UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA ED INFORMATICA
Ph. D in Computer Science

Federico Savasta

TWO-PARTY AND THRESHOLD ECDSA CONSTRUCTIONS WITH BANDWIDTH EFFICIENT INSTANTIATIONS FROM CLASS GROUPS OF IMAGINARY QUADRATIC FIELDS

Final Dissertation

Supervisor:
Prof. Dario Catalano - University of Catalania, Italy

Examiners:
Prof. Michel Abdalla - École Normale Supérieure, Paris, France
Prof. Manuel Barbosa - University of Porto, INESC TEC, Portugal

Academic Year 2020/2021

Introduction

One of the most interesting challenges in Cryptography is building distributed protocols which are efficient and secure. Distributed protocols are thought to compute a function among a set of connected parties, where these functions can range from, for example, computing a decryption of encrypted data to signing documents together. Previous example tasks belong to an area of Cryptography called *Secure Multi-Party Computation*, or simply Multi-Party Computation (MPC). MPC covers several situations where two or more connected parties wish to jointly compute a function on their own private inputs. The aim of MPC is to carry out a distributed task securely. The security properties an MPC should satisfy are not fixed and they depend on what we want to guarantee, the model considered or the kind of adversary we face. However, some properties are quite general, as *privacy* (no more than expected information from the output has to be obtained, then no additional infos about the inputs can be deduced), *correctness* (it is guaranteed that the output received is correct), or *indistinguishability of inputs* (malicious parties cannot choose their inputs as a function of honest parties inputs). Actually, in some contexts, protocols also *guarantee output delivery* to honest parties or *fairness*. Anyway, not all of these property are always guaranteed. In particular, latter two properties are not satisfied when an adversary can corrupt more than an half of the parties[1].

Threshold Cryptography The domain of the entire manuscript is Threshold Cryptography, a particular setting of the more general Multi-Party Computation. In Threshold Cryptography we consider n players that take part in a distributed protocol, where at most $t < n$ players cooperating cannot obtain a specific goal, but at least $t + 1$ of them can run the protocol to an end. Threshold Cryptography ([Des88, DF90, GJKR96b, SG98, Sho00, Boy86, CH89, MR04]) has a lot of advantages in the real life, e.g. it does not permit to an individual to sign sensitive documents that need to be approved by a quorum. This versatility sparked intense research efforts that, mainly in the decade from the early 1990s to the early 2000s, produced efficient threshold versions of most commonly used cryptographic schemes. If we look at the specific case of signatures, a set of at least $t + 1$ parties have to cooperate to generate a signature of a chosen message. The main interesting point consists in how threshold signature protocols permit to split the secret key in a way that each party is not able to reconstruct it in its entirety, and at the same time it is possible to compute the signature. As a consequence, it is clear that an attack to reconstruct the secret key would require the corruption of a significant number of parties. With a threshold scheme it is possible to avoid the problem of the *single point of failure* which is typical to centralized schemes. Indeed, the significance

[1]We do not detail about this, we refer to [HL10]

of the threshold value t is an indication on how many parties a malicious adversary $\mathcal{A}$ needs to corrupt to totally break the protocol, i.e. to obtain the secret key and being able to sign messages as $\mathcal{A}$ is impersonating a valid population of signers. Of course, obtaining the secret key and creating forgeries is not the only possible attack, it is also fundamental to detect misbehaving actions and to avoid that the protocol reaches an end revealing private informations of the honest players. Indeed, in some contexts it is necessary to take in account what infos a malicious adversary can obtain only seeing if an honest player aborts or not.

In this manuscript, we will consider exclusively Threshold ECDSA, the threshold variant of the Elliptic Curve Digital Signature Algorithm. We informally introduce threshold signatures, then we will look at the specific case motivating the aim of the manuscript during this introduction. As the name suggests, threshold signatures concern with the study of the techniques for building secure and efficient threshold signatures scheme. The task of a threshold signature scheme is computing a distributed signature function without revealing information on the secret key, or anything for what an adversary can compute a valid signature as a group of valid signers. An usual example is that of an agency that is going to accept a project, but it is necessary that a certain number of potential voters give their consensus. Certainly, no one should be able to accept the document individually, or the meaning of a distributed protocol is missed.

From the beginning of '80s and motivated by applications, cryptographers started to think on how widely used protocols can be converted in their distributed variant, noting that for some of them this task was hard. For example, initially multi-party version of RSA or Schnorr Signatures were presented, but for the case of ECDSA – the Elliptic Curve based variant of DSA – we had to wait for years. Recent years have seen renewed interest in the field (e.g. [GGN16, Lin17, GG18, DKLs18, LN18, DKLs19]) for several reasons. First a number of start-up companies are using this technology to protect keys in real life applications [Ser, Unb, Sep]. Moreover, Bitcoin and other cryptocurrencies – for which security breaches can result in concrete financial losses – use ECDSA as underlying digital signature scheme. While multisignature-based countermeasures are built-in to Bitcoin, they offer less flexibility and introduce anonymity and scalability issues (see [GGN16]).

How to build efficient solutions for distributed ECDSA is our main scope in this manuscript. We are not the first who propose an efficient solution for distributed ECDSA, but we will present our contributions in the construction of efficient distributed ECDSA schemes that improve the state of art. In this introduction chapter we introduce the issues behind the construction of secure and efficient threshold ECDSA schemes and we recall some of the most relevant works on this topic with the purpose to give an idea about the state of art and the starting point of our contributions. Detailed explanations, extended introductions to our works and the technical parts of our constructions will be given later in dedicated chapters. In this intro chapter we introduce previous works that inspired our constructions and we only briefly introduce our works.

Public parameters: An Elliptic Curve group $\mathbb{G}$, of prime order q, with generator P.

Secret Key: Random $x \in \mathbf{Z}/q\mathbf{Z}$.

Public Key: $Q \leftarrow x \cdot P \in \mathbb{G}$.

EC-Schnorr Sign Algorithm:

- Sample $k \hookleftarrow \mathbf{Z}/q\mathbf{Z}$
- $R \leftarrow k \cdot P$
- $e \leftarrow H(R||m)$
- $s \leftarrow k - x \cdot e \bmod q$
- Output (e, s)

ECDSA Sign Algorithm:

- Sample $k \hookleftarrow \mathbf{Z}/q\mathbf{Z}$
- $R \leftarrow k \cdot P$
- $r \leftarrow r_x \bmod q$ where $R = (r_x, r_y)$
- $\boxed{s \leftarrow k^{-1} \cdot (H(m) + rx) \bmod q}$
- Output (r, s)

Figure 1: EC-Schnorr/ECDSA Signing

Towards distributed ECDSA

How to build an efficient threshold ECDSA scheme became an interesting challenge, in particular from the '90s to the early 2000, motivated by the difficulty of this problem. In general, researchers started to think about how to construct distributed versions for most used schemes, such as RSA, DSA or Schnorr signatures. A stumbling block was presenting an efficient solution for threshold DSA/ECDSA. The best solution proposed until the early 2000 is that one of [MR04] for the two party case, but it is not efficient enough for applications. Furthermore, the lack of direct applications of Threshold ECDSA has put the interest in this problem a little apart. The main bottleneck in the construction of a distributed variant of ECDSA relies on its non linear structure, which does not permit to split all the elements involved in additive shares. To understand what this last sentence means, we give a informal description of ECDSA comparing it with Schnorr signature scheme ([Sch90]) adapted to the elliptic curve case in Figure 1.
Looking at Figure 1, it is clear that in Schnorr Signing algorithm the nonce k and the secret key x appears only in linear expressions. At the same time, a signature in ECDSA is computed as an expression depending on k^{-1} and on $k^{-1}x$. As a consequence, Schnorr algorithm can be adapted to a distributed version using additive shares of k and x among the players. Indeed, suppose that two players P_i and P_{3-i}, $i \in \{1, 2\}$, join the protocol and each P_i owns k_i and x_i. Then, each player can compute $R_i \leftarrow k_i \cdot P$ and share R_i. After receiving R_{3-i}, P_i can compute $R = R_i + R_{3-i}$. Under the hardness of the discrete logarithm in the Elliptic Curve, R_i does not reveal informations about k_i. After that, P_i computes $s_i \leftarrow k_i - x_i \cdot e \mod q$, shares it and receives s_{3-i}. Finally, the signature is computed as

$$s = s_i + s_{3-i} = (k_i + k_{3-i}) - (x_i + x_{3-i}) \cdot e \mod q = k - x \cdot e \mod q$$

where $k = k_i + k_{3-i}$ and $x = x_i + x_{3-i}$. A similar reasoning does not work in the case of ECDSA, since it is not clear how to compute additive shares of k^{-1} and $k^{-1}x$ from additive shares of k and x. The most intuitive idea to approach this problem is to use multiplicative shares instead of additive ones. This is the idea behind the solution of [MR04] in the two-party case, that was improved by Lindell ([Lin17] and generalized by our work ([CCL$^+$19]) in a different setting. Before talking about the improvements, we first introduce the technique used in [MR04] in the next section. About [Lin17], a brief introduction will be given in this chapter, while in Chapter 3 is entirely dedicated to our work [CCL$^+$19].

Two-party ECDSA

When we talk about the security of a distributed protocols, a relevant factor is the number of parties that an adversary can corrupt. In particular, it is of fundamental importance if the adversary is able to corrupt a majority of the parties – i.e. more than an half – or not. In the specific case of two party, an adversary can corrupt at most 1 party to guarantee that there is at least one honest party joining the protocol. Of course, in this specific case honest majority with corruptions cannot be achieved. This is the reasoning why it is better to split in two different situations two-party protocols and more general threshold protocols. That means that the ideas behind constructions of two-party and threshold protocols are quite different. In this section, we give a brief description of some of the solutions that bring two-party ECDSA protocols to the today's state-of-art. The choice of the following results is also done in relation to their importance to our works.

The MacKenzie-Reiter idea As discussed above, MacKenzie and Reiter ([MR04]) proposed a two-party protocol to compute DSA/ECDSA signatures. The authors treat the case of DSA, but we will use the elliptic curve notation to describe it. Their idea is using a multiplicative sharing of k and x distributed among the players. The advantage is avoiding the difficulty to compute additive shares of k^{-1} or $k^{-1}x$ from additive shares of k and x. Indeed, if $k = k_1k_2$, then $k^{-1} = k_1^{-1}k_2^{-1}$. Furthermore, their idea is to use an homomorphic encryption scheme to send and operate on encrypted values, which is Pailler in their specific case. The resulting signing public key and nonce are $Q = x_1x_2 \cdot P$ and $R = k_1k_2 \cdot P$, respectively, where P is a generator of points in the chosen group. Even if the idea is quite simple, the resulting protocol is not efficient for several reasons. To understand better where is the bottleneck, we explain how the basic idea works first. The basic idea is not the original protocol in [MR04], but it is a general framework followed by these authors for first, and by others afterwards, including our work [CCL$^+$19].
Let P_1 and P_2 be the two players involved. We split the computation in two phases, a key generation phase $\mathsf{IKeyGen}$ and a signing one ISign. In the $\mathsf{IKeyGen}$ phase, P_1 and P_2, with secrets $x_1, x_2 \leftarrow \mathbf{Z}/q\mathbf{Z}$, respectively, run a key exhange to compute $Q = x_1 \cdot Q_2 = x_2 \cdot Q_1$, where $Q_i = x_i \cdot P$. In the ISign phase:

- P_1 and P_2 run a key exchange to compute $R = k_1 \cdot R_2 = k_2 R_1$, where $R_i = k_i \cdot P, k_i \leftarrow \mathbf{Z}/q\mathbf{Z}$ as done in $\mathsf{IKeyGen}$
- P_1 uses Paillier encryption to compute $c_1 = \mathsf{Enc}(\mathsf{pk}_1, k_1^{-1} \cdot H(m))$ and $c_2 = \mathsf{Enc}(\mathsf{pk}_1, k_1^{-1} x_1 r)$. Then, P_1 sends c_1 and c_2 to P_2

- P_2 can then complete the signature, using the homomorphic properties of the scheme as follows

$$c \leftarrow c_1^{k_2^{-1}} c_2^{k_2^{-1}x_2} = \mathsf{Enc}(\mathsf{pk}_1, k^{-1} \cdot H(m))\mathsf{Enc}(\mathsf{pk}_1, k^{-1}xr) = \mathsf{Enc}(\mathsf{pk}_1, k^{-1}(H(m) + xr))$$

- P_2 concludes the protocol by sending back c to P_1
- P_1 can extract the signature $s \leftarrow k^{-1}(H(m) + xr)) \mod q$ from c using its secret key sk_1 of the encryption scheme.

However, proving that each party follow the protocol correctly turned out to be hard. The main bottleneck in [MR04] is the cost of the zero-knowledge proofs involved. We presented above the main points followed by the protocol of [MR04], but we do not present the more complex scheme in [MR04] and we recall only the main issues. In particular, in [MR04] protocol, player P_1 has to send additional encrypted values to player P_2, such for example an encryption of k_1^{-1} and of $k_1^{-1}x_1$, and both of the players have to prove the consistency of the sent elements and of the computations done during subsequent steps. Furthermore, for encryptions it is necessary to run range proofs, which are expensive. The reason why range proofs are required is linked to the different moduli involved, the composite N for Paillier encryption from one side and the prime q order of the group from the other side. As a consequence, checks can fail if a wrap around occurs. This last issue is inherited from more recent solutions that uses the same framework with Paillier encryption scheme for homomorphic operations. For this reason, we will return to it when we will talk about our technical contributions.

Lindell Improvement In 2017, Lindell ([Lin17]) managed to provide a much simpler and efficient protocol. What Lindell did is a crucial observation. He noticed that in the two party ECDSA signing protocol, dishonest parties are able to create only little trouble. In brief, in the IKeyGen phase P_1 sends an encryption $c_{\mathsf{key}} = \mathsf{Enc}(\mathsf{pk}_1, x_1)$ to P_2 using Paillier homomorphic encryption, where pk_1 is P_1's public encryption key and x_1 is P_1's share of signing secret key. In addition, they compute the public key Q running a Diffie-Hellman key exchange. After this phase, in the ISign phase:

- P_1 and P_2 compute R from a Diffie-Hellman key exchange
- P_2 homomorphically computes an encryption $c \leftarrow \mathsf{Enc}(k_2^{-1} \cdot H(m) + k_2^{-1}rx_2x_1)$ from c_{key} and finally it sends back c to P_1
- P_1 decrypts c obtaining α and it computes the signature as $s \leftarrow k_1^{-1}\alpha$.

Now, if P_1 is corrupted, since P_1 has to prove in zero-knowledge that it used the same x_1 in c_{key} and in its share $Q_1 = x_1 \cdot P$ of the signing public key, essentially what it can do is participating in the generation of $R = k_1k_2 \cdot P$. However, the computation of R consists in a Diffie-Hellman protocol, for which very efficient and robust protocols exists. On the other hand, if P_2 is corrupted, what she can do is taking part in the generation of R – and we have seen that she cannot cheat during this phase – and create a bad $\widetilde{c}$ to be sent to P_1. After receiving $\widetilde{c}$, P_1 can decrypt it and check if the resulting signature is valid and as a consequence, she detect that P_2 cheated. This last situation does not require zero-knowledge proofs, improving the efficiency of the protocol in [MR04]. As a result, Lindell's protocol is simple and efficient. We will give more details of Lindell's solution in Chapter 3, where we will compare it with our two-party ECDSA protocol based on Hash Proof Systems ([CCL$^+$19]). For completeness, a brief description of our solution is given in the next paragraph.

Castagnos-Catalano-Laguillaumie-Savasta-Tucker two-party ECDSA In 2019, Castagnos et al. ([CCL$^+$19][2]) proposed a new two party ECDSA protocol from a more general framework. Their scheme follows the idea of [Lin17]. Even if the structure of both the protocols is quite similar, the fundamental contribution in [CCL$^+$19] is the proposal of a general framework for building two-party ECDSA protocol based on Hash Proof Systems ([CS98]). Without going too deeply with details in this preliminary discussion, HPSs have the main advantage to set an encryption scheme which is homomorphic, in the sense of [HO09], and which works modulo any q. HPS are useful to avoid the costly range proofs in [Lin17], since the modulo of the encryption scheme can be set to be same prime number which is the order of the elliptic curve defining the signature algorithm. They also presents a concrete instantiation of the protocol using Class Groups of an Imaginary Quadratic Field, whose properties are perfectly in line with the general construction from HPS. This work is discussed in details in the dedicated Chapter 3.

The general case - Threshold ECDSA

In a more general case, there are n parties that join the signature protocol. The protocol is also parametrized by a threshold $t < n$ which indicates what is the maximum number of parties that together cannot compute a signature. The range of possible thresholds for a scheme is fundamental to define which properties the scheme can achieve. In contexts where an adversary corrupts less than half of the parties, it could be possible to reconstruct the signature from honest values and ending the protocol with a valid output ([GJKR96a]). In general this is not true in contexts where an adversary can corrupt a majority of the involved parties and failures of the protocol in the task of computing a valid output are more frequent. To achieve security against any threshold $t < n$ it is required to consider security with aborts. Informally, if at a certain point a party misbehaves, the protocol will abort and no informations about honest parties' values are revealed, or computed by seeing honest parties behaviour. Anyway, after an abort occurs it is not possible to reconstruct the final output. Under this kind of security, recently efficient full threshold ECDSA solution were proposed. Starting from the construction of Gennaro and Goldfeder ([GGN16]), we give a description of the ideas that permitted to construct efficient full threshold ECDSA schemes. Some points are similar to the two-party constructions, i.e. using an homomorphic encryption scheme as a building block, but using multiplicative share is not possible anymore. We generalize the result of [GG18] in our work [CCL$^+$20] in the context of Class Groups. Chapter 4 is dedicated to our [CCL$^+$20].

Gennaro et al. full threshold ECDSA We give a brief description of two relevant results from Gennaro et al.

[2]This is one of our works

[GGN16] In 2016, Gennaro et al. (cf. [GGN16]) took the challenge to construct an efficient and optimal threshold DSA scheme. Gennaro et al. proposed a threshold DSA scheme without honest majority as an extension to a context of more than 2 parties ([MR04]). However, they did not extend directly the construction of [MR04], but they proposed a new scheme which uses the threshold variant of Paillier encryption scheme as a building block. However, they also thought about an extension of the [MR04] scheme for the general $(t, n)-$ threshold case, noting how much inefficient was the result. Indeed, the extension works with $N = O(q^{3t-1})$, a storage of $O(n^t)$ for each party and a non constant number of rounds, i.e. $O(t)$ rounds. On the other hand, the proposal in [GGN16] has 6 rounds, i.e. a constant number, and $N > q^8$ to avoid wrap-arounds. Finally, their scheme presents a series of zero-knowledge to prove the correctness of the computations and it ends with a final decryption step in which the signature is computed as the threshold decryption of a value generated in previous rounds using the homomorphic properties of the scheme.

[GG18] In 2018, Gennaro and Goldfeder (cf. [GG18]) proposed a new optimal threshold ECDSA protocol in the context of a malicious adversary and dishonest majority. The scheme improved the previous result of [GGN16] (and a successive improvement in [BGG17]). The new protocol requires less communication, it is simpler and solves the problem of a very costly distributed key generation part which is present in [GGN16]. Furthermore, authors improved on the number of the costly zero-knowledge proofs. The new protocol uses a Shamir secret sharing for generating the public key and the secret key (implicitly) for the signature scheme. Furthermore, a threshold variant of Paillier encryption is not required anymore and relevant products for the computation of the final signature are computed from a distributed subprotocol that converts multiplicative shares to additive shares. This clearly permits to compute the product $k \cdot x$ in the signature, where $x := \mathsf{sk}$ and $R = g^{k^{-1}}$ as usual in ECDSA (see Figure 1 for notation), from additive shares. Finally, some range proofs are necessary to avoid that a failure in generating the signature can reveal sensitive informations.

Castagnos-Catalano-Laguillaumie-Savasta-Tucker full threshold ECDSA Following the framework of [GG18], Castagnos et al. ([CCL⁺20]) proposed a threshold ECDSA scheme based on Class Groups of Imaginary Quadratic Fields. They noted that changing the Paillier encryption scheme used in [GG18] with the Castagnos-Laguillaumie linearly homomorphic encryption scheme (CL) brings to advantages in terms of communication. Even if this seems a little change, we see how much stronger this change is and how efficiency issues from [GG18] disappears. Unfortunately, new issues relative to proving statement for CL appeared and a relevant contribution of [CCL$^+$20] relies on how we managed to solve them. Looking at the advantages, first of all working with two different modulus – N for Paillier encryption and q for the Elliptic curve – requires the usage of running expensive range proofs. The first advantage in using CL is that the messages are chosen in $\mathbf{Z}/q\mathbf{Z}$ where q is freely chosen (with very weak conditions). Therefore, using the same prime of the elliptic curve avoids to run range proofs. Second, the advantage is in bandwidth consumption, since ciphertext in CL are shorter than Paillier ciphertext for the same level of security. However, for the structure of CL it is required to prove that a ciphertext comes from a valid message since the scheme is not surjective with respect to the couple (message, randomness). A second contribution of the paper is proposing a new efficient zero knowledge to do this task. The resulting scheme is better in terms of communication and faster for high levels of security. A more detailed description of our work, i.e. [CCL+20], is given in the dedicated Chapter 4.

Further Improvements on Threshold ECDSA

Previous solutions lack some crucial properties which are useful for real life situations. To complete our description of recent results on threshold ECDSA, we recall here a brief description of the results in [CGG+20] and of our work [CCL+21].

Canetti et al. In [CGG+20], Canetti et al. proposed two threshold ECDSA schemes which guarantee proactivity, identifiable aborts and adaptive security against $t = n - 1$ corruptions in the Universal Composability model. Informally, a scheme has identifiable aborts if it is possible to detect a misbehaving player which caused an abort and to exclude her; a scheme is proactive if its key can be updated instead of being recomputed after a certain time period to avoid corruptions of parties one by one during a long time. Both the schemes are divided in four phases: Key generation, Key refresh, Presigning and Signing. The scheme is thought to have an offline presigning phase and an online signing phase. During the Presigning phase, parties compute the necessary values which permits a fast and simple Signing procedure (which consists only in sending a share of the signature which belongs to $\mathbf{Z}/q\mathbf{Z}$). Their two schemes are principally different in the identification procedure when an abort occurs caused by a misbehaving player. The first identification procedure requires a cost of $O(n^2)$ and three rounds of communication Presigning phase, while the second identification procedure requires a cost of $O(n)$ but six rounds of communication in Presigning. Indeed, the best choice depends on the condition behind the usage of the scheme, i.e. if in the application is more important the cost or the latency.

Castagnos-Catalano-Laguillaumie-<u>Savasta</u>-Tucker enhanced threshold ECDSA In [CCL+21] we improved our solution from [CCL+20], to take in account the same properties of accountability, proactivity and the split in offline/online phases. We present two schemes, one which is secure against a static adversary that corrupts up to $n - 1$ player and another proven secure against an adaptive adversary which corrupts exactly $n - 1$ player. Actually, the two schemes can be seen as only one since the unique difference is in the key generation protocol, which is optimized for $t = n - 1$. The details about this specific choice are given in Chapter 5. We also present new zero-knowledge for proving that a CL ciphertext decrypts to some message or to $\perp$ useful for our identification protocol specific for using CL. The main difference with [CGG+20] consists in the model considered. Our schemes are provably secure, but they are not proven secure in the UC model. The main advantage of this choice is efficiency. Our solutions have comparable communication and computational costs with [CCL+20]. Details of [CCL+21] are given in the dedicated Chapter 5.

Goal and resume of the manuscript

Short description of our works This manuscript is based on our works in [CCL$^+$19], [CCL$^+$20] and [CCL$^+$21]. The three works can be seen as an unique series of works on the same topic that step by step generalize the context we work with. As already introduced, all our works treat the problem of building a distributed ECDSA protocol which is secure and efficient, and we do that instantiating our solutions with Class Groups of Imaginary Quadratic Fields. The principal difference with other constructions relies on the encryption scheme used as a building block and the advantages that derives from it. One of the notable advantages in using CL is the reduction of the communication costs with respect to other schemes, thanks to the shorter representation of ciphertexts with regard to the security parameter. The strongest of the three works is [CCL$^+$19]: in this work we proposed a general framework based on Hash Proof Systems for the two-party ECDSA and we proved its security in the simulation real/ideal model. Then, we presented an efficient instantiation with Class Groups, which are adaptable to the properties we need. In [CCL$^+$20], we propose a solution based on CL for threshold ECDSA which is secure against a malicious adversary that can corrupt up to $n-1$ parties joining the protocol, where n is the total number of them. We presented a game based security proof of it. The proposed protocol is also bandwidth efficient, thanks also to the smaller communication cost from ciphertexts representation in CL. Finally, in [CCL$^+$21] we extended the protocol of [CCL$^+$20] to take in account the possibility to identify misbehaving players when an abort occurs and to make it proactive. As opposed to Paillier based solutions, we do not need to prove that a key is well-formed and as a consequence the refreshing of a key can be done without an expensive proof of validity.

The main body of the thesis is divided in five Chapters, where the first two chapter are dedicated to parts of the tools required for the next ones. The remaining tools are introduced and explained when necessary, to avoid confusion in the reading. Next Chapters deal with the works which include the author of this thesis as a coauthor. These last three Chapter are sequential both in time and in the topics covered.

Structure of the manuscript

- We began with this Intro Chapter.
- In Chapter 1, we give a list of some cryptographic and mathematical tools required for next chapters.
- Chapter 2 is written as a brief manual on Class Groups of Imaginary Quadratic Fields, which is of independent interest since it is thought to give a good, but not exaggeratedly detailed, background on these mathematical object and to introduce cryptographers to this interesting topic. Furthermore, it is useful to understand the Castagnos-Laguillaumie encryption scheme. The Castagnos-Laguillaumie scheme (CL) is a building block of our threshold ECDSA scheme, since it is the underlying linearly homomorphic encryption scheme used in our constructions, and then the author thought it deserves a significant part of Chapter 2. This chapter does not introduce new results, but it is an original survey of the author of this manuscript.

- CHAPTER 3 is completely dedicated to the work of [CCL+19] that we introduced above. It deals with the construction of a two-party ECDSA protocol from Hash Proofs Systems.

- CHAPTER 4 is built on the work of [CCL+20]. The chapter deals with a more general $(t, n)-$threshold ECDSA construction instantiated using Class Groups of Imaginary Quadratic Fields.

- CHAPTER 5 is built on the work of [CCL+21]. Differently from the other two papers in previous items, at the time of writing this work ([CCL+21]) has not yet published. This work considers a next step of improvements, in terms of additional features, of the full $(t, n)-$threshold ECDSA scheme from [CCL+20], i.e. it takes in account identifiable aborts, proactivity and a specific case of $(n-1, n)-$adaptive security of the scheme. Furthermore, a novelty, even if not relevant as in the previous two papers (cf. [CCL+19], [CCL+20]), is the introduction of a proof for proving correct decryption with Castagnos-Laguillaumie scheme which permits to identify misbehaving players in any case.

Notes for the reader In this manuscript, each of our works has a dedicated chapter and then it is usual that we say Our/Chapter 3/[CCL+19], Our/Chapter 4/[CCL+20] and Our/Chapter 5/[CCL+21] interchangeably. Also, class groups and the CL scheme are used in all the three works and to avoid repetitions, we often refer to Chapter 2, instead of recalling it. Finally, part of the notation used in next chapters is given at the beginning of Chapter 1.

Final note All the three works detailed in this thesis ([CCL+19], [CCL+20] and [CCL+21]) are joint works with my supervisor Dario Catalano (University of Catania), Guilhem Castagnos (University of Bordeaux), Fabien Laguillaumie (University of Montpellier) and Ida Tucker (IMDEA Software Institute Madrid).

Contents

Introduction **i**

1 Preliminaries **5**

1.1 Introduction to Public Key Cryptography 6

1.1.1 Security model 6

1.2 Public Key Encryption 8

1.2.1 Homomorphic Encryption 10

1.3 Digital Signatures 11

1.3.1 The elliptic curve digital signature algorithm 13

1.4 Equivocable commitment schemes 15

1.5 Zero-knowledge proofs 15

1.5.1 Sigma protocols 17

1.5.2 Zero Knowledge Proofs concerning DL 19

1.6 Verifiable secret sharing 20

2 Background on Class Group of Imaginary Quadratic Fields **23**

2.1 Towards the definition of the Ideal Class Groups: the Algebraic Part . . 24

2.1.1 Binary Quadratic Forms 25

2.1.1.1 The reduction algorithm 27

2.1.1.2 The composition of forms 31

2.1.1.3 The Form Class Group 32

2.1.2 Imaginary Quadratic Fields, Orders and Fractional $\mathcal{O}-$Ideals . . . 34

2.1.2.1 Maximal and non-maximal orders 35

2.1.2.2 Representation and inclusions of orders 36

2.1.2.3 Fractional $\mathcal{O}-$ideals 37

2.1.2.4 Product of $\mathcal{O}-$ideals 39

2.1.3 The Ideal Class Group 41

2.1.3.1 Relations between the ideal class group and the form class group 43

2.2 Class groups in Cryptography: the Castagnos-Laguillaumie encryption scheme 46

2.2.1 Computing square roots in $Cl(\mathcal{O}_\Delta)$ 47

2.2.2 The odd part of the Class Group 49

2.2.2.1 A choice for a $2-$Sylow of order 2 50

2.2.2.2 The Cohen-Lenstra Heuristic 50

2.2.3 Switching between a maximal order and a non maximal suborder 52

2.2.4 A subgroup with an easy discrete logarithm problem 55
2.2.4.1 Additional modification to CL 57
2.2.4.2 Construction of the direct product G_q and F 59

3 Two-Party ECDSA from Hash Proofs Systems 61
3.1 Ideal functionalities 64
3.2 Background on HPS, a HPS-based PKE scheme and ECDSA-friendly HPS 66
3.2.1 Defining HPS and PHF 67
3.2.2 Hard assumptions for PHF 68
3.2.3 Homomorphic Properties 69
3.2.4 Resulting Encryption Scheme 70
3.2.5 ECDSA friendly Projective Hash Families 72
3.2.6 Zero-Knowledge Proofs 76
3.3 Two-Party ECDSA Signing Protocol with Simulation-Based Security . . 77
3.3.1 The Two-Party ECDSA protocol 78
3.3.2 Simulation-based security of the Two-Party ECDSA scheme . . . 80
3.4 Instantiation from Class Group of Imaginary Quadratic Fields 90
3.4.1 A Smooth Homomorphic Hash Proof System from HSM 91
3.4.2 A zero-knowledge proof for $\mathsf{R}_{\mathsf{CL-DL}}$ 94
3.4.3 Two-Party Distributed ECDSA Protocol from HSM 97
3.5 Efficiency comparison 97
3.5.1 Lindell's Protocol with Paillier's Encryption Scheme 99
3.5.2 Our Protocol with $\mathsf{HSM} - \mathsf{CL}$ Encryption Scheme 99
3.5.3 Comparison 101
3.6 Instantiation of our Generic Construction Using DCR 102

4 Bandwidth-efficient Threshold ECDSA from Class Groups 103
4.1 Preliminaries 105
4.1.1 Building blocks from Class Groups 105
4.1.2 Algorithmic assumptions 107
4.2 Threshold EC-DSA protocol 109
4.2.1 ZKAoK ensuring a CL ciphertext is well formed 110
4.2.2 Interactive set up for the CL encryption scheme 113
4.2.3 Resulting threshold ECDSA protocol 117
4.2.3.1 Key generation. 117
4.2.3.2 Signing. 118
4.3 Security 120
4.4 Further improvements 131
4.4.1 An improved ZKPoK which kills low order elements. 131
4.4.2 Assuming a standardised group 132
4.5 Efficiency comparisons 135

5 Improvements on Threshold ECDSA 139
5.1 Improved Threshold ECDSA protocol 143
5.2 Identifying Aborts 146
5.2.1 Arguing knowledge of a decrypted message 148

5.2.2 Aborting and detection of misbehaving parties 149
5.3 Security 154
5.3.1 Security of the Full Threshold Protocol with Identifiable Aborts against Static Adversaries 154
5.3.1.1 Indistinguishability of Real and Simulated Environments 160
5.3.2 Security Against Adaptive Adversaries 166
5.3.2.1 Indistinguishability of real and simulated executions against adaptive adversaries 174
5.3.2.2 Concluding the proof 176
5.4 Efficiency comparisons 176

Conclusion **177**

References **181**

Acknowledgement **191**

Chapter 1

Preliminaries

This chapter has the aim to recall some of the cryptographic tools and definitions required in technical chapters which follow. This chapter does not include all the necessary background, since more specific tools are directly recalled in chapters that require them. Then, the following sections only consist of a collection of known results in Cryptography (e.g. [KL14] for an introductory background, or [Lin16] for a tutorial on simulation technique, or [HL10] for secure multiparty computation notions).

