups: New Homomorphic Primitives and Applications

Advisor:
Ch.mo Prof. Dario Catalano

ANNO ACCADEMICO 2013-2014

Contents

Acknowledgments

First of all I want to thank my Ph.D. advisor Prof. Dario Catalano. He was like a mentor for my studies. I met him during my master degree in math and he impressed me with the cryptographic world since I had never known it before. Every talk with him inspired my studies, was a relevant opportunity to my personal and professional growth, and helped me to be now ready to discuss this thesis. For all these and other reasons I am very thankful to him.
Moreover I want to thank Prof. Riccardo Re for introducing me in the cryptographic science during my bachelor degree in his course of "Teoria dei Codici e Crittografia", and thank to him because he introduced me to Dario.
During my Ph.D. I spent one year at the University of Californina in San Diedo (UCSD). I want to thank all people I knew during this amazing experience, in particular Prof. Daniele Micciancio. It was an honor to know him and take his class.
I gratefully acknowledge the funding sources that made my Ph.D. studies possible. In particular I want to thank my Ph.D. program chair Prof. Giovanni Russo, and the Department chair Prof. Giuseppe Mulone.
My time in Catania was enriched by others friend. I want to thank each of them, in particular Antonio, co-author of the paper on which is based this thesis.
Lastly, I would like to thank my life partner Cecilia, my brother, my family and all my friends for supporting and encouraging me during this experience.

Ad Maiora!

Chapter 1

Introduction

During the last 10 years we have seen a huge development of many on-line activities: looking at internet web pages, sharing personal information on social networks, making payments on online store, or connecting to the network through personal mobile phones. All these activities have become standard daily activities.
Recently network security has become more popular but, in the mean time, even more breaks are found on big company servers and personal data are revealed by anonymous hackers (very recently there were two very big attacks: one on the Apple iCloud and one on the JPMorgan Servers).
A possible reason of this is that the very fast expansion of Internet, and the growing demand for specific service by users, don't go together with the actual security offer.
In this context the role of cryptography is to provide the appropriate tools in response to the user's network security safety request.

1.1 A few words about cryptography history

The original, and also historical, purpose of cryptography was to keep secrets in communications.
First applications of cryptography can be found in military and government service. The Cesar's cipher is an example of what is called *"Classical Cryp-*

tography". It can be easily described by a shifting of the alphabetical letters by some number of positions. For several centuries, cryptography was related only to military, government or diplomatic applications, and its only purpose, as said before, was to preserve confidentiality of communications.
Classical cryptography was followed by *Modern Cryptography* which provide not just secrecy, but other services too like *Authenticity*, *Non Repudiation*, *Data Integrity*[1]
It's commonly accepted that Modern Cryptography starts whit the paper *"Communication Theory of Secrecy Systems"* of Shannon in 1949 [71].
In this paper Shannon looks at cryptography from an information theory point of view and introduces formally the concept of perfect security.
In this scenario the request of security growth and cryptographic research was related not only for government applications, but for private users too. In this period cryptography become a science.
1976 was the year of the milestone of modern cryptography. W. Diffie and M. Hellman in this year publish the paper *New Direction in Cryptography* [33]. There they introduced the concept of public key encryption and propose a new key exchange mechanism to share the same key using a public channel. Security of this construction is based on the hardness of the Discrete Logarithmic problem (that will be defined in the next chapter).
We will focus in chapter 2 on public key encryption. Here we give just the basic idea.
In secret key cryptography users need to share the same secret key to communicate secretly, .
In public key cryptography, instead, there are two keys. The public key, revelated to everyone, allow users to communicate whit the owner of the associate secret key. The secret key must be keep private by the owner which use this key to decrypt the conversation.

[1] formal definition of those cryptographic goals can be found in chapter 2

1.2 From Encryption to Homomorphic Encryption

The work of Diffie and Helmann was followed by many other contributions in public key setting. One of them is the work of Rivest, Adleman and Shamir "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*" [67] in 1978. The encryption scheme there described, today is known as RSA. In 1979 Rivest, Adleman ad Dertouzos in "*On Data Bank and privacy homomorphism*"[65] analyze the problem of construct a cryptographic primitive in which ciphertext can be manipulated, so that can be possible to execute arbitrary functions on them. They call a such property *privacy homomorphism*. They also propose some constructions, which unfortunately have been proven to be insecure [21]

Homomorphic encryption has many useful applications in different areas. For example it enables private queries to search engine, search over encrypted data, multiparty computation, NIZK proofs. In 1996 D. Boneh and R. Lipton in [16] have proven that every deterministic algebraic homomorphic scheme[2] is not secure.

After this negative result many researcher started to think that fully homomorphic encryption might be impossible to achieve. Surprisingly in 2009, C. Gentry (a PhD student at the Stanford University) show in [44] and later in [45] the first fully homomorphic encryption scheme.

This result solves the Rivest, Adleman and Dertouzos problem after more than 30 years and now is consider to be an important milestone in cryptographic history.

After this result homomorphic encryption become one of the mostly investigated cryptographic topic, and many papers propose different versions of homomorphic encryption schemes [73, 20, 31, 47, 72], mostly of them following the same Gentry's blueprint. Many of those constructions have a fairly poor performance.

Recently Gentry and Halevi in [46], and Brakerski and Vaikuntanathan in

[2] An encryption scheme is said to be algebraic homomorphic if from $E(x_1)$ and $E(x_2)$ is possible to compute $E(x_1 + x_2)$ and $E(x_1 \cdot x_2)$, where E is the encryption algorithm and x_1, x_2 two messages.

[19] found a different way to obtain FHE. Following [19] Brakerski, Gentry and Vaikuntanathan in [18] present a new FHE scheme achieving better performance than the previous scheme. This last scheme was later implemented by S. Halevi and V. Shoup in [53]. There they propose the new C++ library, HElib, that is the best optimized open source implementation of [18].

1.3 What about homomorphic signatures?

Cryptography, as said before, is not just encryption. Another important primitive is digital signature. Very briefly, signatures allows users to be convinced of authenticity of a digital message.
As for encryption scheme is possible to define homomorphic signatures schemes. The concept of homomorphic signature scheme was originally introduced in 1992 by Desmedt [32],and then refined by Johnson, Molnar, Song, Wagner in 2002 [55].
Very informally homomorphic signatures allow user to validate computation over authenticated data[3].
Regard homomorphic signatures, it's possible, and useful, define a weak homomorphic property called linear homomorphism[13][4]. That is, instead to validate a generic computation f over authenticated data, here f must be a linear function. Following [13] many other works further explored the notion of homomorphic signatures by proposing new frameworks and realizations [42, 9, 15, 14, 27, 10, 28, 38, 11, 26]. In the symmetric setting constructions of homomorphic message authentication codes have been proposed by [13, 43, 25].

Recently Libert *et al.* [61] introduced and realized the notion of *Linearly Homomorphic Structure Preserving signatures* (LHSPS for short). Structure Preserving cryptography provides a simple and elegant methodology to com-

[3]That is a signer holding a dataset $\{m_i\}_{i=1,\ldots,t}$ can produce corresponding signatures $\sigma_i = \mathbf{Sign}(\mathrm{sk}, m_i)$ and store the signed dataset can on a remote server. Later the server can (publicly) compute $m = f(m_1, \ldots, m_t)$ together with a (succinct) valid signature σ on it.

[4]See chapter 3 for a very useful application of linearly homomorphic signatures to prevent pollution attacks in network coding

pose algebraic tools within the framework of Groth Sahai proof systems [52]. In the last years, this methodology has been widely used to design simple and modular realizations of cryptographic protocols and primitives. These include structure preserving signatures (SPS) [5, 3, 4, 1, 2, 22, 29, 30, 39, 51, 54], commitments [50, 6] and encryption schemes [23].

Informally LHSPS are like ordinary SPS but they come equipped with a linearly homomorphic property that makes them interesting even beyond their usage within the Groth Sahai framework. In particular Libert *et al.* showed that LHSPS can be used to enable simple verifiable computation mechanisms on encrypted data. More surprisingly, they observed that linearly homomorphic SPS (generically) yield efficient simulation sound trapdoor commitment schemes [40], which in turn imply non malleable trapdoor commitments [34] to group elements.

In chapter 3 we will define a new primitive called *Linearly homomorphic signature scheme to sign element in bilinear groups* (LHSG for short) and propose a very simple construction of it which RMA-security[5] will based on a variant of the Computational Diffie-Hellman assumption introduced by Kunz-Pointcheval in [60].

The main difference between LHSG and LHSPS is that the first allow for more flexibility, as the signature is explicit allowed to contain components which are not group elements.

However, as explained in remark 4, if it's possible set the file identifier fid as a group element then our LHSG will be an LHSPS.

As an interesting application we will show how this simple primitive can be use in the context of OnLine/OffLine (linearly homomorphic) signatures. Very informally, on-line/off-line signatures allow to split the cost of signing in two phases. An (expensive) offline phase that can be carried out *without* needing to know the message m to be signed and a much more efficient online phase that is done once m becomes available.

In this sense, on-line/off-line homomorphic signature could bring similar efficiency benefits to protocols relying on homomorphic signatures. For instance,

[5]Later we will discuss about different kind of possible attack to a signature scheme. The Random Message Attack is one of them and will be carefully described in section 2.2.4.4

they could be used to improve the overall efficiency of linear network coding routing mechanisms employing homomorphic signatures to fight pollution attacks[6].
We show that RMA-secure LHSG naturally fit this more demanding on-line/off-line scenario. Specifically, we prove that if one combines a RMA-secure LHSG with (vector) Σ protocols with some specific homomorphic properties, one gets a fully fledged linearly homomorphic signature achieving a very efficient online phase. Moreover, since the resulting signature scheme supports vectors of arbitrary dimensions as underlying message space, our results readily generalize to the case of network coding signatures [13].
Finally we present a generic methodology to convert an LHSG secure under RMA attacks into ones CMA-secure. This transformation is totally generic and can be use for linearly homomorphic signatures in general.

1.4 From a concrete problem to a new primitive

Lets now consider the follows scenario. Suppose that a teacher wants to store the grades of homework of his students on a cloud server. A possible way can be as follows: he creates a file identifier fid for each class, and set a generic record as (fid, STUD_ID,GRADE), where STUD_ID is a student identifier. Then he signs each record and stores everything off line. This solution have two problems. First, since data are stored in clear, outsourcing them off line might violated the privacy of students. Second, suppose that the teacher want to compute a linear function on the grades previous stored, how do that without download all data locally?
To solve those two problems we search for a primitive such that the following properties are satisfied:

Privacy: to assure student grades privacy.

[6]This is because the sender could preprocess many off-line computations at night or when the network traffic is low and then use the efficient online signing procedure to perform better when the traffic is high.

Authenticity: to assure that nobody can modify a record.

Homomorphicity: to allow evaluate and validate, *using just a public key*, computations over encrypted (and authenticated) data.

To do that we define in chapter 4 a new primitives that we call *Linearly Homomorphic Authenticated Encryption with Public Verifiability* (LAEPUV for short). Informally, this primitive allows to authenticate computation on (outsourced) encrypted data, with the additional benefit that the correctness of the computation can be publicly verified.
We will show an efficient realization of this primitive supporting Paillier's ciphertext [64]. It works by combining Paillier's encryption scheme with some appropriate additively homomorphic signature scheme. The main idea is as follows. One first encrypts the message in a ciphertext C. Then decrypts a "masking" of C and signs this masked plaintext using a linearly homomorphic signature. In all we validate the computation on the ciphertext by basically authenticating computations on the masked plaintext.
So, in this way, we will obtain the first simple and efficient LAEPuV based on Paillier cryptosystem.
Previous (efficient) solutions for this problem rely on linearly homomorphic structure preserving signatures [61] and, as such, only supported cryptosystem defined over pairing-friendly groups. Since, no (linearly homomorphic) encryption scheme supporting exponentially large message spaces is known to exist in such groups, our construction appears to be the first one achieving this level of flexibility.
On the negative side, our construction is proved secure in the random oracle model.
Another result provided by us in this area is a generalization of our scheme. In particular, we will show how to replace Paillier encryption scheme with any other encryption scheme which had some well defined homomorphic properties.
Interestingly, this includes many well known linearly homomorphic encryption schemes such as [49, 63, 57].
An important feature of LAEPuV is that they allow for public verifiability. This means that everybody can verify correctness of the computation, even

though the possibility of decrypting the result remains feasible only for the holder of the secret decryption key. In the context of verifiable computation on encrypted data, we believe that public verifiability is important for (at least) two reasons. First, it implies that learning whether or not a given computation has been carried out correctly or not does not compromise security in any way. This is in sharp contrast with privately verifiable schemes where revealing this same information *might* create security concerns (and this was actually the case, for instance, for the verifiable computation scheme from [41], see [41] for details). Second, in case of dispute, public verifiability allows third parties to determine whether or not the computation was performed correctly, without requiring the client to disclose his/her private key. This means, once again, that the privacy of the underlying data remains preserved when settling such disputes.

1.5 Organization of this thesis

This thesis is divided in 5 chapters organized as follows:

Chapter 2: Preliminaries and notations. This chapter provides the reader all basic cryptographic tools so that it can be possible to understand all the results of this thesis.

Chapter 3: Linearly homomorphic signatures scheme to sign elements in bilinear groups. In this chapter we provide formal definition for Linearly homomorphic signatures and introduce the new concept of Linearly homomorphic signature scheme to sign elements in bilinear groups (LHSG).

Another goal of this chapter is to achieve a general conversion method for LHSG from Random Message Security to Chosen Message Security.

Chapter 4: (Publicly) Verifiable delegation of computation on outsourced ciphertexts. In this chapter, we first describe a new primitive that we call Linearly Homomorphic Authenticated Encryption with Public Verifiability (LAEPuV). Next we provide two instantiations of this primitive.

Chapter 5: Applications to On-Line/Off-Line Linearly Homomorphic signatures. In this chapter we formally provide definition for On-Line/Off-Line Linearly Homomorphic signatures and for vector Σ protocols. Then we show how to combine a RMA-secure LHSG with a vector Σ protocol to construct an On-Line/Off-Line Linearly Homomorphic signatures

Chapter 6: Conclusions It contains a summary of the main results of this thesis and some open problems

Chapter 2

Preliminaries and notations

2.1 Basic Notations

If $n \in \mathbb{N}$, we denote by 1^n the unary vector of dimension n. We denote by $\{0,1\}^n$ the set of n-bit dimension string (or vector), by $\{0,1\}^*$ the set of binary string without any limitation on the dimension and by $[n]$ the set $\{1,\ldots,n\}$.
If x and y are two arbitrary binary string we denote by $x\|y$ the concatenation of the strings.
If x is a binary string we denote by $|x|$ its length and by $[x]_i$ its i-th bit.

2.1.1 Probabilistic notation

In this paper we use standard probabilistic notation well known in cryptography. Let A be a non empty finite set. We denote by $a \underset{X}{\leftarrow} A$ the procedure consisting of sampling an element a from the set A according to a probability distribution X. When X is omitted we'll refer to the uniform distribution.
An algorithm $\mathcal{A}$ is said to be PPT if it's modeled as a probabilistic Turing machine that runs in polynomial time in its inputs.
A function f is said to be negligible if for all polynomial p there exists $n_0 \in \mathbb{N}$ such that for each $n > n_0$

$$|f(n)| < \frac{1}{p(n)}.$$

Given two algorithms M and A we denote by $M^A(x)$ the output of the algorithm M on input x, when given oracle access[1] to A.

2.1.2 Number Theory

We denote with $\mathbb{N}$ the set of natural numbers, with $\mathbb{Z}$ the set of integers, with $\mathbb{Z}_p$ the set of integers modulo p, with $\mathbb{Z}_p^*$ the multiplicative group of invertible integers modulo p .
Let $\mathbb{G}$ be a group.

Definition 1 *The number of elements of $\mathbb{G}$, denoted by $|\mathbb{G}|$ is called the order of $\mathbb{G}$. We said that a group $\mathbb{G}$ is finite iff $|\mathbb{G}|$ is finite.*

Definition 2 *We say that an element $g \in \mathbb{G}$ is a generator of $\mathbb{G}$ if $\forall a \in \mathbb{G}\ \exists i \in \mathbb{N}$ such that $a = g^i$.*

2.1.3 Pairings

Definition 3 *Let P denote a generator of $\mathbb{G}_1$, where $\mathbb{G}_1$ is a group of prime order p, and let $\mathbb{G}_T$ another multiplicative group such that $|\mathbb{G}_1| = |\mathbb{G}_T|$. A pairing is a map*

$$e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$$

such that the following properties are true:

- *The map e is bilinear: Given $Q, W, Z \in G_1$ we have*

  $e(Q, W \cdot Z) = e(Q, W) \cdot e(Q, Z)$ *and* $e(Q \cdot Z, W) = e(Q, W) \cdot e(Z, W)$.

  So, for any $a, b \in \mathbb{Z}_q$:

  $$e(Q^a, W^b) = e(Q, W)^{ab} = e(Q^{ab}, W) = e(Q, W^{ab})$$

- *The map e is non-degenerate: $e(P, P) \neq 1_{G_T}$ where $1_{\mathbb{G}_T}$ is the identity element of $\mathbb{G}_T$.*

- *The map e is efficiently computable.*

[1] That is, M can ask for the result of $A(x)$ using a black box access to the algorithm A

Remark 1 *It's very easy to verify that such map is symmetric, that is* $e(Q, W) = e(W, Q)$ *for all* $Q, W \in \mathbb{G}_1$.

Remark 2 *The previous definition can be trivially adapted to the case when* $\mathbb{G}_1$ *is an additive group (for example the group of points on an elliptic curve over a finite field.)*

2.1.4 Computational assumptions

Most of cryptographic primitives are based on number theoretical problems. Here we recall a few computational assumptions used to prove the security of our constructions.
Let $\mathbb{G}$ be a finite (multiplicative) group of prime order p.

Definition 4 (DL) *We say that the discrete logarithm assumption hold in* $\mathbb{G}$ *if, given a random generator* $g \in \mathbb{G}$*, there exist no PPT adversary* $\mathcal{A}$ *that on input* (g, g^x) *outputs* x *with more than negligible probability. Here the probability is taken on the uniform choice of* x *and the internal coin tosses of* $\mathcal{A}$.

Definition 5 (CDH) *We say that the Computational Diffie-Hellman assumption holds in* $\mathbb{G}$ *if, given a random generator* $g \in \mathbb{G}$*, there exists no PPT* $\mathcal{A}$ *that on input* g, g^a, g^b *outputs* g^{ab} *with more than negligible probability. Here the probability is taken over the uniform choices of* $a, b \xleftarrow{\$} \mathbb{Z}_p$ *and the internal coin tosses of* $\mathcal{A}$.

The CDH problem, introduced by Diffie and Hellman in [33] is trivially related to de Discrete Logarithm (DL) problem. It's easy to prove that using a DL oracle we can solve the CDH problem. So CDH is at least as strong as DL hardness assumption. In some groups [37], [62] it's possible to prove the viceversa, and so these the two assumption are equivalent, but in general proving equivalence of these two problems is an open question.

The 2-out-of-3 Computational Diffie-Hellman assumption was introduced by Kunz-Jacques and Pointcheval in [59] as a relaxation of the standard CDH assumption. It is defined as follows.

Definition 6 (2-3CDH) *We say that the 2-out-of-3 Computational Diffie-Hellmann assumption holds in $\mathbb{G}$ if, given a random generator $g \in \mathbb{G}$, there exists no PPT $\mathcal{A}$ that on input (g, g^a, g^b) (for random $a, b \xleftarrow{\$} \mathbb{Z}_p$) outputs h, h^{ab} $(h \neq 1)$ with more than negligible probability.*

It's simple to prove that 2-out-of-3 CDH is at most as difficult as CDH.
All the previous computational problems are strictly related to the first problem, the DL problem. Another class of computational problems, instead, are based on a different problem, the factorization problem. The Decisional Composite Residuosity Assumption, described below, was introduced by Paillier in [64] and it's possible to prove that solving this problem is more difficult than factoring.

Definition 7 (DCRA) *We say that the Decisional composite residuosity assumption (DCRA) holds if there exists no PPT $\mathcal{A}$ that can distinguish between a random element from $\mathbb{Z}^*_{N^2}$ and one from the set $\{z^N | z \in \mathbb{Z}^*_{N^2}\}$ (i.e. the set of the N-th residues modulo N^2), when N is the product of two random primes of proper size.*

2.2 Primitives and Security

2.2.1 Users and primitives

Users and primitives are the two main character in cryptography. Users, sometimes called players, are algorithms which use cryptographic primitives to achieve a fixed object: i.e. to secretly share some information they use a specific primitive called Asymmetric Encryption scheme, but they can't use the same primitive to achieve others object (for example to prove the authenticity of a message). Other commonly used primitives are:

- One Way (Hash) Function
- Symmetric Encryption (called also Private Key Encryption)
- Digital signatures

A detailed explanation of such primitives can be found in the next section. However, in concrete instantiations, it's necessary to achieve more than one of the security property guaranteed by each one of those primitives.
There are two possible solutions: combine different cryptographic primitives to make a security protocol, or define new primitives (with related security definitions) matching the fixed problem.
The first method, though it may seem simpler, is affected by errors due to the way the primitives are implemented, combined and used.
In chapter 4 of this thesis we'll follow the second approach by defining a new primitive, LAEPuV, described very informally in the introduction. Before starting a detailed explanation of the primitives used in this thesis, we want to focus on the definition of security.
Cryptography and security are two very related concept. Indeed we often use the word "secure" in the introduction, but we did not explain what this formally means.
Let us start by considering a specific primitive, i.e. private key encryption. Informally it consists of 3 algorithms: **KeyGen**, **Enc**, **Dec**. **KeyGen** produces a secret key sk, **Enc** produces an encryption for a message m given as input using the secret key sk, **Dec** decrypts a ciphertext c given as input using the secret key sk.
Suppose that two users (Alice and Bob), use such primitive to have a "secure" conversation. Supposing they shared the same private key sk, Alice sends to Bob an encryption c of a message m through a public channel. What's security in this contest? Suppose there exists a user (called Adversary) able to capture any ciphertext that flows on the channel between the two parties. Thus he can collect a set of ciphertext and try to extract some information from them. For example, if there exists an adversary able to extract the secret key sk after seeing a polynomial number of ciphertext, then the scheme used by Alice and Bob is trivially insecure. But it's not the only case! If an adversary $\mathcal{A}$ can recover just a single bit of the message m from its ciphertext c without knowing the key sk then the scheme should also be considered insecure. Indeed the specific bit recovered by $\mathcal{A}$ contains information about the message.
Overall there are some information that are known from any user and adver-

sary: for example the dimension of the message space, the dimension of the key, the language used in the conversation and sometimes the main topic of the conversation.
So we desire a definition which captures security in the stronger possible sense, as described very informally above.
Now there are two formalization of security: perfect security and computational (or provable) security.
The first is a very strong notion, introduced by Shannon [71] based on information theory . In this notion even when adversary has an unlimited computing power, ciphertext must provide no information about the plaintext, beyond the *a priori* information it had before seeing the ciphertext.
Unfortunately, achieving perfect security implies a key as long as the message and this is a very bad limitation.
In 1982 Goldwasser and Micali in [48] introduced a new notion of security, called semantic security, but from this definition is very difficult to prove security of any scheme. So, some year later in [49] they prove that this definition is equivalent to another practical definition called computational (or indistinguishability) security.
In the next subsection we'll formalize this security notion providing formal definitions.

2.2.2 Indistinguishability under CPA and CCA

Let $SE = (\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ a symmetric (or private key) encryption scheme, and let $\mathcal{A}$ an adversary. Suppose he has access to an oracle $\mathcal{O}$. This oracle receives in input two message M_0 and M_1 of the same length and returns a ciphertext c. c can be computed in two ways, corresponding to two possible worlds in which adversary "lives" (world-0, world-1).
In the world-0 $\mathcal{A}$ receives an encryption on the message M_0, In the world-1 $\mathcal{A}$ receives an encryption on the message M_1
He can ask a polynomial number of such couples (M_0, M_1). Finally $\mathcal{A}$ outputs a bit: 0 if he thinks to be in the world 0, 1 if he thinks to be in the world 1. Formally we can describe this game through the following experiments, where K is the key space:

$\mathbf{Exp}_{SE}^{IND-CPA-1}(\mathcal{A})$	$\mathbf{Exp}_{SE}^{IND-CPA-0}(\mathcal{A})$
$k \xleftarrow{\$} K$	$k \xleftarrow{\$} K$
$d \leftarrow \mathcal{A}^{\mathbf{Enc}_k(LR(\cdot,\cdot,1))}$	$d \leftarrow \mathcal{A}^{\mathbf{Enc}_k(LR(\cdot,\cdot,0))}$
return d	return d

where the function $LR(\cdot, \cdot, b)$ takes as input two message m_0, m_1 and a bit b and outputs the message m_b. Finally we define the IND-CPA advantage of $\mathcal{A}$ as

$$Adv_{SE}^{IND-CPA}(\mathcal{A}) = Pr\left[\mathbf{Exp}_{SE}^{IND-CPA-1}(\mathcal{A}) = 1\right] - Pr\left[\mathbf{Exp}_{SE}^{IND-CPA-0}(\mathcal{A}) = 1\right]$$

The advantage is a number in $[0, 1]$. If it is large (close to 1), it means that $\mathcal{A}$ is able to distinguish the difference of the worlds, so SE is not secure.
If it's close to zero, than it means that $\mathcal{A}$ is outputting 1 about as often in world-0 as in world-1.
Clearly if for a specific adversary $\mathcal{A}$ its advantage is close to zero, it's not really true that SE is secure. Indeed it's possible that exist another unknown adversary $\mathcal{B}$ such that $Adv_{SE}^{IND-CPA}(\mathcal{B}) \approx 1$.
So, to be SE secure, it must be that $Adv_{SE}^{IND-CPA}(\mathcal{A}) \approx 0$ for all PPT adversary $\mathcal{A}$.

2.2.3 The asymptotic approach

Throughout this thesis we will adopt the computational security model, using the asymptotic approach to define the adversary's advantage, success probability, running time and moreover.
More formally, the keys generation algorithm of each primitive will take as input a security parameter λ. Each algorithm, and adversaries too, will be PPT, so, for some constants a, c the algorithm will run in time $a\lambda^c$. However the adversary, although requested to run in polynomial time, may be much more powerful than the honest parties, like in the real world[2].

[2]For example, an user to login in its bank account can perform all operations using his mobile phone, which have a very limited computational power. However bad organizations can perform attacks using very expensive hardware with big (but anyway PPT) computational power.

Using asymptotic notation we can said that a scheme is secure if every PPT adversary $\mathcal{A}$ has negligible advantage in the security parameter λ.
About the choice of λ, it depends first on the computational assumption on which is based the security proof. Longevity and potential attacks are other elements to keep in mind in order to fix this parameter. A very exhaustive description of possible values for λ can be found at the website *www.keylength.com*.

2.2.4 Primitives

In the following sections we will describe some cryptographic primitives, along with their security definitions, that are related to this work.

2.2.4.1 Hash Function

Strictly speaking an hash function is a function that compress: it takes as input a string in $\{0,1\}^*$ and output a short element in $\{0,1\}^\ell$ for some fixed ℓ. So the output is shorter than the input.
Formally an hash function can be defined as follow.