Notations. We present below the notation principally used in Chapters 1, 3, 4 and 5. Chapter 2 has its own specific notation.

- **Distributions**: given a distribution $\mathcal{D}$, we write $d \hookleftarrow \mathcal{D}$ to refer to d being sampled from $\mathcal{D}$ and $b \overset{\$}{\leftarrow} B$ if b is sampled uniformly from a set B.

- **Groups**: we use the $\mathbb{G}$ notation for the elliptic curve group and G for the class group. We denote with P the generator of $\mathbb{G}$, and each point of the curve is denoted with the notation $Q = x \cdot P$, for some $x \in \mathbf{Z}/q\mathbf{Z}$, where q is the order of $\mathbb{G}$.

- **Interactive protocols**: in an interactive protocol IP, between parties $P_1, \ldots, P_n$ for some integer $n > 1$, we denote by $\mathsf{IP}\langle x_1; \ldots; x_n\rangle \to \langle y_1; \ldots; y_n\rangle$ the joint execution of parties $\{P_i\}_{i\in[n]}$ in the protocol, with respective inputs x_i, and where P_i's private output at the end of the execution is y_i. If all parties receive the same output y we write $\mathsf{IP}\langle x_1; \ldots; x_n\rangle \to \langle y\rangle$. For the specific two-party case, we use the same style, i.e. $\mathsf{IP}\langle x_1; x_2\rangle \to \langle y_1; y_2\rangle$, for the joint execution of parties $\{P_i\}_{i\in\{1,2\}}$, with respective inputs x_i, and where P_i's private output at the end of the execution is y_i. We use $\mathsf{sans\text{-}serif}$ to indicate protocols (e.g. KeyGen).

- **Arrows**: in an interactive protocol, we use the single arrow $\xrightarrow{\alpha}$ from a party P_i to a party P_j to indicate P_i sending α to P_j. We use doublearrow $\overset{\alpha}{\Rightarrow}$ from a party P_i to indicate that P_i sending α in boradcast. In some specific cases, we use the left-right simple arrow $\overset{\pi}{\leftrightarrow}$ for interactive protocols π between parties.

- **Algorithms**: a (P)PT algo stands for an algorithm running in (probabilistic) polynomial time with regard to the length of its inputs. We use calligrafic style ($\mathcal{S}$,$\mathcal{A}$, $\mathcal{F}$) to denote the adversary, a simulator and ideal functionalities. .

1.1 Introduction to Public Key Cryptography

The seminal work of Diffie and Hellman ([DH76]) put the basis of a new type of cryptography, i.e. cryptography based on a public key. This is know as *Public Key Cryptography*, or *Asymmetric Cryptography.* The reason why it is called asymmetric, in constrast with the *symmetric* case, is related to the asymmetric roles of the sender and the receiver. It is well known that the two principal problems in Cryptography are *privacy* – i.e. an encrypted message must not reveal information on the plaintext – and *authentication* – which means that an adversary must not be able to impersonate someone else and being able to sign message for her. [DH76] introduces a duality of keys, a public key pk to compute an encryption (or check a signature) publicly and a secret key sk to compute private operations as decrypting a ciphertext (or signing a message). The keys are strongly linked under some hard cryptographic assumption. This means that obtaining the secret key from the knowledge of the public key needs to be unfeasible. In an asymmetric context, it sufficient that each party shares its public key to everyone without too much worries thanks to the hardness in computing its secret key. Even if asymmetric cryptography presents a lot of advantages with regards to symmetric one, it is in general less efficient in terms of computation, and when possible a symmetric option is preferred.
In general, a public key scheme is composed by three elements: a protocol, a security definition and some assumptions. This is a very easy way to define it, but substantially this is what we need to construct a secure scheme. First, we need to define what *secure* means and if necessary, which assumptions are assumed hard to guarantee the validity of the security definition. Certainly, a security proof relies also on the model or the type of adversary considered.

1.1.1 Security model

In general, a security definition is composed by two elements, a threat model (the specification of an adversary, its power, its corruption strategy) and a security goal. To be secure with respect to the definition of security considered, a protocol has to satisfy that security definition. Without going deeper, we can consider substantially two types of security: information theoretic security and computational security. Informally, the former includes definitions of security against unbounded adversaries which are not able to break the scheme with a non negligible probability, where the meaning of breaking the scheme depends on security task that we want to be guarantee. Generally, information theoretic security is a strong requirement for real applications. The latter, i.e. the computational security, includes the definitions of security against a computational bounded adversary. In public key cryptography it is usual to consider computational definitions of security.

The adversary An adversary is an entity which corrupts parties which join a protocol. There is not a specific description of an adversary, but the strength of a protocol is also based on the type of the adversary considered. In general, we cannot deduce the strategy adopted by the adversary, but we can describe different types of adversary by its way to misbehave and by the corruption strategy it adopts. We present only the types of adversary which are often considered in works. Referring to the way it can misbehave we distinguish:

Honest but-curious An adversary is said *honest but curious*, or *semi-honest* or *passive*, if it follows the protocol as an honest party does. The aim of this type of adversary is not to harm the right execution of the protocol, but it is to obtain private infos about honest players, i.e. data that are not expected to be known.

Malicious An adversary is called *malicious*, or *active*, if it can decide to not follow the protocol. This means that the adversary is able to send wrong data, or it can decide to do not answer to a request (Denial of Service (DoS) attack), or to choose values that can compromise the distribution of relevant values in the protocol revealing private data. In brief, a malicious adversary can decide to deviate from the protocol everytime it desires.

Another characterization of how an adversary can cheat is linked to the type of corruption it does, i.e. when it can corrupt and how. From the works we will present in next chapters, we recall type of corruption of our interest, that coincide with the most used in literature. In this definition we do not consider the corruption power, i.e. how many corruptions the adversary can do, for a clear description.

Static A *static* adversary corrupts a fixed number of parties *before* the protocol starts. When coping with static adversaries, it is known a the beginning who are the corrupted parties. New parties cannot be corrupted after the protocol starts.

Adaptive An *adaptive* adversary is able to corrupt a certain number t of parties during the protocol running. There is no limitation about when a party is corrupted. This type of adversary can corrupt parties in any moment, obtaining their internal state, in any moment and with the condition that the total number of corrupted parties does not exceed t. Furthermore, an adaptive adversary cannot decide to "decorrupt" a party to free a slot of corruption, i.e. once a party is corrupted it will be so until the end.

Mobile A *mobile* adversary, or *proactive* adversary ([OY91], [CH94]), is able to corrupt up to t parties during a time period called *epoch*. After an epoch, a mobile adversary is able to corrupt another set of t parties. This behaviour is that of a virus which spreads over machine during the time. The condition about mobile adversaries is that it corrupts at most t parties in an epoch.

Game-based and Simulation-based definitions

We can distinguish between two types of approach to prove the security of a scheme. The approach depends on the model considered and the type of security we want to guarantee.

Game-based security. The first approach is to prove security from game-based definitions of security. In a game-based context, the security relies on an *experiment* (or game). The experiment consists in a challenge chosen by an entity $\mathcal{C}$, called a *challenger*, and an adversary $\mathcal{A}$ whose aim is to break the problem behind the challenge. A scheme is said secure in the sense of a game-based definition if the advantage of a PPT adversary in solving the challenge is negligible. Game-based definitions for digital signatures in general consist in being able to create a forgery. i.e. a valid signature for a message whose signature is unknown. Our works [CCL$^+$20] and [CCL$^+$21] treated in Chapters 4 and 5, respectively, are game-based secure.

Simulation-based security and ideal functionalities. The second approach is to prove security from simulation-based definitions. It is usual to prove the security of a scheme in the simulation *real/ideal* paradigm. The real/ideal paradigm consists of two world, called *real* and *ideal*. In the basic definition of the ideal world, parties can communicate only with a functionality sending an input and receiving an output, while in the real world they exchange messages among them. The ideal world is assumed secure by definition, since the functionalities are not corruptible. We say that a protocol is secure in this paradigm if the real world is indistinguishable by the ideal world, i.e. from a world where there are no security issues. The technique behind this type of proof is called *simulation*. Informally, let f be a functionality which is secure by definition and π a protocol that does the same task of f. To prove that π *compute* f we construct a simulator $\mathcal{S}$ that receives the output from the functionality f and simulates the interaction of an adversary $\mathcal{A}$ and parties in a real world. The idea to prove security is constructing $\mathcal{S}$ that simulates the view of $\mathcal{A}$ such that this view is computationally indistinguishable from its view in a real world where it takes part in a protocol communicating with parties. Additionally, $\mathcal{S}$ has to generate a view which is consistent with the output received from the functionality.

The hybrid model In the hybrid model, parties can communicate as in a real protocol π, but they also have access to ideal calls to a set of functionalities $f_1, f_2, \ldots, f_n$. This means that they are instructed to send inputs to a set of functionalities and receiving back the outputs. In general, when it is asked for an ideal call to a functionality f_i, π continues after the output from f_i is received.

For an more extended and detailed explanation of how to write simulators and prove security via the simulation paradigm we refer the reader to the tutorial of [Lin16] and to [HL10].

1.2 Public Key Encryption

Public Key Encryption is the part of Public Key Cryptography which comprises the theory of encryption in the asymmetric setting. We introduce below the definition of a public key encryption scheme and some standard definitions about the security of the scheme.

Definition 1.2.1 (Public key Encryption Scheme [DH76]). A *public key encryption scheme* ia s triple of PPT algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- The *key generation algoritm* KeyGen takes as input the security parameter 1^λ and outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$, called *public key* and *secret key*, respectively.

- The *encryption algorithm* Enc takes as input a public key pk and a message m from some message space $\mathcal{M}$. It outputs a ciphertext $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$.

- The deterministic *decryption algorithm* Dec takes as input a private key sk and a ciphertext from some ciphertext space $\mathcal{C}$, and outputs a message $m := \mathsf{Dec}(\mathsf{sk}, c)$ or a special symbol $\perp$ denoting failure.

It is required that, except with negligible probability over $(\mathsf{pk}, \mathsf{sk})$ output by $\mathsf{KeyGen}(1^\lambda)$, we have $\mathsf{Dec}(\mathsf{sk}, (\mathsf{Enc}(\mathsf{pk}, m)) = m$ for any message $m \in \mathcal{M}$.

Semantic Security Informally, an encryption scheme is *semantically secure* if what can be computed from a ciphertext is almost the same of what can be computed from a priori knowledge, without the knowledge of the ciphertext. Let $h : \{0,1\}^* \to \{0,1\}^*$ denote an arbitrary auxiliary function of the plaintext, i.e. h describes partial information on the plaintext that may be leaked by adversary. Let $f : \{0,1\}^* \to \{0,1\}^*$ a function of the message that the adversary wants to learn from the knowledge of the ciphertext and h. The original definition of semantic security was introduced in [GM82], while we recall here that of the treatment in [Gol04] (in the form that it is found in [HL10]).

Definition 1.2.2 (Semantic security in public-key encryption)*.* Let $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public encryption scheme, and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$ a couple of public/private keys output by the key generation algorithm. Π is *semantically secure* (in the public-key model) if for every PPT algorithm $\mathcal{A}$, there exists a PPT algorithm $\mathcal{A}'$ such that for every ensemble $\{X_\lambda\}_{\lambda \in \mathbf{N}}$ with $|X_\lambda| \leq \text{poly}(\lambda)$, every pair of polynomially-bounded functions $f, h : \{0,1\}^* \to \{0,1\}^*$, every polynomial $p(\cdot) > 0$ and all sufficiently large λ:

$$\begin{aligned}\Pr_{\mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)}[\mathcal{A}(1^\lambda, \mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, X_\lambda), 1^{|X_\lambda|}, h(1^\lambda, X_\lambda)) = f(1^\lambda, X_\lambda)]\\ < \Pr[\mathcal{A}'(1^\lambda, 1^{|X_\lambda|}, h(1^\lambda, X_\lambda)) = f(1^\lambda, X_\lambda)] + \frac{1}{p(n)}\end{aligned}$$

Security against Chosen-Plaintext Attacks The notion of indistinguishability against adaptive chosen-plaintex attack ($\mathsf{ind\text{-}cpa}$) resumes the inability of a PPT adversary $\mathcal{A}$ to distinguish between the encryptions of two different messages chosen by the adversary itself. Since $\mathsf{ind\text{-}cpa}$ security against a single encryption or multiple encryptions are proved to be equivalent, we consider the experiment in the case of multiple challenges. Consider a public key encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and an adversary $\mathcal{A}$, the LR experiment is depicted in Figure 1.1.

Definition 1.2.3. Consider the experiment in Figure 1.1. A public key encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is said be $\mathsf{ind\text{-}cpa}$ secure with multiple encryptions if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that

$$\Pr[\mathsf{Expt}_{\mathcal{A},\Pi}^{\mathsf{LR-cpa}}(\lambda) = 1] \leq \frac{1}{2} + \mu(\lambda)$$

Remark 1. Goldwasser and Micali also proved the equivalence between semantic security and $\mathsf{ind\text{-}cpa}$ security.

The LR experiment $\mathsf{Expt}_{A,\Pi}^{\mathsf{LR-cpa}}(\lambda)$

1. Run $\mathsf{KeyGen}(1^\lambda)$ which outputs $(\mathsf{pk}, \mathsf{sk})$
2. Choose a uniform bit $b \in \{0, 1\}$
3. $\mathcal{A}$ is given the public key pk and the access to an oracle $\mathsf{LR}_{\mathsf{pk},b}$ which takes on input a couple of messages m_0 and m_1 and returns the $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.
4. $\mathcal{A}^{\mathsf{LR}_{\mathsf{pk},b}(\cdot,\cdot)}(\mathsf{pk}) \to b'$
5. If $b = b'$ the output of the experiment is 1, otherwise it is 0. $\mathcal{A}$ succeeds if $\mathsf{Expt}_{A,\Pi}^{\mathsf{LR-cpa}}(\lambda) = 1$

Figure 1.1: The LR experiment $\mathsf{Expt}_{A,\Pi}^{\mathsf{LR-cpa}}(\lambda)$

1.2.1 Homomorphic Encryption

In some context, as for example distributed ones, it can be relevant to be able to operate directly on the ciphertext without the necessity to know a decryption key or the underlying plaintext. Some schemes, as for example Paillier encryption scheme, have homomorphic properties that permits to modify the encryption c of a message m to an encryption c' of a message m', where m' is a linear transformation of the original m. A formal definition is the following:

Definition 1.2.4 (Homomorphic encryption scheme [KL14],[HL10]). A public key encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is *homomorphic* if for all security parameter λ and all $(\mathsf{pk}, \mathsf{sk})$ output by $\mathsf{KeyGen}(1^\lambda)$, it is possible to define two groups $(M, +)$ and $(C, \oplus)$ (depending only on pk) such that:

- M is the message space, and all ciphertexts output by $\mathsf{Enc}(\mathsf{pk}, \cdot)$ are elements of C.
- For any couple of messages $m_1, m_2 \in M$, any valid encryptions $c_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1)$ and $c_2 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_2)$, it holds that

  $$\mathsf{Dec}(\mathsf{sk}, c_1 \oplus c_2) = m_1 + m_2$$

  .

Furthermore, the distribution on ciphertexts obtained by the product of the encryptions of m_1 and m_2 is identical to the distribution of the ciphertexts obtained by encrypting $m_1 + m_2$.

Paillier Encryption Scheme Paillier ([Pai99]) presented one of the most used encryption schemes in applications, called the *Paillier Encryption Scheme* from the name of the author. We recall here Paillier Encryption Scheme principally for two reasons: first, it is a good example of an homomorphic encryption scheme and second, it is at the basis of the threshold ECDSA protocols we will compare our solutions with in next chapters. Indeed, a relevant difference between other efficient solutions and ours is the different homomorphic encryption scheme at the basis of constructions and analyzing pros and cons in using one scheme or the other one is of particular interest. We postpone the description of the encryption scheme we use in our works ([CCL$^+$19], [CCL$^+$20], [CCL$^+$21]) to Chapter 2, since it requires understanding more complex objects. In this paragraph we look at Paillier encryption scheme, which is proved ind-cpa secure under the hardness of the Decisional Composite Residuosity problem.

Definition 1.2.5 (DCR assumption). Let be $\mathsf{GenMod}(1^\lambda)$ be a poly-time algorithm which on input the security parameter λ returns a tuple (N, p, q) such that $N = pq$ and p, q are prime of lenght λ, except for a negligible function of λ. The *Decisional Residuosity Problem* (DCR) is hard relative to GenMod if for all PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Big|\Pr[\mathcal{A}(N, r^N \mod N^2) = 1] - \Pr[\mathcal{A}(N, r) = 1]\Big| \leq \mathsf{negl}(\lambda),$$

where the probabilities are taken from the experiment in which $\mathsf{GenMod}(1^\lambda)$ returns (N, p, q) and after that r is chosen in $\mathbf{Z}^*_{N^2}$.

Paillier encryption scheme consists of a tuple of algorithm $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ where:

KeyGen : the probabilistic key generation algorithm KeyGen takes in input the security parameter λ and generate two large primes p, q of equal length. Finally, set $N = pq$.

Enc Consider the Carmichael function of N, defined as $\lambda(N) = \mathrm{lcm}(p-1, q-1)$, and denote Γ a generator of a subgroup of order N in $Z^*_{N^2}$ [1]. To encrypt a message, Enc takes in input a message $m \in \mathbf{Z}_N$, it selects a randomness $r \in \mathbf{Z}^*_N$ and it returns a ciphertext $c = \Gamma^m r^N \mod N^2$.

Dec : Consider the function $L : \{u \in Z_{N^2} : u = 1 \mod N\} \to \mathbf{Z}$ that sends u to $L(u) = \frac{(u-1)}{N}$. To decrypt a ciphertext $c \in Z_{N^2}$, the decryption algorithm Dec takes in input c and it returns a message $m = \frac{L(c^\lambda(N))}{L(\Gamma^\lambda(N))} \mod N$.

Paillier scheme as said above, satisfies some homomorphic properties. Let denote $\oplus, \odot, \cdot$ the homomorphic sum in $\mathbf{Z}_{N^2}$, homomorphic product by constant in $\mathbf{Z}_{N^2}$ and the standard product in $\mathbf{Z}_N$, respectively. If we take two ciphertexts $c_1, c_2 \in \mathbf{Z}_{N^2}$, we have that $c_1 \oplus c_2 = c_1 c_2 \mod N^2$ is a valid encryption of $\mathsf{Dec}(\mathsf{sk}, c_1) \cdot \mathsf{Dec}(\mathsf{sk}, c_2)$. Also, if $c \in \mathbf{Z}_{N^2}$ and $k \in \mathbf{Z}_N$ is a constant, then $k \odot c = c^k \mod N^2$ is a valid encryption of $k \cdot \mathsf{Dec}(\mathsf{sk}, c)$. Finally, about the security of Paillier encryption scheme it is proved that:

Theorem 1.2.1. The Paillier encryption scheme is ind-cpa secure under the assumption that the DCR problem is hard relative to GenMod.

1.3 Digital Signatures

As discussed in the introduction, in Chapters 3, 4 and 5 we will see how to build distributed versions of ECDSA. In this section we give a description of the basic version of ECDSA and its $(t, n)-$threshold variant with all the necessary security definitions.

[1]A choice can be $\Gamma = 1 + N$

Definition 1.3.1. A *digital signature scheme* is a tuple $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ of probabilistic polynomial-time algorithms defined as follows:

$\mathsf{KeyGen}(1^\lambda)$ The key generation algorithm KeyGen takes in input a security parameter 1^λ and returns a couple $(\mathsf{vk}, \mathsf{sk})$ which consists of a (public) verification key vk and a (private) signing key sk.

$\mathsf{Sign}(\mathsf{sk}, m)$ The signing algorithm takes in input a signing key sk and a message m from some message space and returns a signature σ.

$\mathsf{Verif}(\mathsf{pk}, m, \sigma)$ The verification algorithm takes in input a verification key vk, a message m and a signature σ. It returns a bit $b = 1$ or $b = 0$ if the signature is valid for m or not, respectively

A standard security notion required for digital signature schemes is that of existential unforgeability under chosen message attacks ($\mathsf{eu\text{-}cma}$) [GMR88]. This notion is defined from an experiment between a challenger and an adversary.

The $\mathsf{eu\text{-}cma}$ experiment $\mathsf{Exp}^{\mathsf{eu\text{-}cma}}_{\mathcal{A},\mathsf{S}}(\lambda)$

Let λ be a security parameter and $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ be a digital signature scheme. Consider the steps:

1. Run $\mathsf{KeyGen}(1^\lambda)$ to obtatin a couple $(\mathsf{vk}, \mathsf{sk})$ of keys as defined in Definition 1.3.1
2. It is given vk and access to a signature oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$ to the adversary $\mathcal{A}$. $\mathcal{A}$ interacts with the signing oracle and returns a couple (m, σ).
3. Let $\mathcal{M}$ denote the set of all queries $\mathcal{A}$ asked to the signing oracle. $\mathcal{A}$ succeeds if and only if
   - $\mathsf{Verif}(\mathsf{vk}, m, \sigma) = 1$ and
   - m was never asked to the signing oracle

   In that case the result of the experiment is 1.

Figure 1.2: The $\mathsf{eu\text{-}cma}$ experiment $\mathsf{Exp}^{\mathsf{eu\text{-}cma}}_{\mathcal{A},\mathsf{S}}(\lambda)$.

Definition 1.3.2 (Existential unforgeability [GMR88]). Let $\mathsf{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ be a digital signature scheme, λ a security parameter and consider a PPT algorithm $\mathcal{A}$, which is given as input a verification key vk output by $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$ and oracle access to the signing algorithm $\mathsf{Sign}(\mathsf{sk}, \cdot)$ to whom it can (adaptively) request signatures on messages of its choice as in experiment in Figure 1.2. Let $\mathcal{M}$ be the set of queried messages. S is existentially unforgeable if for any such $\mathcal{A}$, there is a negligible function negl such that

$$\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathcal{A},\mathsf{S}}(\lambda) = \Pr[\mathsf{Exp}^{\mathsf{eu\text{-}cma}}_{\mathcal{A},\mathsf{S}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

that $\mathcal{A}$ produces a valid signature on a message $m \notin \mathcal{M}$ is a negligible function of λ.

1.3.1 The elliptic curve digital signature algorithm

Elliptic curve digital signature algorithm. ECDSA is the elliptic curve analogue of the Digital Signature Algoritm (DSA). It was put forth by Vanstone [Van92] and accepted as ISO, ANSI, IEEE and FIPS standards. It works in a group $(\mathbb{G}, +)$ of prime order q (of say μ bits) of points of an elliptic curve over a finite field, generated by P and consists of the following algorithms.

$\mathsf{KeyGen}(\mathbb{G}, q, P) \to (x, Q)$ where $x \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ is the secret signing key and $Q := xP$ is the public verification key.

$\mathsf{Sign}(x, m) \to (r, s)$ where r and s are computed as follows:

1. Compute m': the μ leftmost bits of $\mathsf{SHA256}(m)$ where m is to be signed.
2. Sample $k \xleftarrow{\$} (\mathbf{Z}/q\mathbf{Z})^*$ and compute $R := k^{-1}P$; denote $R = (r_x, r_y)$ and let $r := r_x \mod q$. If $r = 0$ choose another k.
3. Compute $s := k \cdot (m' + r \cdot x) \mod q$.

$\mathsf{Verif}(Q, m, (r, s)) \to \{0, 1\}$ indicating whether or not the signature is accepted.

About the security of ECDSA it is known that it is secure in the generic group model ([Sho97] for this model), i.e. assuming that the underlying group is a generic group, and under the assumption that the hash function considered is collision resistant ([Bro00] for the security proof and analysis, and [Bro02] for a revision from the same author to [Bro00]). In our application in next chapters, we deal with the construction of distributed version of ECDSA. During the last decade several paper about new solution to construct efficient distributed version of ECDSA were proposed ([GGN16], [Lin17], [GG18], [DKLs19], [CCL+19], [CCL+20], [CGG+20]). The motivation behind this intensive study is strongly linked with real application of the protocol. Indeed, ECDSA is a building block for signing Bitcoin transaction and Threshold ECDSA could further improve the state of art of the Bitcoin technology. For the case of $(t, n)-$threshold ECDSA we give a formal description and the security definitions that are needed. In particular, we recall the notion of *threshold signature unforgeability* from [GJKR96b] and a more recent definition, which is called *enhanced existential unforgeability* ([CGG+20]). This latter definition assumes the knowledge of the nonces R of the ECDSA protocol, *before* agreeing on the message to be signed. The motivation behind this last definition will be clear in Chapter 5, when we will require this definition for the first time.

$(t, n)-$threshold ECDSA. For a threshold t and a number of parties $n > t$, threshold ECDSA consists of the following interactive protocols:

$\mathsf{IKeyGen}\langle(\mathbb{G}, q, P); \ldots; (\mathbb{G}, q, P)\rangle \to \langle(x_1, Q); \ldots; (x_n, Q)\rangle$ s.t. $\mathsf{KeyGen}(\mathbb{G}, q, P) \to (x, Q)$ where the values $x_1, \ldots, x_n$ constitute a (t, n) threshold secret sharing of the signing key x.

$\mathsf{ISign}\langle(x_1, m); \ldots; (x_n, m)\rangle \to \langle(r, s)\rangle$ or $\langle\perp\rangle$ where $\perp$ is the error output, signifying the parties may abort the protocol, and $\mathsf{Sign}(x, m) \to (r, s)$.

The verification algorithm is non interactive and identical to that of ECDSA.

Following [GJKR96b], we can present a game-based definition of security analogous to $\mathsf{eu\text{-}cma}$, which is adapted to the $(t, n)-$ threshold case. This notion is called *threshold unforgeability under chosen message attacks* ($\mathsf{tu\text{-}cma}$).

Definition 1.3.3 (Threshold signature unforgeability [GJKR96b]). Consider a (t, n)-threshold signature scheme $\mathsf{IS} = (\mathsf{IKeyGen}, \mathsf{ISign}, \mathsf{Verif})$, and a PPT algorithm $\mathcal{A}$, having corrupted at most t players, and which is given the view of the protocols $\mathsf{IKeyGen}$ and ISign on input messages of its choice (chosen adaptively) as well as signatures on those messages. Let $\mathcal{M}$ be the set of aforementioned messages. IS is unforgeable if for any such $\mathcal{A}$, the probability $\mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\mathsf{IS},\mathcal{A}}$ that $\mathcal{A}$ can produce a signature on a message $m \notin \mathcal{M}$ is a negligible function of λ.

As in [CGG+20] we assume a somewhat enhanced variant of existential unforgeability under chosen message attacks ($\mathsf{e} - \mathsf{eu} - \mathsf{cma}$) for ECDSA. In this notion, for each signature query performed by the adversary $\mathcal{A}$, it gets to see the randomness R used to sign before choosing the message to be signed.

Definition 1.3.4 (Enhanced existential unforgeability [CGG+20]). Consider a PPT algorithm $\mathcal{A}$, which is given as input a verification key Q output by $\mathsf{KeyGen}(\mathbb{G}, q, P) \to (x, Q)$ and access to oracles:

- $\mathcal{O}^R$ to obtain a uniformly random point $R = (r_x, r_y)$ in $\mathbb{G}$;
- $\mathcal{O}^{\mathsf{Sign}(x,m;R)}$ which on input $m \in \mathbf{Z}/q\mathbf{Z}$ chosen by $\mathcal{A}$, returns a valid signature (r, s) on m where $r := r_x \mod q$ for some fresh $R = (r_x, r_y)$ which was output by $\mathcal{O}^R$ but has not been previously used by $\mathcal{O}^{\mathsf{Sign}}$; else it returns $\perp$.

Let $\mathcal{M}$ be the set of queried messages. ECDSA is enhanced existentially unforgeable under chosen message attack ($\mathsf{e} - \mathsf{eu} - \mathsf{cma}$) if for any such $\mathcal{A}$, the probability $\mathsf{Adv}^{\mathsf{e-eu-cma}}_{ECDSA,\mathcal{A}}$ that $\mathcal{A}$ produces a valid signature on a message $m \notin \mathcal{M}$ is a negligible function of λ.

Note that $\mathcal{A}$ *chooses* the messages queried to $\mathcal{O}^{\mathsf{Sign}}$, and *knows* (but does not choose) the randomness. Canetti *et al.* [CGG+20] show that in the generic group model ECDSA is $\mathsf{e} - \mathsf{eu} - \mathsf{cma}$; and that in some cases, enhanced unforgeability of ECDSA follows from standard unforgeability of ECDSA in the random oracle model.

We present a game-based definition of security analogous to $\mathsf{e} - \mathsf{eu} - \mathsf{cma}$: enhanced threshold unforgeability under chosen message attacks ($\mathsf{e} - \mathsf{tu} - \mathsf{cma}$).

Definition 1.3.5 (Enhanced threshold unforgeability). Consider a (t, n)-threshold ECDSA protocol $\mathsf{IS} = (\mathsf{IKeyGen}, \mathsf{ISign}, \mathsf{Verif})$, and a PPT algorithm $\mathcal{A}$, having corrupted at most t players, and which is given the view of the protocols $\mathsf{IKeyGen}$ and ISign on input messages of its choice as well as signatures on those messages. As in Definition 1.3.4, $\mathcal{A}$ can chose these messages adaptively, and after seeing the randomness used in ISign.

Let $\mathcal{M}$ be the set of aforementioned messages. The protocol IS is enhanced threshold unforgeable under chosen message attack ($\mathsf{e} - \mathsf{tu} - \mathsf{cma}$) if for any such $\mathcal{A}$, the probability $\mathsf{Adv}^{\mathsf{e-tu-cma}}_{\mathsf{IS},\mathcal{A}}$ that $\mathcal{A}$ can produce a valid signature on a message $m \notin \mathcal{M}$ is a negligible function of λ.

1.4 Equivocable commitment schemes

An equivocable commitment scheme is used by a sender S to commit to a message m such that it's message is perfectly hidden. When S reveals m and an opening value $\mathsf{d}(m)$ to a receiver R (this is called *opening phase),* S is computationally bound to the committed message. The scheme allows for a trapdoor which allows to open a commitment to arbitrary messages (this is called equivocating the commitment). The trapdoor should be hard to compute efficiently.

Formally a (non-interactive) equivocable commitment scheme consists of four PPT algorithms $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Equiv})$:

Setup The set up procedure $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{tk})$ takes a security parameter λ and it outputs public parameters pp and associated secret trapdoor key tk;

Com The committing algorithm $\mathsf{Com}(m, r) \to [\mathsf{c}(m), \mathsf{d}(m)]$ takes in input a message m and random coins r and outputs the commitment $\mathsf{c}(m)$ and an opening value $\mathsf{d}(m)$ (if S refuses to open a commitment $\mathsf{d}(m)$ is set to $\perp$);

Open The opening algorithm $\mathsf{Open}(\mathsf{c}, \mathsf{d}) \to m$ or $\perp$ takes in input a commitment c and an opening value d and outputs either a message m or an error symbol $\perp$;

Equiv The equivocating algorithm $\mathsf{Equiv}(\mathsf{tk}, m, r, m') \to \widehat{\mathsf{d}}$ allows to open commitments $\mathsf{c}(m)$ to arbitrary values m' if tk is a trapdoor key for pp.