Definition 8 *An hash function family $\mathcal{H}$ is a couple of algorithms $\mathcal{H} = (\mathbf{Gen}, \mathbf{H})$ such that*

$\mathbf{Gen}(1^\lambda)$ *takes as input a security parameter λ and outputs a key K;*

$\mathbf{H}(K, x)$ *takes as input a secret key K and a string $x \in \{0,1\}^*$ and outputs $H(K,x) \in \{0,1\}^\ell$ where ℓ is some fixed polynomial related to the function H*

Remark 3 *Sometimes, during this thesis, we will refer to an hash function as a function of the family $\{H : K \times D \to R\}$ where K is the key space, D is the domain of H, and R its range. Of course to be an hash function we need $|R| \ll |D|$*

A collision for an hash function $H_K \in \mathcal{H}$ is a pair of distinct elements $x_1, x_2 \in D$ such that $H_K(x_1) = H_K(x_2)$. A very important property related to hash functions is the collision resistance. This property can be defined by

the following experiment:

$\mathbf{Exp}^{CR}_{\mathcal{A},\mathcal{H}}(\lambda)$
$K \leftarrow Gen(1^\lambda)$
$(x_1, x_2) \leftarrow \mathcal{A}(K)$
If $x_1 \neq x_2$ and $H_K(x_1) = K_K(x_2)$ return 1 else return 0

The CR property measure the capability of an adversary to find a collision for an instance of a family of $\mathcal{H}$.
By the previous experiment we can define the advantage of $\mathcal{A}$ to break the CR property for $\mathcal{H}$ as:

$$Adv^{CR}_{\mathcal{H}}(\mathcal{A}) = Pr[\mathbf{Exp}^{CR}_{\mathcal{A},\mathcal{H}}(\lambda) = 1]$$

Definition 9 *An hash functions family $\mathcal{H}$ defined as above have the CR property if for all PPT adversary $\mathcal{A}$, $Adv^{CR}_{\mathcal{H}}(\mathcal{A})$ is a negligible function in the security parameter λ.*

Several construction of CR hash functions are known [66, 69]

2.2.4.2 Chameleon Hash Function

Here we describe a slightly different variant of CR hash functions. Roughly speaking chameleon hash functions are hash functions with the additional property that given a "trapdoor" one can efficiently generate collisions. Formally we have the follows definition

Definition 10 *Let $\mathcal{H}$ a set of functions $h_{ek} : \{0,1\}^* \times \Gamma \rightarrow \{0,1\}^\ell$ where Γ is a sufficiently large set used as randomness space, ℓ is a fixed number (or polynomial in the security parameter λ) which defines the image of h_{ek} and each function is identified by a key ek.*
The family of Hash Function $\mathcal{H}$ is a chameleon hash function family if there exist a tuple of algorithm $(\mathbf{Gen}, \mathbf{Eval}, \mathbf{Coll})$ working as follows:

Gen(1^λ) *Takes as input the security parameter λ and outputs an evaluating key ek and a trapdoor key td.*

Eval(ek, m, ρ) *Takes as input the key ek, which identifies a function* $h_{ek} \in \mathcal{H}$*, a message* $m \in \{0,1\}^*$ *and a randomness* ρ*. It outputs* $c \in \{0,1\}^\ell$*.*

Coll(td, m_1, ρ_1, m_2) *Takes as input the trapdoor key td, two messages* $m_1, m_2 \in \{0,1\}^*$ *and a random element* $\rho_1 \in \Gamma$*. It outputs* $\rho_2 \in \Gamma$ *such that* $Eval(ek, m_1, \rho_1) = Eval(ek, m_2, \rho_2)$*.*

Moreover we require the following properties:

Uniformity Property There exists a distribution over Γ denoted by D_Γ such that for all $m \in \{0,1\}^*$ the distributions $(ek, h(ek, m, r))$ and (ek, y) are statistical indistinguishable, where $(ek, tp) \leftarrow Gen(1^\lambda)$, $r \leftarrow D_\Gamma$, $y \xleftarrow{\$} \{0,1\}^\ell$.

Collision Resistance There exists no PPT adversary $\mathcal{A}$ such that on input $\mathcal{H}$ and ek can find m_1, m_2, ρ_1, ρ_2 such that

$$h_{ek}(m_1, \rho_1) = h_{ek}(m_2, \rho_2).$$

Formally

$$Pr[(m_1, \rho_1) \neq (m_2, \rho_2), h_{ek}(m_1, \rho_1) = h_{ek}(m_2, \rho_2) : (ek, tp) \leftarrow Gen(1^\lambda); (m_1, \rho_1, m_2, \rho_2) \leftarrow \mathcal{A}(ek, \mathcal{H})]$$

is negligible for all PPT adversary $\mathcal{A}$.

2.2.4.3 Public Key Encryption

A Public Key Encryption (PKE) scheme is defined by a tuple of PPT algorithms $(\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ working as follows

KeyGen(1^λ) It takes as input the security parameter λ and outputs a public key pk and a secret key sk.

Enc(pk, m) It takes as input the public key pk and a message m lying in the message space $\mathcal{M}$ that may depend on the public key pk. It outputs a ciphertext c.

Dec$(\mathbf{sk}, c)$ It takes as input the secret key sk and a ciphertext c. It outputs a message m or $\perp$ (failure).

We need an additional property which guarantees the correct decryption of a ciphertext.

Correctness

$$Pr[\mathbf{Dec}(\mathrm{sk}, \mathbf{Enc}(pk, m)) = m] = 1 - f(\lambda)$$

where the probability is over all the possible $(pk, \mathrm{sk}) \leftarrow \mathbf{KeyGen}(1^\lambda)$ and f is a negligible function in the security parameter λ.

2.2.4.3.1 Security for Public Key Encryption Schemes. Now we'll describe two security models for PKE. Specifically we will define indistinguishablity against adaptive chosen plaintext attack (CPA) and indistinguishability under chosen ciphertext attack (CCA).
We refer the reader to [12] for a discussion on security notion candidate for PKE and relations among them.
Chosen Plaintext Attack (CPA). We start by the basic security notion for Public Key Encryption scheme: security against chosen plaintext attack (CPA). We formalize this security notion by the follow experiment.
Consider a PKE scheme $\mathcal{E} = (\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ and an adversary $\mathcal{A}$ playing the follow experiment:

$$\begin{aligned}
&\mathbf{Exp}^{IND-CPA}_{\mathcal{A},\mathcal{E}}(k)\\
&(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathbf{KeyGen}(k)\\
&b \xleftarrow{\$} \{0,1\}\\
&(m_0, m_1) \leftarrow \mathcal{A}(\mathrm{pk})\\
&c \leftarrow \mathbf{Enc}(\mathrm{pk}, m_b)\\
&b' \leftarrow \mathcal{A}(c)\\
&\text{If } b' = b \text{ return 1 else return 0}
\end{aligned}$$

Here k is the security parameter. We can suppose that it's still public, as the public key.
We said that $\mathcal{A}$ wins the game if he guesses the correct bit b (that is $\mathbf{Exp}^{IND-CPA}_{\mathcal{A},\mathcal{E}}(k) = 1$). Informally to win this game the adversary $\mathcal{A}$ must be

able to distinguish the encryption of two different ciphertexts of his choice. Two observations about this game:

- Differently from Private Encryption adversary $\mathcal{A}$ can compute itself encryption for different messages so in the queries phase there is just one query.
- From the first observation we can observe that no PKE scheme with a deterministic encryption algorithm can be IND-CPA secure.

We define the advantage

$$\mathbf{Adv}_{\mathcal{E}}^{IND-CPA}(\mathcal{A}) = |2 \cdot Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{E}}^{IND-CPA}(k) = 1\right] - 1|$$

Definition 11 *A PKE sheme $\mathcal{E} = (\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ is IND-CPA secure if for all PPT adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}_{\mathcal{E}}^{IND-CPA}(\mathcal{A})$ is at most a negligible function in the security parameter k.*

Chosen Ciphertext Attack. Security in the Chosen Ciphertext Attack (CCA) model is a trivial extension of CPA security model. Here we allow adversary to ask for decryption queries.
Formally it can be defined by the follow experiment: Let $\mathcal{E} = (\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ a PKE scheme and $\mathcal{A}$ an adversary playing the follow experiment:

$\mathbf{Exp}_{\mathcal{A},\mathcal{E}}^{IND-CCA}(k)$
$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathbf{KeyGen}(k)$
$b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1) \leftarrow \mathcal{A}^{\mathbf{Dec}(\mathrm{sk},\cdot)}(\mathrm{pk})$
$c \leftarrow \mathbf{Enc}(\mathrm{pk}, m_b)$
$b' \leftarrow \mathcal{A}^{\mathbf{Dec}(\mathrm{sk},\cdot)}(c)$
If $b' = b$ return 1 else return 0

The goal of $\mathcal{A}$, similarly in the CPA game, is to guess whether c is an encryption of m_0 or m_1. However $\mathcal{A}$ has access to a decryption oracle (that can be consider like a black box algorithm) through that he can decrypt ciphertexts of his choice.

Of course, in the second decryption phase (line 5) he cannot ask for a decryption on the challenge ciphertext c. In this case, the challenger return $\perp$. We define the IND-CCA advantage for an adversary $\mathcal{A}$ as

$$\mathbf{Adv}_{\mathcal{A},\mathcal{E}}^{IND-CCA} = |2 \cdot Pr\left[\mathbf{Exp}_{\mathcal{A},\mathcal{E}}^{IND-CPA}(k) = 1\right] - 1|.$$

Finally we have the follow definition:

Definition 12 *A PKE sheme $\mathcal{E} = (\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ is IND-CCA secure if for all PPT adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}_{\mathcal{E}}^{IND-CCA}(\mathcal{A})$ is at most a negligible function in the security parameter k.*

It's very easy to check that $\mathbf{Adv}_{\mathcal{A},\mathcal{E}}^{IND-CCA} \geq \mathbf{Adv}_{\mathcal{A},\mathcal{E}}^{IND-CPA}$. In particular, if $\mathcal{E}$ is IND-CCA secure then it's also IND-CPA secure and if $\mathcal{E}$ is not IND-CPA secure then it's not IND-CCA secure.
Now, for sake of completeness and as example of encryption scheme we present the Paillier encryption scheme, very useful in the rest of this thesis.

2.2.4.3.2 Paillier Encryption Scheme First of all we define the following standard functions in group theory:

- let $n = pq$, p, q primes. $\lambda(n) = lcm(p-1, q-1)$.
- Let S_N the set of integers u such that $\{u < N^2 \| u \equiv 1 \mod N\}$. Then we define for each $u \in \mathcal{S}_N$

  $$L(u) = \frac{u-1}{N}.$$

  Note that $L(u) \in \mathbb{Z} \, \forall u \in S_N$.
- We will denote with $\mathcal{B}_\alpha$ the subgroup of $\mathbb{Z}_{N^2}^*$ whose elements have order αN ($\alpha \neq 0$) and with $\mathcal{B}$ the disjoint union of $\mathcal{B}_\alpha$, $\alpha = 1, \ldots, N \; \alpha \mid \lambda(N)$.

Let $g \in \mathcal{B}$. It's possible to prove that the function $\mathcal{E}_g : \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$ defined as follows

$$\mathcal{E}_g(x, y) = g^x y^N$$

is a bijection.
Now we are ready to present the Paillier encryption scheme:

Key Generation Let k the security parameter. The algorithm chooses two primes p and q of equal length k and an element $g \in \mathcal{B}$. It sets $N = pq$, computes $\mu = L(g^{\lambda(N)} \mod N^2)^{-1} \mod N$ and sets (g, N) as the public key, $(\lambda(N), \mu)$ as the private decryption key

Encryption Let $m < N$ a message. It chooses random $r \in \mathbb{Z}_N^*$ and computes the ciphertext c as:

$$c = g^m r^N \mod N^2.$$

Decryption To decrypt a ciphertext c the algorithm first checks that $c < N^2$. If yes it's possible retrieve the message m as

$$m = L(c^{\lambda(N)} \mod N^2) \cdot \mu \mod N$$

otherwise it returns $\perp$

Theorem 1 *If the Decisional Composite Residues Assumption (DCRA) hold, the scheme described above is IND-CPA secure.*

2.2.4.4 Signatures

So far we have only described security models for cryptographic primitives that guarantees privacy, through indistinguishability games. However many times users need a different kind of security.

Suppose the Department of Homeland Security intends to issue a public warning. So the message is public, no encryption scheme is needed, but in this case people want to be sure that the message comes from the HS department and no one has modified it.

So we need two different properties by this new primitive:

Authenticity : like in a real signature, everyone must be sure of the authenticity of the message. Moreover no one can generate a valid "digital signature" for a different user without some private information

Integrity : everyone can be sure that the message have been not modify by a third part.

Another important property is the *non repudiation*, that is, the signer cannot deny to have signed a message.

Now we formally describe a signature scheme by the follow definition.

Definition 13 *A Digital Signature Scheme (DSS)* $\mathcal{S}$ *is defined by a triple of algorithms* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$ *such that*

$\mathbf{KeyGen}(1^\lambda)$ *is a PPT algorithm which on input a security parameter* λ *outputs a signing key sk and a verification key vk. This algorithm implicitly defines a message space* $\mathcal{M}$ *and a signature space* Σ*.*

$\mathbf{Sign}(\mathbf{sk}, m)$ *is a PPT algorithm which takes as input the signing key sk and a message* m *and outputs a string* σ *called signature of* m*.*

$\mathbf{Verify}(\mathbf{vk}, \sigma, m)$ *is a PPT algorithm which given a verification key vk, a signature* σ *and a message* m *returns* 1 *(accept) or* 0 *(reject).*

We also require the follow correctness property: if $(\mathrm{sk}, \mathrm{vk}) \leftarrow \mathbf{KeyGen}(1^\lambda)$ and $\sigma \leftarrow \mathbf{Sign}(\mathrm{sk}, m)$ then for all messages $m \in \mathcal{M}$

$$Pr\left[\mathbf{Verify}(\mathrm{vk}, \sigma, m) = 1\right] \approx 1$$

2.2.4.4.1 Security for Digital Signatures Schemes Now we want to focus on the security definition for Digital Signatures. First of all we observe that there are three basic kind of attacks:

1. Key only Attack: the Adversary knows only the verification key vk.
2. Random Message Attack: The Adversary knows the verification key vk and can ask for couples (m, σ) where m is chosen random by the challenger and σ is a valid signature for the message m.
3. Chosen Message Attack: Similar to the random message attack, but, in this case, Adversary chooses the message m signed by the challenger.

About the success of Adversary in the previous attacks, we can distinguish the following forgeries:

Existential Forgery Adversary outputs a valid signature on a new message, not necessary of its choice.

Strong Existential Forgery Like Existential Forgery but in this case message can be one of previously seen message. Of course, in this case, the related signature must be different from the signature previously seen.

Selective Forgery The adversary outputs a signature for a message of his choice.

Universal Forgery The adversary is able to output a signature on any message.

Total Break Adversary can compute the secret key sk.

Now, in order to clarify those definitions, we describe formally the security games with CMA for each different forgery.

Definition 14 *Let* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$ *a Digital Signature Scheme.* $\mathcal{S}$ *is UF-CMA secure if for all PPT adversary* $\mathcal{A}$, $\mathbf{Adv}^{UF-CMA}(\mathcal{A})$ *is negligible in the security parameter* λ, *where* $\mathbf{Adv}^{UF-CMA}(\mathcal{A})$ *is the probability of winning the follow game:*

Setup *Challenger runs* $\mathbf{KeyGen}(1^\lambda)$ *to obtain* sk, vk. *The verification key* vk *is given to* $\mathcal{A}$.

Queries *Adversary* $\mathcal{A}$ *can asks for a polynomial number of queries* $(m_1, \ldots, m_q)$ *to the signer to obtain the signatures corresponding to each queried message.*

Forgery $\mathcal{A}$ *output a forgery* (m^*, σ^*).

- $\mathcal{A}$ *wins the UF-CMA game if* $\mathbf{Verify}(m^*, \sigma^*) = 1$ *and* $m^* \notin (m_1, \ldots, m_q)$.
- $\mathcal{A}$ *wins the SUF-CMA game if* $\mathbf{Verify}(m^*, \sigma^*) = 1$[3].

[3]Of course, if $m^* \in (m_1, \ldots, m_q)$ then σ^* must be different from the signature previously seen.

- $\mathcal{A}$ *wins the SF-CMA game if* $\mathbf{Verify}(m^*, \sigma^*) = 1$ *where* m^* *was chosen by* $\mathcal{A}$ *before the start of the attack.*
- $\mathcal{A}$ *wins the UF-CMA game if* $\mathbf{Verify}(m^*, \sigma^*) = 1$ *where* m^* *was given to* $\mathcal{A}$ *before the start of the attack.*

In the next section we describe an example of signature scheme, the Waters signature schemes [74], which will be used in some constructions presented in this paper.

2.2.4.4.2 Waters Signature Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map.

KeyGen(p) : It sets $l \leftarrow \lceil \log p \rceil$ and chooses $g, g_2 \xleftarrow{\$} \mathbb{G}$, $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and $A_0, A_1 \ldots, A_l \xleftarrow{\$} \mathbb{G}$. Then it sets $g_1 \leftarrow g^\alpha$ and returns $\mathrm{vk} = (g, g_1, g_2, A_0, A_1, \ldots, A_l)$, $\mathrm{sk} = g_2^\alpha$

Sign$(m, \mathbf{sk})$: This algorithm chooses a random $r \xleftarrow{\$} \mathbb{Z}_p$ and sets $\sigma \leftarrow g^r$ and $\tau \leftarrow g_2^\alpha \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[m]_\zeta} \right)^r$. Then it returns $\Sigma = (\sigma, \tau)$

Verify$(\mathbf{vk}, m, \Sigma)$: It checks that

$$e(g, \tau) = e(\sigma, A_0 \prod_{\zeta=1}^{l} A_\zeta^{[m]_\zeta}) \cdot e(g_1, g_2).$$

If the above equation holds it returns 1, else it returns 0.

Theorem 2 *If the CDH assumption hold then the above signature scheme is UF-CMA secure*

In [17] a strongly secure variant of the previously described scheme (under the same computational assumption) is provided. We will refer to this latter scheme as the *Strong* Waters Signature Scheme.
The Strong Waters Signature Scheme works as follows.
Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear

map and $\mathcal{H} = \{H_K\}_{K \in \mathcal{K}}$ (where $\mathcal{K}$ is the keys' space) a family of collision resistant hash functions

KeyGen(p) : It sets $l \leftarrow \lceil \log p \rceil$ and chooses $g, g_2, h \xleftarrow{\$} \mathbb{G}$, $\alpha \xleftarrow{\$} \mathbb{Z}_p$, $A_0, A_1 \ldots, A_l \xleftarrow{\$} \mathbb{G}$, $K \xleftarrow{\$} \mathcal{K}$. Then it sets $g_1 \leftarrow g^\alpha$ and returns $\mathrm{vk} = (g, g_1, g_2, h, A_0, A_1, \ldots, A_l, k)$, $\mathrm{sk} = g_2^\alpha$

Sign$(m, \mathbf{sk})$: This algorithm chooses random $r, s \xleftarrow{\$} \mathbb{Z}_p$ and sets $\sigma \leftarrow g^r$. Then it computes

$$t \leftarrow H(m\|\sigma), \qquad M \leftarrow H(g^t h^s), \qquad \tau \leftarrow g_2^\alpha \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[M]_\zeta} \right)^r .$$

Next it returns $\Sigma = (\sigma, \tau, s)$.

Verify$(\mathbf{vk}, m, \Sigma)$: To verify a signature $\Sigma = (\sigma, \tau, s)$ the algorithm computes $t \leftarrow H(m\|\sigma)$, $M \leftarrow H(g^t h^s)$. Then it checks that

$$e(g, \tau) = e(\sigma, A_0 \prod_{\zeta=1}^{l} A_\zeta^{[M]_\zeta}) \cdot e(g_1, g_2).$$

If the above equation holds it returns 1, else it returns 0.

Theorem 3 *If the CDH assumption hold then the above signature scheme is SF-CMA secure*

2.2.4.5 Authenticated Encryption

Until now we have introduced two primitives related respectively to two different security goals:

- encryption scheme ⇒ privacy
- digital signature scheme ⇒ authenticity/integrity

In many case users need a primitive achieving both kind of security. Such a primitive is called authenticated encryption scheme.
There are many possible ways to obtain an authenticated encryption scheme. In the symmetric setting a class of possible ways to achieve AE is called *Generic Composition Method*. That is combine standard symmetric encryption scheme with *Message Authentication Code* (which is the equivalent of signature in the symmetric setting) to achieve the desired primitive.

In the asymmetric setting it's possible to generalize those constructions replacing symmetric encryption with public key encryption and MAC with digital signature.

However this is just one method to achieve this primitive. In the rest of this thesis we'll refer to AE in the asymmetric setting considering it as new primitive matching both security goals. For this reason privacy and security notions are exactly the same in encryption scheme and signatures, so are omitted.

2.2.4.6 Sigma Protocol

Now we introduce the last cryptographic tool used in this thesis.
Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be an arbitrary binary relation, with the only restriction that if $(x, w) \in \mathcal{R}$, then the length of w is polynomial in the length of x (typically, $(x, w) \in \mathcal{R}$ if x is part of an NP language L and w is one of its associated witnesses). A Σ-Protocol for $\mathcal{R}$ is an interactive (three rounds) protocol involving two parties: a prover P and a verifier V. We assume that both parties are PPT machines and that they agree on some value x in advance, and the goal of the protocol is to let the prover convince the verifier that he knows w such that $(x, w) \in \mathcal{R}$. The three rounds are carried out as follows: in the first round P sends a message to V, who replies with a string (chosen at random from a well defined set and called a challenge string), and finally gets back a third message from P and outputs 1 or 0 depending on whether he is convinced by this interaction.
More formally, a Σ protocol consists of four PPT algorithms $\Sigma = (\mathbf{\Sigma\text{-}Setup}, \mathbf{\Sigma\text{-}Com}, \mathbf{\Sigma\text{-}Resp}, \mathbf{\Sigma\text{-}Verify})$ defined as follows:

$\mathbf{\Sigma\text{-}Setup}(1^\lambda, \mathcal{R}) \to (x, w)$. It takes as input a security parameter λ and

a relation $\mathcal{R}$. It returns a statement x and a witness w such that $(x, w) \in \mathcal{R}$.

Σ-Com$(x) \rightarrow (R, r)$ Is a probabilistic algorithm run by the prover to get the first message R to be sent to the verifier and some private state r to be stored and used later in the protocol.

Σ-Resp$(x, w, r, c) \rightarrow s$ is an algorithm run by the prover to compute the last (third) message of the protocol (to be sent to the verifier). It takes as input the statement x, its witness w, the challenge string (chosen at random by V in a well defined set **ChSp** and sent as the second message of the protocol), and some state information r. It outputs the third message of the protocol.

Σ-Verify$(x, R, c, s) \rightarrow \{0, 1\}$ is the verification algorithm that on input the message R, a challenge $c \in$ **ChSp** and a response s, outputs 1 (accept) or 0 (reject).

We assume that the protocol satisfies the following three properties:

Completeness $\forall (x, w) \in \mathcal{R}$, any $(R, r) \leftarrow$ **Σ-Com**(x, r), any $c \in$ **ChSp** and $s \leftarrow$ **Σ-Resp**(x, w, r, c) then

$$\textbf{Σ-Verify}(x, R, c, s) = 1$$

with overwhelming probability.

Special Soundness There exists an extractor algorithm **Σ-Ext** such that $\forall x \in L$, $\forall R, c, s, c', s'$ such that **Σ-Verify**$(x, R, c, s) = 1$ and **Σ-Verify**$(x, R, c', s') = 1$, **Σ-Ext**$(x, R, c, s, c', s') = w'$ such that $(x, w') \in \mathcal{R}$

Special Honest Verifier Zero Knowledge (HVZK) There exists a PPT algorithm S such that $\forall c \in$ **ChSp**, $S(x, c)$ generates a pair (R, s) such that **Σ-Verify**$(x, R, c, s) = 1$ and the probability distribution of (R, c, s) is identical to that obtained by running the real algorithms.

2.2.4.6.1 Schnorr Sigma Protocol Here we briefly describe a well know Sigma Protocol, the Schnorr Identification Protocol. This protocol was introduced by Schnorr in [68]. It works as follow.

Definition 15 (Schnorr Σ-Protocol) *Let $\mathbb{G}$ a group of prime order p and $\mathcal{R}$ the DL relation on $\mathbb{G}$, DL= $\{(x,w)|x=(p,g,h),\ \ h=g^w\}$. Let $\overline{g}\in\mathbb{G}$ a group generator. We define $DL_{\overline{g}}=\{(x,w)|x=\overline{g}^w\}$ the restriction of the DL relation to $g=\overline{g}$.*
*The Schnorr Σ-Protocol consists of four PPT algorithm Σ_n = (**Σ-Setup**, **Σ-Com**, **Σ-Resp**, **Σ-Verify**) defined as follows:*

Σ_n-Setup$(1^\lambda, n, \mathcal{R})$ *It chooses a random group generator $g\in\mathbb{G}$ and a witness $w\xleftarrow{\$}\mathbb{Z}_p$. Then it computes the statement $x\leftarrow g^w$ and outputs (x,w).*

Σ_n-Com(x) *It chooses a random $r\in\mathbb{Z}_p$, sets $R\leftarrow g^r$ and returns (r,R).*

Σ_n-Resp(x,w,c) *Let $c\in\mathbb{Z}_p$ the second message of the protocol. This algorithm outputs $s\leftarrow r+cw$.*

Σ_n-Verify(x,R,c) *It checks that*

$$g^s = Rx^c.$$

If the above equation holds, it outputs 1, else outputs 0.

2.2.5 Homomorphic primitives

In the rest of this thesis we will focus on cryptographic primitives with a structure very similar to homomorphism in group theory. Informally a cryptographic primitive $\mathcal{P}$ with message space $\mathcal{M}$ is said to be homomorphic with respect to a class of functions $\mathcal{F}$ if there exists an algorithm **Combine** such that, for all $f\in\mathcal{F}$

$$\textbf{Combine}(f,\mathcal{P}(m_1),\ldots,\mathcal{P}(m_n)) = \mathcal{P}(f(m_1,\ldots,m_n)).$$

We remark that **Combine** algorithm need no secret information, so everyone can compute $\mathcal{P}(f(m_1,\ldots,m_n))$. If we have no restriction on $\mathcal{F}$ a such

primitive it's said to be *fully homomorphic*.
If instead in $\mathcal{F}$ there are only linear functions we will call it *Linearly Homomorphic*[4]

[4]Formally definition for linearly homomorphic encryption and signatures can be found in the following chapters.

Chapter 3

A linearly homomorphic signature scheme to sign elements in bilinear groups

In this chapter we provide formal definitions for Linearly homomorphic signatures and introduce the new concept of Linearly homomorphic signature scheme to sign elements in bilinear groups (LHSG).
We recall that homomorphic signatures allow users, knowing signatures on messages $m_1, \ldots, m_n$, to compute a signature on $f(m_1, \ldots, m_n)$ without using the signing key.
About the function f, if it can be chosen without any specific limitation, the scheme will be called *Fully* Homomorphic Signature Scheme.
In this chapter we will focus on *Linearly* Homomorphic signature, that are signatures where the function f is a linear function. We will provide a formal definition below.
Another goal of this chapter is to achieve a general conversion method for LHSG from Random Message Security to Chosen Message Security. Finally we will observe that this last result works in other contexts as well, like structure preserving signatures.

3.1 Linear Network Coding and Linearly Homomorphic Signatures

In existing computer network, information is transmitted from the source node to the destination node through a chain of intermediate nodes. Network coding, in contrast with the *store and forward* routing, refers to a routing mechanism where each intermediate node *modifies* data packets in transit. There are two main advantages in this routing strategy:

- increase the throughput in certain network topologies[58].
- improving robustness against random network failure.

In *Linear* Network Coding a file is interpreted as an ordered sequence of n-dimensional vectors $\vec{\mathbf{v}}_1, \ldots, \vec{\mathbf{v}}_m \in \mathbb{F}_p^n$, where p is prime.
Before transmitting, each vector is modified by the source node *appending* a canonical basis vector to each $\vec{\mathbf{v}}_i$ in this way: $\mathbf{v_i} = (\vec{\mathbf{v}}_i \| \mathbf{e}_i)$, where $\mathbf{e}_i \in \mathbb{Z}^m$ is the i-th vector of the canonical basis.
Then the source sends these vectors as packets in the network. Each intermediate node of the network after receiving some packets $\mathbf{w_1}, \ldots, \mathbf{w}_\tau$, computes a linear combination $\mathbf{w} = \sum_{i=1}^{\tau} \alpha_i \mathbf{w}_i$ for some random α_i $i = 1, \ldots, \tau$ and sends $\mathbf{w}$ to the next node(s).
The receiver can recover the original file knowing m linear independents vectors $\mathbf{w}_1, \ldots \mathbf{w}_m$. Formally the last node parses $\mathbf{w}_i = (\tilde{\mathbf{v}}_i \| \tilde{\mathbf{e}}_i)$, such that $\tilde{\mathbf{e}}_i \in \mathbb{Z}^m$. Then it computes the matrix

$$G = \begin{bmatrix} \tilde{\mathbf{e}}_1 \\ \vdots \\ \tilde{\mathbf{e}}_m \end{bmatrix}^{-1}$$

Finally it can recover the original file

$$\begin{bmatrix} \vec{\mathbf{v}}_1 \\ \vdots \\ \vec{\mathbf{v}}_m \end{bmatrix} = G \cdot \begin{bmatrix} \tilde{\mathbf{v}}_1 \\ \vdots \\ \tilde{\mathbf{v}}_m \end{bmatrix}$$

Of course, it can compute the matrix G only if the received vectors are linear independent.
Linear Network Coding, as described until now, is affected by the pollution attack. That is, if a malicious adversary corrupts a node, he can send random packets through the network and finally the receiver cannot recover the original file, due to errors introduced in the network by the random packet. Indeed just a single error introduced by a node will be propagated by every nodes following it.
In [13] authors propose a cryptographic approach to this problem and show how linearly homomorphic signatures can be a solution for it.
Specifically their solution works in this way:

- Source node signs each augmented vector of the original file using a LHS scheme and sends these vectors together with corresponding signatures through the network.
- each node, after receiving vectors $\mathbf{w}_1, \ldots, \mathbf{w}_\tau$ together with corresponding signatures, checks each signature using the public verification algorithm (he rejects *no-verified* vectors).
- each node generates a new linear combination only for vectors that have passed the previous test, together with a signature on this combination (he can compute such signature using the homomorphic property of LHS scheme)

It's easy to check that by this strategy is possible to avoid the pollution attack described above.

3.2 Homomorphic Signatures scheme

First here we recall the notion of Homomorphic Signatures as defined by D. M. Freeman in [38].

Definition 16 *An Homomorphic Signature Scheme is a tuple of PPT algorithms* (**HKeyGen**, **HSign**, **HVerify**, **HEval**) *defined as following:*

- **HKeyGen** $(1^\lambda, k)$ *Take as input a security parameter* λ *and a maximum data set size* k*. Return a secret key* sk *and a verification key* vk *(used for function evaluation and verification); the public key* vk *implicitly defines a message space* $\mathcal{M}$ *which is also a group, a file identifier space* $\mathcal{D}$*, a signature space* Σ *and a set* $\mathcal{F}$ *of admissible function* $f : \mathcal{M}^k \to \mathcal{M}$*.*

- **HSign**(sk, τ, m, i) *Take as input the signing key* sk*, a dataset identifier* τ*, a message* $m \in \mathcal{M}$ *and an index* $i \in \{1, \dots k\}$*. Output a signature* $\sigma \in \Sigma$*.*

- **HVerify**(vk, τ, m, σ, f) *Take as input the verification key* vk*, a dataset identifier* τ*, a message* $m \in \mathcal{M}$*, a signature* $\sigma \in \Sigma$ *and an admissible function* $f \in \mathcal{F}$*. Return* 0 *(reject) or* 1 *(accept)*

- **HEval**(vk, τ, f, σ) *Take as input the verification key* vk*, a dataset identifier* fid*, a function* $f \in \mathcal{F}$ *and a vector of signature* $\sigma = (\sigma_1, \dots, \sigma_k) \in \Sigma^k$*. Return* $\sigma \in \Sigma$*.*

3.2.1 Correctness and Security for Homomorphic Signatures

Let $(\text{sk}, \text{vk}) \leftarrow$ **HKeyGen**$(1^\lambda, k)$. For correctness we require that

- Let π_i the standard projection function (i.e. $\pi_i(m_1, \dots, m_k) = m_i$). Then $\pi_i \in \mathcal{F}$ for all $i \in [k]$.

- $\forall \tau \in \{0,1\}^\lambda, m \in \mathcal{M}, i \in [k]$, if $\sigma \leftarrow$ **HSign**(sk, τ, m, i) then **HVerify**$(\text{pk}, \tau, m, \sigma, \pi_i) = 1$

- Let $\tau \in \{0,1\}^\lambda, \mu = (\mu_1, \dots, \mu_k) \in \mathcal{M}^k, f \in \mathcal{F}$. Let $\sigma = (\sigma_1, \dots, \sigma_k) \in \Sigma^k$ signatures produced by zero or more iterative application of **HEval** on the outputs of **HSign**$(\text{sk}, \tau, \mu_i, i)$ and let $(f_1, \dots, f_k, g) \in \mathcal{F}^{k+1}$be any tuple of admissible functions. If

  - **HVerify**$(\text{vk}, \tau, m_i, f_i) = 1$ for some $m_1, \dots, m_k \in \mathcal{M}$;
  - $g(m_1, \dots, m_k) \in \mathcal{M}$;

– $g \circ \mathbf{f}$[1] is admissible;

then, with overwhelming probability

$$\mathbf{HVerify}(\mathrm{vk}, \tau, g(\mathbf{m}), \mathbf{HEval}(\mathrm{vk}, \tau, g, \sigma), g \circ \mathbf{f}) = 1$$

For what concerns security, it's possible to introduce the concept of unforgeability for homomorphic signatures as follows

Definition 17 *Let* $\mathcal{S} = (\mathbf{HKeyGen}, \mathbf{HSign}, \mathbf{HVerify}, \mathbf{HEval})$. *Homomorphic unforgeability is defined by the following game between a challenger and an adversary* $\mathcal{A}$.

Setup *The challenger runs* $(sk, vk) \leftarrow \mathbf{HKeyGen}(1^\lambda, k)$. *Then it gives* vk *to the adversary* $\mathcal{A}$. *Finally let* Q *be an empty set.*

Queries *For each query,* $\mathcal{A}$ *chooses a filename* $F \in \{0,1\}^*$ *and a message* m. *If* $F \notin Q$ *the challenger chooses a tag* $\tau_F \xleftarrow{\$} \{0,1\}^*$, *gives it to* $\mathcal{A}$ *and set a counter* $i_F = 1$ *and* $Q = Q \cup \{F\}$. *Otherwise, the challenger retrieves the associated* τ_F *from* Q *and increments* i_F *by 1 (if* $i_F > k$, *the query is answered with* $\perp$*). Then it gives the signature* $\sigma \leftarrow \mathbf{HSign}(sk, \tau_F, m, i_F)$ *to* $\mathcal{A}$.
The adversary $\mathcal{A}$ *can ask polynomially many such queries. Let* V_F *the tuple of elements* m *queried for filename* F, *listed in the order they were queried.*

Forgery $\mathcal{A}$ *outputs* $(\tau^*, m^*, \sigma^*, f^*)$, *where* σ^* *is a signature,* $m^* \in \mathcal{M}$ τ^* *a dataset identifier and* f^* *an admissible function.*

We say a function f *is well-defined on* $\mathcal{F}$ *if either* $i_F = k$ *or* $i_F < k$ *and* $f(V_F, m_{i_F+1}, \ldots, m_k)$ *does not depend on* $m_{i_F+1}, \ldots, m_k$.
The Adversary wins the game if $\mathbf{HVerify}(vk, fid^*, m^*, \sigma^*, f^*) = 1$ *and one of the following conditions hold:*

1 $\tau^* \neq \tau_F \forall F$ *queried by* $\mathcal{A}$.

[1] Here by $g \circ \mathbf{f}$ we denote the function that maps $\mathbf{x} = (x_1, \ldots, x_k)$ to $g(f_1(\mathbf{x}), \ldots, f_1(\mathbf{x}))$

2 $\tau^* = \tau_F$ *for some filename* F*,* f *is well-defined on* F *and* $m^* \neq f^*(V_F)$

3 $\tau^* = \tau_F$ *for filename* F *and* f^* *is not well-defined on* F[2]*.*

Finally we define the advantage $\mathbf{Adv}^{H-Uf}(\mathcal{A})$ *of* $\mathcal{A}$ *as the probability that* $\mathcal{A}$ *wins the game.*

We say that an Homomorphic Signature scheme is *Unforgeable* secure if $\mathbf{Adv}^{H-Uf}(\mathcal{A}) = negl(\lambda)$ for any PPT adversary $\mathcal{A}$.

3.3 LHSG: Definition

Following [61, 38], our definition of LHSG is essentially equivalent to the one of linearly homomorphic signature scheme (in its strongest variant derived from [7]). To adapt it to our results, we assume that the message space is some bilinear group $\mathcal{M}$ and

- We use as set of functions $\mathcal{F}$ the set of linear combinations of elements of the group, so each function $f \in \mathcal{F}$ can be uniquely expressed as $f(m_1, \ldots, m_k) = \prod_{i=1}^{k} m_i^{\alpha_i}$, and therefore can be identified by a proper vector $(\alpha_1, \ldots, \alpha_k) \in \mathbb{Z}^k$.
- We identify each dataset by a string fid $\in \{0,1\}^*$, and use an additional argument $i \in \{1, \ldots, n\}$ for the signing algorithm to specify that the signed message can be used only as the i-th argument for each function $f \in \mathcal{F}$.

Definition 18 (LHSG) *A Linearly homomorphic signature scheme to sign elements in bilinear groups is a tuple of 4 PPT algorithms (***KeyGen***,* **Sign***,* **Verify***,* **Eval***) such that:*

[2]In [38] author proves the following. Let $\mathcal{S}$ a *Linearly* Homomorphic Signature on a ring R. If $\mathcal{S}$ is secure against type 2 forgery, then it's secure against type 3 forgery.

- **KeyGen**$(1^\lambda, n, k)$ *takes as input the security parameter* λ*, an integer* n *denoting the length of vectors to be signed and an upper bound* k *for the number of messages signed in each dataset. It outputs a secret signing key* sk *and a public verification key* vk*; the public key implicitly defines a message space of the form* $\mathcal{M} = \mathbb{G}^n$*, a file identifier space* $\mathcal{D} = \{0,1\}^{n_d}$ *and a signature space* Σ*, for some* $n_d \in poly(\lambda)$*.*

- **Sign**(sk, m, fid, i) *takes as input the secret key, an element* $m \in \mathcal{M}$*, a dataset identifier* fid*, an index* $i \in \{1, \ldots, k\}$ *and outputs a signature* σ*.*

- **Verify** *(*vk*,* σ*,* m*,* fid*,* f*) takes as input the public key* vk*, a signature* $\sigma \in \Sigma$*, a message* $m \in \mathcal{M}$ *a dataset identifier* $fid \in \mathcal{D}$ *and a function* $f \in \mathcal{F}$ *and outputs* 1 *(accept) or 0 (reject).*

- **Eval** $(vk, fid, f, \{\sigma_i\}_{i=1\ldots k})$ *takes as input the public key* vk*, a dataset identifier* fid*, an admissible function* f *in its vector form* $(\alpha_1, \ldots, \alpha_k)$*, a set of* k *signatures* $\{\sigma_i\}_{i=1\ldots k}$ *and outputs a signature* $\sigma \in \Sigma$*. Note that this algorithm should also work if less than* k *signatures are provided, as long as their respective coefficients in the function* f *are 0, but we don't explicitly account this to avoid heavy notation.*

3.3.1 LHSG: Correctness and Security

The correctness conditions of our scheme are the following:

- *Let* $(sk, vk) \leftarrow$ **KeyGen**$(1^\lambda, n, k)$ *be an honestly generated keypair,* $m \in \mathcal{M}$*,* fid *any dataset identifier and* $i \in 1, \ldots, k$*. If* $\sigma \leftarrow$ **Sign**(sk, m, fid, i)*, then with overwhelming probability*

$$\textbf{Verify}(vk, \sigma, m, fid, e_i) = 1,$$

  where e_i *is the* i*-th vector of the standard basis of* $\mathbb{Z}^k$*.*

- *Let* $(sk, vk) \leftarrow$ **KeyGen**$(1^\lambda, n, k)$ *be an honestly generated keypair,* $m_1, \ldots, m_k \in \mathcal{M}$ *any tuple of messages signed w.r.t the same* fid*, and let* $\sigma_1, \ldots, \sigma_k \in \Sigma$*,* $f_1, \ldots, f_k \in \mathcal{F}$ *such that for all* $i \in \{1, \ldots, k\}$*,*

$\mathbf{Verify}(vk, \sigma_i, m_i, fid, f_i) = 1$. *Then, for any admissible function* $f = (\alpha_1, \ldots, \alpha_k) \in \mathbb{Z}^k$*, with overwhelming probability*

$$\mathbf{Verify}(vk, \mathbf{Eval}(vk, fid, f, \{\sigma_i\}_{i=1\ldots k}), f(m_1, \ldots, m_k), fid, \sum_{i=0}^{k} \alpha_i f_i) = 1$$

About security, roughly speaking, a LHSG is said to be secure if no PPT adversary $\mathcal{A}$ can produce with more than negligible probability one of the following:

- A signature for a message w.r.t. a new fid (i.e. one that it has never seen before)
- A signature w.r.t. a previously seen identifier, for a message m different from the one obtained applying the f to the previously signed messages of the same dataset
- A signature it has not seen but that has been used in the **Eval** algorithm to compute signatures it has seen (under certain independence constraints, see the formal definition for details).

We distinguish between notions where the adversary has no control over the signed messages he can see, and the standard one where he can adaptively choose them by itself.

Definition 19 (Random message attack security) *An LHSG is unforgeable against a random message attack if for all* n, k *the advantage of any PPT adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* λ*:*
Setup *The challenger runs* $\mathbf{KeyGen}(1^\lambda, n, k)$ *and gives* vk *to* $\mathcal{A}$*. The message space* $\mathcal{M}$*, the signature space* Σ *and the dataset space* $\mathcal{D}$ *are all implicitly defined by the verification key.*
Queries $\mathcal{A}$ *can ask a polynomial number of queries of the following types:*

- **Signing Queries** $\mathcal{A}$ *asks for a new message/signature couple w.r.t. to a specific* $fid \in \mathcal{D}$ *and a specific index* $i \in \{1, \ldots, k\}$ *. The challenger checks that this query has not been previously answered (otherwise it returns* $\perp$*), then it picks a random message* $m \xleftarrow{\$} \mathcal{M}$ *and uses the*

secret key sk to compute a signature σ for m w.r.t. fid and the index i. Finally it picks a handle h (from a proper set of identifiers), stores $(h, (\mathit{fid}, m, \sigma, e_i))$ in a table T and returns h to $\mathcal{A}$. Note that $\mathcal{A}$ does not see neither the message nor the signature, and that here $e_i \in \mathbb{Z}^k$ is the i-th vector of the canonical basis, used to indicate the (trivial) function with respect to which the signature has been issued.

- **Derivation Queries** *$\mathcal{A}$ chooses a set of handles $h = (h_1, \ldots, h_k)$ and a vector of coefficients $f = (\alpha_1, \ldots, \alpha_k)$. The challenger retrieves $\{(h_i, (\mathit{fid}_i, m_i, \sigma_i, f_i))\}_{i=1,\ldots k}$ from T and returns $\perp$ if any of these does not exist or if $\mathit{fid}_i \neq \mathit{fid}_j$ for some $i, j \in \{1, \ldots, k\}$ $(i \neq j)$. Else, it computes $m = \prod_{i=1}^{k} m_i^{\alpha_i}$, $\sigma = \mathbf{Eval}(vk, \mathit{fid}, f, \{\sigma_i\}_{i=1\ldots k})$, chooses a handle h, stores $(h, (\mathit{fid}, m, \sigma, \sum_{i=0}^{k} \alpha_i f_i))$ in T and returns h to $\mathcal{A}$.*

- **Reveal Queries** *$\mathcal{A}$ chooses a handle h. If this handle is not in T, the challenger returns $\perp$. Otherwise it retrieves the corresponding record $(h, (\mathit{fid}, m, \sigma, f))$ from table T and gives $(\mathit{fid}, m, \sigma, f)$ to $\mathcal{A}$. Next it adds $(h, (\mathit{fid}, m, \sigma, f))$ to a different table Q.*

Forgery *$\mathcal{A}$ outputs a dataset identifier fid^*, a message m^*, a signature σ^* and a function f^*.*
Let $Q_{\mathit{fid}^*} = \{(h_i, (\mathit{fid}^*, m_i, \sigma_i, f_i))\}_{i=1,\ldots,s} \subseteq Q$ be the set of entries in Q for which $\mathit{fid} = \mathit{fid}^*$.
The Adversary wins the game if $\mathbf{Verify}(vk, \mathit{fid}^*, m^*, \sigma^*, f^*) = 1$ and one of the following conditions hold:

1. *Q_{fid^*} is empty*
2. *f^* (interpreted as a vector) is in the span of $\{f_1, \ldots, f_s\}$ but, for any $\alpha_1, \ldots, \alpha_s$ such that $f = \sum_{i=1}^{s} \alpha_i f_i$, it holds $m^* \neq \prod_{i=1}^{s} m_i^{\alpha_i}$*
3. *f^* (interpreted as a vector) is not in the span of $\{f_1, \ldots, f_s\}$*

Finally we define the advantage $\mathbf{Adv}^{LHSG-RMA}(\mathcal{A})$ of $\mathcal{A}$ as the probability that $\mathcal{A}$ wins the game.

Now we give two more flavors of unforgeability for the LHSG scheme defined above. Roughly speaking first we extend the definition 19 to consider two stronger adversarial model (that we call *Known* Random Message Attack security and Chosen Message Attack security). It's very easy to check that the second security model is stronger than the first.

Definition 20 (Known Random Message Security) *Informally the KRMA security game is almost identical to the RMA game. The only difference concerns Signing queries*

Setup *The challenger runs* $\mathbf{KeyGen}(1^\lambda, n, k)$ *and gives* vk *to* $\mathcal{A}$*. The message space* $\mathcal{M}$ *and the signature space* Σ *are implicitly defined by the verification key.*

Queries $\mathcal{A}$ *can ask a polynomial number of queries of the following types:*

- **Signing Queries** $\mathcal{A}$ *asks for a new message/signature couple w.r.t. to a specific* $fid \in \mathcal{D}$ *and a specific index* $i \in \{1, \ldots, k\}$*. The challenger checks that this query has not been previously answered (otherwise it returns* $\perp$*), then it picks a random message* $m \xleftarrow{\$} \mathcal{M}$ *and uses the secret key* sk *to compute a signature* σ *for* m *w.r.t. fid and the index* i*. Finally it picks a handle* h *(from a proper set of identifiers), stores* $(h, (fid, m, \sigma, e_i))$ *in a table* T *and returns* h *and* m *to* $\mathcal{A}$*. Thus, in this case* $\mathcal{A}$ *actually knows the (random) message (but not the corresponding signature) being signed by the challenger.*
- **Derivation Queries** $\mathcal{A}$ *chooses a set of handles* $h = (h_1, \ldots, h_k)$ *and a vector of coefficients* $f = (\alpha_1, \ldots, \alpha_k)$*. The challenger retrieves* $\{(h_i, (fid_i, m_i, \sigma_i, f_i))\}_{i=1,\ldots k}$ *from* T *and returns* $\perp$ *if any of these does not exist or if* $fid_i \neq fid_j$ *for some* $i, j \in \{1, \ldots, k\}$ $(i \neq j)$*. Else, it computes* $m = \prod_{i=1}^{k} m_i^{\alpha_i}$*,* $\sigma = \mathbf{Eval}(vk, fid, f, \{\sigma_i\}_{i=1\ldots k})$*, chooses a handle* h*, stores* $(h, (fid, m, \sigma, \sum_{i=0}^{k} \alpha_i f_i))$ *in* T *and returns* h *to* $\mathcal{A}$*.*
- **Reveal Queries** $\mathcal{A}$ *chooses a handle* h*. If this handle is not in* T*, the challenger returns* $\perp$*. Otherwise it retrieves the corresponding*

record $(h, (fid, m, \sigma, f))$ *from table* T *and gives* (fid, m, σ, f) *to* $\mathcal{A}$. *Next it adds* $(h, (fid, m, \sigma, f))$ *to a different table* Q.

Forgery $\mathcal{A}$ *outputs a dataset identifier* fid^*, *a message* m^*, *a signature* σ^* *and a function* f^*.
Let $Q_{fid^*} = \{(h_i, (fid^*, m_i, \sigma_i, f_i))\}_{i=1,\dots,s} \subseteq Q$ *be the set of entries in* Q *for which* $fid = fid^*$.
The Adversary wins the game if $\mathbf{Verify}(vk, fid^*, m^*, \sigma^*, f^*) = 1$ *and one of the following conditions hold:*

1 Q_{fid^*} *is empty*

2 f^* *(interpreted as a vector) is in the span of* $\{f_1, \dots, f_s\}$ *but, for any* $\alpha_1, \dots, \alpha_s$ *such that* $f = \sum_{i=1}^{s} \alpha_i f_i$, *it holds* $m^* \neq \prod_{i=1}^{s} m_i^{\alpha_i}$

3 f^* *(interpreted as a vector) is not in the span of* $\{f_1, \dots, f_s\}$

Definition 21 (Chosen message attack security) *An LHSG is unforgeable against chosen message attack if for all* n, k *the advantage of any PPT adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* λ*:*

Setup *The challenger runs* $\mathbf{KeyGen}(1^\lambda, n, k)$ *and gives* vk *to* $\mathcal{A}$. *The message space* $\mathcal{M}$ *and the signature space* Σ *are implicitly defined by the verification key.*

Queries $\mathcal{A}$ *can ask a polynomial number of queries of the following types:*

- **Signing Queries** *When* $\mathcal{A}$ *asks for a signature on the triple* (fid, m, i) *(where fid is a file identifier,* $m \in \mathcal{M}$ *and* $i \in 1, \dots, k$*), the challenger first checks that no other signature of the form* $(fid, \cdot, i)$ *has been requested (if this is not the case, it returns* $\perp$*). Then it uses the secret key sk to compute a signature* σ *for* m *w.r.t. fid and the index* i. *Finally it picks a handle* h *(from a proper set of identifiers), stores* $(h, (fid, m, \sigma, e_i))$ *in a table* T *and returns* h.
- **Derivation Queries** $\mathcal{A}$ *chooses a set of handles* $h = (h_1, \dots, h_k)$ *and a set of coefficients* $f = (\alpha_1, \dots, \alpha_k)$. *The challenger retrieves* $\{(h_i, (fid_i, m_i, \sigma_i))\}_{i=1,\dots k}$ *from* T *and returns* $\perp$ *if any of these does not exists or if* $fid_i \neq fid_j$ *for some* $i, j \in 1, \dots, k$. *Else, it*

computes $m = \prod_{i=1}^{k} m_i^{\alpha_i}$, $\sigma = \mathbf{Eval}(vk, fid, f, \{\sigma_i\}_{i=1\ldots k})$, *chooses a handle* h, *stores* $(h, (fid, m, \sigma, \sum_{i=0}^{k} \alpha_i f_i))$ *in* T *and returns* h *to* $\mathcal{A}$.

- **Reveal Queries** $\mathcal{A}$ *chooses a handle* h. *If this handle is not in* T, *the challenger returns* $\perp$. *Otherwise it retrieves the corresponding record* $(h, (fid, m, \sigma, f))$ *from table* T *and gives* (fid, m, σ, f) *to* $\mathcal{A}$. *Next it adds* $(h, (fid, m, \sigma, f))$ *to a different table* Q.

Forgery $\mathcal{A}$ *outputs a dataset identifier* fid^*, *a message* m^*, *a signature* σ^* *and a function* f^*.
Let $Q_{fid^*} = \{(h_i, (fid^*, m_i, \sigma_i, f_i)\}_{i=1,\ldots,s} \subseteq Q$ *be the set of entries in* Q *for which* $fid = fid^*$.
The Adversary wins the game if $\mathbf{Verify}(vk, fid^*, m^*, \sigma^*, f^*) = 1$ *and one of the following conditions hold:*

1 Q_{fid^*} *is empty*

2 f^* *(interpreted as a vector) is in the span of* $\{f_1, \ldots, f_s\}$ *but, for any* $\alpha_1, \ldots, \alpha_s$ *such that* $f = \sum_{i=1}^{s} \alpha_i f_i$, *it holds* $m^* \neq \prod_{i=1}^{s} m_i^{\alpha_i}$

3 f^* *(interpreted as a vector) is not in the span of* $\{f_1, \ldots, f_s\}$

Finally we define the advantage $\mathbf{Adv}^{LHSG-CMA}(\mathcal{A})$ of $\mathcal{A}$ the probability that $\mathcal{A}$ wins the game.

3.4 A random message secure construction

Here we present a randomly secure instantiation for the case where $n = 1$, that is when the vectors in the message space have only one component. In section 3.6 we show how to to derive a fully secure scheme using the conversion methodology described in section 3.5[3]. Our construction uses as underlying building block a generic signature scheme.

[3]More precisely the scheme proposed in section 3.6 is a slightly optimized version of what one would get by naively converting our random message secure scheme. See section 3.6 for details.

Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map and $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$ a standard signature with message space $\mathcal{M}$. The scheme works as follows:

HKeyGen$(1^\lambda, 1, k)$: Choose a random generator $g \in \mathbb{G}$ and run **KeyGen**(1^λ) to obtain a signing key sk_1 and a verification key vk_1. Pick random $w \xleftarrow{\$} \mathbb{Z}_p$ and set $W \leftarrow g^w$. Select random group elements $h_1, \ldots, h_k, \xleftarrow{\$} \mathbb{G}$.

Set $\mathrm{vk} \leftarrow (\mathrm{vk}_1, g, W, h_1, \ldots, h_k)$ as the public verification key and $\mathrm{sk} = (\mathrm{sk}_1, w)$ as the secret signing key.

HSign$(\mathrm{sk}, m, \mathrm{fid}, i)$: This algorithm stores a list $\mathcal{L}$ of all previously returned dataset identifiers fid (together with the related secret information r and public information σ, τ defined below) and works according to the type of fid it is given in input):

If fid $\notin \mathcal{L}$, then choose $r \xleftarrow{\$} \mathbb{Z}_p$, set $\sigma \leftarrow g^r$, $\tau \leftarrow$ **Sign**$(\mathrm{sk}, \mathrm{fid}, \sigma)$

Else if fid $\in \mathcal{L}$, then retrieve the associated r, σ, τ from memory.

Then set $M \leftarrow m^w, V \leftarrow (h_i M)^r$ (if a signature for the same fid and the same index i was already issued, then abort). Finally output $\pi \leftarrow (\sigma, \tau, V, M)$ as a signature for m w.r.t. the function e_i (where e_i is the i-th vector of the canonical basis of $\mathbb{Z}^n$).

HVerify$(\mathrm{vk}, \pi, m, \mathrm{fid}, f)$: Parse the signature π as (σ, τ, V, M) and f as $(f_1, \ldots, f_k)$. Then check that:

$$\mathbf{Verify}(\mathrm{vk}, \tau, (\mathrm{fid}, \sigma)) = 1$$

$$e(M, g) = e(m, W)$$

$$e(V, g) = e(\prod_{i=1}^{k} h_i^{f_i} M, \sigma)$$

If all the above equations hold output 1, else output 0.

HEval $(\mathrm{vk}, \alpha, \pi_1, \ldots, \pi_k)$: Parse α as $(\alpha_1, \ldots, \alpha_k)$ and π_i as $(\sigma_i, \tau_i, V_i, M_i)$, $\forall i = 1, \ldots, k$. Then, compute $V \leftarrow \prod_{i=1}^{k} V_i^{\alpha_i}$, $M \leftarrow \prod_{i=1}^{k} M_i^{\alpha_i}$ and output $\pi = (\sigma_1, \tau_1, V, M)$ (or $\perp$ if the σ_i are not all equal).

The security of the scheme follows from the following theorem.