Precisely, for any messages m and m', any $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{tk})$, let $\mathsf{Com}(m, r) \to [\mathsf{c}(m), \mathsf{d}(m)]$ and $\mathsf{Equiv}(\mathsf{tk}, m, r, m') \to \widehat{\mathsf{d}}$ then $\mathsf{Open}(\mathsf{c}(m), \widehat{\mathsf{d}}) \to m'$; and such that opening fake and real commitments is indistinguishable. The equivocable commitments we will use in next chapters satisfy the following properties:

- *Correctness:* for all message m and randomness r, if $[\mathsf{c}(m), \mathsf{d}(m)] \leftarrow \mathsf{Com}(m, r)$, then $m \leftarrow \mathsf{Open}(\mathsf{c}(m), \mathsf{d}(m))$.
- *Perfect hiding:* for every message pair m, m' the distributions of the resulting commitments are statistically close.
- *Computational binding:* for any PPT algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ outputs $(\mathsf{c}, \mathsf{d}_0, \mathsf{d}_1)$ such that $\mathsf{Open}(\mathsf{c}, \mathsf{d}_0) \to m_0$; $\mathsf{Open}(\mathsf{c}, \mathsf{d}_1) \to m_1$; $m_0 \neq \perp$; $m_1 \neq \perp$ and $m_0 \neq m_1$ is negligible in the security parameter.
- *Concurrent non-malleability:* a commitment scheme is non-malleable [DDN00] if no PPT adversary $\mathcal{A}$ can "maul" a commitment to a value m into a commitment to a related value $\overline{m}$. The notion of a concurrent non-malleable commitment [DDN00, PR05] further requires non-malleability to hold even if $\mathcal{A}$ receives many commitments and can itself produce many commitments.

1.5 Zero-knowledge proofs

Let R be a relation. A zero-knowledge proof of knowledge (ZKPoK) system for a binary relation R is an interactive protocol (P, V) between two probabilistic algorithms: a prover

P and a PT verifier V. In a ZKPoK, the goal of the prover P is to convince the verifier V that it knows a witness w for a given statement x such that $(x, w) \in \mathsf{R}$, without revealing to V anything than information that is not trivially known or the truth of the statement. Informally, a ZKPoK has to satisfy three properties, giving a *true* output with overwhelming probability if the prover behaves honestly, giving a *false* output with overwhelming probability if the prover is cheating about the knowledge of the witness and do not reveal anything which is not necessary to be known. Zero-knowledge proofs are fundamental in many contexts in Cryptography, since they permit to prove the validity of actions done by parties without revealing secret values. This is of particular relevance in contests where the adversary is malicious and it can deviate from the protocol. Indeed, zero-knowledge proofs can require to prove the validity of some actions, also for the adversary. Of particular interest is their usage in multiparty protocols, and as we will see in Chapter 3,4,5, we build general purpose zero-knowledges for CL, the Castagnos-Laguillaumie encryption scheme ([CL15]) we use for our applications. We below define formally a zero-knowledge proof of knowledge and the concept of computationally convincing proofs of knowledge. Furthermore, we will look at another type of zero-knowledge, i.e. zero-knowledge argument of knowledge (ZKAoK). Arguments of knowledge play an important role in the efficiency improvements of the state of art proofs for the validity of a CL encryption. We will see the original ZKPoK from [CCL$^+$19] in Chapter 3 and new efficient ZKAoKs from [CCL$^+$20] in Chapter 4.
Coming back to ZKPoK, consider a language $\mathcal{L}$. $x \in \mathcal{L}$ is a *true* statement while $x \notin \mathcal{L}$ is a *false* and $1 \leftarrow (P, V)(x)$ (resp. $0 \leftarrow (P, V)(x)$) denotes the case when V interacting with P accepts (resp. rejects) the proof statement.

Definition 1.5.1. Let $\mathcal{L}$ be a language and $x \in \mathcal{L}$ (or $x \notin \mathcal{L}$) be a statement; a ZKPoK must satisfy the following properties:

- *Completeness*: for any $x \in \mathcal{L}$:
$$\Pr[1 \leftarrow (P, V)(x)] > 1/2$$
- *Soundness*: for any prover P^* and for any $x \notin \mathcal{L}$:
$$\Pr[0 \leftarrow (P^*, V)(x)] > 1/2$$
- *Zero-knowledge*: for every probabilistic polynomial time verifier V^*, there exists a probabilistic simulator Sim running in expected polynomial time such that for every $x \in \mathcal{L}$,
$$(P, V^*)(x) \equiv Sim(x).$$
$(P, V)(x)$ is a random variable representing the output of V at the end of an interaction with P, then the zero-knowledge property holds if for any probabilistic polynomial time V^*, the output of V^* after an interaction with P is the same one of the simulator.

For a full explanation on this model see [Gol01] for interactive proofs and [GMR89] for zero-knowledge.
In a zero-knowledge argument of knowledge (ZKAoK), the proof provided by P is computationally sound (P is also a PT algorithm).

To refer to a zero-knowledge, we use the notation introduced by Camenisch-Stadler [CS97], which conveniently expresses the goals of a ZKP (resp. ZKA) scheme:

$$\mathsf{ZKPoK}_x\{(w) : (x, w) \in \mathsf{R}\} \quad \text{and} \quad \mathsf{ZKAoK}_x\{(w) : (x, w) \in \mathsf{R}\},$$

where x is a statement and w a witness. Finally, we provide formal definitions for computationally convincing proofs of knowledge in the next paragraph.

Computationally convincing proofs of knowledge. We here provide some terminology and definitions relating to computationally convincing proofs of knowledge (or arguments of knowledge) as defined in [DF02]. Consider a *relation generator* algorithm $\mathcal{R}$, that takes as input 1^λ and outputs the description of a binary relation R. A prover is a machine P who gets R as an input, outputs a statement x and finally conducts the interactive proof with a verifier V using R, x as common input. From P, define a machine P_{view} which starts in the state P is in after having seen view view and having produced x. P_{view} then conducts the protocol with V following P's algorithm. The view view contains all inputs, messages exchanged and random coins so in particular x is determined by view. We note $\epsilon_{\mathsf{view},P}$ P's probability to make V accept, conditioned on view. The knowledge error function $\kappa(\lambda)$ is the probability that P can make V accept without knowing a witness w s.t. $(x, w) \in \mathsf{R}$ (for a security parameter λ). An extractor is a machine M that gets R and a statement x as an input, has black-box access to P_{view} for some view consistent with x and computes a witness w s.t. $(x, w) \in \mathsf{R}$.

Definition 1.5.2. For some given cheating P^*, extractor M and polynomial $p()$, M fails on view view if $\epsilon_{\mathsf{view},P} > \kappa(\lambda)$, and the expected running time of M using P^*_{view} as oracle, is greater than $\frac{p(\lambda)}{\epsilon_{\mathsf{view},P^*} - \kappa(\lambda)}$.

Definition 1.5.3. Let $\mathcal{R}$ be a probabilistic polynomial time relation generator, and consider a protocol (P, V), a knowledge extractor M, polynomial $p()$ and knowledge error function $\kappa(\lambda)$ be given. Consider the following experiment with input λ: $\mathsf{R} := \mathcal{R}(1^\lambda)$, $x := P^*(\mathsf{R})$ which defines view view. The advantage of P^*, denoted $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)$, is the probability (taken over the random coins of $\mathcal{R}$, P^*) that M fails on the view generated by this experiment.

Definition 1.5.4. Let $\mathcal{R}$ be a PPT relation generator. (P, V) is a computationally convincing proof of knowledge for $\mathcal{R}$, with knowledge error $\kappa()$, failure probability $\nu()$ and time bound $t()$, if the following hold:

Completeness The honest prover P receives $\mathsf{R} \leftarrow \mathcal{R}(1^\lambda)$, produces $(x, w) \in \mathsf{R}$, sends x to V and finally conducts the protocol with V, who accepts with overwhelming probability in λ.

Soundness: There exists a polynomial $p()$ and an extractor M, s.t. for all provers P^* running in time at most $t(\lambda)$, $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda) \leq \nu(\lambda)$.

1.5.1 Sigma protocols

Given a relation R, a Sigma protocol for R is a $3-$ round protocol between a prover P and a verifier V. Informally, in Sigma protocols initially the prover sends a message a to the verifier V, then the verifier V sends back a $k-$bit *challenge* e to P. Finally, P compute an answer z and sends it to V and V checks some relation between the statement and the answer. Formally,

Definition 1.5.5. Let R be a relation. A protocol π is said to be a Sigma protocol for R if it is a $3-$move protocol as above and if it satisfies the following properties:

- *Completeness*: If P and V follow the protocol on input x and witness w to P, where $(x, w) \in \mathsf{R}$, the verifier always accepts.

- *Special soundness*: From any x and any pair of accepting conversations on input x, (a, e, z), (a, e', z') where $e \neq e'$, one can efficiently compute w such that $(x, w) \in \mathsf{R}$.

- *Special honest-verifier zero-knowledge*: There exists a polynomial-time simulator M, which on input x and a random e outputs an accepting conversation of the form (a, e, z), with the same probability distribution as conversations between the honest P, V on input x.

Theorem 1.5.1. *Let π be a Sigma protocol for a relation R with challenge length k. Then π is a proof of knowledge with knowledge error 2^{-k}.*

Random Oracle Model (ROM) and the Fiat-Shamir Heuristic Firstly formally treated by Bellare and Rogaway in [BR93], the *random oracle model* (ROM) is an idealized model to prove security of cryptographic schemes. A *random oracle* is a black-box function $H : \{0,1\}^{t_1} \to \{0,1\}^{t_2}$, acting as an oracle, that takes in input a string of lenght t_1 and return a truly random string of lenght t_2. Furthermore, H is fixed, i.e. it returns the same value on the same string. An advantage in assuming the existence of the random oracle is that it allows to transform interactive zero-knowledge proof in non-interactive. Anyway, it is usual to consider the existence of a random oracle model as a strong assumption. Indeed, normally ROM is instantiated using collision resistant hash functions, but there is no guarantee that a protocol which is proven secure in the random oracle model has also a secure instantiation[2]. Using a hash function in place of the verifier to do non-interactive proofs is also called *Fiat-Shamir heuristic*[3], and this idea comes from [FS87]. Finally, Fiat-Shamir heuristic is useful to transform Honest Verifier Zero-Knowledge property of a Sigma protocol to zero-knowledge property. Indeed, making a Sigma protocol non interactive excludes a dishonest verifier to choose a challenge with some property useful for obtaining informations about the witness. Using a random oracle, the challenge is chosen uniformly in the challenge set. This modus operandi is of particular interest in protocols that cope with active adversaries, motivated by the fact that the Sigma protocol is originally only *honest* verifier zero-knowledge.

[2]Indeed, there exist negative results about the issue of translating a random oracle to a cryptographic hash functions. In [CGH04] the authors prove that there exist encryption and signature schemes which are proven secure in the ROM, but such that there is no secure implementation of the random oracle for which the resulting schemes are secure.

[3]Actually, Fiat-Shamir transform was initially thought to convert identification protocols to signature protocols

1.5.2 Zero Knowledge Proofs concerning DL

We recall in this paragraph simple proofs of the knowledge of a discrete logarithm of an element $Q \in \mathbb{G}$, given a group $\mathbb{G}$ and a generator P, i.e. $x \in \mathbf{Z}$ such that $Q = x \cdot P$ and for the similar relations involving discrete logarithm. The proofs as they are presented, are used in the context of a cyclic group $\mathbb{G}$ of finite order. In Chapters 2, 3, 4 and 5 we will also consider groups of unknown order, and necessary proofs for that case are presented in dedicated sections.

Schnorr Protocol Schnorr (cf. [Sch90]) proposed a zero-knowledge protocol for the relation $\mathsf{R}_{\mathsf{DL}} = \{Q; x : Q = x \cdot P\}$ for some $P, Q \in \mathbb{G}$ where $\mathbb{G}$ is a group of known finite order. The protocol is recalled in Figure 1.3. Schnorr protocol is an easy proof for proving the knowledge of the discrete logarithm of a given element in a finite group $\mathbb{G}$ and requires the computation of only 3 group elements, which are 2 from the receiver and 1 from the verifier.

$$
\begin{array}{ccc}
\text{Prover } (Q, x \in \mathbf{Z}/q\mathbf{Z}) & & \text{Verifier } (Q) \\
\hline
r \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z} & & \\
A := r \cdot P & \xrightarrow{\quad A \quad} & \\
 & & k \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z} \\
 & \xleftarrow{\quad k \quad} & \\
z := r + kx \in \mathbf{Z}/q\mathbf{Z} & \xrightarrow{\quad z \quad} & \text{Check if } z \cdot P = A + k \cdot Q \\
\hline
\end{array}
$$

Figure 1.3: Schnorr Sigma-protocol for discrete logarithm relation R_{DL}

Other proofs for DL based relations In Chapters 4 and 5 we will require proofs for relations which involve the knowledge of discrete logarithm composed by other statements. We present them in this paragraph since they are similar to Schnorr proof. First, consider the relation

$$\mathsf{R}_{\mathsf{double-DL}} = \{(T); (\sigma, \ell) : T = \sigma \cdot P + \ell \cdot H\},$$

where $P, H \in \mathbb{G}$ and $\mathbb{G}$ is finite group of order q prime. An (honest-verifier) zero-knowledge argument for $\mathsf{R}_{\mathsf{double-DL}}$ is presented in Figure 1.4.

The second relation is

$$\mathsf{R}_{\mathsf{triple-DL}} = \{(T, S, R); (\sigma, \ell) : T = \sigma \cdot P + \ell \cdot H \wedge S = \sigma \cdot R\},$$

where $P, H \in \mathbb{G}$ and $\mathbb{G}$ is finite group of order q prime. An (honest-verifier) zero-knowledge argument for $\mathsf{R}_{\mathsf{triple-DL}}$ is presented in Figure 1.5.

$$\begin{array}{ccc}
\text{Prover } (T, \sigma \in \mathbf{Z}/q\mathbf{Z}, \ell \in \mathbf{Z}/q\mathbf{Z}) & & \text{Verifier } (T) \\
\hline
a, b \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z} & & \\
A := a \cdot P + b \cdot H & \xrightarrow{\quad A \quad} & \\
 & & k \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z} \\
 & \xleftarrow{\quad k \quad} & \\
t := a + k\sigma \in \mathbf{Z}/q\mathbf{Z} & & \\
u := b + k\ell \in \mathbf{Z}/q\mathbf{Z} & \xrightarrow{\quad t,u \quad} & \text{Check if } t \cdot P + u \cdot H = A + k \cdot T \\
\hline
\end{array}$$

Figure 1.4: Sigma protocol for relation $\mathsf{R}_{\mathsf{double-DL}}$

$$\begin{array}{ccc}
\text{Prover } (T, S, \sigma \in \mathbf{Z}/q\mathbf{Z}, \ell \in \mathbf{Z}/q\mathbf{Z}) & & \text{Verifier } (T, S) \\
\hline
a, b \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z} & & \\
A := a \cdot R & & \\
B := a \cdot P + b \cdot H & \xrightarrow{\quad A,B \quad} & \\
 & & k \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z} \\
 & \xleftarrow{\quad k \quad} & \\
t := a + k\sigma \in \mathbf{Z}/q\mathbf{Z} & & \\
u := b + k\ell \in \mathbf{Z}/q\mathbf{Z} & \xrightarrow{\quad t,u \quad} & \text{Check if } t \cdot R = A + k \cdot S \\
 & & \text{and } t \cdot P + u \cdot H = B + k \cdot T \\
\hline
\end{array}$$

Figure 1.5: Sigma protocol for relation $\mathsf{R}_{\mathsf{triple-DL}}$

1.6 Verifiable secret sharing

Threshold secret sharing. A (t, n) threshold secret sharing scheme allows to divide a secret s into shares $s_1, \ldots, s_n$, amongst a group of n participants, in such a way that knowledge of any $t + 1$ or more shares allows to compute s; whereas knowledge of any t or less shares reveals no information about s.

Verifiable Secret Sharing A verifiable secret sharing (VSS) protocol allows to share a secret between n parties $P_1, \ldots, P_n$ in a verifiable way. Specifically, it can be used by a party to share a secret with the other ones. The main idea behind VSS is sending secrets with auxiliary infos which do not reveal sensible data under computational assumptions or that are secure in a information theoretic sense. In the context of multiparty computation, protocols as Feldmann's VSS ([Fel87]) and Pedersen's VSS ([Ped92]) are widely used. For what concerns threshold signature protocols in Chapters 4 and 5, we will use Feldman VSS. Feldman VSS relies on Shamir's secret sharing scheme [Sha79], but the former gives additional information allowing to check the sharing is done correctly.

Feldman Verifiable Secret Sharing

Definition 1.6.1 (Feldman Verifiable Secret Sharing). Let $\mathbb{G}$ be a group of order q, g a generator of $\mathbb{G}$, $P_1, \ldots, P_n$ a set of n parties, and suppose that one of the parties, that we call P, wants to share a secret $\sigma \in \mathbf{Z}/q\mathbf{Z}$ with the other ones. To share the secret, it does the following steps:

1. P generates a random polynomial $p \in (\mathbf{Z}/q\mathbf{Z})[x]$ of degree t and with σ as free term. The polynomial is then

$$p(x) = a_t x^t + a_{t-1}x^{t-1} + \ldots + a_2 x^2 + a_1 x + \sigma \mod q,$$

   where $\sigma = p(0) \mod q$. The shares of σ are $\sigma_i = p(i) \mod q$.

2. P sends σ_i to P_i, for all $i \in [n]$.

3. P publishes auxiliary information that other players can use to check the shares are consistent and define a unique secret: $\{v_j = g^{a_j} \in \mathbb{G}\}_{j \in [t]}$ and $v_0 = g^{\sigma} \in \mathbb{G}$.

Each party can check its own share is consistent by verifying if the following condition holds:

$$g^{\sigma_i} = \prod_{j=0}^{t} v_j^{i^j} \in \mathbb{G}$$

If one of the checks fails, then the protocol terminates. Furthermore, the only information that the Feldman's VSS leaks about the secret σ is $v_0 = g^{\sigma}$. The computational security of Feldman VSS is based on the hardness of the Discrete Logarithm Problem in the group $\mathbb{G}$.

Lagrange Interpolation In the construction of a multiparty protocol with a threshold t, it is usual to convert a shared secret among n players in t shares of it. This is useful for example in context where after n players run a key generation protocol, only a part of them compute the rest of relevant values and operations, say t of them. This is the case of some known works on Threshold ECDSA, where n parties compute public and secret (implicitly) keys, while $t < n$ compute the signature of a message. If we take the t shares of the signers as they are, they can be inconsistent with the previously computed signing and verification keys. In Cryptography, a common way to convert n shares of some value in t shares such that they guarantee that consistency is using Lagrange Interpolation. Lagrange Interpolation is a way to construct a degree t polynomial in $\mathbb{F}[x]$, where $\mathbb{F}$ is a field (finite or infinite), from $t+1$ points. In the context we will consider, the secret is implicitly defined in the known term of a polynomial. Then, $t+1$ points are necessary to compute the polynomial, but in general no one knows $t+1$ points and as a consequence no one knows the polynomial in clear. Since a secret can be set as the degree zero term of the resulting polynomial, the fact that the polynomial is not known is a important point. Actually, the most important requirement is that the polynomial where the secret is hidden is defined and it is unique, even if no one knows it. We give the definition of Lagrange Interpolation below. In addition, we are not concerned with the mathematical proof of the existence of the polynomial.

Definition 1.6.2 (Lagrange Interpolation). Let $\{(x_i, y_i)\}_{i\in\{0,\ldots,t\}}$ be a set of $t+1$ points, where $x_i \neq x_j, \forall i \neq j$, the interpolation polynomial in the Lagrange form is a linear combination

$$\mathcal{L}(x) := \sum_{j=0}^{t} y_j \lambda_j(x)$$

of Lagrange basis polynomials

$$\lambda_j(x) := \prod_{\substack{0\leq m\leq t\\ m\neq j}} \frac{x - x_m}{x_j - x_m}$$

where $0 \leq j \leq t$. From the assumption that no two x_j are the same, then (when $m \neq j$) $x_j - x_m \neq 0$, so this expression is always well-defined.

If a secret s is the degree 0 term of the resulting polynomial $\mathcal{L}$, then $s = \mathcal{L}(0) = \sum_{j=0}^{k} y_j \lambda_j(0)$. Notice that in a cryptographic context, if x_i are known to all the parties $P_1, \ldots, P_n$, then everyone can compute $\lambda_j(x)$ and in particular $\lambda_j(0)$. As a consequence, the converted (t, t) share of the original (t, n) share of the secret s of the player P_j is $y_j \lambda_j(0)$.

Chapter 2

Background on Class Group of Imaginary Quadratic Fields

Introduction to the chapter This chapter is dedicated to Class Groups of Imaginary Quadratic Fields, the building block of our instantions of the schemes in Chapter 3, 4 and 5. During the reading of this chapter, sometimes we shortly call them Class Groups, since we will remain inside an *imaginary* quadratic field all the time, and then it is not necessary to specify what kind of field is considered in each occasion. Indeed, even if Class Groups can be built from *real* quadratic fields, we are interested only in the imaginary case motivated by constructions in Cryptography based on them and that we use for our results. Before going on, we specify that this chapter does not include new results, but it is thought as a background on the theory on Class Groups. The idea behind this chapter is put together the study of different authors (cf. [BV07] for the starting background, [Cox14] for more advanced topics at the basis of recent cryptographic construction and [Coh00] for the computational point of view) together with some paper results to explain in a linear way the cryptosystem of our interest in this manuscript. At the same time this chapter can be assumed as a general background about Class Groups theory and complexity in applications.
This chapter is a little longer than a standard background section we can find in a paper because our idea is to give a more detailed description of these objects which can be useful to understand what happens behind the cryptographic protocols we will consider. Indeed, our goal is presenting these objects without leaving relevant concepts and results unexplained or assumed as facts, except in cases where theorems involved require an advanced knowledge of other topics, an unnecessary background or if they require too much space. In latter cases, we will simply give the reasons why those theorems are important and necessary. On the other hand, this Chapter is a lot shorter than a book entirely dedicated to Class Groups, because our aim in this thesis is not presenting a course on this interesting topic. This chapter can be seen as a short manual divided in two parts. In the first part (cf. Section 2.1) it is explained what are class groups of an imaginary quadratic field from an algebraic point of view with the purpose to reduce at minimum the basic knowledge required from courses in Commutative Algebra and Algebraic Number Theory, and making them more accessible. Furthermore, we present also relevant algorithms with their complexity. In the second part (cf. Section 2.2) we look at some of the cryptographic aspects of class groups: which operations are efficient with their elements

– e.g. computing square roots –, which problems are hard and which are the improvements that bring them to be used in Cryptography. Obviously, these improvements are the only ones we are interested in, i.e. how the class groups are useful to build a specific encryption scheme based on them, i.e. the Castagnos-Laguilllaumie encryption scheme (CL from now). Indeed, CL scheme is a building block of the original works on distributed ECDSA in next chapters, i.e. the main topic of this thesis. The second part refers to the first one to understand and motivate actions took by the authors of the contributions we will see. We will refer in particular to the results of Castagnos and Laguillaumie ([CL09], [CL15]) and the improvements in [CLT18a]. Finally, sometimes some concepts will be given as an intuition without an excess of formality because the aim of this Chapter is introducing cryptographers to this mathematical field (where this time ”field” is not intended in a algebraic sense), and not losing the idea of what we are doing. Indeed, the author of this thesis is not a researcher in Pure Algebra, but he is a little interested in.

In general, independently from the context, a class group is a group structure where elements are classes. A class is defined by the elements of the set involved which are equivalent with respect to an equivalence relation. We will detail about the precise structure required in this chapter. Informally, our target is to construct Class Groups in a specific context, i.e. a group of classes inside an Imaginary Quadratic Field. At this point there are a lot of questions to answer and all of them will receive a detailed response. Some of them could be:

- What is the nature of elements defining the classes?
- How is the group operation defined?

In Section 2.1 we give an answer to previous questions and others. After giving the algebraic background, in Section 2.2 we will answer to questions as:

- Which are the difficult problems with Class Groups?
- How does CL uses class groups and under which assumption it is proved secure?

During both the Sections, we recall algorithms that compute the involved operation also looking at their computational cost. Then we will answer to questions as:

- How much efficient are operations in the Class Group?
- How much inefficient are the algorithms which solve hard problems in a Class Group?

Finally, a Class Group of Imaginary Quadratic Fields is called also the *Ideal Class Group* to distinguish it from the *Form Class Group*. We consider both of them, but the Class Group we consider in the end is the ideal one.

2.1 Towards the definition of the Ideal Class Groups: the Algebraic Part

In this section we have as final goal to define Class Groups of Imaginary Quadratic Fields. The style followed in this section is constructive following a bottom-up direction: we start with mathematical entities that at first sight seem cannot have nothing to do with the Class Group, but things will be clear paragraph by paragraph. In the introduction of this chapter we informally said what a class group is in general, without talking about the details as the objects considered and their properties, but saying only that we have a group structure with an unknown defined operation. To sum up, to define Class Groups we need principally two elements: a set and an operation. We start introducing *binary quadratic forms*, which are bivariate homogeneous degree two polynomials that are easy to understand, the operation on them and then we define the Form Class Group. A Form Class Group is strongly linked to an Ideal Class Group in a quadratic field. This link gives a correspondence of the representations of quadratic forms and ideals, translating operations on the form class group to the ideal class group. Indeed, quadratic forms are more accessible and some operations on them are quite simple and fast, and furthermore being able to use the same algorithms in ideal class groups is a clear advantage.

More in detail, our description starts with defining *binary quadratic forms* and important algorithms that are related to them, then we will pass to pure algebraic objects as *imaginary quadratics fields* and *orders*. Next, we will define *ideals* in the orders – which are the elements that represent the ideal classes – and the structure of their product, completing all with the definition of Class Group of Imaginary Quadratic Fields. During the description, some proof is omitted, while fundamental lemmas, propositions, theorem are proved and explained when possible.

Notation. Bold letters $\mathbf{N}, \mathbf{Z}, \mathbf{Q}, \mathbf{R}, \mathbf{C}$ denote the usual sets of natural, integral, rational, real and complex numbers and $\mathbf{A}^n$ denote their repetition for n times where $\mathbf{A}$ is one of the previous sets. In the case of integers, with $a|b$ we indicate that a divides b. Expression as $f(x, y)$ are two-variable functions. $b \mod a$ and $b \bmod_c a$ denote the remainder of b divided by a using the Extended Euclid Algorithm and the Centered Extended Euclid Algorithm, respectively. The latter one gives the remainder $r > 0$ if $-a/2 < r \leq a/2$ and $r - a$ otherwise. $[a]$ denote the integer part of a. With the notation $|\cdot|$ we refer to the absolute value of a number, not the size.

2.1.1 Binary Quadratic Forms

A *binary quadratic form* is a bivariate homogeneous degree 2 polynomial

$$f(x, y) = ax^2 + bxy + cy^2,$$

with a, b and c real coefficients. If the coefficients are integers, the form f is called *integral.* An integral form is called *primitive* if $\gcd(a, b, c) = 1$. From now, we will use the triple representation of f, i.e. $f = (a, b, c)$, or $f(x, y)$ depending on what we are interested in. The triple representation is trivially unique. The quantity $\Delta(f) = b^2 - 4ac$ is the *discriminant* of f. When clear from the context, we will refer to binary quadratic forms as *forms.* Depending on the real solutions of $f(x, y) = 0$, forms can be characterized. Furthermore, the characterization depends only on the discriminant and the sign of a or c.

Definition 2.1.1. A form f is said

- *positive definite* if $f(x, y) > 0$ for each couple $(x, y) \neq (0, 0)$ of real values

- *positive semidefinite* if $f(x, y) \geq 0$ for each couple (x, y) of real values
- *negative definite* if $f(x, y) < 0$ for each couple $(x, y) \neq (0, 0)$ of real values
- *negative semidefinite* if $f(x, y) \leq 0$ for each couple (x, y) of real values
- *indefinite* if there exists at least a couple (x, y) of real values such that if $f(x, y) > 0$ and at least a couple (x, y) of real values such that if $f(x, y) < 0$

Proposition 2.1.1. The form f is

- positive definite if and only if $\Delta(f) < 0$ and $a > 0$
- positive semidefinite if and only if $\Delta(f) \leq 0$ and $(a > 0$ or $c > 0)$
- negative definite if and only if $\Delta(f) < 0$ and $a < 0$
- negative semidefinite if and only if $\Delta(f) \leq 0$ and $(a < 0$ or $c < 0)$
- indefinite if and only if $\Delta(f) > 0$

Remark 2. Note that for integral forms $\Delta = 0, 1 \mod 4$, since $\Delta = b^2 - 4ac \equiv b^2 \mod 4$ and a square $\mod 4$ can be only 0 or 1. We will consider only valid discriminant, and for each valid discriminant there exist a form. In particular, the forms $x^2 + (\Delta/4)y^2$ if $\Delta = 0 \mod 4$ or $x^2 + xy + ((1 - \Delta)/4)y^2$ if $\Delta = 1 \mod 4$ is called the *principal* form of discriminant Δ.

We are interested only in positive definite forms, though there is a full literature about indefinite forms and class groups of a real quadratic field. At this point, we still do not know the relation between forms and ideals, then we premise that positive definite forms are strongly linked with ideals in a imaginary quadratic field, while indefinite forms are linked to ideals in a real quadratic field. We will see later this link. From what we said, forms that are not positive definite are out of our scope and the case of indefinite forms is not part of the class group based cryptography we will consider in next chapters. From now, after having given some general definitions and easy results on forms, we will consider only integral forms, i.e. $f = (a, b, c)$ with integers a, b and c.
Consider a form $f = (a, b, c)$ and write it as a function, i.e. $f(x, y) = ax^2 + bxy + cy^2$. A general linear transformation that applied to a form results in another form is denoted as $U(x, y) = (sx + ty, ux + wy)$. This transformation can be written in a matrix form from its coefficients, i.e. $U = \begin{pmatrix} s & t \\ u & v \end{pmatrix}$. The form after transformation is

$$\begin{aligned} g(x, y) &= f(sx + ty, ux + wy) \\ &= (as^2 + bsu + cu^2)x^2 + (2(ast + cuv) + b(sv + tu))xy + (at^2 + btv + cv^2)y^2 \\ &= f(s, u)x^2 + (2(ast + cuv) + b(sv + tu))xy + f(t, v)y^2 \end{aligned}$$

Two forms f, g are said to be *equivalent* if there exists a matrix $U \in \mathrm{GL}(2, \mathbf{Z})$ such that $g = \det(U)f(U)$, where $f(U)$ represents the form after the linear transformation above. They are said *proper equivalent* if exists a matrix $U \in \mathrm{SL}(2, \mathbf{Z})$ such that $g = f(U)$. Remember that $\mathrm{GL}(2, \mathbf{Z})$ and $\mathrm{SL}(2, \mathbf{Z})$ are the groups of matrices $U \in \mathbf{Z}^{2\times 2}$ such that $\det(U) = \pm 1$ and $\det(U) = 1$, respectively.

Lemma 2.1.2. The discriminant of two equivalent forms f and $g = \det(U)f$ (as defined above) is the same.

Proof. Just notice that $\Delta(g) = (2(ast + cuv) + b(sv + tu))^2 - 4 \cdot (as^2 + bsu + cu^2) \cdot (at^2 + btv + cv^2) = b^2 - 4ac = \Delta(f)$ $\square$

With some calculations, the following proposition is proved.

Proposition 2.1.3. Proper equivalence is an equivalence relation on the set of all integral primitive positive definite forms. Equivalence is an equivalence relation on the set of all integral primitive indefinite forms.