3.4.1 Scheme security

Theorem 4 *If the 2-3CDH assumption holds and $\mathcal{S}$ is a signature scheme unforgeable under adaptive chosen message attack then the scheme described above is a LHSG scheme secure against a random message attack according to definition* 19*.*

Proof. Proving correctness is straightforward, given the bilinear property of the pairing function. For what concerns security, we split the proof in 3 different cases. In each of them, we will show how an adversary that breaks the security of the scheme can be used to build a simulator that breaks the 2-3CDH assumption (or the security of the underlying signature scheme $\mathcal{S}$). In particular, let $m^*, \pi^* = (\mathrm{fid}^*, \sigma^*, \tau^*, V^*, M^*), f^*$ be the forgery returned by the adversary $\mathcal{A}$, Q the set of answers returned to $\mathcal{A}$ in response to its reveal queries, and let $Q_{fid^*} = \{(h_\eta, (fid^*, m_\eta, \pi_\eta, f_\eta))\}_{\eta=1,\ldots,\nu} \subseteq Q$ be the set of signatures seen by $\mathcal{A}$ for which the file identifier is fid^*, where $\pi_\eta = (\mathrm{fid}_\eta, \sigma_\eta, \tau_\eta, V_\eta, M_\eta)$.
Then (at least) one of the following conditions hold:

Case 1: Q_{fid^*} is empty, or $(\mathrm{fid}^*, \sigma^*) \neq (\mathrm{fid}_\eta, \sigma_\eta)$ for all $\eta = 1, \ldots, \nu$ (note that, by construction, all the signatures in Q_{fid^*} share the same σ^* component).
Case 2: $(\mathrm{fid}^*, \sigma^*) = (\mathrm{fid}_\eta, \sigma_\eta)$ for all $\eta = 1, \ldots, \nu$, f^* (interpreted as a vector) is in the span of $\{f_1, \ldots, f_\nu\}$ but, for any $\alpha_1, \ldots, \alpha_\nu$ such that $f = \sum_{\eta=1}^{\nu} \alpha_\eta f_\eta$, it holds $m^* \neq \prod_{\eta=1}^{\nu} m_\eta^{\alpha_\eta}$.
Case 3: $(\mathrm{fid}^*, \sigma^*) = (\mathrm{fid}_\eta, \sigma_\eta)$ for all $\eta = 1, \ldots, \nu$ and f^* (interpreted as a vector) is not in the span of $\{f_1, \ldots, f_\nu\}$.

As one can notice, the simulator can guess in which case he will be in advance with probability at least 1/3.

Case 1. In this case it is possible to reduce the security to the one of the underlying signature scheme $\mathcal{S}$. The simulator is quite simple: it uses its signing oracle for $\mathcal{S}$ to compute the component of each signature authenticating the fid (i.e. τ), and can easily compute the remaining parts of each signature by creating the rest of the secret and public key as in the real case. When $\mathcal{A}$ outputs a forgery, by definition of this case the simulator can output $((\text{fid}^*, \sigma^*), \tau^*)$ as a forgery for $\mathcal{S}$.
Case 2. First of all one can notice that, because the forgery must satisfy (in particular) the third and fourth verification equations, it must be that

$$M^* = (m^*)^w \quad \text{and} \quad V^* = \left(\prod_{i=1}^{k} h_i^{\alpha_i^*} M^* \right)^r$$

Moreover, the same two equations must also hold for the honestly computed signature for the function f^* on the messages signed by the challenger (we call $\overline{m}, \overline{\pi}$ such couple). So it must be that:

$$V^* \overline{V}^{-1} = \left(\prod_{i=1}^{k} M^* M_i^{-\alpha_i^*} \right)^r$$

If the left hand side of the equation is equal to 1 we don't have any forgery (in fact $M^* = \prod_{i=1}^{k} M_i^{\alpha_i}$ and $V^* = \overline{V}$).
Else, in the case when

$$m^* \prod_{i=1}^{k} m_i^{-\alpha_i^*} \neq 1$$

we describe a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break the 2-3CDH assumption. $\mathcal{B}$ works as follows. It takes in input a 2-3CDH tuple (g, g^w, g^r) and guesses the dataset identifier fid' for which it will receive a forgery[4].

Key Generation $\mathcal{B}$ initializes an empty table T (as described in the description of the security game), runs $(\text{sk}_1, \text{vk}_1) \leftarrow \textbf{KeyGen}(1^\lambda)$ and sets

[4] Note that the simulator does not need to predict the exact value of the identifier it will receive a forgery about, but only to pick one among the ones it will be asked to sign (for example, it might pick a random integer i from a large enough domain and choose fid' to be the i-th identifier it will be queried about). So the probability to guess correctly is not negligible and the reduction still works.

$W \leftarrow g^w$ (so w is implicitly part of the secret key). It selects $b_i \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1, \ldots, k$ and for each b_i it computes $m_i = g^{b_i}$, $h_i = g^{\delta_i} m_i^{-w}$, for random $\delta_1, \ldots, \delta_k \xleftarrow{\$} \mathbb{Z}_p$. Finally it gives $(\mathrm{vk}_1, g, W, h_1, \ldots, h_k)$ to $\mathcal{A}$.

Signing Queries To answer to the queries about the dataset fid' and index i from $\mathcal{A}$, $\mathcal{B}$ uses the previously created messages m_i and answers with the following.

if $\mathrm{fid}' \notin T$, it sets

$$\sigma = g^r,$$

$$\tau \leftarrow \mathbf{Sign}(\mathrm{sk}, \mathrm{fid}', \sigma),$$

and stores this data in memory.

if $\mathrm{fid}' \in T$, it retrieves the corresponding (σ, τ) from memory.

Then it sets $V \leftarrow \sigma^{\delta_i}$ and $M \leftarrow W^{b_i} = m_i^w$. By inspection, one can check that τ, M and V are correctly distributed as in the real case.

To answer the other queries with dataset identifier $\mathrm{fid} \neq \mathrm{fid}'$ w.r.t. index i it does the following.

if $\mathrm{fid} \notin T$, it chooses *fresh random* $r \xleftarrow{\$} \mathbb{Z}_p$ and sets

$$\sigma \leftarrow g^r,$$

$$\tau \leftarrow \mathbf{Sign}(\mathrm{sk}, \mathrm{fid}, \sigma),$$

and stores this data in memory.

if $\mathrm{fid} \in T$, it retrieves the corresponding (σ, r, τ) from memory.

Then it sets $m \leftarrow g^{b_i}$ for a random $b_i \xleftarrow{\$} \mathbb{Z}_p$, $V \leftarrow (h_i M_i)^r$, $M \leftarrow W^{b_i}$. In both cases, the signature is not directly returned to $\mathcal{A}$ but associated with a new handle h and stored in a table T.

Derivation and Reveal Queries are handled as in the real experiment.

Forgery Assume that the adversary $\mathcal{A}$ produced a forgery π^* for the function $f^* = (\alpha_1^*, \ldots, \alpha_k^*)$ w.r.t fid*. If fid$^* \neq$ fid$'$, then it aborts. Otherwise it proceeds as follows.
Considering the signature $\overline{\pi} = (\overline{\text{fid}}, \overline{\sigma}, \overline{\tau}, \overline{V}, \overline{M})$ for the message $\overline{m}$ and the function f^* (that the simulator can compute by the **Eval** algorithm from the function f^* provided by $\mathcal{A}$ and the messages m_i chosen by the simulator itself), we can and extract a 2-3CDH solution by the couple $\left(\frac{m^*}{\prod_{i=1}^k m_i^{\alpha_i^*}}, \frac{V^*}{\overline{V}}\right)$; in-fact the elements of the couple are not trivial by the definition of this subcase.

Case 3. In this case we can use exactly the same simulation of case 2, and assume, just to simplify the notation, that the adversary asks for exactly k signing queries (otherwise the simulator can just compute them on his own). In fact, since f^* is not in the span of the vectors $\{f_1, \ldots, f_\nu\}$, the probability that $f^*(m_1, \ldots, m_k) = m^*$ (where $m_1, \ldots, m_k$ are the vectors signed by the simulator in response to signing queries) is negligible and so we can extract a 2-3CDH solution as in the previous case. This is true because, in response to a signing query, the adversary is not even given the message that the simulator chooses at random, but only a handle. So the only information the adversary learns about those messages are the outputs of the reveal queries (where it can basically choose a vector $f = (\alpha_1, \ldots, \alpha_k)$ and learn m such that $m = \prod_{i=1}^k m_i^{\alpha_i}$. Therefore, by the definition of this case, $f^*(m_1, \ldots, m_k)$ is information theoretically hidden from the adversary, and it can only guess it with negligible probability.

Remark 4 *If the practical application allows the fid to be a group element and not simply a string, we can replace the signature $\mathcal{S}$ with a Structure preserving Signature satisfying the same hypothesis of theorem 4 to obtain the first example of a linearly homomorphic structure preserving signature scheme (LHSPS) where all parts of the signature are actually elements of the group (as opposed to [61], where the fid is inherently used as a bit string). In addition, if the identifier can be chosen at random by the signer and not by the adversary, we can even define σ to be the identifier itself and thus further improve efficiency. In practical instantiation it's possible to use the SPS of [4].*

3.5 From random message security to chosen message security

In this section we present a general transform to construct an LHSG secure against chosen message attack from one secure under random message attack. This transform comes in two flavours, depending on whether the underlying scheme is RMA secure or known RMA secure. In this latter case the conversion is totally generic. In the first case, on the other hand, the RMA secure scheme needs to satisfy some additional, but reasonable, requirements. In particular we require it to be *almost deterministic*. Informally, this means that given a file identifier $\text{fid} \in \mathcal{D}$ and a signature on a message m with respect to fid, the signature of any other $m' \in \mathcal{M}$ w.r.t. to any admissible function $f \in \mathcal{F}$ and the same fid is uniquely determined.

Remark 5 *We stress that while we present our theorems in the context of linearly homomorphic signatures (LHSG), if they are applied to linearly homomorphic structure preserving signatures, the structure preserving property is preserved.*

Let $\mathcal{S} = (\mathbf{HKeyGen}, \mathbf{HSign}, \mathbf{HVerify}, \mathbf{HEval})$ be a LHSG which is either known RMA-secure or RMA-secure and almost deterministic. The transformation below shows how to produce a new LHSG $\mathcal{T} = (\mathbf{TKeyGen}, \mathbf{TSign}, \mathbf{TVerify}, \mathbf{TEval})$ which is secure under CMA.

- $\mathbf{TKeyGen}(1^\lambda, n, k)$ takes as input the security parameter λ, the vector size n and an upper bound k for the number of messages signed in each dataset. It runs two times the **HKeyGen** algorithm to obtain $(\text{sk}_1, \text{vk}_1) \leftarrow \mathbf{HKeyGen}(1^\lambda, n, k)$ and $(\text{sk}_2, \text{vk}_2) \leftarrow \mathbf{HKeyGen}(1^\lambda, n, k)$. It outputs $\text{sk} = (\text{sk}_1, \text{sk}_2)$ as the secret signing key and $\text{vk} = (\text{vk}_1, \text{vk}_2)$ as the public verification key. The message space $\mathcal{M}$ is the same of $\mathcal{S}$.
- $\mathbf{TSign}(\text{sk}, \mathbf{m}, \text{fid}, i)$ It chooses random $\mathbf{m}_1 = (m_{1,1}, \ldots, m_{1,n}) \xleftarrow{\$} \mathcal{M}$ and computes $\mathbf{m}_2 \leftarrow \left(\frac{m_1}{m_{1,1}}, \ldots, \frac{m_n}{m_{1,n}}\right)$ (where $\mathbf{m} = (m_1, \ldots, m_n)$). Then it computes $\sigma_1 \leftarrow \mathbf{HSign}(\text{sk}_1, \mathbf{m}_1, i, \text{fid})$, $\sigma_2 \leftarrow \mathbf{HSign}(\text{sk}_2, \mathbf{m}_2, i, \text{fid})$ and outputs $\sigma = (\text{fid}, \mathbf{m}_1, \sigma_1, \sigma_2)$.

- **TVerify**$(\mathrm{vk}, \sigma, \mathbf{m}, \mathrm{fid}, f)$ parses σ as $(\mathrm{fid}, \mathbf{m}_1, \sigma_1, \sigma_2)$, computes $\mathbf{m}_2 \leftarrow \left(\frac{m_1}{m_{1,1}}, \ldots, \frac{m_n}{m_{1,n}}\right)$ and checks that the following equations hold:

$$\mathbf{HVerify}(\mathrm{vk}_i, m_i, \sigma_i, \mathrm{fid}, f) = 1 \quad \text{for} \quad i = 1, 2.$$

- **Eval**$(\mathrm{vk}, \mathrm{fid}, f, \{\sigma^{(i)}\}_{i=1\ldots k})$ parses $\sigma^{(i)}$ as $(\mathrm{fid}^{(i)}, \mathbf{m}_1^{(i)}, \sigma_1^{(i)}, \sigma_2^{(i)})$ and f as $(\alpha_1, \ldots, \alpha_k)$, then checks that $\mathrm{fid} = \mathrm{fid}^{(i)}$ for all i and, if not, aborts. Finally it sets

$$\sigma_1 \leftarrow \mathbf{HEval}(\mathrm{vk}_1, \mathrm{fid}, \{\sigma_1^{(i)}\}_{i=1\ldots k}, f),$$

$$\sigma_2 \leftarrow \mathbf{HEval}(\mathrm{vk}_2, \mathrm{fid}, \{\sigma_2^{(i)}\}_{i=1\ldots k}, f),$$

$$\mathbf{m}_1 = \left(\prod_{i=1}^{k} (m_{1,1}^{(i)})^{\alpha_i}, \ldots, \prod_{i=1}^{k} (m_{1,n}^{(i)})^{\alpha_i}\right)$$

and returns

$$\sigma \leftarrow (\mathrm{fid}, \mathbf{m}_1, \sigma_1, \sigma_2)$$

Theorem 5 *Suppose $\mathcal{S}$ is a LHSG secure against a random message attack with almost deterministic signatures. Moreover assume that the underlying message space is a group where one can efficiently solve systems of group equations. Then the scheme $\mathcal{T}$ described above is a LHSG secure against a chosen message attack.*

Proof. We prove the theorem by reducing the security of $\mathcal{T}$ to the one of $\mathcal{S}$, and showing how to build a simulator $\mathcal{B}$ that uses an adversary $\mathcal{A}$ against $\mathcal{T}$ to break the RMA security of $\mathcal{S}$.
First of all one can notice that, by construction, if $(\mathbf{m}^*, \pi^* = (\mathrm{fid}^*, \mathbf{m}_1^*, \pi_1^*, \pi_2^*), f^*)$ is a forgery for $\mathcal{T}$ then at least one between $(\mathbf{m}_1^*, \pi_1^*, f^*)$ (case 1) and $(\mathbf{m}^*/\mathbf{m}_1^*, \pi_2^*, f^*)$ (case 2) is a forgery for the corresponding instance of $\mathcal{S}$.
The simulator $\mathcal{B}$ works as follows:
It receives a public key vk' for an instance of $\mathcal{S}$ from its challenger. First of all it flips a coin to guess in which case he will be (as usual, his guess will be right with probability at least 1/2). Without loss of generality, we will describe the simulation in the case where its guess is case 1.

Setup $\mathcal{B}$ runs once the **HKeyGen** algorithm to obtain (sk_2, vk_2), sets $\mathrm{vk}_1 \leftarrow \mathrm{vk}'$ and gives $\mathrm{vk} = (\mathrm{vk}_1, \mathrm{vk}_2)$ to $\mathcal{A}$.

Signing Queries Each time $\mathcal{A}$ asks a query of the form $(\mathrm{fid}, \mathbf{m}, i)$, $\mathcal{B}$ forwards a query of the form (fid, i) to its challenger and gets back an handle h (if the challenger returns an error $\perp$, $\mathcal{B}$ simply forwards it to $\mathcal{A}$). Then it chooses a random message[5] $\mathbf{m}_2$, computes $\pi_2 \leftarrow \mathbf{Sign}(\mathrm{sk}_2, \mathbf{m}_2, \mathrm{fid}_2, i)$ and returns h to $\mathcal{A}$. The handle h, the messages $\mathbf{m}$ and $\mathbf{m}_2$, the signature π_2 and the index i are stored in a table T, like in the real experiment.

Derivation Queries In response to a derivation query $(h_1, \ldots, h_k, \mathbf{f})$, the simulator forwards the query to its challenger, and gets back a new handle h (or an error $\perp$, which gets forwarded to $\mathcal{A}$). Then it executes itself the query on the second part of the signature by computing $\pi^{(h)} \leftarrow \mathbf{Eval}(vk, \mathrm{fid}, f, \{\pi^{(h_i)}\}_{i=1,\ldots,k})$, computes the corresponding messages $\mathbf{m}^{(h)} = \prod_{i=1}^{k} (\mathbf{m}^{(h_i)})^{f_i}$, $\mathbf{m}_2^{(h)} = \prod_{i=1}^{k} (\mathbf{m}_2^{(h_i)})^{f_i}$ (the components $\mathbf{m}^{(h_i)}, \mathbf{m}_2^{(h_i)}, \pi^{(h_i)}$ corresponding to each handle h_i are retrieved from the table T). Finally, $\mathcal{B}$ gives h to $\mathcal{A}$ and stores the messages, signature, handles and function f in T.

Reveal Queries When $\mathcal{A}$ provides a handle h in a reveal query, $\mathcal{B}$ forwards the reveal query to the challenger. If the answer is $\perp$, $\mathcal{B}$ simply forwards it to $\mathcal{A}$. Otherwise, it gets a tuple $(\mathrm{fid}, \mathbf{m}_1, \pi_1, f)$. Since the adversary expects to receive a valid signature for a certain message $\mathbf{m}$ (that the simulator knows since it is stored in its own table T together with the handle h, the message $\mathbf{m}_2$ and signature π_2), it must now modify the table T in such a way that it is compliant with the information that the adversary has requested and the ones it has already

[5] We stress that, since the simulator does not know what random message $\mathbf{m}_1$ the challenger has chosen to sign, at this point there is no guarantee that $\mathbf{m} = \mathbf{m}_1\mathbf{m}_2$. However, the adversary only gets a random handle, and we will deal with this problem later.

obtained in the previous reveal queries. In particular, for each reveal query (associated with a function f), the adversary knows a message $\mathbf{m}_2$ such that, called $\mathbf{m}_2^{(1)}, \ldots, \mathbf{m}_2^{(k)}$ the messages corresponding to the second part of the signatures issued by the simulator in response to the signing queries for the same fid, it holds that $\mathbf{m}_2 = \prod_{i=1}^{k} (\mathbf{m}_2^{(h_i)})^{f_i}$. It can modify the table by choosing a random simultaneous solution for all these equations[6] (in the unknowns $\mathbf{m}_2^{(1)}, \ldots, \mathbf{m}_2^{(k)}$) and computing new signatures[7] for each entry in T (except for those who have been already given to the adversary) by either using the **Sign** or the **Eval** algorithm. Finally, it can compute $\mathbf{m}_1 = \mathbf{m}/\mathbf{m}_2$ and give the updated signature to $\mathcal{A}$ in response to the query.

Forgery Suppose $\mathcal{A}$ returns $\mathbf{m}^*, \pi^* = (\mathrm{fid}^*, \mathbf{m}_1^*, \pi_1^*, \pi_2^*), f^*$ as a valid forgery and that $\mathcal{B}$'s guess was correct. Then $\mathcal{B}$ can return $(\mathrm{fid}^*, \mathbf{m}_1^*, \pi_1^*, f^*)$ as a valid forgery against $\mathcal{S}$ to its challenger.

We remark that the proof above becomes much simpler if the simulator were allowed to know the messages signed by the challenger when answering signing queries. Formalizing this observation leads to the following theorem (whose proof is omitted):

Theorem 6 *If $\mathcal{S}$ is a LHSG secure against known random message attack the scheme $\mathcal{T}$ described above is a LHSG secure against a chosen message attack.*

[6] A solution always exists, since if the simulator was given the actual messages chosen by the challenger, he could set $m_2^{(i)} = \mathbf{m}/\mathbf{m}_1^i$ for all i. Moreover, we assumed that such a solution can be efficiently computed

[7] by the the property that the scheme is almost deterministic, the adversary cannot distinguish whether or not the signatures it has not seen have been modified during the game because for each message there is only one signature and therefore this signature does not contain any information about how it was generated.

3.6 A practical instantiation from our LHSG

One can notice that the previous construction can be instantiated using the LHSPS provided in section 3.4. For the sake of completeness, a slightly optimized version of such construction in presented here. Briefly, instead of sign twice the file identifier fid, it's possible to sign it appropriately just one time, reducing the key dimension and the signature dimension too. For this reason security proof for such scheme cannot be trivially derived from theorem 5.

Let $\mathbb{G}$, $\mathbb{G}_T$ be groups of prime order p such that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map,and let $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$ an LHSG with message space $\mathcal{M} = \mathbb{G}^3$.

HKeyGen$(1^\lambda, k)$**:**Runs **KeyGen**(1^λ) to obtain a signing key sk_1 and a verification key vk_1.
Picks random $g \in \mathbb{G}$ $w_1, w_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $W_1 \leftarrow g^{w_1}, W_2 \leftarrow g^{w_2}$.
Selects random group elements $h_1^{(1)}, \ldots, h_1^{(k)}, h_2^{(1)}, \ldots, h_2^{(k)} \xleftarrow{\$} \mathbb{G}$.
Sets $\mathrm{vk} \leftarrow (vk_1, g, W_1, W_2, A_0, \ldots, A_l, h_1^{(1)}, \ldots, h_1^{(k)}, h_2^{(1)}, \ldots, h_2^{(k)})$ as the public verification key and $\mathrm{sk} = (sk_1, w_1, w_2)$ as the secret signing key.

HSign(sk$, m,$ **fid**$, i)$**:** Stores a list $\mathcal{L}$ of all previously returned dataset identifiers fid (together with the related secret information r and public information σ, τ defined below) and works according to the type of fid it is given in input:

If fid $\notin \mathcal{L}$, then it chooses $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $\sigma_1 \leftarrow g^{r_1}$, $\sigma_2 \leftarrow g^{r_2}$, $\tau \leftarrow$ **Sign**$(\mathrm{sk}_1, \mathrm{fid}, \sigma_1, \sigma_2)$.

Else if fid $\in \mathcal{L}$, then it retrieves the associated $r_1, r_2, \sigma_1, \sigma_2, \tau$ from memory.

The message m to be signed is written as $m_1 m_2$ by choosing random $m_1 \xleftarrow{\$} \mathcal{M}$ and computing $m_2 \leftarrow m(m_1)^{-1}$. Then it sets $M_1 \leftarrow {m_1}^{w_1}, M_2 \leftarrow {m_2}^{w_2}, T_1 \leftarrow (h_i^{(1)} M_1)^{r_1}, T_2 \leftarrow (h_i^{(2)} M_2)^{r_2}$ (if a signature for the same fid and the same index i was already issued, then it aborts). Finally it outputs the signature **Sign** $\leftarrow (\mathrm{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, e_j)$, where e_i is the i-th vector of the canonical base of $\mathbb{Z}^k$

HVerify $(\mathrm{vk}, m, \mathbf{Sign}, \vec{f})$: Parses the signature **Sign** as $(\mathrm{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2)$ and $\vec{f}$ as $(f_1, \ldots, f_k)$, computes $m_2 \leftarrow m m_1^{-1}$. Then it checks that:

$$\mathbf{HVerify}(\mathrm{vk}_1, \mathrm{fid}, \sigma_1, \sigma_2) = 1$$

$$e(M_1, g) = e(m_1, W_1)$$

$$e(M_2, g) = e(m_2, W_2)$$

$$e(T_1, g) = e(\prod_{i=1}^{k} {h_1^{(i)}}^{f_i} M_1, \sigma_1)$$

$$e(T_2, g) = e(\prod_{i=1}^{k} {h_2^{(i)}}^{f_i} M_2, \sigma_2)$$

If all the above equations hold outputs 1, else outputs 0.

HEval $(\mathrm{vk}, \vec{\alpha}, \mathbf{Sign}_1, \ldots, \mathbf{Sign}_k)$: Parse $\vec{\alpha}$ as $(\alpha_1, \ldots, \alpha_k)$ and $\mathbf{Sign}_i$ as

$$(\mathrm{fid}_i, \sigma_1^{(i)}, \sigma_2^{(i)}, \tau^{(i)}, T_1^{(i)}, T_2^{(i)}, m_1^{(i)}, M_1^{(i)}, M_2^{(i)}, s_i)\ \forall i = 1, \ldots, k.$$

Then check that all $\mathbf{Sign}_i$ share the same $\mathrm{fid}, \sigma_1, \sigma_2, \tau$ components and, if not, reject. Otherwise, compute $T_1 \leftarrow \prod_{i=1}^{k} {T_1^{(i)}}^{\alpha_i}, T_2 \leftarrow \prod_{i=1}^{k} {T_2^{(i)}}^{\alpha_i}$, $M_1 \leftarrow \prod_{i=1}^{k} {M_1^{(i)}}^{\alpha_i}, M_2 \leftarrow \prod_{i=1}^{k} {M_2^{(i)}}^{\alpha_i}$ and $m_1 \leftarrow \prod_{i=1}^{k} {m_1^{(i)}}^{\alpha_i}$. Finally output $\mathbf{Sign} = (\mathrm{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2)$.

Theorem 7 *If 2-3CDH and the discrete logarithm (DL) assumptions hold and* **Sign** *is a randomly secure LHSG scheme, then the scheme described above is an LHSG scheme unforgeable against chosen message attack.*

Proof. Let $m^*, \mathbf{Sign}^* = (\mathrm{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, T_1^*, T_2^*, m_1^*, M_1^*, M_2^*)$, f^* be the forgery returned by the adversary $\mathcal{A}$, Q the set of answers returned to $\mathcal{A}$ in response to its reveal queries, and let $Q_{fid^*} = \{(h_\eta, (fid^*, m_\eta, \sigma_\eta, f_\eta))\}_{\eta=1,\ldots,\nu} \subseteq Q$ be the set of signatures seen by $\mathcal{A}$ for which $\mathrm{fid} = \mathrm{fid}^*$.
Then (at least) one of the following conditions hold:

Case 1: $(\mathrm{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*) \neq (\mathrm{fid}^{(\eta)}, \sigma_1^{(\eta)}, \sigma_2^{(\eta)}, \tau_\eta)$ for all $\eta = 1, \ldots, \nu$
Case 2: Q_{fid^*} is not empty, $(\sigma_1^*, \sigma_2^*, \tau^*) = (\sigma_1^{(\eta)}, \sigma_2^{(\eta)}, \tau_\eta)$ for all $\eta = 1, \ldots, \nu$,

the function f^* (interpreted as a vector) is in the span of $\{f_1, \ldots, f_\nu\}$ but, for any $\alpha_1, \ldots, \alpha_\nu$ such that $f = \sum_{\eta=1}^{\nu} \alpha_\eta f_\eta$, it holds $m^* \neq \prod_{\eta=1}^{\nu} m_\eta^{\alpha_\eta}$.
Case 3: Q_{fid^*} is not empty, $(\sigma_1^*, \sigma_2^*, \tau^*) = (\sigma_1^{(\eta)}, \sigma_2^{(\eta)}, \tau_\eta)$ for all $\eta = 1, \ldots, \nu$ and f^* (interpreted as a vector) is not in the span of $\{f_1, \ldots, f_\nu\}$.
As one can notice, the simulator can guess in which case he will be in advance with probability at least 1/3.
Case 1. In this case we construct a simulator $\mathcal{B}$ that solves CDH using an adversary $\mathcal{A}$ against the signature scheme described above. The simulator receives as input (g, g^a, g^b) (for a and b he does not know), and behaves as follows:

Key Generation It sets $g_1 \leftarrow g^a, g_2 \leftarrow g_b$ (thus implicitly defining part of the secret key as $g_2^a = g^{ab}$) and picks an hash function $H_K \xleftarrow{\$} \mathcal{H}$. Then it chooses $w_1, w_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $W_1 \leftarrow g^{w_1}, W_2 \leftarrow g^{w_2}$, selects random group elements $h_1^{(1)} = g^{l_1^{(1)}}, \ldots, h_1^{(k)} = g^{l_1^{(k)}}, h_2^{(1)} = g^{l_2^{(1)}}, \ldots, h_2^{(k)} = g^{l_2^{(k)}}, h = g^\ell \in \mathbb{G}$ for random $\ell, l_1^{(1)}, \ldots, l_1^{(k)}, l_2^{(1)}, \ldots, l_2^{(k)} \in \mathbb{Z}_p$.
Next it chooses $A_0, A_1, \ldots, A_l$ in the same way as in security proof of Waters' signature [74]. Because of this choice, there exist two functions $J, K : \{0,1\}^l \to \mathbb{Z}$ (these functions are all kept internal to the simulator) hidden to the adversary such that, for any string $\text{fid} \in \{0,1\}^l$, the expression $Y(\text{fid}) \stackrel{\text{def}}{=} A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\text{fid}]_\zeta}$ can be written as $Y(\text{fid}) = g_2^{J(\text{fid})} g^{K(\text{fid})}$. In addition, it was proven that for any distinct $\tau, \tau_1, \ldots, \tau_q \in \{0,1\}^l$ we will have $J(\tau) = 0 \mod p$ and $J(\tau_i) \neq 0 \, \forall i \in \{1, \ldots, q\}$ with non negligible probability $\eta = \frac{1}{8q(l+1)}$.
Finally, $\mathcal{B}$ creates two empty tables T and Q (used to store the output of signing and reveal queries, as explained in the security definition) and gives vk to $\mathcal{A}$.

Signing queries When $\mathcal{A}$ asks a new signing query (m, fid, i) for a message m w.r.t. dataset identifier fid and index i, $\mathcal{B}$ does the following:

if fid $\in T$, it retrieves the corresponding $(\sigma_1, \sigma_2, r, \tau, s)$ from memory.

if fid $\notin T$, it chooses a random $\gamma \in \mathbb{Z}_p$ and sets $\overline{\text{fid}} \leftarrow H_K(g^\gamma)$. if $J(\overline{\text{fid}}) = 0 \mod p$ it aborts. Else it chooses random $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets[8]

$$\tau \leftarrow (Y(\overline{\text{fid}}))^{r_1+r_2} g_1^{-\frac{K(\overline{\text{fid}})}{J(\overline{\text{fid}})}}$$

$$\sigma_1 \leftarrow g^{r_1} g_1^{-\frac{1}{J(\overline{\text{fid}})}}; \quad \sigma_2 \leftarrow g^{r_2}$$

Then $\mathcal{B}$ sets $t \leftarrow H_K(\text{fid}\|\sigma_1\|\sigma_2)$ and $s \leftarrow \frac{\gamma - t}{\ell}$

The rest of the signature is computed as follows. First $\mathcal{B}$ chooses a random $\lambda \xleftarrow{\$} \mathbb{Z}_p$ and sets $m_1 = g^\lambda$, $m_2 = m/m_1$. Then it sets $M_1 \leftarrow m_1^{w_1}$, $M_2 \leftarrow m_2^{w_2}$, $T_1 \leftarrow \sigma_1^{l_1^{(i)} + \lambda_1 w} = \left(h_1^{(i)} M_1\right)^{r'_1}$, $T_2 \leftarrow \left(h_2^{(i)} M_2\right)^{r'_2}$. The signature $(\text{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s)$ and the message m are not directly returned to $\mathcal{A}$, but associated with a new handle h (together with the trivial function e_i) and stored in the table T.

Derivation and Reveal Queries are handled as in the real experiment

Forgery Once $\mathcal{A}$ provides a forgery $\mathbf{Sign}^* = (\text{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, T_1^*, T_2^*, m_1^*, M_1^*, M_2^*, s^*)$ $\mathcal{B}$ computes $J(\overline{\text{fid}}^*)$ and aborts if $J(\overline{\text{fid}}^*) \neq 0$

From this forgery, $\mathcal{A}$ can extract a CDH solution as follows. First, notice that by correctness the components τ^*,σ_1^* and σ_2^* of the forgery will be of the form

$$\tau^* = g^{ab} \left(Y(\overline{\text{fid}}^*)\right)^{r'_1+r'_2} \quad \sigma_1^* = g^{r'_1} \quad \sigma_2^* = g^{r'_2}$$

Thus the solution of the CDH instance can be computed as

$$g^{ab} = \tau / (\sigma_1^* \sigma_2^*)^{K(\overline{\text{fid}^*})}$$

Case 2. In this case, the adversary produces a forgery for a fid it has seen a signature about, but the part of the forgery used to verify the fid, namely $(\sigma_1^*, \sigma_2^*, \tau^*, s^*)$, is different from what the simulator stored in the table Q (to fix the notation, we will assume $(m, \mathbf{Sign} = (\text{fid}^*, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s), f)$ is recorded in Q during the simulation).

[8] These values are correctly distributed, as one can easily check that they can be written as $\tau = g^{ab}(Y(\overline{\text{fid}}))^{r'_1+r'_2}$, $\sigma_1 = g^{r'_1}, \sigma_1 = g^{r'_2}$ where $r'_1 = r_1 - \frac{a}{J(\overline{\text{fid}})}$, $r'_2 = r_2$.

Let $t^* \leftarrow H_K(\text{fid}^*\|\sigma_1^*\|\sigma_2^*), \overline{\text{fid}}^* \leftarrow H_K(g^{H_K(\text{fid}^*\|\sigma_1^*\|\sigma_2^*)}h^{s^*})$,and let $t, \overline{\text{fid}}$ the corresponding values computed from $(m, \textbf{Sign})$ (as in the real experiment, signatures in Q_{fid^*} will all lead to the same values). Depending on these quantities, we have three different sub-cases:

2.a $\overline{\text{fid}^*} = \overline{\text{fid}}$ and $t^* = t$

2.b $\overline{\text{fid}^*} = \overline{\text{fid}}$ and $t^* \neq t$

2.c $\overline{\text{fid}^*} \neq \overline{\text{fid}}$

Case 2.a. It is easy to build a simulator against the collision resistance of $\mathcal{H}$. Namely, the simulator $\mathcal{B}$ receives in input an hash key k' and has to come up with a couple of elements (y_1, y_2) such that $H_{k'}(y_1) = H_{k'}(y_2)$. Supposing he will get a forgery in this sub-case, $\mathcal{B}$ can just run the ideal experiment but set $K \leftarrow k'$ as the hashing key inside vk. When $\mathcal{A}$ outputs a forgery $(\text{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, T_1^*, T_2^*, m_1^*, M_1^*, M_2^*, s^*)$, by the fact that $\overline{\text{fid}^*} = \overline{\text{fid}}$, we have $H_K(g^{t^*}h^{s^*}) = H_K(g^t h^s)$. So if $s \neq s^*$, because $t = t^*$, we already have a collision (here we require that each element of the group $\mathbb{G}$ and each value in $\mathbb{Z}_p$ have a unique encoding). Otherwise, if $s = s^*$, it must be that $(\sigma_1^*, \sigma_2^*) \neq (\sigma_1, \sigma_2)$ (if this was not the case, then it must be that $\tau^* = \tau$ and therefore this cannot be a case 2 forgery).
So we have that $H_K(\text{fid}^*\|\sigma_1^*\|\sigma_2^*) = t^* = t = H_K(\text{fid}^*\|\sigma_1\|\sigma_2)$ but $\text{fid}^*\|\sigma_1^*\|\sigma_2^* \neq \text{fid}^*\|\sigma_1\|\sigma_2$, and $\mathcal{B}$ can return those values as a collision against the hash function.

Case 2.b. In this case, we can build a simulator $\mathcal{B}$ that breaks the DL problem. $\mathcal{B}$ receives in input a couple (g', h'), and its goal is to output β such that $g'^{\beta} = h'$. $\mathcal{B}$ can run the simulation as follows:

Key Generation $\mathcal{B}$ sets $g \leftarrow g'$, $h \leftarrow h'$, and computes the other elements of the public key vk as in the real case. Then it gives vk to $\mathcal{A}$ and stores the secret key sk.

Queries All types of queries are handled as in the real experiment.

Forgery Suppose $\mathcal{A}$ returns a type 2.*b* forgery. Then it must be that $H_K(g^{t^*}h^{s^*}) = H_K(g^t h^s)$ and $t^* \neq t$. If $g^{t^*}h^{s^*} \neq g^t h^s$ then we have

a collision for H_K (and we can run a simulation similar to the previous case). If $g^{t^*}h^{s^*} = g^t h^s$, then $\mathcal{B}$ can return $\beta = \frac{t-t^*}{s^*-s}$ as a solution for the DL instance (note that it can't be $s^* = s$ because otherwise $t^* = t$).

Case 2.c. Suppose $\mathcal{A}$ returns a $2.c$ type forgery. In this case, we want to reduce the security of this scheme to the one of Waters' weak signature scheme, by showing how to construct a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break that scheme. $\mathcal{B}$ receives in input a public key $\text{vk} = (g, g_1, g_2, A_0, A_1, \ldots, A_l)$ for Waters' weak signature scheme. It needs to output a valid forgery for this scheme.

Key Generation $\mathcal{B}$ chooses $H_K \leftarrow \mathcal{H}$, $a \xleftarrow{\$} \mathbb{Z}_p$ and sets $h \leftarrow g^a$. Then it chooses $w_1, w_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $W_1 \leftarrow g^{w_1}, W_2 \leftarrow g^{w_2}$, selects random $l_1^{(1)}, \ldots, l_1^{(k)}, l_2^{(1)}, \ldots, l_2^{(k)} \in \mathbb{Z}_p$ and sets $h_1^{(1)} \leftarrow g^{l_1^{(1)}}, \ldots, h_1^{(k)} \leftarrow g^{l_1^{(k)}}, h_2^{(1)} \leftarrow g^{l_2^{(1)}}, \ldots, h_2^{(k)} \leftarrow g^{l_2^{(k)}}$. Finally a $\mathcal{B}$ gives to $\mathcal{A}$ the public key $\text{vk}_1 = (g, g_1, g_2, W_1, W_2, A_0, \ldots, A_l, h_1^{(1)}, \ldots, h_1^{(k)}, h_2^{(1)}, \ldots, h_2^{(k)}, K)$.

Signing Queries Each time $\mathcal{A}$ asks for a new query (m, fid, i) on a message m w.r.t. dataset fid and index i, $\mathcal{B}$ responds in this way.

if fid $\in T$ it retrieves the corresponding $(\sigma_1, \sigma_2, \tau, s)$ from the memory.

if fid $\notin T$ it sets $\overline{\text{fid}} = H_K(g^\beta)$ for $\beta \xleftarrow{\$} \mathbb{Z}_p$; then it asks its challenger for a signature on $\overline{\text{fid}}$ and receives (σ_1, τ_1). Next it chooses a random $r_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets $\sigma_2 \leftarrow g^{r_2}$ $t \leftarrow H_k(\text{fid} \| \sigma_1 \| \sigma_2)$, $s \leftarrow \frac{\beta - t}{a}$. Then it chooses $\lambda \xleftarrow{\$} \mathbb{Z}_p$ and sets:

$$m_1 \leftarrow g^\lambda; \quad m_2 \leftarrow m/m_1$$

$$M_1 \leftarrow (m_1)^{w_1}; \quad M_2 \leftarrow (m_2)^{w_2}$$

$$T_1 \leftarrow \sigma_1^{l_1^{(i)} + \lambda w_1}; \quad T_2 \leftarrow (h_2^{(i)} M_2)^{r_2}$$

$$\tau \leftarrow \tau_1 (A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\text{fid}}]_\zeta})^{r_2}.$$

As in the real experiment, the signature $(\text{fid}, \sigma_1, \sigma_2, \tau, T_1, T_2, m_1, M_1, M_2, s)$ is stored in the table T together with an handle h which is returned to $\mathcal{A}$.

Derivation and Reveal Queries are handled as in the real experiment.

Forgery When $\mathcal{A}$ returns a type 2.*c* forgery $(\text{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, T_1^*, T_2^*, m_1^*, M_1^*, M_2^*, s^*)$, $\mathcal{B}$ computes $t^* \leftarrow H_k(\text{fid}^*, \|\sigma_1^*\|\sigma_2^*)$, $\overline{\text{fid}^*} \leftarrow H(g^t h^{s^*})$ and outputs $(\overline{\text{fid}^*}, \sigma_1^*\sigma_2^*, \tau^*)$ as a forgery against Waters' scheme.

We stress that this is a valid forgery for the signature scheme, as no other signature has been requested by the simulator for $\overline{\text{fid}^*}$. In fact, by definition of this subcase we have that $\overline{\text{fid}^*} \neq \overline{\text{fid}}$ (where $\overline{\text{fid}}$ is the one computed from the signatures in Q_{fid^*}). In addition, if there were another $\text{fid}' \in Q$ such that $\overline{\text{fid}^*} = \overline{\text{fid}'}$ then it would be trivial to find a collision for H_K.

Case 3. First of all one can notice that, by definition of a valid forgery, it must be that

$$M_1^* = (m_1^*)^{w_1} \quad \text{and} \quad T_1^* = \left(M_1^* \prod_{i=1}^{k} (h_1^{(i)})^{\alpha_i^*} \right)^{r_1}$$

$$M_2^* = (m_2^*)^{w_2} \quad \text{and} \quad T_2^* = \left(M_2^* \prod_{i=1}^{k} (h_2^{(i)})^{\alpha_i^*} \right)^{r_2}$$

Moreover, the same two equations must also hold for the honestly computed signature for the function f^* computed on the messages originally signed by the simulator (we call $m = \prod_{i=1}^{k} (m^{(i)})^{\alpha_i^*}, \mathbf{Sign}$ such couple, and $(m^{(i)}, \mathbf{Sign}^{(i)})$ each of the message/signature made by the simulator in response to a query for index i and dataset fid^*) . So it must be that:

$$T_1^* T_1^{-1} = \left(M_1^* \prod_{i=1}^{k} M_1^{(i)^{-\alpha_i^*}} \right)^{r_1}, \quad T_2^* T_2^{-1} = \left(M_2^* \prod_{i=1}^{k} M_2^{(i)^{-\alpha_i^*}} \right)^{r_2} \tag{3.1}$$

By the assumption of this subcase, it cannot be that both the left hand sides of these equations are 1. In fact

$$m_1^* m_2^* = m^* \neq m = m_1 m_2 = \prod_{i=1}^{k} \left(m_1^{(i)}\right)^{\alpha_i^*} \left(m_2^{(i)}\right)^{\alpha_i^*}$$

and therefore

$$m_b^* \prod_{i=1}^{k} m_b^{(i)^{-\alpha_i^*}} \neq 1$$

for at least one value of $b \in \{1,2\}$. In this case, we describe a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break the 2-3CDH assumption. $\mathcal{B}$ works as follows. It takes in input a 2-3CDH tuple (g, g^w, g^r) and guesses[9] the dataset identifier fid' for which it will receive a forgery and the value $b \in \{1,2\}$ for which the previous inequality will hold. For the sake of simplicity (and wlog), in the following we will assume it chooses bit $b = 1$

Key Generation $\mathcal{B}$ chooses a random hash key $H_K \xleftarrow{\$} \mathcal{H}$ and random elements $g_2, h \xleftarrow{\$} \mathbb{G}$, sets $W_1 \leftarrow g^w$ (so $w_1 = w$ is not known by the simulator but is implicitly part of the secret key), $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and $g_1 \leftarrow g^\alpha$. Next it chooses $b_i \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1, \ldots, k$ and for each b_i it computes $m_1^{(i)} \leftarrow g^{b_i}$, $M_1^{(i)} \leftarrow {m_1^{(i)}}^{w_1} = {W_1}^{b_i}$, $h_1^{(i)} \leftarrow g^{\delta_i} {m_1^{(i)}}^{-w_1}$, for random $\delta_1, \ldots, \delta_k \xleftarrow{\$} \mathbb{Z}_p$. Finally it picks random $a_0, a_1, \ldots, a_l \xleftarrow{\$} \mathbb{Z}_p$ and defines $A_i \leftarrow g^{a_i}$, $i = 0, \ldots, l$. The other parts of the public key are generated as in the real experiment. Finally $\mathcal{B}$ gives $\text{vk} = (g, g_1, g_2, W_1, W_2, A_0, \ldots, A_l, h, h_1^{(1)}, \ldots, h_1^{(k)}, h_2^{(1)}, \ldots, h_2^{(k)}, K)$ to $\mathcal{A}$ and stores all the other computed values in memory.

Signing Queries To answer to the queries $(m^{(i)}, \text{fid}', i)$ about identifier fid' and index i asked by $\mathcal{A}$, $\mathcal{B}$ works as follows.

First, if this is the first query asked for identifier fid' by $\mathcal{A}$, it chooses random $s, r_2 \xleftarrow{\$} \mathbb{Z}_p$, sets $\sigma_1 \leftarrow g^r$ (note that it does not know r) and computes

$$\sigma_2 \leftarrow g^{r_2}, \quad t \leftarrow H_K(\text{fid}' \| \sigma_1 \| \sigma_2), \quad \overline{\text{fid}'} \leftarrow H_K(g^t h^s),$$

$$\tau \leftarrow g_2^\alpha \left(\sigma_1^{a_0} \prod_{\zeta=1}^{l} \sigma^{a_\zeta [\overline{\text{fid}'}]_\zeta} \right) \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\text{fid}'}]_\zeta} \right)^{r_2} = g_2^\alpha \left(A_0 \prod_{\zeta=1}^{l} A_\zeta^{[\overline{\text{fid}'}]_\zeta} \right)^{r+r_2}.$$

Otherwise, it retrieves all this information from memory.

Then, it fetches the values $m_1^{(i)}, M_1^{(i)}$ generated in the previous phase from memory and computes $m_2^{(i)} \leftarrow m^{(i)} / m_1^{(i)}$ (so that $m^{(i)} = m_1^{(i)} m_2^{(i)}$), $M_2^{(i)} \leftarrow m_2^{(i)}$, $T_1^{(i)} \leftarrow \sigma_1^{\delta_i}$, $T_2^{(i)} \leftarrow (h_2^{(i)} M_2^{(i)})^{r_2}$ (it is easy to check

[9] The probability of guessing correctly is polynomial. See footnote 4 for details.

that the signature is valid and each of its components is correctly distributed).

To answer queries $(m^{(i)}, \text{fid}, i)$ about an identifier $\text{fid} \neq \text{fid}'$, $\mathcal{B}$ works in a different way.
First, if this is the first query asked for identifier fid by $\mathcal{A}$, it chooses random $s, r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$ and sets

$$\sigma_1 \leftarrow g^{r_1}, \quad \sigma_2 \leftarrow g^{r_2},$$

$$t \leftarrow H_K(\text{fid}\|\sigma_1\|\sigma_2), \quad \overline{\text{fid}} \leftarrow H_K(g^t h^s), \quad \tau \leftarrow g_2^{\alpha}(A_0 \prod_{\zeta=1}^{l} A_{\zeta}^{[\overline{\text{fid}}]_\zeta})^{r_1+r_2}$$

Otherwise, it retrieves all this information from memory.
Then, it chooses $c \xleftarrow{\$} \mathbb{Z}_p$, and computes $m_1^{(i)} \leftarrow g^c, m_2^{(i)} \leftarrow m^{(i)}/m_1^{(i)}$ (so that $m^{(i)} = m_1^{(i)} m_2^{(i)}$),

$$M_1^{(i)} \leftarrow W_1^c, \quad M_2^{(i)} \leftarrow (m_2^{(i)})^{w_2}, \quad T_1^{(i)} \leftarrow (h_1^{(i)} M_1^{(i)})^{r_1}, \quad T_2^{(i)} \leftarrow (h_2^{(i)} M_2^{(i)})^{r_2}.$$

In both cases, the signatures are not directly returned to $\mathcal{A}$ but associated with a new handle h and stored in a table T.

Derivation and Reveal Queries are handled as in the real experiment.

Forgery Assume that the adversary $\mathcal{A}$ produced a forgery $\mathbf{Sign}^*$ for the function $f^* = (\alpha_1^*, \ldots, \alpha_k^*)$ and the identifier fid^*. If fid^* was not guessed correctly, $\mathcal{B}$ aborts.
Otherwise it proceeds as follows.
Consider the signature $\overline{\mathbf{Sign}} = (\text{fid}^*, \sigma_1^*, \sigma_2^*, \tau^*, \overline{T_1}, \overline{T_2}, \overline{m_1}, \overline{M_1}, \overline{M_2}, s^*)$ for the function f^* and the message $\overline{m} = \prod_{i=1}^{k} (m^{(i)})^{\alpha_i^*}$ computed using the **Eval** algorithm on the couples $(m^{(i)}, \mathbf{Sign}^{(i)})$ stored in T in response to the signing queries made by $\mathcal{A}$. Then, by the assumption of this subcase and supposing $\mathcal{B}$ guessed the correct index b (otherwise it aborts), it must be that $m_b^* \prod_{i=1}^{k} (m_b^{(i)})^{-\alpha_i^*} \neq 1$). Therefore $\mathcal{B}$ can extract a 2-out-of-3 CDH solution by computing $\left(\frac{m_b^*}{\prod_{i=1}^{k} (m_b^{(i)})^{\alpha_i^*}}, \frac{T_b^*}{\overline{T_b}} \right)$ (this can be easily verified by recalling equation 3.1).

Case 4. The idea to handle this case is the same as the one used in the analogous case 4 of theorem 4. Basically, we use the same simulator of case 3, because the probability that $f^*(m_b^{(1)}, \ldots, m_b^{(k)}) = m_b^*$ is negligible (as in this case $f^*(m_b^{(1)}, \ldots, m_b^{(k)})$ is information theoretically hidden from the adversary).

Chapter 4

(Publicly) Verifiable delegation of computation on outsourced ciphertexts

This chapter contains the main result of this thesis: we introduce a new primitive that we call Linearly Homomorphic Authenticated Encryption with Public Verifiability (LAEPuV). This is done by essentially adapting the general definition of Joo and Yun [56] of homomorphic authenticated encryption to the linear case and adding the useful requirement of public verifiability. We also propose an instantiation of this primitive supporting Paillier's scheme as the underlying encryption mechanism.

4.1 Definition and security

Definition 22 (LAEPuV) *A LAEPuV scheme is a tuple of 5 PPT algorithms (***AKeyGen***,* **AEncrypt***,* **ADecrypt***,* **AVerify***,* **AEval***) such that:*

- **AKeyGen**$(1^\lambda, k)$ *takes as input the security parameter* λ*, and an upper bound* k *for the number of messages encrypted in each dataset. It outputs a secret key* sk *and a public key* vk *(used for function evaluation and verification); the public key implicitly defines a message space* $\mathcal{M}$ *which is also a group, a file identifier space* $\mathcal{D}$ *and a ciphertext space*

$\mathcal{C}$.

- **AEncrypt**(sk, fid, i, m) *is a probabilistic algorithm which takes as input the secret key, an element* $m \in \mathcal{M}$*, a dataset identifier fid, an index* $i \in \{1, \ldots, k\}$ *and outputs a ciphertext* c*.*

- **AVerify**(vk, fid, c, f) *takes as input the pubic key* vk*, a ciphertext* $c \in \mathcal{C}$*, an identifier fid* $\in \mathcal{D}$ *and* $f \in \mathcal{F}$*. It return 1 (accepts) or 0 (rejects).*

- **ADecrypt**(sk, fid, c, f) *takes as input the secret key* sk*, a ciphertext* $c \in \mathcal{C}$*, an identifier fid* $\in \mathcal{D}$ *and* $f \in \mathcal{F}$ *and outputs* $m \in \mathcal{M}$ *or* $\perp$ *(if* c *is not considered valid).*

- **AEval**$(vk, f, fid, \{c_i\}_{i=1\ldots k})$ *takes as input the public key* vk*, an admissible function* f *in its vector form* $(\alpha_1, \ldots, \alpha_k)$*, an identifier fid, a set of* k *ciphertexts* $\{c_i\}_{i=1\ldots k}$ *and outputs a ciphertext* $c \in \mathcal{C}$*. Note that this algorithm should also work if less than* k *signatures are provided, as long as their respective coefficients in the function* f *are 0, but we don't explicitly account this to avoid heavy notation.*

The correctness conditions of our scheme are the following:

- *For any* $(sk, vk) \leftarrow$ **AKeyGen**$(1^\lambda, k)$ *honestly generated keypair, any* $m \in \mathcal{M}$*, any dataset identifier fid and any* $i \in \{1, \ldots, k\}$*, with overwhelming probability*

$$\textbf{ADecrypt}(sk, fid, \textbf{AEncrypt}(sk, fid, i, m), e_i) = m$$

  where e_i *is the* i*-th vector of the standard basis of* $\mathbb{Z}^k$*.*

- *For any* $(sk, vk) \leftarrow$ **AKeyGen**$(1^\lambda, k)$ *honestly generated keypair, any* $c \in C$

$$\textbf{AVerify}(vk, fid, c, f) = 1 \iff \exists m \in \mathcal{M} : \textbf{ADecrypt}(sk, fid, c, f) = m$$

- *Let* $(sk, vk) \leftarrow$ **AKeyGen**$(1^\lambda, k)$ *be an honestly generated keypair, fid any dataset identifier,* $c_1, \ldots, c_k \in \mathcal{C}$ *any tuple of ciphertexts such that*

$m_i = \mathbf{ADecrypt}(sk, fid, c_i, f_i)$. *Then, for any admissible function* $f = (\alpha_1, \ldots, \alpha_k) \in \mathbb{Z}^k$, *with overwhelming probability*

$$\mathbf{ADecrypt}(sk, fid, \mathbf{AEval}(vk, f, fid, \{c_i\}_{i=1\ldots k}), \sum_{i=0}^{k} \alpha_i f_i) = f(m_1, \ldots, m_k)$$

Now we define a linearly homomorphic version of the IND-CCA security game for public key encryption. Although this might seem surprising at first, as CCA encryption is usually deployed to prevent malleability of the ciphertexts, it is possible to give a meaningful definition also in the context of homomorphic encryption. Namely, since we want to allow the ciphertexts to be manipulated only up to a certain extent (i.e. linear operations where the function applied is publicly declared), the best thing that we can do is explicitly disallow the decryption queries on a ciphertext legitimately derived from the challange cyphertext (as they could reveal information about the hidden bit the adversary is trying to guess).
Again, our definition is adapted to the linear case from [56].

Definition 23 (LH-IND-CCA) *Let* $\mathcal{H} = (\mathbf{AKeyGen}, \mathbf{AEncrypt}, \mathbf{ADecrypt}, \mathbf{AVerify}, \mathbf{AEval})$ *a LAEPuV scheme. Linearly Homomorphic IND-CCA is defined by the following game between a challenger and an adversary* $\mathcal{A}$*:*

$\mathbf{LH\text{-}IND\text{-}CCA}_{\mathcal{H},\mathcal{A}}(1^\lambda, k)$:

- **Setup** *The challenger runs* $(sk, vk) \leftarrow \mathbf{AKeyGen}(1^\lambda, k)$. *Then it initializes an empty set* S *and gives* vk *to the adversary* $\mathcal{A}$.