From Proposition 2.1.3 the equivalence relation we consider for the forms is the proper equivalence. From Lemma 2.1.2 and Proposition 2.1.3, it is possible to divide all primitive positive definite forms with the same discriminant in classes of proper equivalence. In each of the classes there are special forms called *reduced*, and in the following subsubsections we will see that actually there is only one reduced form in each (proper) equivalence class.

Definition 2.1.2. A positive definite form $f = (a, b, c)$ is said *reduced* if $-a < b \leq a$, $a \leq c$ and $b \geq 0$ if $a = c$. f is said *normal* if $-a < b \leq a$ (it is not required that $a \leq c$).

2.1.1.1 The reduction algorithm

We introduced the notion of reduced forms above and we said that in each class there is exactly one reduced form. In this subsubsection we prove this fact and we explain how to compute the reduced form equivalent to a given form f. Among all proper transformation, two of them form a set of generators for $\mathrm{SL}(2, \mathbf{Z})$ (cf. [BV07], Ex. 2.9.1) and then each proper transformation U can be represented by a combination of only two matrices. More in detail, this two matrices are $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ and $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, and it is proved that each $U \in \mathrm{SL}(2, \mathbf{Z})$ can be written as $U = T^{s_1}S^{t_1}T^{s_2}S^{t_2} \cdots T^{s_k}S^{t_k}$ for some $k \in \mathbf{N} \setminus \{0\}$ and some exponent $s_i, t_i \in \mathbf{Z}$, $1 \leq i \leq k$. Then, to describe any proper transformation it is enough to work only with S and T. Matrix S exchanges the values of a and c, and it changes the sign of b. It is useful in the reducing process when the condition $a \leq c$ is not true. Furthermore, if $a = c$ it changes $b < 0$ to $b \geq 0$, motivating the condition $b \geq 0$. Matrix $T^s = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix}$ transforms (a, b, c) in $(a, b + 2as, c + bs + as^2)$, then it is useful in reducing $b > a$ or $b \leq -a$ to a value that satisfies the inequality $-a < b \leq a$ without changing the value of a. Indeed, to do that it is enough to choose $s = \left[-\frac{b}{2a}\right]$[1]. Using S and T we prove the existence and unicity of the reduced form in a class. In next proposition, we give implicitly a way to reduce a form and then we present an algorithm in clear.

Proposition 2.1.4 ([Coh00], Proposition 5.3.3)**.** In every equivalence class of positive definite forms there is only one reduced form.

[1]Notice that this is the unique value of s such that $-a < b + 2as \leq a$. It is computed by a centered Euclidean division, i.e. consider $b \mod 2a$ if b satisfies inequalities or $(b \mod 2a) - 2a$ otherwise. (remember that in this case $|b| > a$ and only one of the values satisfy the condition

Proof. We prove in the first part the existence of reduced form in a class, then that if a form is reduced is unique.

Existence Consider all forms (a, b, c) in a given class, and take one for which a is minimal. This implies that for any such form we have $c \geq a$ since (a, b, c) is equivalent to $(c, -b, a)$ (using S transformation). Then, applying the transformation T^s with $s = \left[-\frac{b}{2a}\right]$, the value of a does not change, while the b is in the interval $]-a, a]$. Since a is minimal, we will still have $a \leq c$, hence the form that we have obtained is essentially reduced. If $c = a$, changing (a, b, c) again in $(c, -b, a)$ sets $b \geq 0$ as required.

Unicity If (a, b, c) is reduced, we prove that a is minimal among all the forms equivalent to (a, b, c). Indeed, every other a' has the form $a' = as^2 + bsu + u^2$ with s and u coprime integers – from any proper transformation $U = \begin{pmatrix} s & \cdot \\ u & \cdot \end{pmatrix}$ – and the identities

$$as^2 + bsu + cu^2 = as^2 \left(1 + \frac{b}{a}\frac{u}{s}\right) + cu^2 = as^2 + cu^2 \left(1 + \frac{b}{c}\frac{s}{u}\right)$$

immediately imply our claim, since $|b| \leq a \leq c$. Now in fact these same identities show that the only forms equivalent to (a, b, c) with $a' = a$ are obtained by changing (x, y) into $(x + ky, y)$ (corresponding to $s = 1$ and $u = 0$), and this finishes the proof of the proposition.

□

Next point to see is that the number of classes of primitive positive definite forms is finite. Logically, since in each class there is a unique reduced form and then different reduced forms are representative of distinct classes, the number of classes is the same as the number of primitive reduced forms. Therefore, proving that primitive reduced forms are a finite number is equivalent to prove that the number of classes is finite.

Lemma 2.1.5. If a form (a, b, c) is reduced, then $a \leq \sqrt{|\Delta|/3}$. If $a < \sqrt{|\Delta|/4}$, then (a, b, c) is reduced.

If a form is reduced, $|b| < a \leq \sqrt{|\Delta|/3}$ and c is fixed by a, b, Δ. As a consequence, there are at most $\sqrt{|\Delta|/3} \cdot 2 \cdot \sqrt{|\Delta|/3} = \frac{2}{3} \cdot |\Delta|$ reduced forms of discriminant Δ, which is a finite number. Actually, since b has the same parity of Δ (remember that $\Delta = b^2 \mod 4$), the upper bound is reduced by an half to $\frac{1}{3} \cdot |\Delta|$.

Example 1. The unique reduced class of discriminant $\Delta = -3$ is $f = (1, 1, 1)$, while the reduced classes with $\Delta = -75$ are $f = (3, 3, 7)$ and $f = (1, 1, 19)$. The class $(5, 5, 5)$ is not considered since it is not primitive.

The class number

Definition 2.1.3. The number $h(\Delta)$ of proper equivalence classes of primitive positive definite forms of discriminant Δ is called *class number*. Since there is only one reduced form in each class, $h(\Delta)$ is equal to the number of *primitive reduced* positive definite forms of discriminant Δ.

The inefficiency of algorithms computing the class number for large values of the discriminant is postponed to Section 2.2. For now, we give the basic idea on how compute the class number using an easy inefficient deterministic procedure. We do not write the algorithm in a dedicated figure since it is very easy and we can explain it in few lines. The algorithm uses Lemma 2.1.5 and works computing $c = \frac{b^2-\Delta}{4a}$ for each $0 < a \leq \sqrt{|\Delta|/3}$, $0 \leq b \leq \sqrt{|\Delta|/3}$, such that $b = \Delta \mod 2$ and $b \leq a$, looking if (a, b, c) is a reduced form and considering also $(a, -b, c)$ as reduced if $a \neq c$ and $b \neq a$. Finally, for each reduced form it updates the class number by 1 or 2, where h is increased by 1 if $a = b$ or $a = c$, since in the definition of reduced forms $b \geq 0$ in that cases, and by 2 in the other cases since we obtain (a, b, c) and $(a, -b, c)$ as reduced forms.

We present the reduction algorithm for forms in Algorithm 1 and we call it $\mathsf{Red}(f)$. It consists on applying the two different proper equivalent transformations S and T for a finite number of times. Notice that proper equivalence sends primitive forms to primitive forms. Indeed, let $g(x, y) = f(sx + ty, ux + vy)$ be a proper transformation of $f = (a, b, c)$, where f is not primitive and $d = \gcd(a, b, c) > 1$. Then, $g = (f(s, u), 2(ast + cuv) + b(sv + tu), f(t, v))$, and d divides each of the three coefficients. We conclude that g is not primitive. Anyway we will work on primitive ones since if f is primitive then $\mathsf{Red}(f)$ is primitive because it is properly equivalent to f. Then, it is enough to add the condition "**if** $\gcd(a, b, c) \neq 1$ **abort**" at the beginning, or simply choosing a primitive f. However, notice that if the discriminant Δ is fundamental, all the forms of discriminant Δ are primitive (see Definition 2.1.6). Indeed, if $f = (a, b, c)$ is not primitive and Δ is fundamental, then $f = dg$, where $d = \gcd(a, b, c)$ and g is a form of a valid discriminant $\Delta' = \Delta/d^2$ reaching an absurd.

Looking at Algorithm 1, it seems that $\mathsf{Red}(f)$ computes c differently from applying T, however this is a little optimization which avoids squaring of s.[2]. $\mathsf{Red}(f)$ reaches an end with the correct output in time $O(\text{size}(f)^2)$. Algorithm 1 can be used also to test if two forms f and g are equivalent repeating it two times, one for each form, and verifying if

[2]Notice that $c \leftarrow c - \frac{1}{2}(b + r)s = c - \frac{1}{2}(b + b - 2as)s = c - bs + as^2$ as seen in the description of the reduction. The unique difference is that algorithm from [Coh00] sets $b = 2as + r$ instead of $b + 2as = r$, motivating the $-$ sign in bs. This is only a rewriting used to recall the original algorithm

$\mathsf{Red}(f) = \mathsf{Red}(g)$.

Algorithm 1: $\mathsf{Red}(f)$ – [Coh00], Algorithm 5.4.2

```
Input: A positive definite form $f = (a, b, c)$ of discriminant $\Delta = b^2 - 4ac < 0$
Output: The unique reduced form equivalent to $f$
/* Step 1                                                              */
if $-a < b \leq a$ then
    go to step 3;
/* Step 2                                                              */
Let $b = 2as + r$ with $0 \leq r < 2a$ be the Euclidean division of $b$ and $2a$;
if $r > a$ then
    Set $r \leftarrow r - 2a$;
    $s \leftarrow s + 1$;
Set $c \leftarrow c - \frac{1}{2}(b + r)s$;
Set $b \leftarrow r$;
/* Step 3                                                              */
if $a > c$ then
    Set $b \leftarrow -b$;
    Exchange $a$ and $c$;
    Go to step 2;
else if $a = c$ and $b < 0$ then
    Set $b \leftarrow -b$;
return $(a, b, c)$
```

Lemma 2.1.6. Let (a, b, c) is a positive definite quadratic form of discriminant $\Delta = b^2 - 4ac < 0$ such that $-a < b < a$ and $a < \sqrt{|\Delta|}$. Then either (a, b, c) is already reduced, or the form $f = (c, r, s)$ where $-b = 2cq + r$, $-c < r \leq c$ obtained by one reduction step of Algorithm 1 will be reduced.

Theorem 2.1.7. The deterministic algorithm to compute the class number described above, has running time $O(|\Delta|(\log(|\Delta|)^2)$

Proof. The number of a's and b's in the two cycles is $O(\sqrt{|\Delta|})$, while their binary lenght is $O(\log(|\Delta|)$. Since the number of arithmetic operations on (a, b) is $O(1)$, we conclude that the trivial deterministic algorithm proposed for computing the class number has running time $O(\sqrt{|\Delta|}^2 \cdot \log|\Delta|^2) = O(|\Delta|\log|\Delta|^2)$. □

Remark 3. In [BV07], Chapter 5, the authors give in Algorithm 5.1, 5.2 and 5.3 an extended reduction procedure. The main structure of these algorithms is the same we saw in Algorithm $\mathsf{Red}(f)$, with the difference that they also update the transformation matrix. As a final output they obtain $\mathsf{Red}(f)$ and the matrix P such that $\mathsf{Red}(f) = fP$. The total cost takes in account assignments of values to a 2×2 matrix of integers, then there is an increase in the space required. However, because of we can reassign values to the same locations, the additional space required is the size of 4 integers.

2.1.1.2 The composition of forms

Until now, we talked about operations on a single form – as in the case of the reduction algorithm – and about relations between forms of the same discriminant, i.e. the proper equivalence. We can do operations between forms too, indeed forms can be "multiplied" in a certain way. We are not using the term "multiplication" in the common sense of the component-wise product, but as it is used in group theory to indicate the operation defined in a group, since it is usual to call it *product* or multiplication. The correct term to indicate what we are calling multiplication of forms is *composition*, but we wanted to tell that the composition is the group operation. As we know a form is a degree 2 polynomial and the standard product between forms gives a polynomial of degree 4. The first property a product should satisfy is returning a valid form, and as a result neither the standard product or a classical composition of forms – in the sense of $f(g(x))$ where f and g are forms – are valid candidates. We introduce then the Gauss composition of forms (1798). The Gauss composition copes with primitive positive forms of the same discriminant Δ. We do not explain the work that brings to the composition formula, however the composition gives in output a normalized form, i.e. computed modulo the action Γ on the form f, where Γ is the set of all matrix T^s for each integer s and T is the special matrix of Subsubsection 2.1.1.1. Specifically, the Γ action on f is the $\Gamma-$orbit of f, i.e. the set

$$f\Gamma = \left\{ f\begin{pmatrix}1 & s\\ 0 & 1\end{pmatrix} : s \in \mathbf{Z}, f=(a,b,c)\right\} = \{(a, b+2as, c+bs+as^2) : s \in \mathbf{Z}, f=(a,b,c)\}.$$

Reducing modulo Γ means computing a normal form in $f\Gamma$, i.e. that one such that $-a < b+2as \leq a$. Obviously, if f is normal, then $s=0$. Furthermore, the normal form is unique in the $\Gamma-$orbit, since $s = \left[-\frac{b}{2a}\right]$ is the unique value which normalize the form $f=(a,b,c)$. To conclude, it is clear that reducing modulo Γ means computing $B = \bar{B} \mod 2\bar{a}$ in the range $-\bar{A} < B < \bar{A}$. Furthermore, with the substitution with the normalizing integer s, i.e. $B = \bar{B} + 2\bar{A}s$, we obtain $C = \frac{B^2-\Delta}{4A} = \bar{C} + \bar{B}s + \bar{A}s^2$ where $(\bar{A}, \bar{B}, \bar{C})$ its the output before reduction modulo Γ.

We will see how the normal form in the $\Gamma-$orbit is useful for the standard representation of an ideal in Subsection 2.1.3. The discussion about the $\Gamma-$orbit is necessary to understand the composition formula of forms, since it requires a reduction modulo Γ. We introduce below the composition of two forms:

Definition 2.1.4 ([Coh00]). Let $f=(a_1,b_1,c_1)$ and $g=(a_2,b_2,c_2)$ be two quadratic forms of the same discriminant Δ. Set $s=(b_1+b_2)/2$, $n=(b_1-b_2)/2$ and let u, v, w and d such that

$$ua_1 + va_2 + ws = d = \gcd(a_1, a_2, s)$$

and let $d_0 = \gcd(d, c_1, c_2, n)$. We define the composite of the two forms f and g as the form

$$(A,B,C) = \left(d_0\frac{a_1a_2}{d^2},\ b_2 + \frac{2a_2}{d}(v(s-b_2)-wc_2),\ \frac{B^2-\Delta}{4A}\right).$$

modulo the action of $\Gamma = \left\{\begin{pmatrix}1 & \tau\\ 0 & 1\end{pmatrix}, \tau \in \mathbf{Z}\right\}$.

If we work on classes then with reduced forms, we have to reduce the product using the reduction algorithm since a normalization is not enough to obtain a reduced form. If the product is not reduced, its class is the same, but working with representatives of classes is better because we use a unique representative of the class. Indeed, the composition can be applied both to classes of forms and to single forms. Then, multiplying classes $[f]$ and $[g]$ is equivalent to multiply f and g and reducing the product.

From a computational point of view, computing the composition of two form can be done efficiently. Indeed, Algorithm 2 does the task in $O(\log(|\Delta|)^2)$.

Algorithm 2: Comp(f, g) – [Coh00], Algorithm 5.4.7

```
Input: Two primitive positive definite quadratic forms $f = (a_1, b_1, c_1)$ and
       $g = (a_2, b_2, c_2)$ with the same discriminant $\Delta < 0$
Output: The composition $F = (A, B, C)$ of $f$ and $g$
/* Step 1: Initialize                                                  */
if $a_1 > a_2$ then
    Exchange $f$ and $g$;
    Set $s \leftarrow \frac{1}{2}(b_1 + b_2)$, $n \leftarrow b_2 - s$;
/* Step2: First Euclidean Step                                         */
if $a_1 | a_2$ then
    Set $y_1 \leftarrow 0$ and $d \leftarrow a_1$;
else
    Compute $(u, v, d)$ such that $ua_2 + va_1 = d = \gcd(a_2, a_1)$ (using Euclid's
      extended algorithm);
    Set $y_1 \leftarrow u$;
/* Step 3: Second Euclidean Step                                       */
if $d | s$ then
    Set $y_2 \leftarrow -1$, $x_2 \leftarrow 0$ and $d_1 \leftarrow d$;
else
    Compute $(u, v, d_1)$ such that $us + vd = d_1 = gcd(s, d)$ (using Euclid's
      extended algorithm);
    Set $x_2 \leftarrow u, y_2 \leftarrow -v$;
/* Step 4: Compose                                                     */
Set $v_1 \leftarrow a_1/d_1$, $v_2 \leftarrow a_2/d_1$, $r \leftarrow (y_1 y_2 n - x_2 c_2 \mod v_1)$, $B \leftarrow b_2 + 2v_2 r$,
  $A = v_1 v_2$, $C \leftarrow (c_2 d_1 + r(b_2 + v_2 r))/v_1$ (or $C \leftarrow \frac{B^2-\Delta}{4A}$);
Compute $F \leftarrow$ Red$(A, B, C)$ using Algorithm 1;
return $F$
```

From the efficiency side, Atkin (cf. [Atk90]) proposed two optimized version of Shanks' NUCOMP and NUDUPL, which are two optimized algorithm to do operations with forms. The composition is done by the algorithm NUCOMP, while the second algorithm, NUDUPL, is used to compute squares. The application of these two algorithms can be seen as a Square and Multiply version with forms.

2.1.1.3 The Form Class Group

The Form Class Group represents the first step to make easier the understanding of the structure of the ideal class group in the Subsection 2.1.3. We will see that they are strongly linked together, since an isomorphism between them exists and it is possible to translate easy operations on forms to ideals. Previously, we defined classes of forms from the proper equivalence relation and in the previous subsubsection we defined a product/composition operation for primitive positive definite forms in the case they have the same discriminant. As explained at the beginning of this chapter, they are the necessary ingredients for defining a class group. The remaining property for defining a group is the existence of a neutral element. For any discriminant Δ and the set of positive definite forms of discriminant $\Delta < 0$, the role of the neutral element is covered by the class of the *principal* form.

Definition 2.1.5 (Principal form). Let Δ be a discriminant. The *principal form* of discriminant Δ is $f = (1, 0, -\frac{\Delta}{4})$ if $\Delta \equiv 0 \mod 4$ and $f = (1, 1, \frac{1-\Delta}{4})$ if $\Delta \equiv 1 \mod 4$.

[Cox14] analyzes a subcase of the composition of forms, i.e. the case when $d = \gcd(a, a', \frac{b+b'}{2}) = 1$. Indeed, when composing two forms $f = (a, b, c)$ and $g = (a', b', c')$ the results is a form $F = d(A, B, C)$, which is clearly not primitive if $d \neq 1$, and the discriminant of F is not the same as for f and g. However, Definition 2.1.4 takes in account this fact, using d as a divisor of the result. We conclude that the next theorem from [Cox14] which defines the form class group from the restricted composition with $d = 1$ it is also valid extending to $d \neq 1$ using the composition in Definition 2.1.4.

Theorem 2.1.8 ([Cox14], Theorem 3.9)**.** Let $\Delta < 0$ be a discriminant, i.e. $\Delta \equiv 0, 1 \mod 4$ and let $Cl(\Delta)$ be the set of classes of primitive positive definite forms of discriminant Δ. Then composition induces a well-defined binary operation on $Cl(\Delta)$ which makes $Cl(\Delta)$ into a finite abelian group whose order is the class number $h(\Delta)$. The identity element of $Cl(\Delta)$ is the class of the principal form of discriminant Δ, while the inverse of the class of $f = (a, b, c)$ is the class of $f = (a, -b, c)$.

Last theorem concludes the necessary background on binary quadratic forms. We recall main results of thi section to start talking about Ideals in Quadratic Fields. We defined binary quadratic forms and we focused on positive definite ones. After that we introduced the proper equivalence relation between forms and we divided forms in classes with respect to this equivalence relation. Then, we saw that each class is represented by a reduced form – which is unique for each class – and we explained how to compute it from using the reduction algorithm. Furthermore, we saw that the class number is the number of primitive reduced forms of a fixed discriminant. However, the deterministic algorithm proposed in paragraph **Computing the class number** is not efficient[3]. Once we did a partition of the set of positive definite forms, we presented the composition operation which sends two primitive positive definite forms of discriminant Δ to a primitive positive definite form of discriminant Δ. The reduction of the product of any two forms f and g is equivalent to the reduction of the product of the two reduced forms in the classes of f and g, then the composition of forms can be seen as a composition of classes. Finally, there is always a class represented by the principal form (which is the reduced form of its class) that it is the neutral element class. To sum up, $Cl(\Delta)$ is the Abelian class group of primitive positive definite forms of discriminant Δ and neutral element of the class is

[3]When talking about the class number of the ideal class group in imaginary quadratic fields, which is equal to the class number of the form class group for negative discriminant, we will talk about the best known solutions for computing the discrete logarithm. Computing discrete logarithms gives information about the class number.

the principal form. In next subsections we will introduce the ideal class group $Cl(\mathcal{O}_\Delta)$ of discriminant Δ and we will analyze the relation between $Cl(\Delta)$ and $Cl(\mathcal{O}_\Delta)$ for the case of $\Delta < 0$. To conclude, we present the last definition of the chapter, i.e. the *conductor* of a discriminant. The conductor has a fundamental role in the theory of ideals in a imaginary quadratic field in Subsection 2.1.2.

Conductor of a discriminant Not all the discriminant are the same. In this paragraph we introduce the notion of conductor of discriminant. The conductor is of particular relevance in the context of maximal and non maximal orders. Here, we give only the definition and its importance will be clear in Subsection 2.1.2.

Definition 2.1.6 (Conductor). The *conductor* $f(\Delta)$ of a discriminant Δ is the largest positive integer f such that $\Delta' = \Delta/f(\Delta)^2$ is a discriminant, i.e. $\Delta' = 0, 1 \mod 4$. A discriminant Δ is called *fundamental* if $f(\Delta) = 1$.

An easy characterization of fundamental discriminants is given in the following proposition.

Proposition 2.1.9. A integer discriminant Δ is fundamental if and only if

- $\Delta \equiv 1 \mod 4$ and Δ is square free, or
- $\Delta \equiv 0 \mod 4$, $\Delta/4 \equiv 2, 3 \mod 4$ and $\Delta/4$ is square free

Proof. Suppose that $\Delta \equiv 1 \mod 4$ ad Δ is not square free. Notice that a prime p which divides Δ is odd. Therefore, $p^2 \equiv 1 \mod 4$ and $\Delta/p^2 \equiv \Delta \mod 4 \equiv 1 \mod 4$. Let $t = p_1^{2\alpha_1} \cdot \ldots \cdot p_k^{2\alpha_k}$ the maximal square which divides Δ, then $\Delta' = \Delta/t \equiv 1 \mod 4$. Δ' is a valid disciminant and as a result Δ is not fundamental, which brings to an absurd. On the other side, if Δ is square-free, there is not an integer f such that $f^2|\Delta$, i.e. Δ is fundamental. For the case $\Delta \equiv 0 \mod 4$, we work on $\Delta/4$. Let Δ be fundamental, then $\Delta/4 \not\equiv 0, 1 \mod 4$. With the reasoning in the previous case and using the same notation, $(\Delta/4)/t$ is square-free. On the other side, there is not a square which divides $\Delta/4$. □

2.1.2 Imaginary Quadratic Fields, Orders and Fractional $\mathcal{O}-$Ideals

In this subsection, we introduce *imaginary quadratic fields* and their unitary subrings called *orders*. After that, we introduce *$\mathcal{O}-$ideals*, where $\mathcal{O}$ is an order. $\mathcal{O}-$ideals are the elements of the ideal class group we will work with in the next section and that is at the basis of the instantiations in the next chapters. We start giving some algebraic definition and then we present the required notions. Links with the theory of binary quadratic forms will become more clear after reading this part.

Definition 2.1.7. A field extension is a pair of fields $K \subseteq L$, such that K inherits the operations in L. The extension is indicated with the notion L/K. L may be seen as a $K-$vector space. The *degree* of the field extension is the dimension of L as $K-$vector space, and it is denoted by $[L : K]$.

Example 2. $\mathbf{C}/\mathbf{R}$ is a field extension and $[\mathbf{C} : \mathbf{R}] = 2$. Indeed, $\mathbf{C} = \mathbf{R} + i\mathbf{R}$ and $(1, i)$ is a basis of $\mathbf{C}$ over $\mathbf{R}$.

Definition 2.1.8 (Number Fields). A number field K is a subfield of $\mathbf{C}$ which has finite degree over $\mathbf{Q}$. The degree of K over $\mathbf{Q}$ is denoted $[K : \mathbf{Q}]$.

In particular, we are interested in a subfamily of number fields, i.e. *quadratic fields.*

Definition 2.1.9 (Quadratic Fields). A *quadratic field* K is an algebraic number field of degree two over $\mathbf{Q}$, i.e. $[K : \mathbf{Q}] = 2$.

Example 3. $\mathbf{Q}(i) = \mathbf{Q}+i\mathbf{Q} = \{a+bi|a, b \in \mathbf{Q}\}$, where i is the imaginary unit, is a number field. It strictly includes $\mathbf{Q}$ and $[\mathbf{Q}(i) : \mathbf{Q}] = 2$, indeed $i \in \mathbf{Q}(i)$ and then $(1, i)$ is a $\mathbf{Q}-$basis over $\mathbf{Q}(i)$. Actually, $\mathbf{Q}(i)$ is also a quadratic number field.

If K is a quadratic field, it can be written in a unique way as $K = \mathbf{Q}(\sqrt{N}) = \mathbf{Q}+\sqrt{N}\mathbf{Q}$ where N is squarefree integer and $N \neq 0, 1$. Actually, let N not be a squarefree integer, then $N = f^2M$, where f, M are integers, M squarefree. Then, $\mathbf{Q}+\sqrt{N}\mathbf{Q} = \mathbf{Q}+f\sqrt{M}\mathbf{Q} \subset \mathbf{Q}+\sqrt{M}\mathbf{Q}$ and the inverse direction is given by $\sqrt{M} = 1/f\sqrt{f^2M} \in \sqrt{N}\mathbf{Q}$. Therefore N is assumed squarefree. This little remark motivates the definition of the maximal order we will see later.

2.1.2.1 Maximal and non-maximal orders

Definition 2.1.10 ($R-$module). Let R be a ring. A left $R-$module is a commutative group $(\mathbb{G}, +)$ equipped with a scalar multiplication $R \times \mathbb{G} \to \mathbb{G}$, $(r, g) \to r \cdot g$ such that for $r, s \in R$ and $a, b \in \mathbb{G}$,

- $r \cdot (a + b) = r \cdot a + r \cdot b$
- $(r + s) \cdot a = r \cdot a + s \cdot a$
- $r \cdot (s \cdot a) = (r \cdot s) \cdot a$

In our context we consider $R = \mathbf{Z}$, and the "$\cdot$" operation is commutative, so we do not say the sets we consider are left or right $\mathbf{Z}-$modules, but simply $\mathbf{Z}-$modules. With the natural correspondence $(z, a) \to z \cdot a$ for $z \in \mathbf{Z}$ and $a \in \mathbb{G}$, where $(\mathbb{G}, +)$ is a group, it is clear that a commutative group is a $\mathbf{Z}-$module. Some $R-$modules can be written from a set of generators, for example we saw in the examples at the beginning of this subsection that $\mathbf{C} = \mathbf{R} + i\mathbf{R}$. $\mathbf{C}$ is then generated by elements 1 and i with coefficients in $\mathbf{R}$. A $R-$module $\mathbb{G}$ is said *finitely generated* if there exists a finite set of elements $g_1, \ldots, g_t$ such that $\mathbb{G} = g_1R + \ldots g_tR$, i.e. if for each element $g \in \mathbb{G}$ there exists a tuple $(r_1, \ldots, r_t) \in R^t$ such that $g = \sum_{i=1}^{t} r_i \cdot g_i$. If the generators are linearly independent, i.e. if $\sum_{i=1}^{t} r_i \cdot g_i = 0$ implies $r_1 = \ldots = r_t = 0$, the $R-$module is said *free* and the number of generators is its *rank*. As one can notice, a free $R-$module is quite similar to a vector space. Actually, the main difference between them is that R is a ring and not a field as in the definition of a vector space. If we consider $R = \mathbf{Z}$, we can notice that a free $\mathbf{Z}-$module has a lattice structure. Indeed, typically a lattice is defined as a set $L = \mathbf{v}_1\mathbf{Z} + \ldots + \mathbf{v}_n\mathbf{Z}$, where $\{\mathbf{v}_i\}_{i\in[n]}$ are vectors in $\mathbf{R}^n$ (cf. [GM02]). However, we will consider the components of $\mathbf{v}$ in a subfield $K \subset \mathbf{C}$.
Now, we have the necessary tools to define orders in a quadratic field, and in particular in an imaginary one. Next step is defining orders. Orders are necessary to define $\mathcal{O}-$ideals, which are the building block of ideal class groups and then of particular relevance in building cryptographic instantiations from Class Groups of Imaginary Quadratic Fields.

As pointed at the beginning of this chapter, we will see in the Section 2.2 how these tools were used by Castagnos-Laguillaumie to build a linearly homomorphic encryption scheme.

2.1.2.2 Representation and inclusions of orders

Any quadratic field K can be written in a unique way as $K = \mathbf{Q}(\sqrt{N}) = \mathbf{Q} + \sqrt{N}\mathbf{Q}$. [4] Consider $N \in \mathbf{Z}$, $N \neq 0, 1$ and N is a squarefree integer. We denote the discriminant Δ_K of K as $\Delta_K := N$ if $N \equiv 1 \mod 4$ or $\Delta_K := 4N$, otherwise. Notice that from Proposition 2.1.9, the conditions on the choice of Δ_K imply that Δ_K is a fundamental discriminant.

Definition 2.1.11. An *order* $\mathcal{O}$ in a quadratic field K is a unitary subring of K which is a free $\mathbf{Z}-$module of rank 2.

The ring of integers of K, i.e. the set of elements in K – called the algebraic integers – that are roots of some monic polynomial in $\mathbf{Z}[x]$ is a special order $\mathcal{O}_{\Delta_K}$ in K. $\mathcal{O}_{\Delta_K}$ is the unique maximal order in K and it is represented as

$$\mathcal{O}_{\Delta_K} = \begin{cases} \mathbf{Z} + \frac{1+\sqrt{N}}{2}\mathbf{Z} & \text{if} N \equiv 1 \bmod 4 \\ \mathbf{Z} + \sqrt{N}\mathbf{Z} & \text{if} N \not\equiv 1 \bmod 4 \end{cases}$$

where N is linked to the discriminant Δ_K of K as above. However, we can choose an integer from the first component $\mathbf{Z}$ of $\mathcal{O}_{\Delta_K}$ such that $z + \Delta_K \bmod 2 \in \{0, 1\}$ concluding that both cases can be written in a unique expression as $\mathcal{O}_{\Delta_K} = \mathbf{Z} + \frac{\Delta_K + \sqrt{\Delta_K}}{2}\mathbf{Z}$. We do not prove here how the structure of $\mathcal{O}_{\Delta_K}$ is obtained neither the fact it is the unique maximal order (see [BV07], Theorem 8.2.5 for the proof). A final observation is that since $\sqrt{\Delta_K}$ is an imaginary number, it is clear that $\mathcal{O}_{\Delta_K}$ is a free $\mathbf{Z}-$module of rank 2 generated by 1 and $\omega_K = \frac{\Delta_K + \sqrt{\Delta_K}}{2}$. In general, given a quadratic field K, orders can be classified in maximal and non-maximal and the difference depends on the conductor of the discriminant.

Theorem 2.1.10 ([BV07] – Theorem 7.2.2 readapted)**.** The quadratic orders are exactly the two dimensional $\mathbf{Z}-$modules of the form $\mathcal{O} = a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z}$, where $f = (a, b, c)$ is a principal form of some discriminant Δ.