- **Queries I** $\mathcal{A}$ *can ask a polynomial number of encryption and decryption queries. Firsts are of the form* (fid, m_i, i) *(where fid is a dataset identifier,* $m_i \in \mathcal{M}$ *is a message and* $i \in \{1, \ldots, k\}$ *is an index). The challenger computes* $c_i \leftarrow \mathbf{AEncrypt}(sk, fid, i, m)$, *gives* c_i *to* $\mathcal{A}$ *and updates the set* $S \leftarrow S \cup \{(fid, m_i, i, c_i)\}$. *No two queries differing only on the* m_i *component can be asked by the adversary (if this happens, the answer to the second query is* $\perp$*). Decryption queries instead are of the form* (fid, c_i, f) *and* $\mathcal{A}$ *gets the corresponding output of*

$\mathbf{ADecrypt}(sk, fid, c, f)$ *(It can be* $\perp$ *if* c *is a not valid ciphertext).*

- **Challenge** $\mathcal{A}$ *produces a challenge tuple* (fid^*, i^*, m_0, m_1) *(as in the previous phase, if a query of the form* $(fid^*, i^*, \cdot)$ *has already been answered, the challenger returns* $\perp$*). The challenger chooses a random bit* $b \xleftarrow{\$} \{0,1\}$ *and gives* $c^* \leftarrow \mathbf{AEncrypt}(sk, fid, i, m_b)$ *to* $\mathcal{A}$*. Then it updates the set* $S \leftarrow S \cup \{(fid, m_b, i, c_i)\}$*.*

- **Queries II** *This phase is carried out as the previous one, the only difference being that the decryption queries made w.r.t.* fid^* *and a function* f *where* $f_{i^*} \neq 0$ *are answered with* $\perp$*.*

- **Output** *Finally* $\mathcal{A}$ *outputs a bit* b'*. The challenger outputs* 1 *if* $b = b'$*, and* 0 *otherwise.*

The advantage of $\mathcal{A}$ *in the LH-IND-CCA game is defined as*

$$\mathbf{Adv}_{\mathcal{H}}^{LH\text{-}IND\text{-}CCA}(\mathcal{A}) \stackrel{def}{=} \left| Pr\left[LH\text{-}IND\text{-}CCA_{\mathcal{H},\mathcal{A}}(1^\lambda) = 1\right] - \frac{1}{2} \right|$$

We say that a LAEPuV scheme is secure against a LH-IND-CCA attack if $\mathbf{Adv}_{\mathcal{H}}^{LH\text{-}IND\text{-}CCA}(\mathcal{A}) = negl(\lambda)$ *for any PPT adversary* $\mathcal{A}$*.*

Definition 24 (LH-Uf-CCA) *A LAEPuV scheme is Linearly Homomorphic Unforgeable against chosen ciphertext attack if the advantage of any PPT adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* λ*:*

- **Setup** *The challenger runs* $(sk, vk) \leftarrow \mathbf{AKeyGen}(1^\lambda, k)$*. Then it initializes an empty set* Q *and gives* vk *to the adversary* $\mathcal{A}$*.*

- **Queries** $\mathcal{A}$ *can ask a polynomial number of encryption and decryption queries. First are of the form* (fid, m_i, i) *(where fid is a dataset*

*identifier, $m_i \in \mathcal{M}$ is a message and $i \in \{1, \ldots, k\}$ is an index). The challenger computes $c_i \leftarrow$ **AEncrypt**(sk, fid, i, m), gives c_i to $\mathcal{A}$ and updates the set $Q \leftarrow Q \cup \{(fid, m_i, c_i, e_i)\}$. No two queries differing only on the $\mathbf{m}_i$ component can be asked by the adversary (if this happens, the answer to the second query is $\perp$). Decryption queries instead are of the form (fid, c_i, f) and $\mathcal{A}$ gets the corresponding output of* **ADecrypt**(sk, fid, c, f) *(It can be $\perp$ if c is a not valid ciphertext).*

- **Forgery** *$\mathcal{A}$ outputs (fid^*, c^*, f^*), where c^* is a ciphertext, fid^* a file identifier and f^* an admissible function.*

Let $Q_{fid^*} = \{(fid^*, m_i, c_i, f_i)\}_{i=1,\ldots,s} \subseteq Q$ be the set of entries in Q for which $fid = fid^*$.
The Adversary wins the game if **ADecrypt**$(vk, fid^*, c^*, f^*) = m^* \neq \perp$ *and one of the following conditions hold:*

1 Q_{fid^*} is empty

2 f^* (interpreted as a vector) is in the span of $\{f_1, \ldots, f_s\}$ but, for any $\alpha_1, \ldots, \alpha_s$ such that $f^* = \sum_{i=1}^{s} \alpha_i f_i$, it holds $m^* \neq \prod_{i=1}^{s} m_i^{\alpha_i}$

3 f^* (interpreted as a vector) is not in the span of $\{f_1, \ldots, f_s\}$

Finally we define the advantage $\mathbf{Adv}^{LH\text{-}Uf\text{-}CCA}(\mathcal{A})$ of $\mathcal{A}$ as the probability that $\mathcal{A}$ wins the game.

4.2 An instantiation supporting Paillier's encryption

Let (**HKeyGen**, **HSign**, **HVerify**, **HEval**) be a secure linearly homomorphic signature scheme whose message space is $\mathbb{Z}_N$ (where N is the product of two distinct (safe) primes). Moreover, let $\mathcal{H}$ be a family of collision resistant hash functions (whose images can be interpreted as elements of $\mathbb{Z}_{N^2}^*$). Then we can construct a LAEPuV scheme as follows.

AKeyGen$(1^\lambda, k)$: Choose two primes p, q of size $\lambda/2$, set $N \leftarrow pq$ and choose a random element $g \in \mathbb{Z}^*_{N^2}$ of order N. Run[1] **HKeyGen**$(1^\lambda, k, N)$ to obtain a signing key sk' and a verification key vk'. Pick a hash function $H \leftarrow \mathcal{H}$. Return $\text{vk} \leftarrow (\text{vk}', g, N, H)$ as the public verification key and $\text{sk} = (\text{sk}', p, q)$ as the secret signing key.

AEncrypt$(\text{sk}, m, \text{fid}, i)$: Choose random $\beta \leftarrow \mathbb{Z}^*_{N^2}$, compute $C \leftarrow g^m \beta^N \mod N^2$. Set $R \leftarrow H(\text{fid}||i)$, and use the factorization of N to compute $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}^*_N$ such that $g^a b^N = RC \mod N^2$. Compute $\sigma \leftarrow$ **HSign**(sk', fid, i, a) and return $c = (C, a, b, \sigma)$.

AVerify$(\text{vk}, \text{fid}, c, f)$: Parse $c = (C, a, b, \sigma)$ and $\text{vk} \leftarrow (\text{vk}', g, N, H)$, then check that:

$$\textbf{HVerify}(\text{vk}', \text{fid}, a, f, \sigma) = 1$$

$$g^a b^N = C \prod_{i=1}^{k} H(\text{fid}||i)^{f_i} \mod N^2$$

If both the above equations hold output 1, else output 0.

ADecrypt$(\text{sk}, \text{fid}, c, f)$: If **AVerify**$(\text{vk}, \text{fid}, c, f) = 0$, return $\perp$. Otherwise, use the factorization of N to compute (m, β) such that $g^m \beta^N = C \mod N^2$ and return m.

AEval$(\text{vk}, \alpha, \text{fid}, c_1, \ldots, c_k)$: Parse $\alpha = (\alpha_1, \ldots, \alpha_k)$ and $c_i = (C_i, a_i, b_i, \sigma_i)$, set

$$C \leftarrow \prod_{i=i}^{k} C_i^{\alpha_i} \mod N^2, \quad a \leftarrow \sum_{i=i}^{k} a_i \alpha_i \mod N,$$

$$b \leftarrow \prod_{i=i}^{k} b_i^{\alpha_i} \mod N^2, \quad \sigma \leftarrow \textbf{HEval}(vk', \text{fid}, f, \{\sigma_i\}_{i=1,\ldots,k})$$

and return $c = (C, a, b, \sigma)$.

[1]Notice that the signature scheme must support $\mathbb{Z}_N$ as underlying message space. This is why we give N to the **HKeyGen** algorithm as additional parameter. Note that, this means that, in general, the signature algorithm cannot not use the factorization of N as part of its private key.

Remark 6 (Supporting datasets of arbitrary size). *In the construction above the number k of ciphertext supported by each dataset needs to be fixed once and for all at setup time. This might be annoying in practical scenarios where more flexibility is preferable. We remark, that in the random oracle model, the scheme can be straightforwardly modified in order to remove this limitation. The idea would be to use the random oracle* also *in the underlying (homomorphic) signature scheme given in section 4.2.3. More precisely, rather than publishing the h_i as part of the public key, one computes different h_i's on the fly for each dataset by setting $h_i = H'(fid, i)$ (where H' is some appropriate random oracle). Slightly more in detail, the elements from dataset fid are then authenticated by replacing the h_i with $h_{fid,i} = H'(fid, i)$. Using this simple trick brings the additional benefit that the public key can be reduced to constant size.*

The security of the scheme is provided by the following theorems:

Theorem 8 *Assuming that the DCRA holds, if* (**HKeyGen**, **HSign**, **HVerify**, **HEval**) *is a secure linearly homomorphic signature scheme for messages in $\mathbb{Z}_N$ and H is a random oracle, the scheme described above is LH-IND-CCA secure according to definition 23.*

Theorem 9 *If $\Sigma =$ (**HKeyGen**, **HSign**, **HVerify**, **HEval**) is a secure linearly homomorphic signature scheme for messages in $\mathbb{Z}_N$ then the scheme described above is LH-Uf-CCA secure according to definition 24.*

4.2.1 Proof of theorem 8

We reduce the security of the scheme to the one of the DCRA: we use an adversary $\mathcal{A}$ that wins the LH-IND-CCA game to build a distinguisher $\mathcal{D}$ against the DCRA with advantage $\mathbf{Adv}_{\mathcal{D}} > \mathbf{Adv}_{\mathcal{A}}$. The distinguisher $\mathcal{D}$ receives in input (y, N) and runs the simulation as follows:

Key generation phase The distinguisher chooses $g \in \mathbb{Z}^*_{N^2}$ as a random element of order N, runs $(\mathrm{sk}', \mathrm{vk}') \leftarrow \mathbf{HKeyGen}(1^\lambda, k, N)$ and gives (vk', g, N) to $\mathcal{A}$ (the function H is substituted with a random oracle).

Queries The adaptively chosen queries asked by $\mathcal{A}$ are handled as follows.

Random Oracle Queries $\mathcal{D}$ guesses in advance on which couple (fid^*, i^*) the adversary will ask its challenge (since $\mathcal{A}$ is polynomial, $\mathcal{D}$ will be right with non negligible probability). It chooses u^*, v^* at random and sets $R^* \leftarrow g^{u^*} v^{*N} y^{-1} \mod N^2$ as the output of $H(\text{fid}^* || i^*)$. For any other oracle query (fid, i) it chooses random u, v and sets $R = g^u v^N \mod N^2$ as the output. It stores (fid, i, u, v) in a table T (and, if the same query is asked more than once, the same answer computed from the table T is returned).

Encryption Queries On input a query (fid, i, m), $\mathcal{D}$ retrieves the associated u, v from the table T (if the query (fid, i) has not been already asked, $\mathcal{D}$ simulates it and populates the table T accordingly). Then it computes $C \leftarrow g^m \beta^N$ for a random β, $a \leftarrow u + m$, $b \leftarrow \beta v$, $\sigma \leftarrow \textbf{Sign}(\text{sk}', \text{fid}, i, a)$ and returns $c = (C, a, b, \sigma)$ to $\mathcal{A}$

Decryption Queries On input a triple (fid, c, f), where $c = (C, a, b, \sigma)$ is a ciphertext, fid an identifier and f an admissible function, $\mathcal{D}$ verifies the signature on a and that the equation

$$g^a b^N = C \prod_{i=1}^{k} H(\text{fid} || i)^{f_i} \mod N^2$$

holds (the oracle queries appearing in the above equation with non-zero exponents must exist in T, otherwise $\mathcal{D}$ can just assign to these queries a value that does not satisfy the equation and return $\perp$ to $\mathcal{A}$). If this is not the case, it returns $\perp$ to $\mathcal{A}$. Otherwise, it retrieves the couples (u_i, v_i) associated with each query (fid, i) from the table T. Then it computes $m \leftarrow a - \sum_{i=1}^{k} f_i u_i$ and returns m to $\mathcal{A}$. It is easy to see that if c is a valid ciphertext, than the message m is a correct decryption.

Challenge query On input $(\text{fid}^*, i^*, m_0, m_1)$, if $\mathcal{D}$ guessed the right fid and index, it chooses a bit z, computes $C \leftarrow g^{m_z} \beta^N y$ for a random β, $a \leftarrow u + m_z$, $b \leftarrow \beta v$, $\sigma \leftarrow \textbf{Sign}(\text{sk}', \text{fid}, i, a)$ and returns $c^* = (C, a, b, \sigma)$. Otherwise, the simulation is aborted.

Queries II after the challenge phase another queries phase takes place and all queries are handled as before, the only exception being that decryption queries involving identifier fid^* and a function f whose i^*-th component is non-zero are answered with $\perp$.

Output phase When $\mathcal{A}$ outputs a bit z', $\mathcal{D}$ guesses that y is a residue if $z = z'$ and that y is not a residue if $z \neq z'$ or if $\mathcal{A}$ aborts at any time.

First of all, one can notice that all the answers to the queries made by $\mathcal{A}$ (apart from the challenge query) are distributed as in the real case. Moreover, this is also the case for the challenge query provided y is an N-th residue, while c^* contains no information about the bit z in the other case. This is because, if we write y as $g^{y_1} y_2^N$, from c^* an unconditionally powerful adversary could deduce 3 *dependent* equations in the 3 variables y_1, m_z, u^*, which makes their simultaneous solution undetermined.
Therefore, in the case where y is not a residue $\mathcal{A}$ is playing the proper security game and $\mathcal{D}$ wins as long as $\mathcal{A}$ is successful, which happens with probability $1/2 + \mathbf{Adv}_A$, and in the case where y is not a residue $\mathcal{A}$ can only guess at random, which makes $\mathcal{D}$ successful with probability at most $1/2$. In sum, since we are assuming the DCRA problem to be hard, it must be that

$$negl(n) > \mathbf{Adv}_D \geq |1/2 + \mathbf{Adv}_A - 1/2| \geq \mathbf{Adv}_A$$

which is our thesis.

4.2.2 Proof of theorem 9

The reduction is very simple because, since we are not breaking any assumption related to the modulus N, its factors can be known by the simulator $\mathcal{D}$. $\mathcal{D}$ receives in input a public key vk' for Σ, and prepares the public key for $\mathcal{A}$ as in the real case, with the only difference that it uses vk' instead of running the key generation algorithm of Σ. The encryption and decryption queries are handled as in the real case, with the only exception that the σ component of each ciphertext is requested by $\mathcal{D}$ to its signing oracle instead of being computed by $\mathcal{D}$ itself. When $\mathcal{A}$ outputs a forged ciphertext

$c^* = (C^*, a^*, b^*, \sigma^*)$ w.r.t. an identifier fid and a function f^*, its σ^* component must be a valid forgery for Σ. If this was not the case, it would imply that, called $c_i = (C_i, a_i, b_i, \sigma_i)$ the signatures returned by the simulator to $\mathcal{A}$ in response to its query (fid, i, m_i),

$$\left(\frac{b^*}{\prod_{i=1}^{k} b_i^{f_i^*}}\right)^N = \frac{C^*}{\prod_{i=1}^{k} C_i^{f_i^*}}.$$

Therefore this would not be a forgery as C^* would be in the same residuosity class (and hence contain the same plaintext) of the ciphertext obtained by computing the function f^* on the honestly generated ciphertext $c_1, \ldots, c_n$.

4.2.3 Instantiating the underlying signature scheme

As a concrete instantiation of the linearly homomorphic signature scheme (**HKeyGen**, **HSign**, **HVerify**, **HEval**), one can use use a simple variant of the (Strong) RSA based scheme from [28] adapted to use $\mathbb{Z}_N$ as underlying message space. In this section we will describe such signature scheme and prove its security.
In [28] Catalano et al. present a Network Coding signature scheme based on the strong RSA assumption. Here we describe a simple variant of the scheme that allows a user to sign messages in a bigger space.

KeyGen$(1^k, N)$ Let N product of two arbitrary safe primes each one of length $k'/2$. The **KeyGen** algorithm chooses two random (safe) primes $\hat{p}, \hat{q}$ of length $k/2$ each such that $gcd(N, \phi(\hat{N})) = 1$[2] where $\hat{N} = \hat{p}\hat{q}$ and proceeds by choosing $g, g_1, h_1, \ldots, h_m$ at random (in $\mathbb{Z}^*_{\hat{N}}$). Moreover it chooses a chameleon hash function $H : \{0,1\}^* \to \{0,1\}^\ell$ that maps to primes of length $\ell < k'/2$. The public key is set as $(N, H, \hat{N}, g, g_1, h_1, \ldots, h_m)$, while the secret key is $(\hat{p}, \hat{q})$.

Sign$(\text{sk}, \text{fid}, M, i)$ Let e_i the i-th vector of the canonical basis on $\mathbb{Z}^m$. The signing algorithm proceeds as follows. First it chooses a randomness ρ

[2]Supposing $\hat{p}$ and $\hat{q}$ are two safe prime of equal length $k/2$ we can write $\hat{p} = 2\hat{p}' + 1$ and $\hat{q} = 2\hat{q}' + 1$ so $\phi(\hat{N}) = 4\hat{p}'\hat{q}'$ where $|\hat{p}'| = |\hat{q}'| \approx \frac{k}{2} - 1$. So, fixed N, it's enough choose $\hat{p}, \hat{q}$ of length at least $k'/2 + 2$ to verify $gcd(N, \phi(\hat{N})) = 1$

and maps the random identifier fid to prime: $e \leftarrow H(\rho, \text{fid})$. It chooses random elements $s \in \mathbb{Z}_{eN}$ and uses its knowledge of $\hat{p}$ and $\hat{q}$ to solve the following equation

$$x^{eN} = g^s h_i g_1^M \bmod \hat{N}$$

We denote with $\sigma = (\rho, e, s, x)$ the signature for the message M w.r.t. the function e_i and the identifier fid.

Verify$(\text{vk}, \sigma, M, \mathbf{f})$ To verify a signature σ for a message M w.r.t. an identifier fid and a function $\mathbf{f}$, the verification algorithm proceeds as follows

- Compute $e \leftarrow H(\rho, \text{fid})$
- Check that M, s are in $\mathbb{Z}_{eN}$.
- Define $\mathbf{f}' = \frac{\mathbf{f} - \mathbf{f} \bmod eN}{eN}$ and $\hat{x} = \frac{x}{\prod_{j=1}^{m} h_j^{f'_j}}$
- Finally check that the equation

$$\hat{x}^{eN} = g^s \prod_{j=1}^{m} h_j^{f_j} g_1^M$$

  is satisfied
- If all the checks above are satisfied, output 1, otherwise 0.

Combine$(\text{vk}, \text{fid}, \hat{\mathbf{f}}, \sigma_1, \ldots, \sigma_m)$ To combine signatures σ_i, (corresponding to the messages M_i) sharing the same fid it works as follows. Let $\hat{\mathbf{f}} = (\alpha_1, \ldots, \alpha_m)$. It sets $s = \sum_{i=1}^{m} \alpha_i s_i \bmod eN$, $s' = (\sum_{i=1}^{m} \alpha_i s_i - s)/(eN)$. It outputs the signature $\sigma = (\rho, e, s, \text{fid}, x)$ which is obtained by computing

$$x = \frac{\prod_{i=1}^{m} x_i^{\alpha_i}}{g^{s'}} \bmod \hat{N}.$$

Security follows very easily from the proof of the original signature scheme in [28]. For completness we explicitly prove it again here.

Theorem 10 *Under the Strong-RSA assumption, the scheme described above is an unforgeable signature scheme under chosen messages attack according to [14] definition.*

Let $\mathcal{A}$ be an efficient adversary against the security of the scheme. This means that, with non negligible probability, $\mathcal{A}$ is able to produce a valid forgery $\sigma^* = (\rho^*, e^*, s^*, x^*)$ for the message M^* w.r.t. a function f^* and an identifier fid^*. We show how to build an efficient adversary $\mathcal{B}$ that "uses" it to break the strong RSA assumption (for the case when the challenge is a quadratic residue). Let t be the maximum number of signatures queried by $\mathcal{A}$. Let $e_j = H(\text{fid}_j)$. Using a chameleon hash function it is possible to fix its the possible output at the beginning of the game. More details are given below. If one considers e^* and the set $\{e_1, \ldots, e_t\}$ it is possible distinguish two types of forgeries:

Type I the adversary outputs a signature containing an Ne^* such that $Ne* \nmid N^t \prod_{i=1}^{t} e_i$,

Type II the adversary outputs a signature containing an Ne^* such that $Ne* \mid N^t \prod_{i=1}^{t} e_i$.

At the beginning of the game we guess on the type of forgery will be provided by $\mathcal{A}$ in order to set up an appropriate simulation accordingly. This guess will be right with probability at least $1/2$.

Type I. $\mathcal{B}$ takes as input $(\hat{N}, \tau)$ where $\hat{N}$ is the product of two safe primes $\hat{p}, \hat{q}$ (where $\hat{p} = 2p' + 1$ and $\hat{q} = 2q' + 1$) and $\tau \in QR_N$. The goal here is to find an e-th root y of τ for e of $\mathcal{B}$'s choice.

In the following we describe the simulator $\mathcal{B}$ during the three phases of the simulation.

Setup $\mathcal{B}$ chooses a function H and an integer N as prescribed by the **KeyGen** algorithm and randomly chooses t random file identifiers $\text{fid}_1, \ldots, \text{fid}_t$ of the appropriate length. Next it chooses a randomness ρ and computes $e_i = H(\rho, \text{fid}_i)$ $\forall 1 = 1, \ldots, t$. Then it generates the public key as follows.

- pick random $\alpha_0, \alpha_1, \beta_1, \ldots, \beta_m \leftarrow \{1, \ldots, \hat{N}^2\}$
- let $E = N^t \prod_{i=1}^{t} e_i$ and set $g = \tau^{E\alpha_0}$, $g_1 = g^{\alpha_1}$ and $h_i = g^{\beta_i}$ for all $i = 1$ to m.

Finally $\mathcal{B}$ gives vk $= (N, H, g, h_1 \ldots, h_m, g_1)$ to $\mathcal{A}$

let $\alpha_1 = bp'q' + c$ where $0 \leq c < p'q'$. Since α_1 is chosen from a suitably large interval, the distributions of $(\alpha_1 \bmod p'q')$ is statistically indistinguishable from the uniform distribution over $\mathbb{Z}_{p'q'}$. So g_1 is distributed like random quadratic residues of $\mathbb{Z}_{\hat{N}}$. Moreover the conditional distribution of b given c is statistically indistinguishable from the uniform distribution over $\{0, \ldots, \lfloor \hat{N}^2/p'q' \rfloor\}$. The same argument applies to g and all the h_i's.

Signing queries At this stage $\mathcal{A}$ is allowed to adaptively query signatures on messages M w.r.t an identifier fid and a position i.

By these positions each signature query is managed as follows. It uses the private chameleon hash key to compute $\tilde{\rho}_k$ such that $e_k = H(\tilde{\rho}_k, \text{fid})$ and chooses at random $s_i \in \mathbb{Z}_{Ne_k}$, . Next, $\mathcal{B}$ computes the solution of $x_i^{Ne_k} = g^{s_i} \cdot h_i \cdot g_1^M$ as follows:

- let $E_k = N^{t-1} \prod_{j=1, j \neq k}^{t} e_j$
- $\forall i = 1, \ldots, m : x_i = (\tau^{E_k \alpha_0})^{s_i + \beta_i + M\alpha_1}$

Finally $\mathcal{B}$ gives $\sigma = (\tilde{\rho}_k, e_k, s_i, i, M, \text{fid}_k, x_i)$ to $\mathcal{A}$. It is easy to see that x_i are valid solution for the equation above (and that the equation is distributed as in the real case).

Challenge Once the previous phase is over, $\mathcal{A}$ is supposed to output a forgery $\sigma^* = (\rho^*, e^*, s^*, f^*, M^*, \text{fid}^*, x^*)$. By definition of valid forgery it has to be the case that

$$x^{Ne^*} = g^{s^*} \cdot \prod_{j=1}^{m} h_j^{f_j^*} \cdot g_1^{M^*} = \tau^{E\alpha_0(s^* + \sum_{j=1}^{m} f_j \beta_j + M^* \alpha_1)}.$$

Let $E' = E\alpha_0(s^* + \sum_{j=1}^{m} f_j^* \beta_j + M^* \alpha_1)$ and $d = gcd(Ne^*, E')$. Provided that $Ne^* \nmid E'$ $\mathcal{B}$ can use standard techniques (i.e. Shamir's trick) to extract an (Ne^*/d)-th root y of τ and thus it can output $(e^*/d, y^N)$ to break Strong-RSA.

Therefore we are left with the task of showing that $Ne^* \nmid E'$ with non-negligible probability. About e^*, since all the e exponents are primes and we are assuming a Type I forgery, it has to be the case that $e^* \nmid E$. It remains to show that $e^* \nmid \alpha_0(s^* + \sum_{j=1}^{m} f_j^* \beta_j + M^* \alpha_1)$ with non-negligible probability.

As pointed out before, we set $\alpha_1 = bp'q' + c$. Since each b is information theoretically hidden to $\mathcal{A}$, e^* might depend only on c (the same holds for the β_i's). Moreover as $e^* \nmid p'q'$ the probability that $e^* \mid \alpha_0(s^* + \sum_{j=1}^{m} f_j^* \beta_j + M^* \alpha_1)$, or equivalently $\alpha_0(s^* + \sum_{j=1}^{m} f_j^* \beta_j + M^* \alpha_1) = 0 \bmod e^*$, is close to $1/e^*$. This means that $e^* \nmid E'$ with probability close to $1 - 1/e^*$.

Type II. This encompasses the case when $\mathcal{A}$ "reuses" some previously seen exponent when producing its forgery. This is because, being all the e_i's primes, the fact that $Ne* \mid N^t \prod_{i=1}^{t} e_i$, implies that $e^* = e_k$ for some k. Again let $\sigma^* = (\rho^*, e^*, s^*, x^*)$ be the forgery provided by the adversary for the message M^* w.r.t. a function f^* and an identifier fid^*. Since in this case we assume that $e^* = e_k$, $\mathrm{fid}^* = \mathrm{fid}_k$. Thus, in order for the provided signature to be a valid forgery, it has to be the case that $M^* \neq \sum_{i=1}^{m} f_i M_i \bmod Ne^*$ or $\rho^* \neq \rho_k$. In the last case the adversary trivially break the chameleon hash function so it has to be that $M^* \neq \sum_{i=1}^{m} f_i M_i \bmod Ne^*$. Let $z = M^* - \sum_{i=1}^{m} f_i M_i \bmod Ne^* \neq 0 \bmod Ne^*$.

In what follows we will require the simulator to guess both the index k Thus, $\mathcal{B}$'s guess will be correct with probability $1/t$. Now, if we consider the forgery provided by the adversary and the values $x_1, \ldots, x_m$ obtained from the signatures on the identifier $e_k = e^*$ provided by the simulator, we distinguish two additional subcases

(a) $x^* = \prod_{j=1}^{m} x_j^{f_j^*} \bmod \hat{N}$

(b) $x^* \neq \prod_{j=1}^{m} x_j^{f_j^*} \bmod \hat{N}$

We provide different simulations for the two cases. In particular we describe Type-II.b first.

Type-II.b We describe a simulator $\mathcal{B}$ that solves Strong RSA for the case of Type-II.b forgeries.

Setup $\mathcal{B}$ chooses $\mathrm{fid}_1, \ldots, \mathrm{fid}_t$ at random, ρ_i in the random space and computes $e_i = H(\rho_i, \mathrm{fid}_i)\ \forall i = 1, \ldots, t$. Next, $\alpha_1, \omega_1, \ldots, \omega_m, \beta_1, \ldots, \beta_m \leftarrow \{1, \ldots, 2\hat{N}\}$.

Let $E = N^t \prod_{i=1}^{t} e_i$ and $E_k = N^{t-1} \prod_{i=1, i\neq k}^{t} e_i$. $\mathcal{B}$ proceeds by creating the public key as follows. $g = \tau^{E_k}$, $g_1 = g^{\alpha_1}$, $h_i = g^{e_k\omega_i - \beta_i}\ \forall i = 1, \ldots, m$. Finally it gives the public key to $\mathcal{A}$. It is easy to get convinced that the distribution of the so generated public key is statistically close to that of a "true" public key.

Signing queries $\mathcal{B}$ answers $\mathcal{A}$'s signature queries as follows.

Let fid_i be the i-th queried identifier. For all $i \in \{1, \ldots, t\}\backslash\{k\}$ $\mathcal{B}$ first it use the private chameleon hash key to compute $\tilde{\rho}_i$ such that $e_i = H(\tilde{\rho}_i, \mathrm{fid})$ it sets $e_i \leftarrow H(\mathrm{fid}_i)$. Supposing to sign a message M in the position j it chooses random $s_j \in \mathbb{Z}_{Ne_i}$ and sets

$$x_j = (\tau^{\prod_{l\neq k,i} e_l})^{s_j + e_k\omega_j - \beta_j + \alpha_1 M}.$$

It is easy to verify that x_j is such that $x_j^{Ne_i} = g^{s_j} \cdot \prod_{l=1}^{m} h_l^{u_l^{(j)}} \cdot g_1^M$.

For $i = k$ a different machinery is required. Let j the position for which M must be signed, $\mathcal{B}$ sets $s = \beta_j - \alpha_1 M \bmod Ne_i$ and computes

$$x_j = \tau^{E_k\omega_j}. = \sqrt[Ne_k]{g^s \cdot h_j \cdot g_1 M}.$$

Finally $\mathcal{B}$ provides the signature created above to $\mathcal{A}$. Notice that such signature follows a distribution which is statistically close with respect to that that would have been produced by a genuine signer.

Challenge In this phase $\mathcal{A}$ will output a type II forgery (defined by $(\rho^*, e^*, s^*, f^*, M^*, \mathrm{fid}^*, x^*)$), we show that $\mathcal{B}$ can extract an e^*-th root of τ as follows.

First, let

$$\begin{aligned} x_1^{Ne_k} &= g^{s_1} h_1 g_1^{M_1} \\ &\vdots \\ x_m^{Ne_k} &= g^{s_m} h_m g_1^{M_m} \end{aligned} \tag{4.1}$$

denote the verification equations arising from $\mathcal{B}$'s signatures on position $1, \ldots, m$. Combining them with the received forgery one gets.

$$\begin{aligned}\left(\frac{x^*}{\prod_{j=1}^{m} x_j^{f_j^*}}\right)^{Ne^*} &= g^{(s^* - \sum_{l=1}^{m} f_l^* s_l)} g_1^{(M^* - \sum_{l=1}^{m} f_l^* M_j)} \\ &= (\tau^{E_k})^{(s^* - \sum_{l=1}^{m} f_l^* s_l) + \alpha_1 z}\end{aligned}$$

Thus we can rewrite the equation above as

$$\left[\left(\frac{x^*}{\prod_{j=1}^{m} x_j^{f_j^*}}\right)\right]^{Ne^*} = \tau^{E_k(s^* - \sum_{l=1}^{m} f_l^* s_l + \alpha_1 z)}.$$

Let $E' = E_k(s^* - \sum_{l=1}^{m} f_l^* s_l + \alpha_\nu z_\nu)$. In order to extract a root of τ we have to show that $Ne^* \nmid E'$ with non-negligible probability. Observe that $e^* \nmid E_k$ and that $\alpha_1 = bp'q' + c$ where $b \in \{0, 1\}$ (with probability close to 1) and b is information theoretically hidden to the adversary. We show that $\Pr[Ne^* \nmid (s^* - \sum_{l=1}^{m} f_l^* s_l + \alpha_1 z)]$ is at least $1/2$. To see this, assume by contradiction that $\Pr[Ne^* \mid (s^* - \sum_{l=1}^{m} f_l^* s_l + \alpha_1 z)]$ is non-negligibly higher than $1/2$. Then it must be that $Ne^* \mid (s^* - \sum_{l=1}^{m} f_l^* s_l + cz)$ and $Ne^* | (s^* - \sum_{l=1}^{m} f_l^* s_l + (\phi(\hat{N}) + c)z)$, which implies that $Ne^* | z\phi(\hat{N})$. Now $z \in \mathbb{Z}Ne^*$ so $Ne^* \nmid z$. Thus Ne^* must be a non trivial factor of $\phi(\hat{N})$. Therefore $Ne^* \nmid E'$ with probability at least $1/2$ and in this case $\mathcal{B}$ can use standard techniques (i.e., Shamir's trick) to extract an (Ne^*/d)-th root y of τ where $d = gcd(Ne^*, E')$.

Type-II.a For the case of Type-II.a forgeries the simulator performs basically the same Setup and Signing queries phases as in the Type-I simulation. The only difference here is that in the setup we set $g_1 = \tau^E$. Once a forgery is provided $(\rho^*, e^*, s^*, f^*, m^*, \text{fid}^*, x^*)$ being it a Type-II.a one we have that

$$g^{s^*} \prod_{j=1}^{m} h_j^{f_j^*} g_1^{M^*} = g^{\sum_{l=1}^{m} s_l f_l^*} \prod_{j=1}^{m} h_j^{f_l^*} g_1^{\sum_{l=1}^{m} M_l f_l^*}$$

This leads to the following

$$\tau^{E(\alpha_0(s^* - \sum_{j=1}^{m} s_j f_j^*) + z} = 1 \bmod \hat{N}$$

where, again, z is $M^* - \sum_{j=1}^{m} f_j^* M_j \bmod Ne^*$. Let $\gamma = (\alpha_0(s^* - \sum_{j=1}^{m} s_j f_j^*) + z)$. Notice that each $\alpha_j = b_j p'q' + c_j$ where b_j is information theoretically hidden to the adversary and that $z \neq 0 \bmod Ne^*$ (this is the simulator's guess). Therefore, with non-negligible probability we have an integer $E\gamma \neq 0$ such that $E\gamma = 0 \bmod \phi(\hat{N})$, that allows to factor and thus to trivially solve the Strong RSA problem.

4.3 A General Result

In this section we show how to generalize our results to support arbitrary encryption schemes satisfying some well defined homomorphic properties.
In such schemes, the message, randomness and ciphertext spaces are assumed to be finite groups, respectively denoted with $\mathcal{M}, \mathcal{R}, \mathcal{C}$ (the key spaces are treated implicitly). To adhere with the notation used in the previous section, we will denote the operation over $\mathcal{M}$ additively and the ones over $\mathcal{R}$ and $\mathcal{C}$ multiplicatively. We assume $\mathcal{T}$ to be an IND-CPA secure public key encryption scheme satisfying the following additional properties:

- We require the group operation and the inverse of an element to be efficiently computable over all groups, as well as efficient sampling of random elements. The integer linear combinations are thus defined and computed by repeatedly applying these operations.
- For any $m_1, m_2 \in \mathcal{M}, r_1, r_2 \in \mathcal{R}$, any valid public key pk it holds
$$\mathbf{Enc}_{\mathrm{pk}}(m_1, r_1) \cdot \mathbf{Enc}_{\mathrm{pk}}(m_2, r_2) = \mathbf{Enc}_{\mathrm{pk}}(m_1 + m_2, r_1 \cdot r_2)$$
- For any honest key pair $(\mathrm{pk}, \mathrm{sk})$ and any $c \in \mathcal{C}$ there exists $m \in \mathcal{M}$ and $r \in \mathcal{R}$ such that $\mathbf{Enc}_{\mathrm{pk}}(m, r) = c$ (i.e. the encryption function is surjective over the group $\mathcal{C}$). Moreover, we assume that such m and r are efficiently computable given the secret key.

Now, let $(\mathbf{HKeyGen}, \mathbf{HSign}, \mathbf{HVerify}, \mathbf{HEval})$ be a secure linearly homomorphic signature scheme for elements in $\mathcal{M}$, let $\mathcal{H}$ be a family of collision resistant hash functions $H_K : \{0,1\}^* \to \mathcal{C}$ and let $\mathcal{T} = \{\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec}\}$ be an encryption scheme as above.

We construct a LAEPuV scheme as follows:

AKeyGen$(1^\lambda, k)$: Run **HKeyGen**$(1^\lambda, k)$ to obtain a signing key sk' and a verification key vk' and **KeyGen**(1^λ) to obtain a public key $\overline{\mathrm{pk}}$ and a secret key $\overline{\mathrm{sk}}$. Pick a hash function $H \leftarrow \mathcal{H}$. Return $\mathrm{vk} \leftarrow (\mathrm{vk}', \overline{\mathrm{pk}}, H)$ as the public verification key and $\mathrm{sk} = (\mathrm{sk}', \overline{\mathrm{sk}})$ as the secret key.

AEncrypt$(\mathrm{sk}, m, \mathrm{fid}, i)$: Choose random $r \leftarrow \mathcal{R}$, compute $C \leftarrow \mathbf{Enc}_{\overline{\mathrm{pk}}}(m, r)$ and compute, using the secret key sk, $\overline{m}$ and $\overline{r}$ such that $\mathbf{Enc}_{\overline{\mathrm{pk}}}(\overline{m}, \overline{r}) = H(\mathrm{fid}||i)$. Compute $\sigma \leftarrow \mathbf{HSign}(sk', \mathrm{fid}, i, m + \overline{m})$ and return $c = (C, m + \overline{m}, r \cdot \overline{r}, \sigma)$.

AVerify$(\mathrm{vk}, \mathrm{fid}, c, f)$: Parse $c = (C, a, b, \sigma)$ and $\mathrm{vk} \leftarrow (\mathrm{vk}', \overline{\mathrm{pk}})$, then check that:

$$\mathbf{HVerify}(\mathrm{vk}', \mathrm{fid}, a, f, \sigma) = 1$$

$$\mathbf{Enc}_{\overline{\mathrm{pk}}}(a, b) = C \prod_{i=1}^{k} H(\mathrm{fid}||i)^{f_i}$$

If both the above equations hold output 1, else output 0.

ADecrypt$(\mathrm{sk}, \mathrm{fid}, c, f)$: Parse $c = (C, a, b, \sigma)$. If **AVerify**$(\mathrm{vk}, \mathrm{fid}, c, f) = 0$, return $\perp$. Otherwise, use the secret key $\overline{\mathrm{sk}}$ to compute $m \leftarrow \mathbf{Dec}_{\overline{\mathrm{sk}}}(C)$

AEval$(\mathrm{vk}, \alpha, \mathrm{fid}, c_1, \ldots, c_k)$: Parse $\alpha = (\alpha_1, \ldots, \alpha_k)$ and $c_i = (C_i, a_i, b_i, \sigma_i)$, set

$$C \leftarrow \prod_{i=i}^{k} C_i^{\alpha_i}, \quad a \leftarrow \sum_{i=i}^{k} a_i \alpha_i,$$

$$b \leftarrow \prod_{i=i}^{k} b_i^{\alpha_i}, \quad \sigma \leftarrow \mathbf{HEval}(vk', \mathrm{fid}, f, \{\sigma_i\}_{i=1,\ldots,k})$$

and return $c = (C, a, b, \sigma)$.

Theorem 11 *Assuming $\mathcal{T}$ is an is IND-CPA secure public key encryption scheme satisfying the conditions detailed above,* (**HKeyGen**, **HSign**, **HVerify**, **HEval**) *is a secure linearly homomorphic signature scheme supporting $\mathcal{M}$ as underlying message space and H is a random oracle, then the scheme described above has indistinguishable encryption according to definition 23.*

Theorem 12 *If* $\Sigma = (\mathbf{HKeyGen}, \mathbf{HSign}, \mathbf{HVerify}, \mathbf{HEval})$ *is a secure linearly homomorphic signature scheme for messages in* $\mathcal{M}$ *then the scheme described above is unforgeable according to definition 24.*

4.3.1 Proof of theorem 11

We reduce the security of the scheme to the one of the encryption scheme: we use an adversary $\mathcal{A}$ that wins the LH-IND-CCA game to build a distinguisher $\mathcal{D}$ against the IND-CPA with advantage $\mathbf{Adv}_{\mathcal{D}} > \mathbf{Adv}_{\mathcal{A}}$. The distinguisher $\mathcal{D}$ receives in input $\bar{\text{pk}}$, set the challenge as $m_0, m_1 \xleftarrow{\$} \mathcal{M}$, receives a challenge ciphertext $y^* \in \mathcal{C}$ and runs the simulation as follows:

Key generation phase The distinguisher runs

$$(\text{sk}', \text{vk}') \leftarrow \mathbf{HKeyGen}(1^\lambda, k, N)$$

and gives $(\text{vk}', \bar{\text{pk}})$ to $\mathcal{A}$ (the function H is substituted with a random oracle).

Queries The adaptively chosen queries asked by $\mathcal{A}$ are handled as follows.

Random Oracle Queries $\mathcal{D}$ guesses in advance on which couple (fid^*, i^*) the adversary will ask its challenge (since $\mathcal{A}$ is polynomial, $\mathcal{D}$ will be right with non negligible probability). It chooses $\bar{m}, \bar{r}$ at random and sets $R^* \leftarrow \mathbf{Enc}(\bar{m}, \bar{r}) y^{*-1}$ as the output of $H(\text{fid}^* || i^*)$. For any other oracle query (fid, i) it chooses random u, r and sets $R = \mathbf{Enc}(u, r)$ as the output. It stores (fid, i, u, r) in a table T (and, if the same query is asked more than once, the same answer computed from the table T is returned).

Encryption Queries On input a query (fid, i, m), $\mathcal{D}$ retrieves the associated u, r from the table T (if the query (fid, i) has not been already asked, $\mathcal{D}$ simulates it and populates the table T accordingly). Then it computes $C \leftarrow \mathbf{Enc}(m, r_1)$ for a random r_1, $a \leftarrow u + m$, $b \leftarrow r r_1$, $\sigma \leftarrow \mathbf{Sign}(\text{sk}', \text{fid}, i, a)$ and returns $c = (C, a, b, \sigma)$ to $\mathcal{A}$

Decryption Queries On input a triple (fid, c, f), where $c = (C, a, b, \sigma)$ is a ciphertext, fid an identifier and f an admissible function, $\mathcal{D}$ verifies the signature on a and that the equation

$$\mathbf{Enc}(a, b) = C \prod_{i=1}^{k} H(\text{fid}||i)^{f_i}$$

holds (the oracle queries appearing in the above equation with non-zero exponents must exist in T, otherwise $\mathcal{D}$ can just assign to these queries a value that does not satisfy the equation and return $\perp$ to $\mathcal{A}$). If this is not the case, it returns $\perp$ to $\mathcal{A}$. Otherwise, it retrieves the couples (u_i, r_i) associated with each query (fid, i) from the table T. Then it computes $m \leftarrow a - \sum_{i=1}^{k} f_i u_i$ and returns m to $\mathcal{A}$. It is easy to see that if c is a valid ciphertext, than the message m is a correct decryption.

Challenge query On input $(\text{fid}^*, i^*, m_0, m_1)$, if $\mathcal{D}$ guessed the right fid and index, it chooses a bit z, computes $C \leftarrow \mathbf{Enc}(m_z, r)y^*$ for a random r, $a \leftarrow \bar{m} + m_z$, $b \leftarrow \bar{r}r$, $\sigma \leftarrow \mathbf{Sign}(\text{sk}', \text{fid}, i, a)$ and returns $\hat{c}^* = (C, a, b, \sigma)$. Otherwise, the simulation is aborted.

Queries II after the challenge phase another queries phase takes place and all queries are handled as before, the only exception being that decryption queries involving identifier fid^* and a function f whose i^*-th component is non-zero are answered with $\perp$.

Output phase When $\mathcal{A}$ outputs a bit z', if $z = z'$ then $\mathcal{D}$ output 0 else if $z \neq z'$ or $\mathcal{A}$ abort, $\mathcal{D}$ output 1.

First of all, one can notice that all the answers to the queries made by $\mathcal{A}$ (apart from the challenge query) are distributed as in the real case. Moreover, if $z \neq z'$ this is also the case for the challenge query provided c^* is an correct encryption for one of the two messages, while $\hat{c}^*$ contains no information about the bit z. In the other case the challenger query c^* is again a correct encryption, but in this case $\hat{c}^*$ contain some information about the bit z Therefore, let b the bit chooses by the challenger in the game of $\mathcal{D}$. In the

case where $b = 0$ $\mathcal{A}$ is playing the proper security game and $\mathcal{D}$ wins as long as $\mathcal{A}$ is successful, which happens with probability $1/2 + \mathbf{Adv}_A$, and in the case where $b = 1$ $\mathcal{A}$ can only guess at random, which makes $\mathcal{D}$ successful with probability at most $1/2$. In sum, since we are assuming the underlying encryption scheme is IND-CPA secure, it must be that

$$negl(n) > \mathbf{Adv}_D \geq |1/2 + \mathbf{Adv}_A - 1/2| \geq \mathbf{Adv}_A$$

which is our thesis. .

Chapter 5

Applications to On-Line/Off-Line Homomorphic Signatures

In this chapter, we show a general construction to build (efficient) on-line/off-line homomorphic (and network coding) signature schemes by combining LHSG unforgeable against a random message attack (like the one described in section 3.4) with a certain class of sigma protocols. The intuitive idea is that in order to sign a certain message m, one can choose a Σ-Protocol whose challenge space contains m, next one signs the first message of the Σ-Protocol with a standard signature (this can be done off-line) and use its knowledge of the witness of the protocol to later compute the response (third message) of the protocol associated to the challenge m. This is secure because, if an adversary is able to produce a second signature with respect to the same first message, by the special soundness of the Σ-Protocol, he would be able to recover the witness itself. We show how, if both the signature scheme and the Σ-Protocol have specific homomorphic properties, this construction can be extended to build (linearly) homomorphic signatures as well.
Informally the properties we require from the underlying sigma protocol are: (1) it is linearly homomorphic, (2) its challenge space can be seen as a vector space and (3) the third message of the protocol can be computed in a very efficient way (as it is used in the online phase of the resulting scheme). First,

we adapt the definition of linearly homomorphic signature (LHSG) to the On-line/Off-line case. Then, we precisely define the properties required by the sigma protocol, and as a last step we will describe and prove the security of our construction.

5.1 On-line/Off-line signatures

On-Line/Off-Line digital signature were introduced by Even, Goldreich and Micali in [35]. In such schemes the signature process consists of two parts: a computationally intensive one that can be done Off-Line (i.e. when the message to be signed is not known) and a much more efficient online phase that is done once the message becomes available. There are two general ways to construct on-line/off-line signatures: using one time signatures [35] or using chameleon hash [70].
The first construction works by combining two different kind of digital signatures: standard signatures and one time signatures[1].
The second construction works by combining standard signatures with chameleon hash functions.
In [24] Catalano *et al.*, unified the two approaches by showing that they can be seen as different instantiations of the same paradigm.

5.2 Linearly Homomorphic On-line/Off-line signatures

First, we remark that the only difference between a LHSG and a LHOOS is in the signing algorithm. When signing m the latter can use some data prepared in advance (by running a dedicated algorithm **OffSign**) to speed up the signature process. The definitions of unforgeability are therefore analogous to the ones of traditional LHSG schemes and are omitted to avoid repetition[2].

[1]Differently form the first, One Time signatures allow users to sign only one message using the same signing key.

[2]We stress, however, that those definitions are stronger than the ones traditionally introduced for network coding (i.e. the adversary is more powerful and there are more

Definition 25 (LHOOS) *A Linearly Homomorphic On-line/Off-line signature scheme is a tuple of PPT algorithms (***KeyGen***,* **OffSign***,* **OnSign***,* **Verify***,* **Eval***) such that:*

- **KeyGen**$(1^\lambda, n, k)$ *takes as input the security parameter* λ*, an integer* n *denoting the length of vectors to be signed and an upper bound* k *for the number of messages signed in each dataset. It outputs a secret signing key* sk *and a public verification key* vk*; the public key implicitly defines a message space that can be seen as a vector space of the form* $\mathcal{M} = \mathbb{F}^n$ *(where* $\mathbb{F}$ *is a field), a file identifier space* $\mathcal{D}$ *and a signature space* Σ*.*

- **OffSign**(sk) *takes as input the secret key and outputs some information* I*.*

- **OnSign**$(sk, fid, I, \mathbf{m}, i)$ *takes as input the secret key, an element* $\mathbf{m} \in \mathcal{M}$*, an index* $i \in \{1, \ldots, k\}$*, a dataset identifier* fid *and an instance of* I *output by* **OffSign***. This algorithm must ensure that all the signatures issued for the same* fid *are computed using the same information* I *(i.e. by associating each* fid *with one specific* I *and storing these couples on a table). It outputs a signature* σ*.*

- **Verify** $(vk, \sigma, \mathbf{m}, fid, f)$ *takes as input the public key* vk*, a signature* $\sigma \in \Sigma$*, a message* $\mathbf{m} \in \mathcal{M}$*, a dataset identifier* $fid \in \mathcal{D}$ *and a function* $f \in \mathbb{Z}^k$*; it outputs* 1 *(accept) or 0 (reject).*

- **Eval**$(vk, fid, f, \{\sigma_i\}_{i=1\ldots k})$ *takes as input the public key* vk*, a dataset identifier* fid*, an admissible function* f *in its vector form* $(\alpha_1, \ldots, \alpha_k)$*, a set of* k *signatures* $\{\sigma_i\}_{i=1\ldots k}$ *and outputs a signature* $\sigma \in \Sigma$*. Note that this algorithm should also work if less than* k *signatures are provided, as long as their respective coefficients in the function* f *are 0, but we don't to explicitly account this to avoid heavy notation.*

The correctness conditions of our scheme are the following:

---

types of forgeries), and therefore our efficient instantiation perfectly integrates in that framework.

- *Let* $(sk, vk) \leftarrow \mathbf{KeyGen}(1^\lambda, n, k)$ *be an honestly generated keypair,* $\mathbf{m} \in \mathcal{M}$*, fid any dataset identifier and* $i \in 1, \ldots, k$*.*
  If $\sigma \leftarrow \mathbf{Sign}(sk, fid, \mathbf{OffSign}(sk), \mathbf{m}, i)$*, then with overwhelming probability*
  $$\mathbf{Verify}(vk, \sigma, \mathbf{m}, fid, e_i) = 1,$$
  where e_i *is the* i^{th} *vector of the standard basis of* $\mathbb{Z}^k$*.*

- *Let* $(sk, vk) \leftarrow \mathbf{KeyGen}(1^\lambda, n, k)$ *be an honestly generated keypair,* $\mathbf{m}_1, \ldots, \mathbf{m}_k \in \mathcal{M}$ *any tuple of messages signed (or derived from messages originally signed) w.r.t the same fid (and therefore using the same offline information I), and let* $\sigma_1, \ldots, \sigma_k \in \Sigma$*,* $f_1, \ldots, f_k \in \mathcal{F}$ *such that for all* $i \in \{1, \ldots, k\}$*,* $\mathbf{Verify}(vk, \sigma_i, \mathbf{m}_i, fid, f_i) = 1$*. Then, for any admissible function* $f = (\alpha_1, \ldots, \alpha_k) \in \mathbb{Z}^k$*, with overwhelming probability*
  $$\mathbf{Verify}(vk, \mathbf{Eval}(vk, fid, f, \{\sigma_i\}_{i=1\ldots k}), f(\mathbf{m}_1, \ldots, \mathbf{m}_k), fid, \sum_{i=0}^{k} \alpha_i f_i) = 1$$

Remark 7 *As for the case of LHSG, relaxing the requirements for the fid (i.e. assuming that it can be chosen offline independently of the message or that it can even be completely random) typically improves efficiency. See remark 9 for an example of how this idea applies to our instantiation.*

Definition 26 (LHOOS CMA) *An LHOOS is unforgeable against a chosen message attack if for all n the advantage of any PPT adversary* $\mathcal{A}$ *in the following game is negligible in the security parameter* λ*:*
Setup *The challenger runs* $\mathbf{KeyGen}(1^\lambda, n, k)$ *and gives vk to* $\mathcal{A}$*. The message space* $\mathcal{M}$*, the signature space* Σ *and the dataset space* $\mathcal{D}$ *are all implicitly defined by the verification key.*
Signing Queries $\mathcal{A}$ *can ask a polynomial number of queries of the form* $(\mathbf{m}_i, fid, i)$ *(where* $\mathbf{m}_i \in \mathcal{M}$ *is a message, fid is a dataset identifier and* $i \in \{1, \ldots, k\}$ *is an index), and get the corresponding signatures by the challenger. No two queries where only the message* $\mathbf{m}_i$ *changes can be asked by the adversary (if this happens, the answer to the second query is* $\perp$*).*
Forgery $\mathcal{A}$ *outputs a dataset identifier* fid^**, a message* $\mathbf{m}^*$*, a signature* σ^* *and a vector* $\alpha^* \in \mathbb{Z}^k$*.*

The Adversary wins the game if $\mathbf{Verify}(vk, \mathit{fid}^*, \mathbf{m}^*, \sigma^*, \alpha^*) = 1$ *and, called* $\mathbf{m}_1, \ldots, \mathbf{m}_k$ *the messages (possibly) queried by the adversary for the identifier* fid^**, either*

- *there exists* i *such that* $\alpha_i^* \neq 0$*, but no message w.r.t. index* i *and* fid^* *has been queried by the adversary.*
- *the previous condition does not occur, and* $\mathbf{m}^* \neq \sum_{i=1}^{k} \alpha_i \mathbf{m}_i$

Finally we define the advantage $\mathbf{Adv}^{LHOOS-RMA}(\mathcal{A})$ *of* $\mathcal{A}$ *as the probability that* $\mathcal{A}$ *wins the game.*

One can give an analogous notion of strong security against a chosen message attack, where even a new signature for a previously received (or derived) message is considered a forgery. Formally second condition can be replace by:

- The previous condition does not occur, $\mathbf{m}^* = \sum_{i=1}^{k} \alpha_i \mathbf{m}_i$ but $\sigma^* \neq \mathbf{Eval}(\mathrm{vk}, \mathrm{fid}, (\alpha_i)_{i=1,\ldots,k}, \{\sigma_i\}_{i=1,\ldots,k})$.

5.3 Vector and Homomorphic Σ-protocols

Briefly speaking, a Σ-Protocol can be described as a tuple of four algorithms (**Σ-Setup**, **Σ-Com**, **Σ-Resp**, **Σ-Verify**), where the first one generates a statement/witness couple, **Σ-Com** and **Σ-Resp** generate the first and third message of the protocol, and **Σ-Verify** is used by the verifier to decide on the validity of the proof (a more formal and detailed description was given in section 2.2.4.6). This notion can be extended to the vector case[3]. For this purpose we adapt the notion of Homomorphic Identification Protocol originally introduced in [8] to the Sigma protocol framework.

Given a language L and an integer $n \in \mathbb{N}$, we can consider the language $L^n = \{(x_1, \ldots, x_n) \mid x_i \in L \ \forall i = 1, \ldots, n\}$. A natural witness for a tuple (vector) in this language is the tuple of the witnesses of each of its components

[3]The intuition is that it should be more efficient to run a vector Σ-Protocol once than a standard Σ-Protocol multiple times in parallel)

for the language L. As before we can consider the relation $\mathcal{R}^n$ associated to L^n, where $(\vec{x}, \vec{w}) = (x_1, \ldots, x_n, w_1, \ldots, w_n) \in \mathcal{R}^n$ if $(x_1, \ldots, x_n)$ is part of L^n and w_i is a witness for x_i. A *vector* Σ-Protocol for $\mathcal{R}^n$ is a three rounds protocol defined similarly as above with the relaxation that the special soundness property is required to hold in a weaker form. Namely, we require the existence of an efficient extractor algorithm $\mathbf{\Sigma_n}$**-Ext** such that $\forall\, \vec{x} \in L^n$, $\forall\ R, \vec{c}, \vec{s}, \vec{c'}, \vec{s'}$ such that $(c, s) \neq (c', s')$, $\mathbf{\Sigma_n}$**-Verify**$(\vec{x}, R, \vec{c}, \vec{s}) = 1$ and $\mathbf{\Sigma_n}$**-Verify**$(\vec{x}, R, \vec{c'}, \vec{s'}) = 1$, outputs $(x, w) \leftarrow \mathbf{\Sigma_n}$**-Ext**$(\vec{x}, R, \vec{c}, \vec{s}, \vec{c'}, \vec{s'})$ where x is one of the components of $\vec{x}$ and $(x, w) \in \mathcal{R}$.
Another important requirement for our construction to work is the following property.

Definition 27 *A Σ-Protocol $\Sigma = (\mathbf{\Sigma}$**-Setup**$, \mathbf{\Sigma}$**-Com**$, \mathbf{\Sigma}$**-Resp**$, \mathbf{\Sigma}$**-Verify**$)$ for a relation $\mathcal{R}$ is called* group homomorphic *if*

- *The outputs of the* $\mathbf{\Sigma}$**-Com** *algorithm and the challenge space of the protocol can be seen as elements of two groups $(\mathbb{G}_1, \odot)$ and $(\mathbb{G}_2, \otimes_2)$ respectively*

- *There exists a PPT algorithm* **Combine** *such that, for all $(x, w) \in \mathcal{R}$ and all $\alpha \in \mathbb{Z}^n$, if transcripts $\{(R_i, c_i, s_i)\}_{i=1,\ldots,n}$ are such that* $\mathbf{\Sigma}$**-Verify**$(x, R_i, c_i, s_i) = 1$ *for all i, then*

$$\mathbf{\Sigma}\textbf{-Verify}\left(x, \bigodot_{i=1}^{n} R_i^{\alpha_i}, \bigotimes_{i=1}^{n} c_i^{\alpha_i}, \textbf{Combine}(x, \alpha, \{(R_i, c_i, s_i)\}_{i=1,\ldots,n})\right) = 1$$

Although it is given for the standard case, this property can easily be extended to vector Σ-Protocols: in particular, the group $\mathbb{G}_2$ can be seen as the group of vectors of elements taken from another group $\mathbb{G}$.