Definition 2.1.12. An order is called *maximal* if its discriminant is fundamental, i.e. its conductor $f(\Delta) = 1$. Otherwise, it is called *non-maximal.*

Theorem 2.1.10 implies that for quadratic orders, $a = 1$ and $b = \Delta \mod 2$ and that for each discriminant there is only one quadratic order. Indeed, for each discriminant there is only one principal form of that discriminant. This implies that $\mathcal{O} = \mathbf{Z} + \frac{\Delta \mod 2 + \sqrt{\Delta}}{2}\mathbf{Z}$ which is equal to $\mathcal{O} = \mathbf{Z} + \frac{\Delta+\sqrt{\Delta}}{2}\mathbf{Z}$. Then all the orders have the same representation of maximal ones, as in the definition of $\mathcal{O}_{\Delta_K}$.
We gave a correspondence between primitive forms and orders. In particular the discriminant $\Delta(\mathcal{O})$ of an order $\mathcal{O}$ is $\Delta(\mathcal{O}) = \Delta(f)$, where f is the principal quadratic form that

[4]This is the notation to denote the smallest field K which has $\mathbf{Q}$ as a subfield and that contains $\sqrt{N}$ as element.

represents $\mathcal{O}$ in Theorem 2.1.10. We see here why fundamental discriminants represent maximal orders, i.e. why that orders are maximal in a chain of inclusion. Proposition 2.1.11 below proves that the standard inclusion of orders depends only on their discriminant. From the factorization of the discriminant it is possible to identify all the maximal orders and their suborders.

Proposition 2.1.11 ([BV07], Lemma 7.2.6 + Proposition 7.2.7)**.** A quadratic order $\mathcal{O}'$ is contained in an other quadratic order $\mathcal{O}$ if and only if $\Delta(\mathcal{O}') = d^2\Delta(\mathcal{O})$ for some integer d, i.e. if $\Delta(\mathcal{O}')/\Delta(\mathcal{O})$ is the square of a integer.

Proof. Set $\Delta := \Delta(\mathcal{O})$ and $\Delta' := \Delta(\mathcal{O}')$. If $\Delta' = d^2\Delta$ for some integer d, then

$$\begin{aligned}\mathcal{O}' := \mathcal{O}_{\Delta'} = \mathcal{O}_{d^2\Delta} &= \mathbf{Z} + \frac{d^2\Delta + \sqrt{d^2\Delta}}{2}\mathbf{Z} = \mathbf{Z} + \frac{(d^2\Delta - d\Delta) + d\Delta + \sqrt{d^2\Delta}}{2}\mathbf{Z} \\ &= \mathbf{Z} + \frac{d\Delta + d\sqrt{\Delta}}{2}\mathbf{Z} = \mathbf{Z} + d\frac{\Delta + \sqrt{\Delta}}{2} \subset \mathcal{O},\end{aligned}$$

since $(d^2\Delta - d\Delta)/2 \in \mathbf{Z}$, and the generators of $\mathcal{O}'$, i.e. $1, d\frac{\Delta+\sqrt{\Delta}}{2}$ belong to $\mathcal{O}$. For the other direction, $\frac{\Delta'+\sqrt{\Delta'}}{2} \in \mathcal{O}' \subset \mathcal{O}$, i.e. $\frac{\Delta'+\sqrt{\Delta'}}{2} \in \mathcal{O}$. Then there exist $x, y \in \mathbf{Z}$ such that $\frac{\Delta'+\sqrt{\Delta'}}{2} = x + y\frac{\Delta+\sqrt{\Delta}}{2}$. This implies $x + y\Delta = \Delta'$ and $\sqrt{\Delta'} = y\sqrt{\Delta}$. The solution of the system is $(x, y) = \left(\frac{\Delta'-y\Delta}{2}, \frac{\sqrt{\Delta'}}{\sqrt{\Delta}}\right)$, which implies $\Delta' = y^2\Delta$ proving the second part[5]. $\square$

2.1.2.3 Fractional $\mathcal{O}-$ideals

A class group of an imaginary quadratic field consists of classes of $\mathcal{O}-$*ideals*. In this subsection we introduce the notion of $\mathcal{O}-$ideal and its properties. Then, we will analyze the link between $\mathcal{O}-$ideals and quadratic forms looking at how we can use algorithms we saw for quadratic forms, as composition and reduction, to compute the product and reduction of $\mathcal{O}-$ideals. The correspondence between operations on forms and ideals makes easy to understand how to work with ideals and the efficiency of operations with forms is translated in the efficiency of the operations with ideals. In the next definition, the notion of multiplication is general, but we will inherit the multiplication from binary quadratic forms theory.

Definition 2.1.13. Let K be a quadratic number field and $\mathcal{O} \subset K$ an order of discriminant Δ.

- An $\mathcal{O}-$*ideal* is an additive subgroup of $\mathcal{O}$ which is an $\mathcal{O}-$module with respect to multiplication.
- An $\mathcal{O}-$ideal $\mathfrak{a}$ is called *primitive* if does not exist $m \in \mathbf{N}$ and another $\mathcal{O}-$ideal $\mathfrak{b}$ such that $\mathfrak{a} = m\mathfrak{b}$.
- A subset $\mathfrak{b} \subset K$ is called a *fractional* $\mathcal{O}-$ideal if $d\mathfrak{b}$ is an $\mathcal{O}-$ideal for some positive $d \in \mathbf{N}$.
- An $\mathcal{O}-$ideal $\mathfrak{a}$ is called *principal* if $\mathfrak{a} = \alpha\mathcal{O}$ for some $\alpha \in K^*$. α is called a *generator*.

[5]Notice that the proof of Proposition 2.1.11 is valid both for positive and negative discriminants. The unique observation to add is that Δ and Δ' has the same sign.

Standard representation of fractional $\mathcal{O}$-ideals Fractional $\mathcal{O}-$ideals have a unique standard representation which permits to identify them with few elements. Of particular interest are the *reduced* $\mathcal{O}-$ideals. The notions of reduced ideals will be given soon, but for now it is enough to know that they are one of the most important type of ideals we will work with and that they are linked to reduced forms. Indeed, we saw that in each form class there is a a reduced form. As we will see the same is valid for classes of ideals and the correspondence between forms and ideals permits us to represent reduced ideals with two integers (a, b) smaller than $\sqrt{|\Delta|}$ using that standard representation. The next two propositions give the standard representation we require.

Proposition 2.1.12 ([BV07], Proposition 8.4.4)**.** Let $\mathfrak{b} \subset K$, then $\mathfrak{b}$ is a fractional $\mathcal{O}$-ideal if and only if $\mathfrak{b} = q\left(a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\right)$ for some $q \in \mathbf{Q}^+$ and where (a, b) are coefficient of an integral form (a, b, c) of discriminant Δ (where c is not relevant since uniquely determined by a,b and Δ.)

Proposition 2.1.13 ([BV07], Proposition 8.4.5)**.** Let $\mathfrak{b} = q\left(a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\right)$ a fractional $\mathcal{O}$-ideal . Then the value of q is uniquely determined and $f = (a, b, c)$ is an integer form which is unique modulo Γ. $\mathfrak{b}$ is integral if and only if $q \in \mathbf{N}$ and $\mathfrak{b}$ is integral primitive if and only if $q = 1$.

Looking at Proposition 2.1.12 and 2.1.13, we can see that a primitive reduced $\mathcal{O}-$ideal can be denoted by (a, b) where $0 < a \leq \sqrt{|\Delta|/3}$ and $-a < b \leq a$. This implies that the representation of a reduced ideals require two integers of the same size of $\sqrt{|\Delta|}$, i.e $O(\log(\sqrt{|\Delta|})$.

Even if we do not have yet defined a correspondence between forms and ideals, we will see in Subsection 2.1.3 that it is easier to define an isomorphism between the form class and the ideal class. However, from previous propositions we know a link between forms and representations of ideals.

Remark 4. In Subsection 2.1.1 we defined how Γ acts on a form f as $f\Gamma = \{(a, b + 2as, c + bs + as^2) : f = (a, b, c), s \in \mathbf{Z}\}$, then reducing f mod Γ consists in sending $g = (a', b', c') = (a, b', c') \in f\Gamma$ in $(\bar{a}, \bar{b}, \bar{c}) = (a, b \mod_c 2a, \bar{c})$, where the remainder $b \bmod a$ is centered, i.e. $-a < \bar{b} \leq a$, and $\bar{c} = \frac{\bar{b}^2-\Delta}{4\bar{a}}$. For example, consider the form $f = (4, 5, 6)$ for $\Delta = -71$. The standard representation of $\mathfrak{a}$ from f is not $\mathfrak{a} = 4\mathbf{Z} + \frac{-5+\sqrt{-71}}{2}\mathbf{Z}$, but $\mathfrak{a} = 4\mathbf{Z} + \frac{-5 \mod_c 8+\sqrt{-71}}{2}\mathbf{Z} = 4\mathbf{Z} + \frac{3+\sqrt{-71}}{2}\mathbf{Z}$, i.e. where b is the unique value $\mod 2a$ such that $-a < b \leq a$. However, there is no problem in using a not normal form to represent an ideal, even if the standard form is unique. Indeed, $a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z} = a\mathbf{Z} + \frac{-(b+2as)+\sqrt{\Delta}}{2}\mathbf{Z}$ for each $s \in \mathbf{Z}$ [6] and $f = (a, b, c)$ is equivalent to $g = (a, b + 2as, \bar{c})$ for a uniquely determined $\bar{c}$. This is perfectly in line with the correspondence between standard representations and $\Gamma-$orbits of forms.

From Proposition 2.1.13, we understand that above representations (a, b) of integral primitive $\mathcal{O}-$ideals are in biunivocal correspondence with $\Gamma-$orbits $f\Gamma = \{(a, b + 2a\tau, c + b\tau + a\tau^2) : f = (a, b, c), \tau \in \mathbf{Z}\}$ of integral forms $f = (a, b, c)$ of discriminant Δ, i.e. the standard representation of $\mathfrak{a}$ is defined by the unique *normal* form – i.e.

[6]Notice that $2as/2 = as \in a\mathbf{Z}$

that one with $-a < b < a$ – in $f\Gamma$. From this reasoning, the *standard* representation of $\mathfrak{a}$ is given by the normal form $f_{\mathfrak{a}} = (a, b, c)$ linked to $\mathfrak{a}$. From now, we will use the notation $f_{\mathfrak{a}}$ to indicate the normal form in the standard representation of $\mathfrak{a}$. From previous proposition we know the unique representation modulo Γ of a fractional $\mathcal{O}-$ideal. i.e. the standard representation. To conclude the set of representations of $\mathcal{O}-$ideals, we see that at first sight principal $\mathcal{O}-$ideals does not satisfy the form of a standard representation since they are of the form $\alpha\mathcal{O}$ for $\alpha \in K^*$, and not simply in $\mathbf{Q}$. Actually, a principal $\mathcal{O}-$ideal is a fractional one and then it should can be rewritten in the standard representation, too. Indeed, if $\alpha \in K^*$ then $\alpha = \frac{m}{n} + \frac{s}{t}\sqrt{\Delta}$, for some integers $m, n, s, t \in \mathbf{Z}$. Since $(1, \frac{\Delta+\sqrt{\Delta}}{2})$ is a basis of $\mathcal{O}$, then $(\alpha, \alpha \cdot \frac{\Delta+\sqrt{\Delta}}{2})$ is a basis of $\alpha\mathcal{O}$. Rewriting we have

$$\begin{cases} \alpha & = \frac{2mt+2ns\sqrt{\Delta}}{2nt} \\ \alpha \cdot \frac{\Delta+\sqrt{\Delta}}{2} & = \frac{(mt+ns)\Delta+(mt+ns\Delta)\sqrt{\Delta}}{2nt} \end{cases}$$

and applying the following proposition we obtain a standard representation for principal ideals. Actually, since we can see fractional $\mathcal{O}$-ideals as two-dimensional lattices, they can be also written in a lattice form as $\alpha_1\mathbf{Z} + \alpha_2\mathbf{Z}$, where $\alpha_1, \alpha_2 \in K$. Fortunately, the next proposition permits to transform a lattice representation of a fractional $\mathcal{O}$-ideal in a standard one.

Proposition 2.1.14 ([BV07], Proposition 8.4.8)**.** Let $\mathfrak{b}$ a fractional $\mathcal{O}$-ideal and let (α_1, α_2) be a $\mathbf{Z}-$basis of $\mathfrak{b}$. Write $\alpha_i = \frac{x_i+y_i\sqrt{\Delta}}{2d}$, $x_i, y_i \in \mathbf{Z}$, $i = 1, 2$ and $d \in \mathbf{Z}^+$. Let $\tau = \gcd(y_1, y_2) = u_1y_1 + u_2y_2$, $u_1, u_2 \in \mathbf{Z}$ (from Extended Euclid Algorithm). Then $\mathfrak{b} = q\left(a\mathbf{Z} + \frac{b+\sqrt{\Delta}}{2}\right)$ where

$$a = \left|\frac{y_2x_1 - y_1x_2}{2\tau^2}\right|, \qquad b = \frac{u_1x_1 + u_2x_2}{\tau}, \qquad c = \frac{b^2 - \Delta}{4a}, \qquad q = \frac{\tau}{d}$$

.

Applying Proposition 2.1.14 to the fractional $\mathcal{O}$-ideal $\alpha\mathcal{O} = \alpha\mathbf{Z} + \alpha\frac{\Delta+\sqrt{\Delta}}{2}\mathbf{Z}$, i.e. choosing $x_1 = 2mt$, $y_1 = 2ns$, $x_2 = (mt+ns)\Delta$, $y_2 = mt+ns\Delta$ and $d = nt$ we obtain the standard representation of $\alpha\mathcal{O}$ from a and b formulas.

2.1.2.4 Product of $\mathcal{O}-$ideals

The product between ideals of the same discriminant Δ works essentially as for binary quadratic forms. We saw that algorithms on forms are efficient, then a translation of them in the context of ideals permits to do operation on them efficiently. Indeed, the product of two forms (a_1, b_1, c_1) and (a_2, b_2, c_2) of discriminant Δ gives the form $(a_1a_2, B, \frac{B^2-\Delta}{4a_1a_2})$ where the value of B can be computed from the following lemma, which is the composition formula version for ideals.

Lemma 2.1.15 ([Coh00], Lemma 5.4.5)**.** Let $I_1 = a_1\mathbf{Z} + \frac{-b_1+\sqrt{\Delta}}{2}$ and $I_2 = a_2\mathbf{Z} + \frac{-b_2+\sqrt{\Delta}}{2}$ be two ideals, set $s = (b_1 + b_2)/2$, $d = \gcd(a_1, a_2, s)$, and let u, v, w be integers such that $ua_1 + va_2 + ws = d$. Then we have

$$I_1 \cdot I_2 = d\left(A\mathbf{Z} + \frac{-B+\sqrt{\Delta}}{2}\mathbf{Z}\right)$$

where
$$A = d_0 \frac{a_1 a_2}{d^2}, \qquad B = b_2 + \frac{2a_2}{d}(v(s-b_2) - wc_2)$$
and $d_0 = 1$ if at least one of the forms (a_1, b_1, c_1) or (a_2, b_2, c_2) is primitive and in general $d_0 = \gcd(a_1, a_2, s, c_1, c_2, n)$ where $n = (b_1 - b_2)/2$.

Invertible $\mathcal{O}-$ideals We saw that fractional $\mathcal{O}$-ideals can be multiplied. This product is at the basis of the definition of a class group. Anyway, we need a group structure and the set of fractional $\mathcal{O}$-ideals of a given discriminant Δ with the product operation is not enough because it defines only a semigroup, not a group[7]. Indeed, the product is closed, commutative and associative, but it is not true that all the elements are *invertible*. The meaning of invertible is the usual one, i.e. a fractional $\mathcal{O}$-ideal $\mathfrak{a}$ is invertible if exists a fractional $\mathcal{O}$-ideal $\mathfrak{b}$ such that $\mathfrak{ab} = \mathcal{O}$, where $\mathcal{O}$ has the role of the neutral element. It is clear that if we restrict the set of fractional $\mathcal{O}$-ideals to its subset $I(\mathcal{O})$ of invertible ones, the structure of group is achieved. Indeed, the product is still closed because if $\mathfrak{a}, \mathfrak{b} \in I(\mathcal{O})$, then $\mathfrak{ab}$ is invertible with inverse $\mathfrak{b}^{-1}\mathfrak{a}^{-1} = \mathfrak{a}^{-1}\mathfrak{b}^{-1}$. Notice that the group $P(\mathcal{O})$ of principal $\mathcal{O}-$ideals is contained in $I(\mathcal{O})$ and the inverse of some $\alpha\mathcal{O}$ is trivially $\frac{1}{\alpha}\mathcal{O}$.

Remark 5 (On the inversion of $\alpha \in K^*$). If $\alpha \in K$, then $\alpha = \lambda + \mu\sqrt{N}$, where $\lambda, \mu \in \mathbf{Q}$. It is easy to check that $\alpha^{-1} = -\frac{\lambda^2}{\mu^2 N - \lambda^2} + \frac{\mu}{\mu^2 N - \lambda^2}\sqrt{N}$. Remember that K is a field and then each element except 0 is invertible, then there are no problem with the denominator of α^{-1}. However, this is a consequence of a check that we can do here: an element cannot be invertible if it is 0 or if $\mu^2 N - \lambda^2 = 0$, i.e. if $N = \frac{\lambda^2}{\mu^2}$. However, this implies that N is the square of the rational number $\frac{\lambda}{\mu}$. Since N is an integer, the equality is valid if $\frac{\lambda}{\mu} \in \mathbf{Z}$ and if N is a square. But, N is not a square by definition of K. Therefore, any element except 0 is invertible in K (as expected since K is a field), and we saw hot to compute its inverse.

The classic definition of invertible ideals does not give the standard representation that permits to distinguish them from non invertible ones. For this, the following proposition helps us:

Proposition 2.1.16 ([Coh00], Proposition 5.2.5)**.** Let $\mathfrak{a} = a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z}$ be a fractional $\mathcal{O}-$ideal, and let $f = (a, b, c)$ be the corresponding quadratic form of discriminant Δ. Then $\mathfrak{a}$ is invertible if and only if f is primitive. In that case, we have $\mathfrak{a}^{-1} = \mathbf{Z} + \frac{b+\sqrt{\Delta}}{2a}\mathbf{Z} = \frac{1}{a}\left(a\mathbf{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbf{Z}\right)$.

Notice that each invertible $\mathcal{O}-$ideal is trivially fractional, but in general the inverse is not true. Since the ideal class group works on classes of invertible ideals, it is not relevant knowing when the two definition coincide, even if the proof is very easy. However, to prove the equivalence is necessary to introduce the notion of *ring of multipliers* that can be avoided to do not burden the discussion. For completeness, we recall only a corollary of what we saw for fractional $\mathcal{O}-$ideals.

Corollary 2.1.16.1. If $\mathcal{O}$ is a maximal order, then all fractional $\mathcal{O}$-ideals are invertible.

[7]The set of *integral* $\mathcal{O}-$ideals also forms a semigroup with the product.

2.1.3 The Ideal Class Group

Finally, we have all the tools to understand Class Groups of Imaginary Quadratic Fields. What we need is again an equivalence relation and an operation. We saw in Subsubsection 2.1.2.4 how to compute the product of two ideals, so we have an operation. It remains to define elements of the classes and an equivalence relation. Given an order $\mathcal{O}$ of discriminant Δ, recall $I(\mathcal{O})$ the group of all invertible $\mathcal{O}-$ideals and let $P(\mathcal{O})$ be the group of all principal $\mathcal{O}-$ideals. Furthermore, sometimes ideals will be denote by their standard representation $\mathfrak{a} = a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z}$ or from their base $\mathfrak{a} = [\alpha_1, \alpha_2]$ as $\mathbf{Z}-$module and sometimes classes of ideals are denoted by their representative forms as $[\mathfrak{a}] = [a, b, c]$ depending on which characteristics we are focus on.

Reduction of $\mathcal{O}-$ideals The characterization of reduced $\mathcal{O}-$ideals is almost the same as in the case of binary forms. Indeed, from previous sections we know that the standard representation of a fractional $\mathcal{O}-$ideal $\mathfrak{a}$ is in biunivocal correspondence with the normal form $f_{\mathfrak{a}}$. Then:

Definition 2.1.14. An $\mathcal{O}-$ideal $\mathfrak{a} = m\left(a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z}\right)$ is reduced if $m = 1$ and $f_{\mathfrak{a}} = (a, b, c)$ is reduced.

Before giving the characterization of reduced $\mathcal{O}-$ideals, we require the notion of *norm* of an $\mathcal{O}-$ideal.

Definition 2.1.15 (Norm of an ideal). Let $\mathfrak{a}$ a fractional $\mathcal{O}$-ideal of discriminant Δ. The norm of $\mathfrak{a}$ is defined as $N_\Delta(\mathfrak{a}) = N_{\mathcal{O}}(\mathfrak{a}) = [\mathcal{O} : \mathfrak{a}]$.

To give an easy way to compute the $N(\mathfrak{a})$ with respect to $\mathcal{O}$, we see $\mathfrak{a}$ and $\mathcal{O}$ as lattices. The representation is exactly the same we saw, then $\mathfrak{a} = q\left(a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z}\right)$ and $\mathcal{O} = \mathbf{Z} + \frac{\Delta+\sqrt{\Delta}}{2}\mathbf{Z}$. The basis of these $\mathbf{Z}-$module (or two-dimensional lattices) are $(\alpha_1, \alpha_2) = (qa, q\frac{-b+\sqrt{\Delta}}{2})$ and $(\beta_1, \beta_2) = (1, \frac{\Delta+\sqrt{\Delta}}{2})$, respectively. From lattice theory, $[\mathcal{O} : \mathfrak{a}] = |\det(T)|$ where $T \in \mathrm{GL}(2, \mathbf{R})$ is the matrix which transform a basis of $\mathcal{O}$ in a basis for $\mathfrak{a}$. We do not explore here the invariance of the determinant and of the two basis in the result, but we give a good choice for T. It is clear that $(\alpha_1, \alpha_2) = (1, \frac{\Delta+\sqrt{\Delta}}{2})\begin{pmatrix} qa & -\frac{b+\Delta}{2}q \\ 0 & q \end{pmatrix}$. The matrix T has $|\det(T)| = \det(T) = q^2a \in \mathbf{Q}$ [8]. We deduce that $N(\mathfrak{a}) = q^2a$ in $\mathcal{O}$. However, notice that if the order changes the norm changes too and this can happens for example when considering the norm of an $\mathcal{O}-$ideal, where $\mathcal{O}$ is non maximal, with respect to $\mathcal{O}$ and with respect to its maximal order $\mathcal{O}_{\Delta_K}$.

Example 4. Let $\mathfrak{a} = q\left(a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z}\right)$ be an $\mathcal{O}-$ideal, where $\mathcal{O}$ is non maximal of conductor f and let $\mathcal{O}_{\Delta_K}$ be its maximal order. Then $N_{\mathcal{O}}(\mathfrak{a}) = q^2a$, while $N_{\mathcal{O}_{\Delta_K}}(\mathfrak{a}) = fq^2a$.

If the order is clear from the context, it can be omitted from the norm notation. The characterization of reduced forms (see Lemma 2.1.5) is valid in the context of reduced ideals but considering the norm of the ideal:

Proposition 2.1.17. Let $\Delta < 0$, then

- $\mathfrak{a}$ is reduced if and only if $a = N(\mathfrak{a}) = \min\{N(\alpha) : \alpha \in \mathfrak{a}, \alpha \neq 0$ and $b \geq 0$ if $a = c\}$.

[8] $|\det(T)| = \det(T)$ because of $a > 0$ for negative discriminants.

- if $\mathfrak{a}$ is reduced, then $N(\mathfrak{a}) < \sqrt{|\Delta|/3}$
- if $N(\mathfrak{a}) < \sqrt{|\Delta|/4}$, then $\mathfrak{a}$ is reduced

To reduce an $\mathcal{O}-$ideal $\mathfrak{a}$ it is enough to reduce the form $f_{\mathfrak{a}}$ in its representation. As a consequence, reduced ideals can be computed efficiently computing $\mathsf{Red}(f_{\mathfrak{a}})$ in Algorithm 1. Furthermore, remember that if we do not have the standard representation of $\mathfrak{a}$, but another representation from a form f in the same $\Gamma-$orbit of $f_{\mathfrak{a}}$, as observed in Subsubsection 2.1.2.3, it is enough to normalize f and then reduce it.

Equivalence of ideals Even ideals can be put in a equivalence relation which brings to a partition of them. Two fractional $\mathcal{O}$-ideals $\mathfrak{a}$ and $\mathfrak{b}$ are said *equivalent* if $\mathfrak{a} = \alpha\mathfrak{b}$ for some $\alpha \in K^*$, and *properly equivalent* if $\mathfrak{a} = \alpha\mathfrak{b}$ for some $\alpha \in K^*$ and $N(\alpha) > 0$.[9] Since in a imaginary quadratic field K the norm of each element is positive, we will consider only the proper equivalence of ideals. It is easy to check that proper equivalence is an equivalence relation of fractional $\mathcal{O}-$ideals and it divides $\mathcal{O}-$ideals in $\mathcal{O}-$ideal classes.

Definition 2.1.16. The *ideal class group* of $\mathcal{O}$ is defined as $Cl(\mathcal{O}) = I(\mathcal{O})/P(\mathcal{O})$. It is an abelian group and its order is the class number $h(\mathcal{O})$.

Remark 6. If we define $P^+(\mathcal{O})$ as the group of principal $\mathcal{O}-$ideals which have a generator of positive norm, we can define the narrow class group of $\mathcal{O}$ as $Cl^+(\mathcal{O}) = I(\mathcal{O})/P^+(\mathcal{O})$. In the case we consider, i.e. with $\Delta < 0$, all the fractional $\mathcal{O}$-ideals have positive norm and this is also valid for all the principal $\mathcal{O}-$ideals, since they are in particular fractional ideals. As a result $P(\mathcal{O}) = P^+(\mathcal{O})$ and $Cl(\mathcal{O}) = Cl^+(\mathcal{O})$ if $\Delta < 0$.

At first sight, the quotient which defines the ideal class group does not say a lot about its meaning, but we give a deeper look on it. First, elements in $Cl(\mathcal{O})$ are taken from $I(\mathcal{O})$, the set of fractional invertible $\mathcal{O}-$ideals. In particular, we saw that $I(\mathcal{O})$ has a group structure with the product of ideals we take from Lemma 2.1.15. We prove now that $Cl(\mathcal{O})$ is exactly the same set obtained by considering classes of invertible fractional $\mathcal{O}-$ideals with the properly equivalence relation. Let $\mathfrak{a}$ and $\mathfrak{b}$ be properly equivalent fractional $\mathcal{O}-$ideal, i.e. $\mathfrak{a} = \alpha\mathfrak{b}$ for some $\alpha \in K^*$. From group theory, if we consider a quotient group G/H from an equivalence relation then $a, b \in G$ belongs to the same class if $ab^{-1} \in H$. In our context $\mathfrak{a} = \alpha\mathfrak{b}$ implies

$$\mathfrak{a} = \mathfrak{b}\alpha\mathcal{O} \Rightarrow \mathfrak{a}\mathfrak{b}^{-1} = \alpha\mathcal{O} \in P(\mathcal{O}),$$

i.e. $[\mathfrak{a}] = [\mathfrak{b}]$ in $Cl(\mathcal{O})$. On the other side, if $[\mathfrak{a}] = [\mathfrak{b}]$ in $Cl(\mathcal{O})$, then $\mathfrak{a}\mathfrak{b}^{-1} = \alpha\mathcal{O}$ for some $\alpha \in K^*$ of positive norm (recall $P(\mathcal{O}) = P^+(\mathcal{O})$) and therefore $\mathfrak{a} = \mathfrak{b}\alpha\mathcal{O} = \alpha\mathfrak{b}$. We conclude that $\mathfrak{a}$ and $\mathfrak{b}$ are properly equivalent.
From what said above, to test that two $\mathcal{O}-$ideals are in the same class, essentially it seems to be enough to reduce their representatives, and check if the resulting reduced representative are the same. Actually, it is more complicated than using their normal forms $f_{\mathfrak{a}}$ representative because of the imaginary part. However reduction of ideals and forms are in some way similar, but we have to take in account multiplication of ideal by

[9]The norm of an element $\alpha = \lambda + \mu\sqrt{N} \in K^*$ is defined as usual for complex numbers, i.e. $N(\alpha) = \lambda^2 + \mu^2|N| > 0$.

the field elements. We avoid to extend the discussion about checking if two $\mathcal{O}-$ideals are equivalent to be concise. In brief, we have to extend the reduction procedure of forms to ideals. This brings to a reduction "factor". If two $\mathcal{O}-$ideals have the same factor, they are equivalent. This is motivated by the fact that if $\mathfrak{a} = \alpha\mathfrak{b}$, $\alpha \in K^*$, the ideals are equivalent, but α in general is not an integer.

Group operations in $Cl(\mathcal{O})$ Basic operations on the class group are quite efficient. Basic operations are: decide if two classes are equal, computing the inverse of a class and compute the product between classes.

Equality Each class is represented also by its reduced form, then it is enough to compare reduced representatives and verifying if two ideals have the same ones. Verifying two representation are the same – not that they are equivalent – consists of reading and comparing their triple representation (a, b, c), then the equality test takes time $O(\log(|\Delta|)$, since $|b| < a < \sqrt{|\Delta|/3}$ and they have the same c if a and b are the same (Δ is fixed). Notice that the size of a and b is $O(\log\sqrt{|\Delta|})$, however $c = \frac{b^2+|\Delta|}{4a} \leq \frac{a^2+|\Delta|}{4a} \leq \frac{a+|\Delta|}{4} \leq \frac{\sqrt{|\Delta|/3}+|\Delta|}{4}$ is of size $O(\log(|\Delta|))$.

Inverse Notice that from Proposition 2.1.16, the inverse of an ideal $\mathfrak{a}$ has a multiplying factor $1/a$. However, $1/a \in K^*$ and therefore $[\mathfrak{a}^{-1}] = [\frac{1}{a}\mathfrak{a}^{-1}]$ in $Cl(\mathcal{O})$. The inverse of $[a, b, c] \in Cl(\mathcal{O})$ is therefore computed as $[a, b, c]^{-1} = [a, -b, c]$. We conclude that the inverse of a class is computed in time $O(\log(|\Delta|)$, i.e. the lenght of its representation as pointed in the Equality item above.

Product A product of two classes requires the time of doing the product of representatives and reducing the result to the reduced form. The total time is $O((\log(|\Delta|)^2)$.

2.1.3.1 Relations between the ideal class group and the form class group

Definition 2.1.17. $\mathfrak{a}$ is said a *proper* $\mathcal{O}-$ideal if $\mathcal{O} = \{\beta \in K : \beta\mathfrak{a} \subset \mathfrak{a}\}$

The definition of proper ideals is needed to generalize the notion of invertible $\mathcal{O}-$ideal. Indeed, from Corollary 2.1.16.1, we know that all fractional $\mathcal{O}_{\Delta_K}-$ideals are invertible, when $\mathcal{O}_{\Delta_K}$ is maximal. When talking about classes, we consider only integral ideals since from the standard representation of any fractional ideals, we know it is obtained by the multiplication of an integral one by a constant belonging to the field K, i.e. they are in the same class. Applying the corollary, we have that in a maximal order $\mathcal{O}_{\Delta_K}$ the class of every $\mathcal{O}_{\Delta_K}-$ideal is invertible. In general, a fractional $\mathcal{O}-$ideal is invertible if and only if it is proper (cf. [Cox14], Proposition 7.14).

Theorem 2.1.18 ([Cox14], Theorem 7.7)**.** Let $\mathcal{O}$ be the order of discriminant Δ in an imaginary quadratic field K. Then

- if $f(x, y) = ax^2 + bxy + cy^2$ is a primitive positive definite quadratic form of discriminant Δ, then $\left[a, \frac{-b+\sqrt{\Delta}}{2}\right]$ is a proper ideal of $\mathcal{O}$
- the map sending $f(x, y)$ to $\left[a, \frac{-b+\sqrt{\Delta}}{2}\right]$ induces an isomorphism between the form class group $Cl(\Delta)$ and the ideal class group $Cl(\mathcal{O})$

If $h(\Delta) := |Cl(\Delta)|$ and $h(\mathcal{O}) := |Cl(\mathcal{O})|$, from item 2 of the previous theorem we deduce $Cl(\Delta) = Cl(\mathcal{O})$ for $\Delta < 0$, and as a result $h(\Delta) = h(\mathcal{O})$. Theorem 2.1.18 is fundamental since it permits to use operations on forms, e.g. the reduction algorithm, with more complex objects as ideal classes.

The class number $h(\mathcal{O})$ We have seen in Theorem 2.1.18 that from a discriminant Δ there exists a bijection between the ideal class group and primitive positive definite quadratic form class group. Since bijection is a one-to-one correspondence, it is clear that the number of ideal classes is the same as the number of reduced primitive positive definite forms. Let $h(\mathcal{O})$ denote the number of ideal classes of an order $\mathcal{O}$ of discriminant Δ, we have $h(\mathcal{O}) = h(\Delta)$. All the correspondences between ideal classes and form classes are fundamental since they give us an easy representation of ideals and they permits us to use the algorithm from forms theory for computing different tasks.However, as we have seen in Subsection 2.1.1, computing the class number is hard in computational terms. Indeed, the complexity of computing the class number derives from the computational costs of algorithms doing the task, even if an analytic formula exists. Consider the *Dirichlet series*

$$L(1, \chi_\Delta) = \sum_{n=1}^{\infty} \frac{\chi_\Delta(n)}{n} = \prod_p \left(1 - \frac{\chi_\Delta(p)}{p}\right)^{-1},$$

where $\chi_\Delta(n) = \left(\frac{\Delta}{n}\right)$ and $\left(\frac{\cdot}{\cdot}\right)$ denotes the Kronecker symbol.

Theorem 2.1.19. For $\Delta < 0$, the class number is $h(\mathcal{O}_\Delta) = \frac{w\sqrt{|\Delta|}}{2\pi} \cdot L(1, \chi_\Delta)$, where w denotes the number of the roots of unity in $\mathcal{O}_\Delta$.

Remark 7. For a negative discriminant $\Delta < 0$, we have $w = 2$ for any value of $\Delta < -4$ since the set of units in $\mathcal{O}_\Delta$ is $\{\pm 1\}$. For the remaining cases $\Delta = -3, -4$, we have $w_{-3} = 6$ and $w_{-4} = 4$. Without going too deeply in the explanation of this fact which is linked to automorphisms of forms given by equivalence transformations, for $\Delta = -3, -4$ the set of units of $\mathcal{O}_\Delta$ are the sixth roots and fourth roots of unity respectively. Then for $\Delta < -4$, we have $h(\mathcal{O}_\Delta) = \frac{\sqrt{\Delta}}{\pi} L(1, \chi_\Delta)$.

Now, we can give the formula which explains the relation between an order $\mathcal{O}$ (non-maximal) and its (maximal) order $\mathcal{O}_{\Delta_K}$. We recall the notation $[\mathcal{O}^*_{\Delta_K} : \mathcal{O}^*_{\Delta_f}]$ that indicates the dimension of $\mathcal{O}^*_{\Delta_K}$ as a $\mathcal{O}^*_{\Delta_f}-$vectorial space. We introduced this notation in Subsection 2.1.2.

Theorem 2.1.20 ([Cox14], Theorem 7.24)**.** Let $\mathcal{O}_{\Delta_f}$ be the order of conductor f in an imaginary quadratic field K. Then

$$h(\mathcal{O}_{\Delta_f}) = \frac{h(\mathcal{O}_{\Delta_K})f}{[\mathcal{O}^*_{\Delta_K} : \mathcal{O}^*_{\Delta_f}]} \prod_{p|f} \left(1 - \left(\frac{\Delta_K}{p}\right)\frac{1}{p}\right)$$

where the product is computed for each prime p dividing the conductor f. Furthermore, $h(\mathcal{O}_{\Delta_f})$ is always an integer multiple of $h(\mathcal{O}_{\Delta_K})$.

In Remark 7, we saw that in general $w = 2$ for a negative discriminant, except for the cases $\Delta = -3, -4$. Since the fundamental discriminant we will consider are large enough (in absolute value) to guarantee security properties, we can assume $\Delta_K < -4$, and of course non-maximal orders have a discriminat $\Delta_f < \Delta_K < -4$. As a result, in Theorem 2.1.20 we have $|\mathcal{O}^*_{\Delta_K}| = |\mathcal{O}^*_{\Delta_f}| = 2$ and then $[\mathcal{O}^*_{\Delta_K} : \mathcal{O}^*_{\Delta_f}] = 1$. The formula becomes

$$h(\mathcal{O}) = h(\mathcal{O}_{\Delta_K}) f \prod_{p|f} \left(1 - \left(\frac{\Delta_K}{p}\right) \frac{1}{p}\right).$$

For example, in the construction of the CL encryption scheme in Section 2.2, it is chosen $\Delta_K = -pq$ and $\Delta_q = -pq^3$, where p, q are primes, then the conductor is $f = q$. Since q is the unique prime divisor of f and $q|\Delta_K$, then $\left(\frac{\Delta_K}{q}\right) = 0$ and $h(\mathcal{O}_{\Delta_q}) = q \cdot h(\mathcal{O}_{\Delta_K})$. Even if it is hard to compute the class number, Theorem 2.1.20 gives us an analytic formula which links the class numbers of a maximal order and a non maximal suborder of it.

Even if it is hard to compute the class number from the discriminant, we know which is its asymptotic behaviour from the Brauer-Siegel theorem and a upper bound used in application. Even if it is proved in the more general case of number fields we recall a corollary of it for fields of a fixed degree over $\mathbb{Q}$ – as in the case of quadratic fields – readapted for the case of imaginary quadratic fields.

Theorem 2.1.21 (Brauer-Siegel)**.** Let K vary in a family of number fields of fixed degree over $\mathbb{Q}$. Then, as $|\Delta| \to \infty$, we have

$$\ln(h(\Delta_K)) \sim \ln(\sqrt{|\Delta_K|}).$$

As a consequence, $h(\Delta_K) \sim \sqrt{|\Delta_K|}$ asymptotically. This was a relevant result used in [HM00], where the authors use Brauer-Siegel theorem to propose suitable choices for the size and the form of the discriminant which avoid known attacks that aim to compute discrete logarithm in the class group. Furthermore, it is known that

$$h(\Delta) < \frac{1}{\pi}\sqrt{|\Delta|}\ln(|\Delta|)$$

From an algorithmic point of view, computing class number is quite hard with increasing of the discriminant ($h(\Delta) \sim \sqrt{|\Delta|}$ for the Brauer-Siegel theorem, cf. Theorem 2.1.21). Jacobson ([Jac00]) proposed an algorithm to solve the discrete logarithm in class group of imaginary quadratic field of discriminant based on an index-calculus method. Next, Biesse *et al.* ([BJS10]), analyze the computational cost of variants of the Jacobson proposed by Biasse and Vollmer and they conjectured that the state of the art implementation has complexity $L_{|\Delta|}[1/2, o(1)]$ [10]. They also proposed new values for the size of the discriminant with respect to the recommended NIST levels of security (112,128,192,256). Furthermore, the best known algorithm to compute class numbers in the case of fundamental discriminants is based on the same method and it has the same complexity as conjectured. Notice that we do not go deeper in the description of Jacobson proposal, since it requires a section dedicated on the structure of $Cl(\mathcal{O})$ and how to find a system of relation and generators for $Cl(\mathcal{O})$. We limits us to say that computing the structure of the class group consists in writing $Cl(\mathcal{O})$ as the direct products of cyclic groups. An attack on it is based on computing these cyclic groups and using their orders to find a solution, when it exists, of a system of modular equations using the Chinese Remainder Theorem.

[10] $L_x[a, b] = e^{b(\log x)^a(\log\log x)^{1-a}}$, where $a, b, x \in \mathbf{R}$, $0 \le a, b \le 1$ and $x > e$, i.e. the Euler constant.

2.2 Class groups in Cryptography: the Castagnos-Laguillaumie encryption scheme

Castagnos and Laguillaumie ([CL15]) proposed an El-Gamal fashion linear homomorphic encryption scheme whose security relies only on the DDH assumption. They also proposed a concrete instantiation based on Class Groups of Imaginary Quadratic Fields. The idea of their protocol consists in building the Class Group – whose order is unknown even if the discriminant is public[11] – such that it is possible to define a subgroup of prime order where it is easy to compute the discrete logarithm. After some years, the protocol has been modified and new versions were proposed. The most significant variation is in [CLT18a], where the DDH assumption on the elements of the group was changed by an hard subset membership assumption (HSM). In the original version from 2015, the message space is $\mathbf{Z}/q\mathbf{Z}$ where q is a random prime computed by the algorithm that generates the parameters, and its elements are encoded in the subgroup of known order. The encryption of a message m is of the form $(c_1, c_2) = (g^r, \mathsf{pk}^r f^m)$, where g generates the group they work with, f is the generator of the easy discrete log subgroup, $\mathsf{pk} = g^{\mathsf{sk}}$ is the public key and sk is the secret key which is took from some distribution $\mathcal{D}$. At that point, to decrypt a ciphertext it is enough to compute $m = \log_f(c_2/c_1^{\mathsf{sk}})$.
In this section we will talk a little about the encryption scheme focusing more on its parameters generation, since it is the part which reflects what we saw in previous section. Then, the starting point is building the subgroup where the discrete logarithm is easy, from a group which is defined by a discriminant. After giving an informal explanation of a DDH group with an easy DL subgroup in the original version, we present the instantiation of the generation of parameters in the variation of CL we will consider, i.e. that one in [CCL+19]. We start with some results on the structure of the class group for a specific choice of the discriminant and the consequences of this choice. Before considering that construction of CL, we will look at the novelty of [CLT18a] in which the class group is "split" in a direct product of two subgroups, where one of them is the subgroup with an easy DL. The main difference with the original version is that this time a generator of a subgroup G^q is used instead of a generator of the group. The group G and its subgroup G^q define the HSM problem under which the encryption scheme is ind-cpa secure. In some cases what we will recall are not always proved results, but some of them are conjectures motivated by experiments, as the Cohen-Lenstra heuristic which is very used in the context of class groups. Unfortunately, from a mathematical/computational point of view, it is quite hard to compute the order of the class group or its structure. However, from the cryptographic side, this hardness it is used for the security. To obtain a class group which is a product of two disjoint subgroup where one is of a chosen prime order, it is clear that the discriminant must be chosen with respect to some proved results in Algebraic Number Theory.

Structure of the section: Castagnos and Laguillaumie (cf. [CL15], pag. 22) observed that it is necessary to work with squares of the Class Group, otherwise it is easy to break the DDH assumption in the group. Then, in Subsection 2.2.1, we present some result in the *genus theory* and we see how to test that a class is a square and that it is possible

[11]The group is finite and unknown order means that it cannot be efficiently computable.

to compute square roots efficiently. In Subsection 2.2.2, we will see for which choices of the discriminant we can minimize the maximal power of 2 dividing the class number. In Subsection 2.2.3 we explain how to switch between a maximal order $\mathcal{O}_{\Delta_K}$ and a non maximal suborder $\mathcal{O}_{\Delta_q} \subset \mathcal{O}_{\Delta_K}$, that is necessary in the proof of Proposition 2.2.11 and in the Gen algorithm of CL based on that proposition. Finally, in Subsection 2.2.4 we present the definition of a HSM group with an easy DL subgroup, we see how the choice of previous subsections fits with the definition and we recall CL scheme with its characteristics.

2.2.1 Computing square roots in $Cl(\mathcal{O}_\Delta)$

Gauss (and others independently) studied the *genus theory* applied to binary quadratic forms. A *genus* is a set of classes of forms with the same discriminant Δ which represent the same values in $(\mathbf{Z}/\Delta\mathbf{Z})^*$. The genus cointaning the class of the principal form is called *principal genus.* In particular, characters χ of order 2 (i.e. $\chi(a)^2 = 1$ for each a in the set consided) are useful to distinguish squares from non squares. These characters are called *genus characters* and they are essentially Legendre symbols applied to number properly represented by a class $[f]$ of forms. Before going on, a form $f(x, y)$ properly represents a number n if there exists a couple $(\bar{x}, \bar{y})$ such that $f(\bar{x}, \bar{y}) = n$ and $\gcd(\bar{x}, \bar{y}) = 1$. If a form g is properly equivalent to f, then the sets of elements represented by f and g are the same. Indeed, if $g = f(U(x, y))$ for some proper transformation U, then n is equal to $g(U^{-1}(\bar{x}, \bar{y})) = f(UU^{-1}(\bar{x}, \bar{y})) = f(\bar{x}, \bar{y})$. As a consequence, we can see the class of proper equivalence $[f]$ of the form f as representative of the set of elements represented by f. Then, all genus character can be obtained by a basis of specific characters called *assigned characters.* Assigned characters are defined from each odd prime divisor $p_1, \dots, p_k$ of the discriminant and some special ones if $\Delta \equiv 0 \mod 4$, i.e. $\chi_1(\cdot) = (\cdot/p_1), \chi_2(\cdot) = (\cdot/p_2), \dots, \chi_k(\cdot) = (\cdot/p_k)$ for the prime divisor of Δ and $\chi_{-4}(t) = (-1)^{(t-1)/2}$, $\chi_8(t) = (-1)^{(t^2-1)/1}$ and $\chi_{-8}(t) = \chi_{-4}(t)\chi_8(t)$, taken depending on $-\Delta/4 \mod 8$ if Δ is even. Let μ denote the number of assigned characters[12]. We do not repeat the table of assigned characters chosen for even discriminant, that can be found in [Cox14], Theorem 3.15 proof, but we will give soon a look at some example. We call a *complete character* of a class the tuple of characters $\Psi([t]) = (\chi_1([t]), \chi_2([t]), \dots, \chi_\mu([t]))$, where $[t]$ denotes the class of t in $(\mathbf{Z}/\Delta\mathbf{Z})^*$ with the usual meaning of integers prime to Δ with the same remainder $t \mod \Delta$. Complete characters allows us to distinguish squares from non squares as we will see from next results:

Theorem 2.2.1 ([Cox14], Theorem 3.15)**.** Let $\Delta \equiv 0, 1 \mod 4$, $\Delta < 0$, then

- There are $2^{\mu-1}$ genera of forms of discriminant Δ
- The principal genus (the genus containing the principal form) consists of the classes in $Cl(\Delta)^2$, the subgroup of squares in the class group $Cl(\Delta)$.

Example 5. The reduced forms with discriminant $\Delta = -56$ are $(1, 0, 14), (2, 0, 7), (3, -2, 5)$ and $(3, 2, 5)$, Their classes belongs to two different genera of order 2, i.e. $H = \{[1, 0, 14], [2, 0, 7]\}$ and $H' = \{[3, -2, 5], [3, 2, 5]\}$. Notice

[12] If Δ odd, then $\mu = k$, otherwise $\mu = k + c$, where c is the number of the assigned characters for the even discriminant considered.

that elements in the principal genus H are squares since $[1,0,14] = [1,0,14]^2$ and $[2,0,7] = [3,2,5]^2$. However, we did not still give a method to identify squares efficiently, but we are going to do it.

Lemma 2.2.2 ([Cox14], Lemma 3.17)**.** The homomorphism $\Psi : (\mathbf{Z}/\Delta\mathbf{Z})^* \to \{\pm 1\}^\mu$ is surjective and its kernel is the subgroup H of values represented by the principal form. Thus Ψ induces an isomorphism $(\mathbf{Z}/\Delta\mathbf{Z})^*/H \xrightarrow{\sim} \{\pm 1\}^\mu$.

Theorem 2.2.1 proves that squares and principal genus are the same set, i.e. if an element lies in the principal genus it is a square, otherwise it is not a square. Lemma 2.2.2 proves that the elements represented by principal genus are the kernel of the complete character. This implies that a class is in the principal genus and then it is a square if $\Psi([f]) = (1, \dots, 1)$, i.e. if each assigned character is equal to 1. However, we need the last step necessary to compute the complete character. Indeed, we need a properly represented number which is prime to the discriminant since assigned characters are computed on them.

Remark 8. Special assigned characters for even discriminants, i.e. $\chi_{-4}(t) = (-1)^{(t-1)/2}$, $\chi_8(t) = (-1)^{(t^2-1)/1}$ and $\chi_{-8}(t) = \chi_{-4}(t)\chi_8(t)$, are defined for an odd t and chosen depending on the remainder of $-\Delta/4 \mod 8$. Then, to calculate the complete character of a class with an even discriminant, we need a *odd* properly represented number prime to Δ, while for odd Δ the number can be even.

The final question is if a properly represented number prime to the discriminant exists. Fortunately, the answer is positive.

Lemma 2.2.3 ([Cox14], Lemma 2.25)**.** Given a form $f(x,y)$ and an integer M, then $f(x,y)$ properly represents at least one number relative prime to M.

Example 6. We give two examples:

1. In the case of $\Delta = -pq$, consider a form class $[a,b,c]$ representing n such that $\gcd(n,\Delta) = 1$. If $\chi_p(n) = \chi_q(n) = 1$, then $[a,b,c]$ is a square.
2. In the previous example, $\Delta = -56 = -2^3 \cdot 7 = -4 \cdot 14$, then $-\Delta/4 \equiv 6 \mod 8$ and the complete character is $\Psi(\cdot) = (\chi_7(\cdot), \chi_{-8}(\cdot))$. If applied to $[2,0,7]$, the form represent $9 = f(1,1)$ which is prime to the discriminant. Since $\chi_7(9) = 1$ and $\chi_{-8}(9) = 1$, we conclude that the form is a square. Applied to $[3,5,2]$, we choose $3 = f(1,0)$ prime to the discriminant and notice that $\chi_7(3) = -1$ and $\chi_{-8}(3) = 1$, then $[3,5,2]$ is not a square.

Lagarias (cf. [Lag80]) proposed an algorithm to compute efficiently the square root of a form. First of all, it proved that the computation of the complete character is done in $O(\mu \log(\Delta)^2)$ elementary operations in the worst case, where μ is the number of assigned characters. We do not recall the entire algorithm which breaks the square root because it requires too much space and additional notions, but we limit ourselves to recall the resulting theorem, which satisfies all the condition we will assume in the construction of the CL scheme. Notice that, it is usual to call *determinant* the value N such that the discriminant is $\Delta = N$ or $\Delta = 4N$ depending from the cases $\Delta \equiv 0, 1 \mod 4$ to indicate the part outside the 4.

Theorem 2.2.4. Given a properly reduced form f of nonsquare determinant N such that

1. A complete factorization of N is provided.
2. A quadratic nonresidue n, is given for each prime p_i dividing N.
3. f is in the principal genus.

There is an algorithm which produces a proper primitive form g such that $[g] \cdot [g] = [f]$. This algorithm terminates in $O(\mu \log(N)^3)$ elementary operations in the worst case.

In our application $\Delta = N = -pq$ and the factorization is public, satisfying the first condition of Theorem 2.2.4. The second condition requires the computation of a complete character and we saw it can be done efficiently. Finally, since we will work in the subgroup of squares in $Cl(\mathcal{O}_\Delta)$, the third condition is always true. We conclude that in our application in next chapters we need to pay particular attention on the exponents of elements because, as we will see, the computation of square root compromises the security.

2.2.2 The odd part of the Class Group

We will work in the odd part of the class group, i.e. the subgroup of the elements of odd order. The choice is motivated by the Cohen-Lenstra heuristic in Subsubsection 2.2.2.2, which gives information about the structure of the class group. The starting question is:

If we do not know the class number, how do we know that we are in the odd part?

This question can be answered in an easy way for discriminants of a specific form that we will see in this section, since they generate a class group where the 2−Sylow, i.e. each subgroup of order 2^k where k is the maximum possible is of order two. At this point it is enough to work with squares, since the square of an element in a 2−Sylow is sent to the unit and as a consequence the squares of elements which are different from the unit have an odd order. In this subsection, we give the required definition and mathematical results/conjectures that are at the basis of the class groups used to build the CL encryption scheme.

Theorem 2.2.5. Let $\mathbb{G}$ be a finite group, and let $\mathrm{ord}(\mathbb{G})$ be the order of $\mathbb{G}$. Then, for each prime p and each integer r such that p^r divides $\mathrm{ord}(\mathbb{G})$, there is a subgroup of $\mathbb{G}$ of order p^r.

Definition 2.2.1 (p−Sylow subgroup). Let $\mathbb{G}$ a finite group, and let p be a prime number such that p divides $\mathrm{ord}(\mathbb{G})$. Let $\mathrm{ord}(\mathbb{G}) = p^k m$, where p does not divide m. Each subgroup of $\mathbb{G}$ of order p^k is called a *p−Sylow subgroup* (or simply a *p−Sylow*) of $\mathbb{G}$.

A p−Sylow is not necessarily unique, indeed if p^k is the order of the p−Sylow, it is possible to have more subgroups of $\mathbb{G}$ of order p^k. For example, $\langle 9 \rangle$ and $\langle 11 \rangle$ are both 3−Sylow of $\mathbb{G} = \mathbf{Z}_{14}^*$, where $\mathrm{ord}(\mathbb{G}) = 6$.
We are not interested in how many p−Sylow (or 2−Sylow in our specific case) there are, but in how large they are. As noticed before, we are interested in discriminants which permits to generate groups where the 2−Sylow is of order 2.

2.2.2.1 A choice for a $2-$Sylow of order 2

Remember that our final goal is explaining how the CL encryption scheme from Castagnos-Laguillaumie is built, and of particular interest in this construction it is the $2-$Sylow in $Cl(\mathcal{O}_{\Delta_K})$. In particular, [HM00] recommends to construct the discriminant Δ_K such that the two-part of $Cl(\mathcal{O}_{\Delta_K})$, i.e. the $2-$Sylow, is small. Then if the $2-$Sylow has order 2, the subgroup of squares has an odd order, since elements of order 2 squared give the neutral element in $Cl(\mathcal{O}_{\Delta_K})$. [Kap78] studied the link between the order of the $2-$Sylow of the class group and its discriminant, also for our specific case. To recall it, in our construction we look at using a discriminant of the form $\Delta = -pq$ where p, q are odd primes such that $p \equiv -q \mod 4$ and $\left(\frac{p}{q}\right) = -1$.[13]

Before analyzing our situation, we need a new easy notion. Fixed a discriminant, the principal form could not be the unique reduced form with $b = \Delta \mod 2$, e.g. $(1, 0, 14)$ and $(2, 0, 7)$ for $\Delta = -56$. This type of reduced forms are called *ambiguous* forms. Ambiguous forms are that ones which squared give the principal form, i.e. the form of order ≤ 2. Doing some calculation it is easy to prove that all reduced ambiguous forms (a, b, c) are that ones with $b = 0$, or $a = b$ or $a = c$. It was proved by Gauss (cf. [Kap78], Proposition 0 (Gauss)) that if the discriminant is negative, then in a class there is exactly 1 ambiguos form. If we call S_2 the $2-$Sylow subgroup, Gauss proved that if $\Delta = \pm p \equiv -1 \mod 4$, or $\Delta = \pm 2p$, or when $\Delta = \pm pq \equiv 1 \mod 4$ (i.e. our case), S_2 is cyclic non trivial, i.e. S_2 contains the class of an ambiguous form which square is the class of the principal form. Furthermore, [Kap78] reports that if S_2 is cyclic non trivial, then 4 divides its order $h_2(\Delta)$, i.e. the even part of $h(\Delta)$, if and only if the class of the ambiguos form inside S_2 is a square.
Looking at our choice of Δ, we exclude the case $b = 0$ since Δ is odd. Ambiguous forms are given by (a, a, c) and (a, b, a). If $a = b$, then $\Delta = -pq = a(a - 4c)$, then $a = p$ and $4c - a = q$ (remember that $a > 0$) or $a = q$ and $4c - a = p$. In the latter case we obtain the ambiguous form $(p, p, \frac{p+q}{4})$, which is reduced if $q > 3p$ (equality $q = 3p$ implies that it is not primitive). Notice that $f(1, 4) = 9p + 4q$ is prime to $\Delta = -pq$ (if not $p|q$ or $q|p$, for $p, q > 3$). Then we conclude this is not a square since $\left(\frac{9p+4q}{p}\right) = \left(\frac{4q}{p}\right) = \left(\frac{4}{p}\right)\left(\frac{q}{p}\right) = 1\cdot(-1) = -1$ and $\left(\frac{9p+4q}{q}\right) = \left(\frac{9p}{q}\right) = \left(\frac{9}{q}\right)\left(\frac{p}{q}\right) = 1\cdot(-1) = -1$ by the choice of p and q. Setting $a = c$ and doing the same reasoning, we obtain the two ambiguous forms $(\frac{p+q}{4}, \pm\frac{p-q}{2}, \frac{p+q}{4})$. We see that this is not a square taking the representative $f(2, 0) = p + q$.
To recap we saw that for the choice $\Delta = -pq$, $p \equiv -q \mod 4$ and $\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right) = -1$, the $2-$Sylow subgroup of $Cl(\mathcal{O}_\Delta)$ has order 2, i.e. the order of the squares in $Cl(\mathcal{O}_\Delta)$ is odd.

2.2.2.2 The Cohen-Lenstra Heuristic

Some relevant results about the structure of a class group or the properties of the class number with regard to the discriminant are little known. Cohen and Lenstra ([CL84]) presented a conjecture about probability of different events occuring in class groups, such as the probability that the group is cyclic or that a small prime divides the class number.

[13]Actually, the order of choice of p and q in $\left(\frac{p}{q}\right)$ is irrelevant, since from Kronecker symbol properties and the choice of $p, q \mod 4$, $\left(\frac{p}{q}\right) = (-1)^{(p-1)(q-1)/4}\left(\frac{q}{p}\right) = \left(\frac{q}{p}\right)$.

These probabilities were proved by experiments done by the authors and we present the proposed conjecture – which we took from [Coh00], Conjecture 5.10.1. The heuristic consider the odd part of the class group, i.e. the subgroup of elements of odd order, which we call Cl_{odd}. From Subsubsection 2.2.2.1, we saw that depending on the choice of the discriminant, we know some information about the even part of the class group. With our particular choice, the even part is isomorphic to $\mathbf{Z}/2\mathbf{Z}$, then the subgroup of square is of odd order t such that $t|h(\Delta)/2$. Actually, for the choice $\Delta = -pq$, there are only two genera, i.e. the principal genus and another one. Since each genus has the same number of classes and principal genus consists only of squares, the size of the subgroup of squares is exactly $t = h(\Delta)/2$.

Definition 2.2.2 (Riemann ζ function)*.* The Riemann $\zeta(\cdot)$ function is defined as

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s} = \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \cdots, \text{ if } \operatorname{Re}(s) > 1$$

where $\operatorname{Re}(s)$ is the real part of the complex number s.

Conjecture 2.2.6 (Cohen-Lenstra Heuristic)**.** For any odd prime p and any $r \in \mathbf{Z} \cup \{\infty\}$, set $\eta_r(p) = \prod_{1 \le k \le r}(1 - p^{-k})$ and $A = \prod_{k \ge 2} \zeta(k) \approx 2.29486$.

- The probability that Cl_{odd} is *cyclic* is equal to

$$\frac{\zeta(2)\zeta(3)}{3 \cdot \eta_\infty(2) \cdot A \cdot \zeta(6)} \approx 0.977575$$

- If p is an odd prime, the probability that $p|h(\Delta)$ is equal to

$$f(p) = 1 - \eta_\infty(p) = \frac{1}{p} + \frac{1}{p^2} - \frac{1}{p^5} - \dots$$

- If p is an odd prime, the probability that the $p-$Sylow subgroup of $Cl(\Delta)$ is isomorphic to a given finite Abelian $p-$group G is equal to $(p)_\infty/|\operatorname{Aut}(G)|$, where $\operatorname{Aut}(G)$ denotes the group of automorphisms of G.

- If p is an odd prime, the probability that the $p-$Sylow subgroup of $Cl(\Delta)$ has rank r (i.e. is isomorphic to a product of r cyclic groups) is equal to $\frac{\eta_\infty(p)}{p^{r^2}\eta_r(p)^2}$

Remark 9*.* The values used in Item 1 of the Cohen-Lenstra conjecture are $\zeta(2) = \pi^2/6 \approx 1.645$, $\zeta(3) \approx 1.202$, $\zeta(6) = \pi^6/945 \approx 1.017$ and $\eta_\infty(2) \approx 0.289$. In Item 2, to understand the dimension of $f(p)$, we have $f(3) \approx 0.43987$, $f(5) \approx 0.23967$ and $f(7) \approx 0.16320$.

We give a little explanation of the Cohen-Lenstra Conjecture: first item tells us that we expect that after having chosen a suitable discriminant, the odd part of the class group is cyclic except in 1 case out of about 45. Then, if the $2-$Sylow subgroup is isomorphic to $\mathbf{Z}/2\mathbf{Z}$, there is large probability that the subgroup Cl_{odd} of order $h(\Delta)/2$ is cyclic. Indeed, first item does not depend on the choice of the discriminant. However, item 2 works on a random variabile which is the discriminant. In second item, this is translated in the probability that chosing a random discriminant Δ an odd prime divides $h(\Delta)$ and as a consequence we cannot say anything on a specific choice. Furthermore, it does not guarantee that the odd part is not divisible by small primes. To give an example, if we choose $p = 3$ the Cohen-Lenstra heuristic tells that $p|h(\Delta)$ with probability $\tau_3 = 1/3 + 1/9 \approx 0.4444$. Doing some tests counting the number of times $3|h(\Delta)$ for each negative $\Delta > -50000$ and $\Delta > -150000$, we obtain the empirical probabilities 0.4470 and 0.4643, respectively. For $\Delta > -50000$ and $p = 5$, the heuristic returns $\tau_5 = 0.2400$, while empirically is 0.2131. Item 4 tells how large is the probability that a $p-$Sylow is the direct product of r cyclic subgroup. For completeness, each class group can be written in a unique way as the direct product of cyclic subgroup (cf. [BV07], Theorem 9.4.5), then Item 4 is in line with this last theorem.

2.2.3 Switching between a maximal order and a non maximal suborder

This subsection is the final part of the math results on class groups needed. Even if the first part of this section is related to the previous section, it is put here because it is a starting point of the cryptographic construction we will see. We start giving properties of class groups useful to understand a fundamental proposition of [CL15] which permits to build a group with a subgroup where the discrete logarithm problem is easy. This subgroup represents the space of encodings of messages.

Definition 2.2.3 (Coprime Ideals). Two $\mathcal{O}-$ideals $\mathfrak{a}$ and $\mathfrak{b}$ are said *coprime* if $\mathfrak{a} + \mathfrak{b} = \mathcal{O}$. Given an order $\mathcal{O}$ of conductor f with respect to a maximal order $\mathcal{O}_{\Delta_K}$, an $\mathcal{O}-$ideal $\mathfrak{a}$ is said *prime* to the conductor f is $\mathfrak{a} + f\mathcal{O} = \mathcal{O}$, which is equivalent to say that $\mathfrak{a}$ and $\mathfrak{b} = f\mathcal{O}$ are coprime.

Example 7. If $\Delta = -95$, $\mathfrak{a} = 2\mathbf{Z} + \frac{-1+\sqrt{-95}}{2}\mathbf{Z}$ and $\mathfrak{b} = 3\mathbf{Z} + \frac{-1+\sqrt{-95}}{2}\mathbf{Z}$ are coprime. Indeed, $\mathfrak{a} + \mathfrak{b} = 2\mathbf{Z} + 3\mathbf{Z} + \frac{-1+\sqrt{-95}}{2}\mathbf{Z} = \mathbf{Z} + \frac{-1+\sqrt{-95}}{2}\mathbf{Z} = \mathbf{Z} + \frac{1+\sqrt{-95}}{2}\mathbf{Z} = \mathcal{O}_{-95}$.