To sum up, we define a class of vector Σ-Protocols having all the properties required by our construction:

Definition 28 (1-n (vector) Σ-Protocol) *Let $(\mathbb{G}_1, \odot)$, $(\mathbb{G}_2, \otimes)$ be two computational groups. A 1-n vector sigma protocol consists of four PPT algorithm $\Sigma_n = (\mathbf{\Sigma_n}$**-Setup**$, \mathbf{\Sigma_n}$**-Com**$, \mathbf{\Sigma_n}$**-Resp**$, \mathbf{\Sigma_n}$**-Verify**$)$ defined as follows:*

$\mathbf{\Sigma_n}$**-Setup**$(1^\lambda, n, \mathcal{R}^n) \rightarrow (\mathbf{x}, \mathbf{w})$. *It takes as input a security parameter λ, a vector size n and a relation $\mathcal{R}^n$ over a language L^n. It returns a vector of statements and witnesses $(x_1, \ldots, x_n, w_1, \ldots, w_n)$. The challenge space is required to be* $\mathbf{ChSp} \subseteq \mathbb{G}_2^n$.

$\mathbf{\Sigma_n}$**-Com**$(\mathbf{x}) \rightarrow (R, r)$. *It's a PPT algorithm run by the prover to get the first message R to send to the verifier and some private state to be stored. We require that $R \in \mathbb{G}_1$.*

$\mathbf{\Sigma_n}$**-Resp**$(\mathbf{x}, \mathbf{w}, r, \mathbf{c}) \rightarrow s$. *It's a PPT algorithm run by the prover to compute the last message of the protocol. It takes as input the statements and witnesses* $(\mathbf{x}, \mathbf{w})$ *the challenge string* $\mathbf{c} \in \mathbf{ChSp}$ *(sent as second message of the protocol) and some state information r. It outputs the third message of the protocol, s.*

$\mathbf{\Sigma_n}$**-Verify**$(\mathbf{x}, R, \mathbf{c}, s) \rightarrow \{0, 1\}$. *It's the verification algorithm that on input the message R, the challenger* $\mathbf{c} \in \mathbf{ChSp}$ *and a response s it outputs* 1 *(accept) or* 0 *(reject).*

We require this protocol to be group homomorphic and to satisfy the completeness and special honest verifier zero knowledge properties. Moreover, the protocol must guarantee either the vector special soundness outlined above or a stronger soundness property that we define below. Roughly speaking, this property requires that the extractor, upon receiving the witnesses for all but one statements of the vector $\vec{x}$, has to come up with a witness for the remaining one.

Definition 29 (Strong (Vector) Special Soundness) *Let* $\Sigma = (\mathbf{\Sigma}$**-Setup**, $\mathbf{\Sigma}$**-Com**, $\mathbf{\Sigma}$**-Resp**, $\mathbf{\Sigma}$**-Verify**$)$ *be a 1-n Σ-Protocol for a relation $\mathcal{R}^n$. We say that Σ has the* Strong Special Soundness *property if there exist an efficient extractor algorithm* $\mathbf{\Sigma_n}$**-Ext** *such that* $\forall\, \vec{x} \in L^n, \forall j^* \in \{1, \ldots, n\}$, $\forall\, R, \vec{c}, \vec{s}, \vec{c'}, \vec{s'}$ *such that* $c_{j^*} \neq c'_{j^*}$, $\mathbf{\Sigma_n}$**-Verify**$(\vec{x}, R, \vec{c}, \vec{s}) = 1$ *and* $\mathbf{\Sigma_n}$**-Verify**$(\vec{x}, R, \vec{c'}, \vec{s'}) = 1$, *outputs* $w_{j^*} \leftarrow \mathbf{\Sigma_n}$**-Ext**$(\vec{x}, R, \vec{c}, \vec{s}, \vec{c'}, \vec{s'}, \{w_j\}_{j \neq j^*})$ *such that* $(x_{j^*}, w_{j^*}) \in \mathcal{R}$.

In the next section we show that a simple variant of the well known identification protocol by Schnorr is a 1-n Σ-Protocol (with Strong Vector Special Soundness).

5.3.1 Schnorr 1-n Σ-Protocol

Here we describe a 1-n vector Σ-Protocol satisfying the Strong Vector Special Soundness Property.

Definition 30 (Schnorr 1-n Σ-Protocol) *Let $\mathbb{G}$ a group of prime order p and $\mathcal{R}$ the DL relation on $\mathbb{G}$, DL= $\{(x,w)|x=(p,g,h),\ \ h=g^w\}$. Let $\overline{g}\in\mathbb{G}$ a group generator. We define $DL_{\overline{g}}=\{(x,w)|x=\overline{g}^w\}$ the restriction of the DL relation to $g=\overline{g}$.*
The (Strong) Schnorr (Vector) 1-n Σ-Protocol consists of four PPT algorithm $\Sigma_n=(\mathbf{\Sigma_n}\text{-}\mathbf{Setup},\ \mathbf{\Sigma_n}\text{-}\mathbf{Com},\mathbf{\Sigma_n}\text{-}\mathbf{Resp},\mathbf{\Sigma_n}\text{-}\mathbf{Verify})$ defined as follows:

- $\mathbf{\Sigma_n}$-$\mathbf{Setup}(1^\lambda,n,\mathcal{R})$ *It chooses a random group generator $g\in\mathbb{G}$ and a vector of witnesses $\vec{w}=(w_1,\ldots,w_n)\xleftarrow{\$}\mathbb{Z}_p^n$. Then it computes the vector of statements $(x_1,\ldots,x_n)\leftarrow(g^{w_1},\ldots,g^{w_n})$ and sets $\vec{x}\leftarrow(x_1,\ldots,x_n,g)$. Then it outputs $(\vec{x},\vec{w})$. Obviously the couple $(x_j,w_j)\in DL_g\ \ \forall j=1,\ldots,n$.*

- $\mathbf{\Sigma_n}$-$\mathbf{Com}(\vec{x})$ *It chooses a random $r\in\mathbb{Z}_p$, sets $R\leftarrow g^r$ and returns (r,R).*

- $\mathbf{\Sigma_n}$-$\mathbf{Resp}(\vec{x},\vec{w},r,\vec{c})$ *Let $\vec{c}\in\mathbb{Z}_p^n$ the second message of the protocol. This algorithm outputs $s\leftarrow r+\sum_{j=1}^n c_jw_j$.*

- $\mathbf{\Sigma_n}$-$\mathbf{Verify}(\vec{x},R,\vec{c},s)$ *It checks that*

$$g^s=R\prod_{j=1}^n x_j^{c_j}.$$

  If the above equation holds, it outputs 1, else outputs 0.

5.4 Signatures and Σ-Protocols

Signatures and Σ-Protocols are very related. In this section we recall the Fiat-Shamir method to transform Identification Protocols in Digital Signatures as described in [36]. This construction work as follow.
Let $\Sigma=(\mathbf{\Sigma}\text{-}\mathbf{Setup},\mathbf{\Sigma}\text{-}\mathbf{Com},\mathbf{\Sigma}\text{-}\mathbf{Resp},\mathbf{Verify})$ a standard sigma protocol

where the Special Soundness property holds and $\mathcal{H}$ a family of CR hash functions. Then we can construct a signature scheme $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$ where:

KeyGen(1^λ). It runs $(x, w) \leftarrow$ **Σ-Setup**(1^λ). Then it picks an hash function $H \in \mathcal{H}$ and sets $\text{vk} \leftarrow (w, H)$, $\text{sk} \leftarrow x$.

Sign(sk, m). It runs $(r, R) \leftarrow$ **Σ-Com**(x), sets $c \leftarrow H(R, m)$ and produces $z \leftarrow$ **Σ-Resp**(x, w, r, c). Then output $\sigma \leftarrow (R, z)$.

Verify(vk, m, σ). Let $\sigma = (R, z)$. It compute $c \leftarrow H(R, m)$. If **Σ-Verify**$(x, R, c, z) = 1$ then accept else reject.

Theorem 13 *If* $\Sigma = (\mathbf{\Sigma\text{-}Setup}, \mathbf{\Sigma\text{-}Com}, \mathbf{\Sigma\text{-}Resp}, \mathbf{Verify})$ *is a standard sigma protocol where the Special Soundness property holds and* $\mathcal{H}$ *a family of CR hash functions then the signature scheme* $\mathcal{S}$ *is uf-cma secure in the random oracle model.*

5.5 A Linearly Homomorphic On-Line/Off-Line Signature

Suppose $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ is a randomly secure LHSG (even one that only allows to sign scalars), $\Sigma_n = (\mathbf{\Sigma_n\text{-}Setup}$,$\mathbf{\Sigma_n\text{-}Com}$,$\mathbf{\Sigma_n\text{-}Resp}$,$\mathbf{\Sigma_n\text{-}Verify}$) is a 1-n Σ-Protocol and $\mathcal{H} = (\mathbf{CHGen}, \mathbf{CHEval}, \mathbf{CHFindColl})$ defines a family of chameleon hash functions[4]. Moreover, suppose that the LHSG's message space is the same as the group $\mathbb{G}_1$ of the outputs of $\mathbf{\Sigma_n\text{-}Com}$. Our generic construction uses the challange space of the Σ-Protocol as a message space and works as follows:

ON/OFFKeyGen $(1^\lambda, k, n)$: It runs $(\text{vk}_1, sk_1) \leftarrow$ **KeyGen**$(1^\lambda, 1, k)$, $(\mathbf{x}, \mathbf{w}) \leftarrow$**Σ$_\mathbf{n}$-Setup** $(1^\lambda, n, \mathcal{R}^n)$ and $(\text{hk}, \text{ck}) \leftarrow$ **CHGen**(1^λ). It outputs $\text{vk} \leftarrow (vk_1, \mathbf{x}, \text{hk})$, $\text{sk} \leftarrow (sk_1, \mathbf{w}, \text{ck})$.

[4]a formal definition of chameleon hash functions can be found in section 2.2.4

OFFSign (sk): This algorithm runs the $\mathbf{\Sigma_n}$**-Com** algorithm k times to obtain $(R_i, r_i) \leftarrow \mathbf{\Sigma_n}$**-Com** $(\mathbf{x})$, chooses a random string fid' from the dataset identifiers' space and randomness ρ' and sets $\overline{\mathrm{fid}} \leftarrow$ **CHEval**(hk, fid', ρ'). Then it signs each R_i using the LHSG signing algorithm $\overline{\sigma_i} \leftarrow$ **Sign**$(sk_1, R_i, \overline{\mathrm{fid}}, i)$ and outputs $I_{\mathrm{fid}'} = \{(i, r_i, R_i, \overline{\sigma_i}, \mathrm{fid}', \rho')\}_{i=1,\ldots,k}$.

ONSign $(\mathrm{vk}, \mathrm{sk}, \mathbf{m}, \mathrm{fid}, I_{\mathrm{fid}'}, i)$: It parses $I_{\mathrm{fid}'}$ as $\{(i, r_i, R_i, \overline{\sigma_i}, \mathrm{fid}', \rho')\}_{i=1,\ldots,k}$, computes $s \leftarrow \mathbf{\Sigma_n}$**-Resp** $(\mathbf{x}, \mathbf{w}, r_i, \mathbf{m})$, $\rho \leftarrow$ **CHFindColl**$(\mathrm{ck}, \mathrm{fid}', \rho', \mathrm{fid})$ and outputs $\sigma \leftarrow (R_i, \overline{\sigma}_i, s, \rho)$. As explained in the definition, this algorithm must ensure that all the messages signed with respect to the same fid are computed from the same information $I_{\mathrm{fid}'}$

ON/OFFVerify $(\mathrm{vk}, \sigma, \mathbf{m}, \mathrm{fid}, f)$: It parses σ as $(R, \overline{\sigma}, s, \rho)$ and vk as $(\mathrm{vk}_1, \mathbf{x}, \mathrm{hk})$. Then it checks that **Verify**$(\mathrm{vk}_1, \overline{\sigma}, R,$ **CHEval**$(\mathrm{fid}, \rho), f) = 1$ and $\mathbf{\Sigma_n}$**-Verify**$(\mathbf{x}, R, \mathbf{m}, s) = 1$. If both the above equations hold it returns 1, else it returns 0.

ON/OFFEval $(\mathrm{vk}, \alpha, \sigma_1, \ldots, \sigma_k)$: it parses σ_i as $(R_i, \overline{\sigma}_i, s_i, \rho)$ for each $i = 1, \ldots, k$ and vk as $(\mathrm{vk}_1, \mathbf{x})$. Then it computes:

$$R \leftarrow R_1^{\alpha_1} \odot \cdots \odot R_k^{\alpha_k}, \quad \overline{\sigma} \leftarrow \mathbf{Eval}(\mathrm{vk}_1, \alpha, \overline{\sigma}_1, \ldots, \overline{\sigma}_k),$$

$$s \leftarrow \mathbf{Combine}\left(\vec{x}, \alpha, \{(R_i, c_i, s_i)\}_{i=1,\ldots,k}\right).$$

Finally it returns $(R, \overline{\sigma}, s, \rho)$ (as a signature for the message $\mathbf{m}_1^{\alpha_1} \otimes \cdots \otimes \mathbf{m}_k^{\alpha_k}$).

Remark 8 *The construction presented above applies to any LHSG. However, if the LHGS itself is obtained as described in section 3.4, the use of the chameleon hash function could be avoided by substituting the signature scheme $\mathcal{S}$ used for the fid with an on-line/off-line one. This improves efficiency.*

Remark 9 *Loosening the requirements on the fid can also help improving the performance of the scheme. For example, if the fid is allowed to be chosen randomly by the signer and not by the adversary, the use of the chameleon hash function can be avoided. Moreover, in this case one could use a LHSG that achieves better efficiency by choosing the fid itself (an example is described in remark 4).*

Theorem 14 *If* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ *is a randomly secure LHSG,* $\Sigma_n = (\mathbf{\Sigma_n\text{-}Setup}, \mathbf{\Sigma_n\text{-}Com}, \mathbf{\Sigma_n\text{-}Resp}, \mathbf{\Sigma_n\text{-}Verify})$ *is a* 1*-*n Σ*-Protocol for a non trivial relation* $\mathcal{R}^n$*, and* $\mathcal{H}$ *implements a family of chameleon hash functions then the LHOOS described above is secure against a chosen message attack according to definition 21.*

Proof of theorem 14 Here we present a proof sketch in the simplified case where each signing query is immediately followed by the corresponding reveal query. In this setting we can just assume that the evaluation queries are computed by the adversary itself and that for each encryption query it gets the corresponding message/signature couple. A more detailed proof for the general case is deferred to a full version of the paper.
Suppose $\mathbf{m}^*, (R^*, \overline{\sigma}^*, s^*)$ is the forgery returned by an adversary $\mathcal{A}$ w.r.t the identifier fid^* and the vector $\vec{\alpha}^* = (\alpha_1^*, \ldots, \alpha_k^*)$. Let $\{\sigma_1, \ldots, \sigma_k\}$ the set of signatures seen by $\mathcal{A}$ w.r.t. the same identifier fid^* and the messages $(\mathbf{m}_1, \ldots, \mathbf{m}_k)$. Note that, because $\mathcal{S}$ is secure against random message attacks, and $\mathcal{H}$ is collision resistant, it must be that $\prod_{i=1}^k R_i^{\alpha_i} = R^*$ and $\text{fid}^* = \text{fid}$ for some fid that $\mathcal{A}$ has received during the security game. Therefore, by the security definition, the only kind of forgery the adversary can make is one where $\mathbf{m}^* \neq \mathbf{m}_1^{\alpha_1^*} \otimes \cdots \otimes \mathbf{m}_k^{\alpha_k^*}$.
In this case we describe a simulator $\mathcal{B}$ that uses $\mathcal{A}$ to extract a witness for the language L such that L^n is the language associated to the relation $\mathcal{R}^n$. We will assume first that the underlying Σ-Protocol has the vector special soundness, and explain later how to modify the proof in the case where the strong vector special soundness holds. $\mathcal{B}$ takes as input a vector of statements $\vec{x} \in L^n$. It must then return a couple (x, w) such that x is a component of $\vec{x}$ and $(x, w) \in \mathcal{R}$. It works as follows.
Key Generation. $\mathcal{B}$ runs $(\text{vk}_1, \text{sk}_1) \leftarrow \mathbf{KeyGen}(1^\lambda, k, n)$ and gives to $\mathcal{A}$ $\text{vk} = (\text{vk}_1, \vec{x})$. It is easy to check that this key is correctly distributed as in the real case.
Signing queries. Each time $\mathcal{A}$ asks for a signature on a message $\mathbf{m}_i$ w.r.t. an identifier fid and to an index $i \in 1, \ldots, k$, $\mathcal{B}$ uses the HVZK simulator of the sigma protocol to compute $(R_i, s_i) \leftarrow S(\mathbf{x}, m)$. Then it computes a signature $\overline{\sigma_i} \leftarrow \mathbf{Sign}(\text{sk}_1, \text{fid}, R_i, i)$ on R_i and returns the signature $\sigma \leftarrow (R_i, \overline{\sigma}_i, s_i)$ to $\mathcal{A}$.

Forgery Suppose $\mathcal{A}$ returns a forgery of type 1 $(R^*, \overline{\sigma}^*, s^*)$ for the message $\mathbf{m}^*$. Let $\overline{\mathbf{m}} = \mathbf{m}_1^{\alpha_1^*} \otimes \cdots \otimes \mathbf{m}_k^{\alpha_k^*}$ and $\overline{\mathbf{s}} = \mathbf{Combine}\,(\mathbf{x}, \alpha^*, \{(R_i, \mathbf{m}_i, s_i)\}_{i=1,\ldots,k})$.
$\mathcal{B}$ uses the extractor $(x, w) \leftarrow \mathbf{\Sigma\text{-}Ext}\ (\mathbf{x}, R, \mathbf{m}^*, s^*, \overline{\mathbf{m}}, \overline{\mathbf{s}})$ from the vector special soundness to obtain a witness for one of the components of $\vec{x}$.

In the case where the strong vector special soundness holds, since $\mathbf{m}^* \neq \mathbf{m}_1^{\alpha_1^*} \otimes \cdots \otimes \mathbf{m}_k^{\alpha_k^*}$, there exists and index j such that the j-th components of the two sides of the equation are different. $\mathcal{B}$ can guess the index j in advance and prepare the public key by generating $n-1$ couples $(x_i, w_i) \in L$ for $i \in \{1, \ldots, n\} \setminus j$, setting $x_j \leftarrow x$ (where x is the statement $\mathcal{B}$ received in input) and $\vec{x} \leftarrow (x_1, \ldots, x_n)$. The rest of the public key and the other phases of the simulation are carried as in the previous case. When $\mathcal{A}$ provides a forgery, assuming $\mathcal{B}$ guessed the correct index (this will happen with non negligible probability, otherwise $\mathcal{B}$ aborts), $\mathcal{B}$ can use the extractor for the strong vector special soundness to get the missing witness $w_j \leftarrow \mathbf{\Sigma\text{-}Ext}$ $(\mathbf{x}, R, \mathbf{m}^*, s^*, \overline{\mathbf{m}}, \overline{\mathbf{s}}, \{w_i\}_{i \neq j})$.

The security obtained by this construction can be strengthened by assuming additional properties on the underlying LHSG scheme: if $\mathcal{S}$ is strongly secure against a random message attack, then we can prove that the resulting construction is strongly secure (against a CMA) as well.

Theorem 15 *If* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify}, \mathbf{Eval})$ *is a LHSG scheme strongly unforgeable against a random message attack and* $\Sigma_n = (\mathbf{\Sigma_n\text{-}Setup}, \mathbf{\Sigma_n\text{-}Com}, \mathbf{\Sigma_n\text{-}Resp}, \mathbf{\Sigma_n\text{-}Verify})$ *is a* 1*-*n Σ*-Protocol for a non trivial relation* $\mathcal{R}^n$*, then the on-line/off-line scheme described above is* ***strongly*** *unforgeable against chosen message attacks.*

The proof is straightforward and similar to the previous one and therefore is omitted.

Chapter 6

Conclusions

In this thesis we focus on linearly homomorphic primitives and, in particular, we concentrate our attention on the homomorphic version of two different primitives: signatures scheme and authenticated encryption schemes.
This thesis essentially consists of two parts: the first (chapter 1) contains some standard definitions related to signatures and encryption schemes and it can be considered like a briefly summary, in order to provide the reader all basic notions related to the following chapters. Each chapter of the second part (chapters 2 and following), instead, focuses on a new different primitives (LAEPuV, LHSG, LHOOS).
Each of these last chapters is divided in two parts: in the first part we formally provide definitions for each new primitives, then we propose different instantiations for each of them.
More in detail. About LHSG we show first a random message secure construction based on the 2-3 CDH problem. Then we show a general transform to construct CMA LHSG from RMA LHSG. Finally we propose a concrete instantiation, slightly optimized, of a CMA secure LHSG.
These results about LHSG are interesting for two reasons. First the transformation from RMA to CMA secure LHSG can be applied to structure preserving signatures too. Second, in chapter 5 we show that the RMA secure version of our LHSG in enough to construct an LHOOS secure against chosen message attack.
About LAEPuV, first we present an instantiation supporting Paillier's ci-

phertexts, then we show how to generalize our result to support arbitrary schemes satisfying some well defined homomorphic properties.
This result is very interesting because it's the first efficient construction of a Linearly Homomorphic Authenticated Encryption Scheme. We additionally show,in the introduction chapter, a nice application, in order to motivate this primitive. This example is related to the increasing relevant scenario where a user wants to store encrypted data on the cloud in a way such that he can later delegate the cloud to perform computation on this data.
Our research also leaves open some interesting questions:

- First we observe that our LGSG scheme can support only a polynomial size bounded dataset identifier. Indeed, if we want to sign a dataset with exponential size, we need an exponentially bigger key. A first way to attempt to this observation is to use a random oracle H. More in detail, it is possible to prove that the proposed LHSG is still secure by replacing the public key elements h_i with a random oracle H and setting $h_i = H(i)$. However, in this case, it is possible to prove the security of such modified scheme only in the random oracle model. A first interesting question is: is it possible to sign an exponential bigger (or unbounded) dataset, in the standard model, using a short public key?

- The second open question is related to our LAEPuV construction. We prove its security only in the random oracle model. Prove its security in the standard model can be an interesting problem.

- The final open question is related again to our LAEPuV construction. A bad limitation of our construction is that it can be applied only to linear functions. Is it possible to extend it to a wider functionalities class?

Bibliography

[1] Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 4–24. Springer, December 2012.

[2] Masayuki Abe, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Tagged one-time signatures: Tight security and optimal tag size. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 312–331. Springer, February / March 2013.

[3] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236. Springer, August 2010.

[4] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 649–666. Springer, August 2011.

[5] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Signing on elements in bilinear groups for modular protocol design. Cryptology ePrint Archive, Report 2010/133, 2010. http://eprint.iacr.org/.

[6] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Group to group commitments do not shrink. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 301–317. Springer, April 2012.

[7] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 1–20. Springer, March 2012.

[8] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 319–333. Springer, December 2009.

[9] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 17–34. Springer, March 2011.

[10] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 367–385. Springer, December 2012.

[11] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013:*

16th International Workshop on Theory and Practice in Public Key Cryptography, volume 7778 of *Lecture Notes in Computer Science*, pages 386–404. Springer, February / March 2013.

[12] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, August 1998.

[13] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 68–87. Springer, March 2009.

[14] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 149–168. Springer, May 2011.

[15] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 1–16. Springer, March 2011.

[16] Dan Boneh and Richard J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 283–297. Springer, August 1996.

[17] Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In Moti Yung, Yevgeniy

Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 229–240. Springer, April 2006.

[18] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS 12, pages 309–325. ACM, 2012.

[19] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106. IEEE Computer Society Press, October 2011.

[20] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, August 2011.

[21] Ernest F. Brickell and Yacov Yacobi. On privacy homomorphisms (extended abstract). In David Chaum and Wyn L. Price, editors, *Advances in Cryptology – EUROCRYPT'87*, volume 304 of *Lecture Notes in Computer Science*, pages 117–125. Springer, April 1988.

[22] Jan Camenisch, Maria Dubovitskaya, and Kristiyan Haralambiev. Efficient structure-preserving signature scheme from standard assumptions. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 76–94. Springer, September 2012.

[23] Jan Camenisch, Kristiyan Haralambiev, Markulf Kohlweiss, Jorn Lapon, and Vincent Naessens. Structure preserving CCA secure encryption and applications. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances*

in Cryptology – ASIACRYPT 2011, volume 7073 of *Lecture Notes in Computer Science*, pages 89–106. Springer, December 2011.

[24] Dario Catalano, Mario Di Raimondo, Dario Fiore, and Rosario Gennaro. Off-line/on-line signatures: Theoretical aspects and experimental results. In Ronald Cramer, editor, *PKC 2008: 11th International Conference on Theory and Practice of Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 101–120. Springer, March 2008.

[25] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 336–352. Springer, May 2013.

[26] Dario Catalano, Dario Fiore, Rosario Gennaro, and Konstantinos Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 680–699. Springer, mar 2012.

[27] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Adaptive pseudo-free groups and applications. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 207–223. Springer, May 2011.

[28] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 680–696. Springer, May 2012.

[29] Julien Cathalo, Benoît Libert, and Moti Yung. Group encryption: Non-interactive realization in the standard model. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 179–196. Springer, December 2009.

[30] Melissa Chase and Markulf Kohlweiss. A new hash-and-sign approach and structure-preserving signatures from DLIN. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 131–148. Springer, September 2012.

[31] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 487–504. Springer, August 2011.

[32] Yvo Desmedt. Computer security by redefining what a computer is. NSPW, 1993.

[33] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[34] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552. ACM Press, May 1991.

[35] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[36] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, August 1987.

[37] David Fifield. The equivalence of the computational diffie–hellman and discrete logarithm problems in certain groups, 2012.

[38] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7293

of *Lecture Notes in Computer Science*, pages 697–714. Springer, May 2012.

[39] Georg Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. Cryptology ePrint Archive, Report 2009/320, 2009. http://eprint.iacr.org/.

[40] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 177–194. Springer, May 2003.

[41] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, August 2010.

[42] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 142–160. Springer, May 2010.

[43] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, page 290. Springer, December 2012.

[44] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[45] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 116–137. Springer, August 2010.

[46] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 107–109. IEEE Computer Society Press, October 2011.

[47] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, May 2011.

[48] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. *STOC '82 Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 365–377, 1982.

[49] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[50] J. Groth. Homomorphic trapdoor commitments to group elements. Cryptology ePrint Archive, Report 2009/007, 2009. http://eprint.iacr.org/.

[51] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, December 2006.

[52] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, April 2008.

[53] Shai Halevi and Victor Shoup. Algorithms in helib. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 554–571, 2014.

[54] Dennis Hofheinz and Tibor Jager. Tightly secure signatures and public-key encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 590–607. Springer, August 2012.

[55] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer, February 2002.

[56] Chihong Joo and Aaram Yun. Homomorphic authenticated encryption secure against chosen-ciphertext attack. Cryptology ePrint Archive, Report 2013/726, 2013. http://eprint.iacr.org/.

[57] Marc Joye and Benoit Libert. Efficient cryptosystems from 2^k-th power residue symbols. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 76–92. Springer, May 2013.

[58] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. Xors in the air: practical wireless network coding. volume 16, pages 497–510, 2008.

[59] Sébastien Kunz-Jacques and David Pointcheval. About the security of MTI/C0 and MQV. In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 156–172. Springer, September 2006.

[60] Sébastien Kunz-Jacques and David Pointcheval. A new key exchange protocol based on MQV assuming public computations. In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 186–200. Springer, September 2006.

[61] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications.

In Ran Canetti; Juan A. Garay, editor, *Advances in Cryptology – CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 289–307. Springer, August 2013.

[62] Ueli Maurer and Stefan Wolf. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms, 1999.

[63] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *ACM CCS 98: 5th Conference on Computer and Communications Security*, pages 59–66. ACM Press, November 1998.

[64] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, May 1999.

[65] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[66] Ronald L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* Internet Activities Board, April 1992.

[67] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[68] Claus-Peter Schnorr. Efficient identification and signatures for smart cards (abstract) (rump session). In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 688–689. Springer, April 1990.

[69] Secure hash standard. National Institute of Standards and Technology, NIST FIPS PUB 180-1, U.S. Department of Commerce, April 1995.

[70] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, August 2001.

[71] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[72] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, May 2010.

[73] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, May 2010.

[74] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, May 2005.