However, there is an easy way to see if a fractional $\mathcal{O}$-ideal $\mathfrak{a}$ is prime to the conductor. Indeed, $\mathfrak{a}$ is prime to conductor f if and only if $\gcd(N(\mathfrak{a}), f) = 1$. The notion of ideal prime to conductor is relevant where the discriminant is not fundamental. Indeed, when working with a maximal order $\mathcal{O}_{\Delta_K}$, the conductor is $f = 1$ and then each $\mathcal{O}_{\Delta_K}-$ideal is automatically prime to the conductor. In the case of a non maximal order $\mathcal{O}$ it is necessary to see how the conductor can influence the class group $Cl(\mathcal{O})$. The following Proposition 2.2.7 presents a link between the size of the class group in a maximal order and in a non maximal suborder with regard to the conductor. Proposition 2.2.9 presents links between $\mathcal{O}_{\Delta_K}$ and $\mathcal{O}-$ideals which are prime to conductor.

Proposition 2.2.7 ([Cox14], Proposition 7.19)**.** The inclusion $I(\mathcal{O}, f) \subset I(\mathcal{O})$ induces an isomorphism

$$I(\mathcal{O}, f)/P(\mathcal{O}, f) \simeq I(\mathcal{O})/P(\mathcal{O}) = Cl(\mathcal{O})$$

Lemma 2.2.8 ([Cox14], Lemma 7.18)**.** Let $\mathcal{O}$ be an order of conductor f

- An $\mathcal{O}-$ideal $\mathfrak{a}$ is prime if and only if its norm $N(\mathfrak{a})$ is relatively prime to f
- Every $\mathcal{O}-$ideal prime to f is proper

Proposition 2.2.9 ([Cox14], Proposition 7.20))**.** Let $\mathcal{O}$ be an order of conductor f in an imaginary quadratic field K.

- If $\mathfrak{A}$ is an $\mathcal{O}_{\Delta_K}-$ideal prime to f, then $\mathfrak{A} \cap \mathcal{O}$ is an $\mathcal{O}-$ideal prime to f of the same norm.
- If $\mathfrak{a}$ is an $\mathcal{O}-$ideal prime to f, then $\mathfrak{a}\mathcal{O}_{\Delta_K}$ is an $\mathcal{O}_{\Delta_K}-$ideal prime to f of the same norm.
- The map $\varphi_f : I(\mathcal{O}, f) \to I(\mathcal{O}_{\Delta_K}, f)$ such that $\mathfrak{a} \mapsto \mathfrak{a}\mathcal{O}_{\Delta_K}$ is an isomorphism

Finally, we can apply the isomorphism of Proposition 2.2.9 to classes, but it is necessary that in the class considered there is an ideal which is prime to conductor. This is not a problem, since there is always such an ideal, from the following Proposition:

Proposition 2.2.10 ([Cox14], Corollary 7.17)**.** Let $\mathcal{O}$ be an order in an imaginary quadratic field. Given a nonzero integer M, then every ideal class in $Cl(\mathcal{O})$ contains a proper $\mathcal{O}-$ideal whose norm is relatively prime to M.

From a theoretical point of view, from Proposition 2.2.9 we know that there exists an isomorphism from ideals prime to conductor of a maximal order $\mathcal{O}_{\Delta_K}$ to ideals prime to conductor of a non maximal suborder $\mathcal{O}_{\Delta_f} \subset \mathcal{O}_{\Delta_K}$, and viceversa. From Proposition 2.2.10 we saw that in each class there is at least one $\mathcal{O}-$ideal which is prime to the conductor. This means that classes in $\mathcal{O}_{\Delta_K}$ are in bijective correspondence with classes in $\mathcal{O}_{\Delta_f}$ with the isomorphism φ_f applied on a prime representative of the class. From a more practical point of view, we need a way to compute ideals which are prime to the conductor and an algorithm to compute φ_f and its inverse φ_f^{-1}. These two tasks can be done with some algorithms proposed in [HJPT98] and [PT00], but not for all values of the conductor. However, even if some algorithm we will see are specific for a prime conductor, this is not a problem for the resulting encryption scheme in Section 2.2.4 since it works with a prime conductor.

A generalization of the composition for different discriminants We can extend Definition 2.1.4 to the case of forms g_1 and g_2 with different discriminants $\Delta(g_1) \neq \Delta(g_2)$, where both discriminants come from the same fundamental discriminant, i.e. $\Delta(g_1) = f_1^2\Delta$ and $\Delta(g_2) = f_2^2\Delta$ for some maximal discriminant Δ. This is perfectly in line with Algorithm 4 from [HJPT98] in next section (or the algorithms of [PT00] not recalled here). The following definition is a readaptation of the formula of the product of two-dimensional lattices in [BV07], Theorem 7.3.16 which follows the line of Definition 2.1.4.

Definition 2.2.4. Let $g_1 = (a_1, b_1, c_1)$ and $g_2 = (a_2, b_2, c_2)$ be two quadratic forms of disciminants $\Delta_1 = f_1^2\Delta$ and $\Delta_2 = f_2^2\Delta$, respectively, for some fundamental discriminant Δ and f_i is the conductor of Δ_i. Set $s = (b_1f_2+b_2f_1)/2$, $n = (b_1f_2-b_2f_1)/2$, $d_1 = \frac{f_1}{\gcd(f_1,f_2)}$ and $d_2 = \frac{f_2}{\gcd(f_1,f_2)}$. Let u, v, w and d such that

$$ua_1d_2 + va_2d_1 + ws = d = \gcd(a_1d_2, a_2d_1, s)$$

and let $d_0 = \gcd(d, c_1d_2, c_2d_1, n)$. We define the composite of the two forms g_1 and g_2 as the form

$$(A, B, C) = \left(d_0\frac{a_1a_2}{d^2}, \frac{1}{d_2}\left(b_2 + \frac{2a_2}{d}(v(s - b_2d_1) - wd_1c_2)\right), \frac{B^2 - \Delta_2/d_2^2}{4A}\right).$$

modulo the action of $\Gamma = \left\{ \begin{pmatrix} 1 & \tau \\ 0 & 1 \end{pmatrix}, \tau \in \mathbf{Z} \right\}$.

If Δ_2 is fundamental, e.g. when switching from a non maximal order to a maximal one, we have $f_2 = 1$ and then $\Delta_2 = \Delta$, $d_2 = 1$, and the composition is

$$(A, B, C) = \left(d_0 \frac{a_1 a_2}{d^2}, b_2 + \frac{2a_2}{d}(v(s - b_2 d_1) - w d_1 c_2, \frac{B^2 - \Delta}{4A} \right).$$

Last formula is useful when switching between a non maximal order $\mathcal{O}_{\Delta_f}$ to a maximal order $\mathcal{O}_{\Delta_K}$ which contains $\mathcal{O}_{\Delta_f}$. We will see in next section how to switch between orders, for now it is enough to know that the class of $\mathcal{O}_{\Delta_f}-$ideal $\mathfrak{a}$ is transformed into a class in $\mathcal{O}_{\Delta_K}$ computing the class of the $\mathcal{O}_{\Delta_K}-$ideal $\mathfrak{a}\mathcal{O}_{\Delta_K}$. Let $\delta := \Delta_K \mod 2$. From Definition 2.2.4, $\mathfrak{a} = a\mathbf{Z} + \frac{-b+\sqrt{\Delta_f}}{2}\mathbf{Z}$ and $\mathcal{O}_{\Delta_K} = \mathbf{Z} + \frac{\delta+\sqrt{\Delta_K}}{2}\mathbf{Z}$, where $\Delta_f = f^2\Delta_K$ imply $g_1 = (a, b, \frac{b^2 - f^2\Delta_K}{4a})$, $g_2 = (1, \delta, \frac{\delta - \Delta_K}{4})$, $d_1 = f$, $d_2 = 1$. Since an assumption used is that $\gcd(a, f) = 1$ [14], then $d = \gcd(a, f, \frac{b+\delta f}{2}) = 1$, $w = 0$ (a and f coprime) and $d_0 = 1$. With the substitution of the element, we obtain

$$(A, B, C) = \left(a, bv + ua\delta, \frac{B^2 - \Delta_K}{4a} \right) \mod \Gamma,$$

Then, the output is reduced and we obtain exactly the output of Algorithm 4.

How to find a prime ideal to the conductor Algorithm 1 in [HJPT98] takes in input a primitive $\mathcal{O}-$ideal $\mathfrak{a}$ and a prime conductor q and returns another $\mathcal{O}-$ideal $\mathfrak{a}$ which is prime to q and equivalent to $\mathfrak{a}$, i.e. it finds an prime to conductor ideal in the class of $\mathfrak{a}$.

Algorithm 3: FindIdealPrimeTo – [HJPT98], Algorithm 1

```
Input: A primitive $\mathcal{O}_\Delta-$ideal $\mathfrak{a} = (a, b)$ and a prime $q$
Output: A primitive $\mathcal{O}_\Delta-$ideal, $\mathfrak{A} = (A, B)$ equivalent to $\mathfrak{a}$, such that
        $\gcd(N(\mathfrak{A}), q) = \gcd(A, q) = 1$
if $\gcd(a, q) > 1$ then
    $c \leftarrow (b^2 - \Delta)/4a$;
    if $\gcd(c, q) > 1$ then
        $A \leftarrow a + b + c$;
        $B \leftarrow -b - 2a$;
    else
        $A \leftarrow c$;
        $B \leftarrow -b$;
    return $(A, B)$
else
    return $(a, b)$
```

The idea behind the algorithm is quite easy. If $\gcd(a, q) = 1$, then $\mathfrak{a}$ has already its norm coprime to conductor. If not, we look at c, and if $\gcd(c, q) = 1$, it is enough to switch a and c using the transformation $(a, b) \mapsto (c, -b)$ which brings to an $\mathcal{O}-$ideal $\mathfrak{b}$

[14]Choosing a prime conductor of a certain size guarantees that this is always true

equivalent to $\mathfrak{a}$.[15] Finally, if both a and c are not coprime to q, then the algorithm chooses $A = a+b+c$, which is coprime to q. If not, $q|b$ and then (a, b, c) is not primitive. Fixed A and B as in the algorithm, we have $C = a$. The form $(a+b+c, -b-2a, c)$ is equivalent to (a, b, c) with the composition of transformations $(x, y) \mapsto (x, x+y)$ and $(x, y) \mapsto (y, -x)$. Furthermore, the algorithm is efficient since it requires the computation of at most two gcd of element of size at most $O(\log(|\Delta|))$, i.e. the max size of c.

How to move between $\mathcal{O}_{\Delta_K}$ and $\mathcal{O}_{\Delta_q}$ Algorithm 3 in [HJPT98] present a practical to compute $\varphi(\mathfrak{a})$ for some $\mathcal{O}_{\Delta_f}$. The algorithm takes in input an $\mathcal{O}_{\Delta_q}$−ideal $\mathfrak{a}$, it obtains an $\mathcal{O}_{\Delta_q}$−ideal $\mathfrak{b}$ prime to conductor using Algoritm FindIdealPrimeTo and it computes $\varphi_f(\mathfrak{b})$.

Algorithm 4: GoToMaxOrder – [HJPT98], Algorithm 3

```
Input: A primitive $\mathcal{O}_{\Delta_q}$−ideal $\mathfrak{a} = (a, b)$, the fundamental discriminant $\Delta_K$ and
       the conductor $q$
Output: A primitive $\mathcal{O}_{\Delta_K}$−ideal $\mathfrak{A} = \varphi^{-1}(\mathfrak{b}) = \mathfrak{b}\mathcal{O}_{\Delta_K} = (A, B)$, where $\mathfrak{b} \sim \mathfrak{a}$
        and $\gcd(N(\mathfrak{b}), q) = 1$
$(A, B) \leftarrow$ FindIdealPrimeTo$(a, q)$;
$\delta \leftarrow \Delta_K \mod 2$;
Solve $1 = \mu q + \lambda A$ for $\mu, \lambda \in \mathbf{Z}$;
$B \leftarrow B\mu + A\delta\lambda \mod 2A$;
return $(A, B)$
```

Algorithm 5: GoToNonMaxOrder(Inverse) – [HJPT98], Algorithm 2

```
Input: A primitive $\mathcal{O}_{\Delta_K}$−ideal $\mathfrak{A} = (A, B)$ and the conductor $q$
Output: A primitive $\mathcal{O}_{\Delta_q}$−ideal $\mathfrak{a} = \varphi(\mathfrak{B}) = \mathfrak{B} \cap \mathcal{O}_{\Delta_q} = (a, b)$, where $\mathfrak{B} \sim \mathfrak{A}$
        and $\gcd(N(\mathfrak{B}), q) = 1$
$(a, b_q) \leftarrow$ FindIdealPrimeTo$(\mathfrak{A}, q)$;
$b \leftarrow b_q q \mod 2a$;
return $(a, b)$
```

Both the algorithms require $O(\log(|\Delta|)^2)$ bit operations, where in GoToMaxOrder $\Delta = \Delta_q$ and in GoToNonMaxOrder $\Delta = \Delta_K$.

2.2.4 A subgroup with an easy discrete logarithm problem

In this subsection we recall the framework of the CL linearly homomorphic encryption scheme, introduced in [CL15] and built from a finite group G which contains a subgroup where DL problem is easy. A concrete instantiation of the scheme is obtained from class groups of imaginary quadratic fields. Next, the scheme has been enhanced in [CLT18a],[CCL+19],[CCL+20]. The original proposed CL in [CL15] is different from its version in [CLT18a] principally for the hard assumption at the basis of the ind-cpa security of the scheme. Indeed, the original scheme is based on the hardness of the decisional Diffie-Hellman assumption in the group G, while the version in [CLT18a] is based on a *hard subset membership* problem, i.e. the difficulty to distinguish elements in the group

[15]Looking at the associated forms, this is the proper transformation $(x, y) \mapsto (y, -x)$. Furthermore, if necessary the ideal has to be normalized.

G from elements in a certain subgroup of G. In this section we do not recall neither the enhanced version in [CLT18a] in its entirety nor the original version in [CL15], but we give only an informal description of them. However, we give a more detailed description of the slightly modified version of CL proposed in [CCL+19], since it is the version we consider in next chapters. In next paragraphs GenGroup refers to the couple of algorithms (Gen, Solve), where Gen outputs the group G with an easy DL subgroup F and necessary parameters, while Solve outputs discrete logarithms in F.

The original CL scheme In [CL15], the authors – inspired by [BCP03] – present a definition of a DDH group with an easy DL subgroup, i.e. a group G where the decision Diffie-Hellman assumption is hard but such that there is a subgroup F of G where computing the discrete logarithm is easy. They also proposed a generic linearly homomorphic encryption scheme based on that definition and they presented an instantiation using class group of imaginary quadratic field. In few words, CL is an El-Gamal fashion encryption scheme where messages are encoded in the exponent of the subgroup F. Indeed, to decrypt a message it is necessary to extract the part in F of the ciphertext using the secret key and computing the easy discrete logarithm of the result with Solve. In the instantiation from the ideal class group, Gen and Solve algorithm are constructed from Proposition 2.2.11. Proposition 2.2.11 is an original result of [CL15] and it gives a way to compute the subgroup of order q where the DL is easy.

Proposition 2.2.11 ([CL15], Proposition 1)**.** Let Δ_K be a fundamental discriminant with $\Delta_K \equiv 1 \bmod 4$ of the form $\Delta_K = -qs$ where q is an odd prime and s a non-negative integer prime to q such that $s > 4q$. Let $\mathfrak{t} = (q^2, q)$ be an ideal of $\mathcal{O}_{\Delta_q}$, the order of discriminant $\Delta_q = \Delta_K \cdot q^2 = -q^3 s$. Denote by $f = [\mathfrak{t}]$ the class of $\mathfrak{t}$ in $Cl(\mathcal{O}_{\Delta_q})$. For $m \in \{1, \ldots, q-1\}$, $\mathsf{Red}(f^m) = (q^2, L(m)q)$ where $L(m)$ is the odd integer in $[-q, q]$ such that $L(m) \equiv 1/m \mod q$. Moreover, f is a generator of the subgroup of order q of $Cl(\mathcal{O}_{\Delta_q})$.

The idea behind the proof of Proposition 2.2.11 consists in considering the surjection $\bar{\varphi}_q$ between $Cl(\mathcal{O}_{\Delta_q})$ and $Cl(\mathcal{O}_{\Delta_K})$ from Proposition 2.2.9 and noticing that $\ker(\bar{\varphi}_q)$ is cyclic of order q. Then, a generator g of a cyclic isomorphic group to $\ker(\bar{\varphi}_q)$ is found and a principal $\mathcal{O}_{\Delta_K}-$ideal is built from it. Finally, using the inverse of the surjection the authors find an $\mathcal{O}_{\Delta_q}-$ideal which is prime to conductor and equivalent to (q^2, q). The class of (q^2, q) defines the generator f of the subgroup $F = \ker(\bar{\varphi}_q)$ of order q. In this subgroup it is easy to compute Dlogs since g^m gives an ideal corrispondent to $(q^2, L(m)q)$ and computing $L(m)^{-1} \mod q$ is a very efficient operation. Furthermore, all elements in F are easily recognizable by their norm q^2.[16] From Proposition 2.2.11 and following a construction of [HJPT98], in [CL15] the authors were able to construct a generation algorithm Gen for a DDH group with an easy DL subgroup in the instantiation with the class group. To sum up, they presented the first version of CL, a linearly homomorphic encryption scheme which is ind-cpa secure if the DDH problem is hard in the group G and they gave a concrete instantiation.

[16]This is in general not true, however if $a = q^2$ and $b = L(m)q$ then $c > a$ and $a > |b|$. As a consequence, f^m is reduced.

Enhanced CL scheme Castagnos et al. (cf. [CLT18a]) enhanced the CL framework by introducing a hard subgroup membership assumption (HSM). They slightly modify the definition of DDH group with an easy DL subgroup to make it suitable to the HSM assumption. First, the integer q from $\mathsf{GenGroup}$ is chosen prime. Second, from Gen algorithm in [CL15] they deduced that instead of giving the generator g of a group as output in Gen, this algorithm can directly output the generator g_q of the subgroup of the $q-$powers G^q, the generator f of F – i.e. $[(q^2, q)] \in \ker(\bar{\varphi}_q)$ – and consider $g = g_q \cdot f$ as the generator of the class group they work with. The order of G^q divides s and $\gcd(q, s) = 1$, then $G^q \cap F = \{[\mathcal{O}]\}$ (the class of the principal ideal, i.e. the unit of the group) and by construction $G = G^q \times F$ is cyclic with generator g. From the construcion, the authors built a CL version which is $\mathsf{ind\text{-}cpa}$ if the HSM assumption is hard, i.e. if it is hard to distinguish between elements in G from that in G^q.

2.2.4.1 Additional modification to CL

[CCL+19, CCL+20, CCL+21] present distributed versions of the threshold ECDSA protocol using a linear homomorphic encryption scheme to compute part of the operations on parties' shares. In their construction the encryption scheme used is CL, but it is adapted to the parameter of the signature scheme. We do not go deeply in what they exactly do because Chapters 3, 4 and 5 are completely dedicated to them. What we are interested in is the fact the instantiations in their scheme are built using the CL scheme with some adaptation to the definition we informally introduced above. In particular, these definitions can be slightly modified to be compatible with the order of the elliptic curve. We now present formal definition of a group with an easy DL subgroup and of HSM in the context of [CCL+19]. However, the costructions in [CCL+20, CCL+21] require additional modifications that we will see in the dedicated chapters. Indeed, in [CCL+20, CCL+21] the generation of the parameters of the class group requires the partecipation of all parties and in those contexts it is required that the distribution of the generator is close to uniformly random, while in the setup of CL the generator is deterministic. Anyway, we postpone this discussion to the dedicated chapters since it is not relevant to present the CL construction below.

Definition 2.2.5 ([CCL+19] version). Let $\mathsf{GenGroup}$ be a pair of algorithms $(\mathsf{Gen}, \mathsf{Solve})$. The Gen algorithm is a group generator which takes as inputs a parameter λ and a prime q and outputs a tuple $(\tilde{s}, g, f, g_q, \widehat{G}, G, F, G^q)$. The set $(\widehat{G}, \cdot)$ is a finite abelian group of order $q \cdot \widehat{s}$ where the bitsize of $\widehat{s}$ is a function of λ and $\gcd(q, \widehat{s}) = 1$. The algorithm Gen only outputs an upper bound $\tilde{s}$ of $\widehat{s}$. It is also required that one can efficiently recognise valid encodings of elements in $\widehat{G}$. The set $(F, \cdot)$ is the unique cyclic subgroup of $\widehat{G}$ of order q, generated by f. The set $(G, \cdot)$ is a cyclic subgroup of $\widehat{G}$ of order $q \cdot s$ where s divides $\widehat{s}$. By construction $F \subset G$, and, denoting $G^q := \{x^q, x \in G\}$ the subgroup of order s of G, it holds that $G = G^q \times F$. The algorithm Gen outputs f, g_q and $g := f \cdot g_q$ which are respective generators of F, G^q and G. Moreover, the DL problem is easy in F, which means that the Solve algorithm is a deterministic polynomial time algorithm that solves the discrete logarithm problem in F:

$$\Pr\Big[x = x^\star : (\tilde{s}, g, f, g_q, \widehat{G}, G, F, G^q) \leftarrow \mathsf{Gen}(1^\lambda, q), x \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}, X \leftarrow f^x, \\ x^\star \leftarrow \mathsf{Solve}(q, \tilde{s}, g, f, g_q, \widehat{G}, G, F, G^q, X)\Big] = 1.$$

Our definition slightly modify the original definition of [CLT18a] to be adapted to our application. In the original Gen algorithm the prime q is generated by the algorithm, while here it is given as input. We output the group $\widehat{G}$ from which the group G with an easy DL subgroup F is produced. Giving a prime q as input to the Gen algorithm does not change the meaning of it since q was independently generated of the rest of the output in [CL15, CLT18a].

A Hard Subgroup Membership Assumption We present the definition of an hard subgroup membership (HSM) problem within a group with an easy DL subgroup as defined in [CCL+19], which is slightly modified from the definition of HSM in [CLT18a]. In Definition 2.2.5, one has $G = F \times G^q$. The assumption says that it is hard to distinguish the elements of G^q in G. To understand the similarity of HSM with other hard assumption, we can compare it to Paillier's DCR assumption. They are closely related, but there is no reduction from one to the other one. Such hard subgroup membership problems are based on a long line of assumptions on the hardness of distinguishing powers in groups. DCR and HSM are essentially the same assumption, but they are defined in different groups. We emphasise that this assumption is well understood both in general, and for the specific case of class groups of quadratic fields, which we will use to instantiate $\mathsf{GenGroup}$. It was first used by [CLT18a] within class groups to prove the $\mathsf{ind\text{-}cpa}$ security of $\mathsf{HSM} - \mathsf{CL}$, i.e. the CL scheme from HSM we use in the manuscript (even if we refer to it as simply CL almost always), but we adapt it to our specific context.

Definition 2.2.6 (HSM assumption, [CCL+19]). We say that $\mathsf{GenGroup}$ is the generator of a HSM group with easy DL subgroup F if it holds that the HSM problem is hard even with access to the Solve algorithm. Let $\mathcal{D}$ (resp. $\mathcal{D}_q$) be a distribution over the integers such that the distribution $\{g^x, x \hookleftarrow \mathcal{D}\}$ (resp. $\{g_q^x, x \hookleftarrow \mathcal{D}_q\}$) is at distance less than $2^{-\lambda}$ from the uniform distribution in G (resp. in G^q). Let $\mathcal{A}$ be an adversary for the HSM problem, its advantage is defined as:

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda) = \Big| 2 \cdot \Pr\Big[& b = b^\star : (\tilde{s}, g, f, g_q, \widehat{G}, G, F, G^q) \leftarrow \mathsf{Gen}(1^\lambda, q), \\ & x \hookleftarrow \mathcal{D}, x' \hookleftarrow \mathcal{D}_q, b \xleftarrow{\$} \{0,1\}, Z_0 \leftarrow g^x, Z_1 \leftarrow g_q^{x'}, \\ & b^\star \leftarrow \mathcal{A}(q, \tilde{s}, g, f, g_q, \widehat{G}, G, F, G^q, Z_b, \mathsf{Solve}(.))\Big] - 1 \Big|\end{aligned}$$

The HSM problem is said to be hard in G if for all probabilistic polynomial time attacker $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda)$ is negligible.

```
Gen(1^λ, q)

1. Let $\mu$ be the bit size of $q$. Pick $\tilde{q}$ a random $\eta(\lambda)-\mu$ bits prime such that $q\tilde{q} \equiv -1 \bmod 4$ and $(q/\tilde{q}) = -1$.
2. $\Delta_K \leftarrow -q\tilde{q}$, $\Delta_q \leftarrow q^2\Delta_K$ and $\widehat{G} \leftarrow Cl(\Delta_q)$
3. $f \leftarrow [(q^2, q)]$ in $Cl(\Delta_q)$ and $F := \langle f \rangle$
4. $\tilde{s} \leftarrow \lceil \frac{1}{\pi} \log|\Delta_K|\sqrt{|\Delta_K|}\rceil$
5. Let $r$ be a small prime, with $r \neq q$ and $\left(\frac{\Delta_K}{r}\right) = 1$, set $\mathfrak{r}$ an ideal lying above $r$.
6. Set $g_q \leftarrow [\varphi_q^{-1}(\mathfrak{r}^2)]^q$ in $Cl(\Delta_q)$ and $G^q \leftarrow \langle g_q \rangle$
7. Set $g \leftarrow g_p \cdot f$ and $G \leftarrow \langle g \rangle$
8. Return $(\tilde{s}, g, f, g_q, \widehat{G}, G, F, G^q)$
```

Figure 2.1: Group generator Gen

2.2.4.2 Construction of the direct product G_q and F

The parameter generation algorithm Gen of the CL cryptosystem is presented in Figure 2.1. First, a prime q is chosen and given to the algorithm. Then, in step 1 and 2 the algorithm pick another prime $\tilde{q}$ such that the resulting fundamental discriminant is $\Delta_K = -q\tilde{q} \equiv 1 \mod 4$. For the choice of $\tilde{q}$ – i.e. such that $(q/\tilde{q}) = -1$ – the resulting class group has a $2-$Sylow subgroup of order 2 as discussed in Subsubsection 2.2.2.1. Then, the class group from a non maximal order of conductor q is uniquely defined and the $2-$Sylow subgroup is again of order 2 because of $h(\Delta_q) = q \cdot h(\Delta_K)$. In step 3 and 4 the algorithm chooses the generator f of the subgroup of order q where the DL is easy following Proposition 2.2.11 and it fixes the known upperbound $\tilde{s}$ of the class number. In Steps 5,6 and 7, alogrithm compute the generator of G^q following the construction in the proof of Proposition 2.2.11 and the construction presented in [HJPT98], obtaining $G = G^q \times F$ as a subgroup of $\hat{G}$. Indeed, the order of G^q divides $\hat{s}$, but it is not assured that it is equal to $\hat{s}$.

Remark 10. The strategy adopted by [HJPT98] is the following: first, select a prime number r such that $(\Delta_q/r) = 1$, i.e. Δ_q is a quadratic residue $\mod r$. Then, build a prime $\mathcal{O}_{\Delta_q}-$ideal $\mathfrak{r} \leftarrow (r, b_r)$ computing b_r, i.e. a square-root of $\Delta_q \mod 4r$. The authors suggest to do this computation using Shanks' probabilistic algorithm RESSOL. A version of RESSOL with expected run time $O((\log(r))^3 + \log(\Delta_q) \cdot \log(r))$, and a deterministic algorithm for computing the Kronecker-symbol (Δ_q/r) in $O((\log(r))^2 + \log(r) \cdot \log(|\Delta_q|))$. Finally, consider $\mathfrak{r}^2$ and reduce it. The exponentiation can be done via NUCOMP and NUDUPL.

```
Algorithm KeyGen(1^λ, q)
  1. $(\tilde{s}, g, f, g_q, \widehat{G}, G, F, G^q) \leftarrow \mathsf{Gen}(1^\lambda, q)$
  2. Pick $x \hookleftarrow \mathcal{D}_q$ and $h \leftarrow g_q^x$
  3. Set $\mathsf{pk} \leftarrow (\tilde{s}, g_q, f, p, h)$
  4. Set $\mathsf{sk} \leftarrow x$
  5. Return $(\mathsf{pk}, \mathsf{sk})$

Algorithm Enc(pk, m)
  1. Pick $r \hookleftarrow \mathcal{D}_q$
  2. Return $(g_q^r, f^m h^r)$

Algorithm Dec(sk, (c_1, c_2))
  1. Compute $M \leftarrow c_2/c_1^x$
  2. Return $\mathsf{Solve}(M)$
```

Figure 2.2: Description of the $\mathsf{HSM}-\mathsf{CL}$ encryption scheme

.

A linearly homomorphic encryption scheme from HSM We here recall the $\mathsf{HSM}-\mathsf{CL}$ scheme of [CLT18a]. Under the hardness of the HSM assumption, the encryption scheme is $\mathsf{ind\text{-}cpa}$ secure scheme (the proof is given in the full version of [CLT18a], i.e. in [CLT18b], Appendix IV). It is a building block for the distributed ECDSA schemes of Chapters 3,4 and 5. Elements from G^q are chosen in $\{g_q^r, r \leftarrow \mathcal{D}_q\}$, where $\mathcal{D}_q$ is a distribution at distance less than $2^{-\lambda}$ from the uniform distribution in G^q. The plaintext space is $\mathbf{Z}/q\mathbf{Z}$, where q is a μ bit prime, with $\mu \geq \lambda$. The scheme (from [CLT18a]) used in [CCL+19] is depicted in Figure 2.2.

On the size of parameters The size $\eta(\lambda)$ of Δ_K is chosen to resist the best practical attacks, which consists in computing discrete logarithms in $Cl(\Delta_K)$ (or equivalently the class number $h(\Delta_K)$). - An index-calculus method to solve the DL problem in a class group of imaginary quadratic field of discriminant Δ_K was proposed in [Jac00]. It is conjectured in [BJS10] that a state of the art implementation of this algorithm has complexity $O(L_{|\Delta_K|}[1/2, o(1)])$, which allows to use asymptotically shorter keys compared to protocols using classical problems that are solved in subexponential complexity $O(L[1/3, o(1)])$ (see Section 3.5.3 for concrete sizes for η).

Chapter 3

Two-Party ECDSA from Hash Proofs Systems

In previous chapters we introduced and we explained some of the tools we need to understand next chapters, starting from this one. As said in the introduction, Chapters 3, 4 and 5 focus on our results in [CCL$^+$19], [CCL$^+$20] and [CCL$^+$21], respectively, and each work has a dedicated chapter. To resume the precise topics of each chapter, this one copes with our two-party ECDSA construction, the next chapter with the more general case of our $(t, n)-$threshold ECDSA and the last one with improvements to our threshold construction to take in account proactivity, the identification of misbehaving players and other properties. More details about our threshold constructions will be given in next chapters.

The starting point Our construction generalizes from the two-party ECDSA protocol of Lindell [Lin17]. To understand the main advantages and contributions of our solution, we introduce the main difficulties that Lindell proposal overcame. Then, we see how to build a more general framework for two-party ECDSA from Hash Proof Systems (HPS), observing how this improves on Lindell's idea. In the introduction chapter, we presented the problem of building an efficient two-party threshold ECDSA starting from [MR04]. As pointed there, their solution is inefficient caused by the number of expensive zero-knowledge required to prove the security of their protocol. We then recalled the idea followed by [Lin17], which managed to provide a much simpler and efficient protocol than previous one in [MR04]. This chapter begins with a little deeper discussion of the solution of Lindell, since it is the starting point of our scheme in [CCL$^+$19]. We limit the description of [Lin17] to this introduction, and the entire chapter is dedicated to our two-party ECDSA construction referring to [Lin17] only to highlight the differences between its and ours constructions when required.
The crucial idea of Lindell's protocol is the observation that dishonest parties can create very little trouble. Indeed, suppose that P_1 is corrupted. Initially, P_1 sends both a Paillier's encryption key *and* an encryption $\mathsf{Enc}(x_1)$ of its share x_1 of the secret signing key to P_2, and after this step all P_1 can do is participate in the generation of $R \leftarrow k_1k_2P$, essentially, where R is the nonce of the ECDSA scheme. Anyway, the generation of R is a Diffie-Hellman protocol, and very efficient and robust protocols for it exist. On the other hand, if P_2 is corrupted all she can do (except again participate in the generation of R)

is to create a bad c as a final response for P_1. However, while P_2 can certainly try that, this would be easy to detect by simply checking the validity of the resulting signature. Even if Lindell's intuition is nice and interesting, the resulting protocol presents caveats given by working with elements of Paillier homomorphic encryption:

1. Paillier's plaintexts space is $\mathbf{Z}/N\mathbf{Z}$ (where N is a large composite), whereas ECDSA signatures live in $\mathbf{Z}/q\mathbf{Z}$ (q is prime). The difference between moduli creates inconsistencies if a wraparound occurs during the whole signature generation process. A solution to avoid such inconsistencies is to make sure that N is taken large enough so that no wraparounds occur. As a consequence, because of $N > q$ it is necessary that that when sending $\mathsf{Enc}(x_1)$ to P_2, P_1 needs to prove that the plaintext x_1 is in the right range (*i.e.* sufficiently small).

2. A second issue is linked to using Paillier's encryption in the proof. Indeed, if one wants to use the scheme to argue indistinguishability of an adversary's view in real and simulated executions, it seems necessary to set up a reduction to the indistinguishability of Paillier's cryptosystem. This means one must design a proof technique that manages to successfully use Paillier's scheme *without* knowing the corresponding secret key. In Lindell's protocol the issue arises when designing the simulator's strategy against a corrupted player P_2. In such a case, P_2 might indeed send a wrong ciphertext c (*i.e.* one that does not encrypt a signature) that the simulator simply cannot recognize as bad. Lindell [Lin17] proposes two alternative proofs, one that relies on game based definition and the other one on simulation. The former avoids the problem by simply allowing the simulator to abort with a probability that depends on the number of issued signatures q_s. This results in a proof of security that is not tight (as the reduction loses a factor q_s). The latter proof is simulation based, avoids the aborts, but requires the introduction of a new interactive non standard assumption regarding Paillier's encryption (which we recall in Figure 3.5).

Two-party ECDSA from Hash Proof Systems From the caveats listed in previous paragraph, we can ask to ourselves if it possible to devise a two party ECDSA signing protocol which is practical (both in terms of computational efficiency and in terms of bandwidth consumption[1]), that does not require interactive assumptions. Furthermore, can such a proposal allow for a tight security reduction? We have a positive answer for the above question and we will present in details our solution from [CCL+19] for answering it. The aim of this chapter is presenting this solution with all the required tools. The first contribution we give in [CCL+19] is providing a generic construction for two-party ECDSA signing from hash proof systems (HPS), which achieves simulation based security. HPS are a mathematical tool introduced by Cramer and Shoup ([CS98],[CS02]) to present a general framework for public key encryption which is secure against adaptive chosen ciphertext attacks. We will recall HPS and the properties they require to satisfy for our protocol in Section 3.2. Our solution can be seen as a generalization of Lindell's scheme [Lin17] to the general setting of HPSs that are homomorphic in the sense of [HO09]. This generic solution is not efficient enough for practical

[1]We note here that the very recent two party protocol of [DKLs18] is very fast in signing time and only relies on the ECDSA assumption. However its bandwidth consumption is much higher than [Lin17].

applications as, for instance, it employs general purpose zero knowledge as underlying building block. Still, beyond providing a clean, general framework which is of interest in its own right, it allows us to abstract away the properties we want to realize. In particular, our protocol allows for a proof of security that is both tight and does not require artificial interactive assumptions when proving simulation security. It is also interesting that in public key encryption schemes based on HPSs, indistinguishability of ciphertexts is not compromised by the challenger knowing the scheme's secret keys as it relies on a computational assumption and a statistical argument. This is a clear difference from using Paillier encryption, as we discuss in previous paragraph. The correctness of our protocol follows from homomorphic properties that we require of the underlying HPS. We define the notion of *homomorphically-extended projective hash families* which ensure the homomorphic properties of the HPS hold for any public key sampled from an efficiently recognisable set, thus no zero-knowledge proofs are required for the public key. The notion of is a fundamental property which makes HPS-based encryption compatible with the ECDSA signature. Finally, the our HPS-based two-party ECDSA scheme and its analysis of security are detailed in Section 3.3.

Concrete instantiation Towards efficient solutions, we then show in Section 3.4 how to instantiate our (homomorphic) HPS construction using class groups of imaginary quadratic fields. We here recall only the relevant points of the CL framework and we refer to Section 2.2 for details on the CL scheme, complexity of the attacks on the discrete logarithm in the class group, reference to works about the choice of the discriminant. Principally, the resulting schemes benefit from (asymptotically) shorter keys. Moreover, interest in the area has been renewed in recent years as it allows versatile and efficient solutions such as encryption switching protocols [CIL17], inner product functional encryption [CLT18a] or verifiable delay functions [BBBF18, Wes19a]. Concretely, the main feature of the Castagnos and Laguillaumie cryptosystem (CL) and its variants is that they rely on the existence of groups with associated easy discrete log subgroups, for which hard decision problems can be defined. In CL there exist a cyclic group $G := \langle g \rangle$ of order qs where s is unknown, q is prime and $\gcd(q, s) = 1$, and an associated cyclic subgroup of order q, $F := \langle f \rangle$. Denoting with $G^q := \langle g_q \rangle$ the subgroup of q-th powers in G (of unknown order s), one has $G = F \times G^q$, and one can define an hard subgroup membership problem. This allows to build a linearly homomorphic public key encryption scheme where the plaintext space is $\mathbf{Z}/q\mathbf{Z}$ for arbitrarily large q. This also means that if one uses the very same q underlying the ECDSA signature, one gets a concrete instantiation of our general protocol which naturally avoids all the inefficiencies resulting from N and q being different. Finally, details on the efficiency comparisons are given in Section 3.5.

A zero knowledge for CL We remark that, similarly to Lindell's solution, our schemes require P_2 to hold an encryption $\mathsf{Enc}(x_1)$ of P_1's share of the secret key. This imposes a somewhat heavy key registration phase in which P_1 has to prove, among other things, that the public key is correctly generated. Fortunately, in our setting we can achieve this without resorting to expensive range proofs, however other difficulties arise: first of all, we work with groups of unknown order and second we cannot assume that all ciphertexts are valid (*i.e.*, actually encrypt a message). The latter derives from the non surjectivity of CL scheme, i.e. not all the ciphertext derives from a couple message/randomness[2]. We address this by developing a new proof that solves both issues at the same time. Our proof is inspired by the Girault *et al.* [GPS06] identification protocol but introduces new ideas to adapt it to our setting and to make it a proof of knowledge. As for Lindell's case, it uses a binary challenge, which implies that the proof has to be repeated t times to get soundness error 2^t.

Improvements on the zero-knowledge In our work [CCL+19], initially we did not propose a solution for enlarging the challenge space of our proof for proving well-formness of ciphertext. This is fundamental to reduce the number of repetitions of the proof and to lead to substantial efficiency improvements. Fortunately, in 2020 we proposed a protocol for threshold ECDSA ([CCL+20]) built from class groups and, motivated by the expensive costs of proofs with CL and a larger number of players that have to run those proofs, we proposed a new very efficient ZKAoK for well-formness of CL ciphertexts. The same idea, with some constraints, can be applied to the zero-knowledge proof we use in our two party construction. We will introduce the improvements on our proof at the end of Subsection 3.4.2, but we will look at it in detail in Chapter 4.

3.1 Ideal functionalities

Notations. See Notation at the beginning of Chapter 1.

The elliptic curve digital signature algorithm. See the description of ECDSA from Subection 1.3.1

Two-party ECDSA. This consists of the following interactive protocols:

$\mathsf{IKeyGen}\langle(\mathbb{G}, q, P); (\mathbb{G}, q, P)\rangle \to \langle(x_1, Q); (x_2, Q)\rangle$ such that $\mathsf{KeyGen}(\mathbb{G}, q, P) \to (x, Q)$ where x_1 and x_2 are shares of x.

$\mathsf{ISign}\langle(x_1, m); (x_2, m)\rangle \to \langle\emptyset; (r, s)\rangle$ or $\langle(r, s); \emptyset\rangle$ or $\langle(r, s); (r, s)\rangle$ where $\emptyset$ is the empty output, signifying that one of the parties may have no output and $\mathsf{Sign}(x, m) \to (r, s)$.

The verification algorithm is non interactive and identical to that of ECDSA.

Interactive zero-knowledge proof systems. A zero-knowledge proof system (P, V) for a language $\mathcal{L}$ is an interactive protocol between two probabilistic algorithms: a prover P and a polynomial-time verifier V. Informally P, detaining a witness for a given statement, must convince V that it is true without revealing anything other to V. A more formal definition is provided in Section 1.5.

Ideal functionalities. In Chapter 1 we discussed the simulation real/ideal paradigm and we saw what it is an hybrid model and how a proof by simulation intuitively works. Our protocol for two-party ECDSA is proved secure in this model and before going

[2]For Paillier's scheme, used in [Lin17], this is not an issue: every ciphertext is valid

- Upon receiving $(\mathsf{prove}, sid, x, w)$ from a party P_i (for $i \in \{1,2\}$): if $(x,w) \notin \mathsf{R}$ or sid has been previously used then ignore the message. Otherwise, send (proof, sid, x) to party P_{3-i}

Figure 3.1: The $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R}}$ functionality

deeply in the protocol description and its security proof, we have to describe functionalities that compose our model. We will use ideal functionalities for commitments, zero-knowledge proofs of knowledge (ZKPoK) and commitments to non interactive zero-knowledge (NIZK) proofs of knowledge between two parties P_1 and P_2.
The intuition behind these ideal functionalities can be described taking the case of ZKPoK as an example. It is straightforward how adapting to other functionalities. Consider the case of a prover P_i with $i \in \{1,2\}$ who wants to prove the knowledge of a witness w for an element x which ensures that (x,w) satisfy the relation R, *i.e.* $(x,w) \in \mathsf{R}$. In an ideal world we can imagine an honest and trustful third party, which can communicate with both P_i and P_{3-i}. In this ideal scenario, P_i could give (x,w) to this trusted party, the latter would then check if $(x,w) \in \mathsf{R}$ and tell P_{3-i} if this is true or false. In the real world we do not have such trusted parties and must substitute them with a cryptographic protocol between P_1 and P_2. Roughly speaking, the ideal/real paradigm requires that whatever information an adversary $\mathcal{A}$ (corrupting either P_1 or P_2) could recover in the real world, it can also recover in the ideal world. The trusted third party can be viewed as the ideal functionality and we denote it by $\mathcal{F}$. If some protocol satisfies the above property regarding this functionality, we call it secure.

Formally, we denote $\mathcal{F}\langle x_1; x_2\rangle \to \langle y_1; y_2\rangle$ the joint execution of the parties via the functionality $\mathcal{F}$, with respective inputs x_i, and respective private outputs at the end of the execution y_i. Each transmitted message is labelled with a session identifier sid, which identifies an iteration of the functionality. The *ideal ZKPoK functionality* [HL10, Section 6.5.3], denoted $\mathcal{F}_{\mathsf{zk}}$, is defined for a relation R by $\mathcal{F}_{\mathsf{zk}}\langle (x,w); \emptyset\rangle \to \langle \emptyset; (x, \mathsf{R}(x,w))\rangle$, where $\emptyset$ is the empty output, signifying that the first party receives no output (cf. Figure 3.1).

The *ideal commitment functionality*, denoted $\mathcal{F}_{\mathsf{com}}$, is depicted in Figure 3.2. We also use an ideal functionality $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R}}$ for *commitments to NIZK proofs* for a relation R (cf. Figure 3.3). Essentially, this is a commitment functionality, where the committed value is a NIZK proof.

The ideal functionality for two-party ECDSA. The ideal functionality $\mathcal{F}_{ECDSA}$ (cf. Figure 3.4) consists of two functions: a key generation function, which is called once, and a signing function, called an arbitrary number of times with the generated keys.

- Upon receiving $(\mathsf{commit}, sid, x)$ from party P_i (for $i \in \{1,2\}$), record (sid, i, x) and send $(\mathsf{receipt}, sid)$ to party P_{3-i}. If some $(\mathsf{commit}, sid, *)$ is already stored, then ignore the message.
- Upon receiving $(\mathsf{decommit}, sid)$ from party P_i , if (sid, i, x) is recorded then send $(\mathsf{decommit},\ sid, x)$ to party P_{3-i}.

Figure 3.2: The $\mathcal{F}_{\mathsf{com}}$ functionality

- Upon receiving $(\mathsf{com} - \mathsf{prove}, sid, x, w)$ from a party P_i (for $i \in \{1,2\}$): if $(x, w) \notin \mathsf{R}$ or sid has been previously used then ignore the message. Otherwise, store (sid, i, x) and send $(\mathsf{proof} - \mathsf{receipt}, sid)$ to P_{3-i}.
- Upon receiving $(\mathsf{decom} - \mathsf{proof}, sid)$ from a party P_i (for $i \in \{1,2\}$): if (sid, i, x) has been stored then send $(\mathsf{decom} - \mathsf{proof}, sid, x)$ to P_{3-i}

Figure 3.3: The $\mathcal{F}^{\mathsf{R}}_{\mathsf{com-zk}}$ functionality

3.2 Background on HPS, a HPS-based PKE scheme and ECDSA-friendly HPS

In this section, we recall Hash Proof System and we present the properties required to build a homomorphic encryption scheme from them. Next, we will look at the additional properties which adapt the construction to ECDSA parameters. In Subsection 3.2.1, we first recall the HPS framework from [CS02], before defining basic properties required for our construction in Subsection 3.2.2. In particular, to guarantee correctness of the protocol (in order for party P_2 to be able to perform homomorphic operations on ciphertexts provided by P_1, which are encryptions of elements in $\mathbf{Z}/q\mathbf{Z}$) the HPS must be homomorphic; and for security to hold against malicious adversaries we also require that the subset membership problem underlying the HPS be hard, and that the HPS be smooth. We note that diverse group systems (often used as a foundation for constructions of HPSs) imply all the aforementioned properties. In Subsection 3.2.3 we present homomorphic properties of HPS. Such HPSs define linearly homomorphic encryption schemes as described in Subsection 3.2.4. In Subsection 3.2.5, we summarise all the properties required to build a simulation secure two party ECDSA from hash proof systems. This includes two new definitions. The first, decomposability, imposes some requirement on the structure of the HPS. It holds for a variety of HPSs (such as the Decision Diffie Hellman based HPS of [CS02], and the class group based HPS presented in Section 3.4). The second, called the double encoding assumption, is slightly more ad-hoc, and is necessary to capture the information leaked from the public parameters of centralised ECDSA. We also back that this assumption seems hard. Finally, before presenting the overall two party signing protocol in next Section 3.3 and proving its security, we describe zero-knowledge proofs

Consider an Elliptic-curve group $\mathbb{G}$ of order q with generator a point P, then:

- Upon receiving $\mathsf{KeyGen}(\mathbb{G}, P, q)$ from both P_1 and P_2:

    1. Generate an $ECDSA$ key pair (Q, x), where $x \xleftarrow{\$} (\mathbf{Z}/q\mathbf{Z})^*$ is chosen randomly and Q is computed as $Q \leftarrow x \cdot P$.
    2. Choose a hash function $H_q : \{0,1\}^* \rightarrow \{0,1\}^{\lfloor \log |q| \rfloor}$, and store $(\mathbb{G}, P, q, H_q, x)$.
    3. Send Q (and H_q) to both P_1 and P_2.
    4. Ignore future calls to KeyGen.

- Upon receiving $\mathsf{Sign}(sid, m)$ from both P_1 and P_2, where keys have already been generated from a call to Keygen and sid has not been previously used, compute an $ECDSA$ signature (r, s) on m, and send it to both P_1 and P_2. (To do this, choose a random $k \xleftarrow{\$} (\mathbf{Z}/q\mathbf{Z})^*$, compute $(r_x, r_y) \leftarrow k \cdot P$ and set $r \leftarrow r_x \mod q$. Finally, compute $s \leftarrow k^{-1}(H_q(m) + rx)$ and output (r, s).)

Figure 3.4: The $\mathcal{F}_{ECDSA}$ functionality

(ZKP) related to the aforementioned HPSs, and justify that they fulfil the $\mathcal{F}_{\mathsf{com}}/\mathcal{F}_{\mathsf{com\text{-}zk}}$ hybrid model.

3.2.1 Defining HPS and PHF

To define a HPS, the initial ingredients we need to consider are a language $\mathcal{L}$, an instance of a *subset membership problem* and a *projective hash family* (PHF). Informally, a subset membership problem consists in deciding if a given random element from some set S belongs to a certain subset S' of S, where S and S' are specified by the instance of the problem. Formally,

Definition 3.2.1. Let $\mathcal{X}$ be a set of words, $\mathcal{L} \subset \mathcal{X}$ be an NP language such that $\mathcal{L} := \{x \in \mathcal{X} \mid \exists w \in \mathcal{W} : (x, w) \in \mathsf{R}\}$ where R is the relation defining the language, $\mathcal{L}$ be the language of true statements in $\mathcal{X}$, and for $(x, w) \in \mathsf{R}$, let $w \in \mathcal{W}$ be a witness for $x \in \mathcal{L}$. The set $(\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R})$ defines an instance of a *subset membership problem*, i.e. the problem of deciding if an element $x \in \mathcal{X}$ is in $\mathcal{L}$ or in $\mathcal{X} \backslash \mathcal{L}$.

We denote $\mathsf{Gen}_{\mathcal{SM}}$ an instance generator, i.e. an algorithm which on input a parameter 1^λ, outputs the description $(\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R})$ of a subgroup membership problem.

As pointed above, the second ingredient we require is a projective hash family. The following definitions come from [CS02] and their re-adaptation in the context of [CCL+19].

Definition 3.2.2 (Hash family, [CS02]). Let $\mathcal{X}$ and Π be finite, non-empty sets. Let K_{hk} be a set, $\mathcal{D}_{\mathsf{hk}}$ be a distribution over K_{hk} and hk be a key sampled in K_{hk} with distribution $\mathcal{D}_{\mathsf{hk}}$. We call K_{hk} an *hash key space* and hk a *secret hash key*. Let $\{\mathsf{hash}_{\mathsf{hk}}\}_{\mathsf{hk} \in K_{\mathsf{hk}}}$ be a collection of functions indexed by hk, so that for every $\mathsf{hk} \in K_{\mathsf{hk}}$, $\mathsf{hash}_{\mathsf{hk}}$ is a function from $\mathcal{X}$ into Π (note that we may have $\mathsf{hash}_{\mathsf{hk}} = \mathsf{hash}_{\mathsf{hk}'}$ for $\mathsf{hk} \neq \mathsf{hk}'$).
We call $(\{\mathsf{hash}\}_{\mathsf{hk} \in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \Pi)$ a *hash family*, and each $\mathsf{hash}_{\mathsf{hk}}$ a *hash function*.

Definition 3.2.3 (Projective Hash Families). Let $(\{\mathsf{hash}\}_{\mathsf{hk}\in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \Pi)$ be a hash family and consider a key generation algorithm $\mathsf{PHF.KeyGen}$ which outputs a secret hashing key hk sampled from distribution of hashing keys $\mathcal{D}_{\mathsf{hk}}$ over a hash key space K_{hk}. Let $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$ be a public *projection key* in *projection key space* K_{hp}. A *projective hash family* PHF is defined by a tuple $\mathsf{PHF} := (\{\mathsf{hash}_{\mathsf{hk}}\}_{\mathsf{hk}\in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \mathcal{L}, \Pi, K_{\mathsf{hp}}, \mathsf{projkg})$, where:

- K_{hk} and K_{hp} are defined as above
- $\mathcal{X}$ and $\mathcal{L}$ are defined as in Definition 3.2.1
- $\mathsf{projkg} : K_{\mathsf{hk}} \mapsto K_{\mathsf{hp}}$ is an efficient auxiliary function
- $\mathsf{hash}_{\mathsf{hk}} : \mathcal{X} \mapsto \Pi$ is an hash function defined by the secret hashing key hk and Π is its range as in Definition 3.2.2

An Hash Proof System (HPS) associates an instance of a subset membership problem to a projective hash family. In addition, a HPS provides efficient algorithms to compute basic operations: sampling an element $\mathsf{hk} \in K_{\mathsf{hk}}$, computing $\mathsf{projkg}(\mathsf{hk}) \in K_{\mathsf{hp}}$ given $\mathsf{hk} \in K_{\mathsf{hk}}$ and computing $\mathsf{hash}_{\mathsf{hk}}(x)$ given $\mathsf{hk} \in K_{\mathsf{hk}}$ and $x \in \mathcal{X}$. $\mathsf{hash}_{\mathsf{hk}}(x)$ can be computed essentially in two ways, which depends on $x \in \mathcal{X}$ or $x \in \mathcal{L}$. Computing $\mathsf{hash}_{\mathsf{hk}}(x)$ given $\mathsf{hk} \in K_{\mathsf{hk}}$ and $x \in \mathcal{X}$ is called a *private evaluation* and it can be done for every $x \in \mathcal{X}$. If $x \in \mathcal{L}$, there exists a witness $w \in \mathcal{W}$ such that $(x, w) \in \mathsf{R}$. Computing $\mathsf{projhash}_{\mathsf{hp}}(x, w) := \mathsf{hash}_{\mathsf{hk}}(x)$ for $(x, w) \in \mathsf{R}$ given $\mathsf{hp} \in K_{\mathsf{hp}}$ is a *public evaluation.* Since hk defines a hash function $\mathsf{hash}_{\mathsf{hk}} : \mathcal{X} \mapsto \Pi$ and hp, public and private evaluation coincide if both restricted to $x \in \mathcal{L}$, and then $\mathsf{projhash}_{\mathsf{hp}}(x, w)$ is correctly defined for $(x, w) \in \mathsf{R}$.

Several properties can be satified by Hash Proof Systems, and in particular by the PHF on which they are built upon. In this subsection, we introduce and discuss standard properties for PHFs and HPSs and other useful ones for the usage of HPS in our two-party construction built upon them. In particular, the aim of this section is to analyze the necessary properties to adapt Hash Proof Systems to ECDSA. To give a brief introduction to what we need, first players in our scheme works homomorphically on encrypted messages, then to use HPS to encrypt messages we require some homomorphic properties. Second, we can construct an ECDSA-friendly PHF such that the order of message space is the same as the order of the Elliptic Curve and then extending to ECDSA friendly HPS when putting together the resulting PHF with a subset membership problem. A description of some standard hard assumptions for PHF are given in Subsection 3.2.2. In Subsection 3.2.3 we discuss homomorphic properties of PHF. In Subsection 3.2.4 we present the resulting encryption scheme from HPS. Finally, a detailed explanation about the construction of an ECDSA-friendly PHF is given in Subsection 3.2.5.

3.2.2 Hard assumptions for PHF

δ_s-smoothness. The standard *smoothness* property of a PHF requires that for any element $x \notin \mathcal{L}$ outside the language and for each secret key hk, the value $\mathsf{hash}_{\mathsf{hk}}(x)$ is uniformly distributed knowing $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$. As introduced in the previous subsection, we require that the order q of the Elliptic curve is part of the parameters and then, that the PHF is adapted to q. First of all, we consider sets $\mathcal{X}$ and Π that are finite abelian groups. The space where messages are encoded is a subgroup F of Π such that F is cyclic of order q and its generator is f. Of course, a necessary condition for the existence of such a group is that the order of Π is a multiple of q. For now suppose this is the case. Once we have a cyclic subgroup F of order q, it is clear how a correspondence one-by-one with the elliptic curve $\mathbb{G}$ of ECDSA may be built. Indeed, for $m \in \mathbf{Z}/q\mathbf{Z}$ the enconding of m in F is f^m, which is an induced efficient isomorphism. The inverse isomorphism $\log_f : f^m \mapsto m$ must also be efficiently computable. We assume that the discrete logarithm is easy in F. Even if it seems a non trivial construction, there exists examples of a group with a subgroup of given order q in which it is easy to compute the discrete logarithm, while it is not easy to compute it outside the subgroup. The encryption scheme CL built from class groups that we saw in Chapter 2 is a clear example of how to generate a group with a subgroup where discrete logarithm is easy and such that this subgroup is the space of encoded messages. We will consider a instantiation of our two-party scheme with CL in the dedicated Section 3.4.

Remark 11. In some instantiations $F = \Pi$, but F may also be a strict subgroup of Π. If we are in the latter case, i.e. $F \subsetneq \Pi$, we require smoothness over $\mathcal{X}$ on F ([CS02, Subsection 8.2.4]).

In the above description, smoothness property takes in account the uniformity of $\mathsf{hash}_{\mathsf{hk}}(x)$ for $x \notin F$. We introduced informally the smoothness property of an PHF, formally it is as follows:

Definition 3.2.4. A projective hash family is δ_s-smooth over $\mathcal{X}$ on F if for any $x \in \mathcal{X}\backslash\mathcal{L}$ (i.e. $x \notin \mathcal{X}$), a random $\pi \in F$ and a randomly sampled hashing key $\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$, the distributions $\mathcal{U} := \{x, \mathsf{projkg}(\mathsf{hk}), \mathsf{hash}_{\mathsf{hk}}(x) \cdot \pi\}$ and $\mathcal{V} := \{x, \mathsf{projkg}(\mathsf{hk}), \mathsf{hash}_{\mathsf{hk}}(x)\}$ are δ_s-close.

$\delta_{\mathcal{L}}-$**hard subset membership problem.** For security to hold $(\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R})$ must be an instance of a hard subset membership problem, i.e. no polynomial time algorithm can distinguish random elements of $\mathcal{X}\backslash\mathcal{L}$ from those of $\mathcal{L}$ with significant advantage. We introduced this problem as part of Definition 3.2.1, and a formal definition is:

Definition 3.2.5. Consider a positive integer λ. We say $\mathsf{Gen}_{\mathcal{SM}}$ is a generator for a $\delta_{\mathcal{L}}(\lambda)$-hard subset membership problem if for any $(\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R})\leftarrow \mathsf{Gen}_{\mathcal{SM}}(1^\lambda)$, $\delta_{\mathcal{L}}$ is the maximal advantage of any polynomial time adversary in distinguishing random elements of $\mathcal{X}\backslash\mathcal{L}$ from those of $\mathcal{L}$. For conciseness, we often simply say $(\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R})$ is a $\delta_{\mathcal{L}}-$hard subset membership problem.

3.2.3 Homomorphic Properties

In the introduction we talked about proposed solutions for building a distributed DSA/ECDSA protocol and we saw that a strategy is using an homomorphic encryption scheme as a building block. We follow the same direction, but to do that our PHF should be homomorphic too. This is a necessary property, but alone is not enough. Indeed, we have to define the remaining properties our PHF should have to be adaptable to ECDSA, or simply to be ”ECDSA-friendly”. In this subsection we continue in defining properties that are necessary for our PHF.

Linearly homomorphic PHF. In order for the homomorphic operations performed to hold in the two party ECDSA protocol, we require that the projective hash family also be homomorphic as defined in [HO09].

Definition 3.2.6 ([HO09]). The projective hash family $\mathsf{PHF} := (\{\mathsf{hash}_{\mathsf{hk}}\}_{\mathsf{hk} \in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \mathcal{L}, \Pi, K_{\mathsf{hp}}, \mathsf{projkg})$ is homomorphic if $(\mathcal{X}, \star)$ and $(\Pi, \cdot)$ are groups, and for all $\mathsf{hk} \in K_{\mathsf{hk}}$, and $u_1, u_2 \in \mathcal{X}$, we have $\mathsf{hash}_{\mathsf{hk}}(u_1) \cdot \mathsf{hash}_{\mathsf{hk}}(u_2) = \mathsf{hash}_{\mathsf{hk}}(u_1 \star u_2)$, that is to say $\mathsf{hash}_{\mathsf{hk}}$ is a homomorphism for each hk.

Linearly homomorphic properties are also valid for the public projective hash function. Indeed, for $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$ and for elements $u_1, u_2 \in \mathcal{L}$:

$$\begin{aligned}\mathsf{projhash}_{\mathsf{hp}}(u_1, w_1) \cdot \mathsf{projhash}_{\mathsf{hp}}(u_2, w_2) &= \mathsf{hash}_{\mathsf{hk}}(u_1) \cdot \mathsf{hash}_{\mathsf{hk}}(u_2) \\ &= \mathsf{hash}_{\mathsf{hk}}(u_1 \star u_2) = \mathsf{projhash}(u_1 \star u_2, w),\end{aligned}$$

for some witness w.

Homomorphically extended PHF. We notice that the co-domain of projkg, which specifies the set of valid projection keys, may not be efficiently recognisable. Though we do not require – as did the protocol of [Lin17] – a costly ZKPoK of the secret key associated to the public key, it is essential in our protocol that even if a public key is chosen maliciously (i.e. there does not exist $\mathsf{hk} \in K_{\mathsf{hk}}$ such that $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$, which may go unnoticed to honest parties in the protocol), the homomorphic properties of the public projective hash function still hold. We thus require that the co-domain of projkg, which defines valid projection keys, be contained in an *efficiently recognisable* space K'_{hp}, such that for all $\mathsf{hp}' \in K'_{\mathsf{hp}}$, $\mathsf{hash}_{\mathsf{hp}'}$ is a homomorphism (respectively to its inputs in $\mathcal{L}$).

Definition 3.2.7 (Homomorphically extended PHF). We say that the projective hash family $\mathsf{PHF} := (\{\mathsf{hash}_{\mathsf{hk}}\}_{\mathsf{hk} \in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \mathcal{L}, \Pi, K_{\mathsf{hp}}, K'_{\mathsf{hp}}, \mathsf{projkg})$ is homomorphically extended if $\mathsf{PHF} := (\{\mathsf{hash}_{\mathsf{hk}}\}_{\mathsf{hk} \in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \mathcal{L}, \Pi, K_{\mathsf{hp}}, \mathsf{projkg})$ is a homomorphic PHF and that there exists an efficiently recognizable space $K'_{\mathsf{hp}} \supseteq K_{\mathsf{hp}}$ such that for any $\mathsf{hp}' \in K'_{\mathsf{hp}}$, the function $\mathsf{projhash}_{\mathsf{hp}'}$ is a homomorphism (respectively to its inputs in $\mathcal{L}$).

Key homomorphic PHF. Our security proofs also requires that projective hash families are linearly homomorphic with respect to the hashing keys.

Definition 3.2.8. A projective hash family PHF is *key homomorphic* if K_{hk} is a cyclic additive Abelian group, Π is a multiplicative finite Abelian group; and $\forall x \in \mathcal{X}$ and $\forall \mathsf{hk}_0, \mathsf{hk}_1 \in K_{\mathsf{hk}}$, it holds that $\mathsf{hash}(\mathsf{hk}_0, x) \cdot \mathsf{hash}(\mathsf{hk}_1, x) = \mathsf{hash}(\mathsf{hk}_0 + \mathsf{hk}_1, x)$.

Remark 12. We note that for correctness and security of our construction, it is not necessary that K_{hk} be cyclic. However imposing this greatly simplifies presentation. It can be verified that our results hold even without this requirement on K_{hk}. We also point out that if one does not require K_{hk} to be cyclic, the resulting definition is that of [BBL17, Definitions 6 and 7].

3.2.4 Resulting Encryption Scheme

We use the standard chosen plaintext attack secure encryption scheme which results from a HPS [CS02]. It consists of the following algorithms:

KeyGen The key generation algorithm runs $\mathsf{PHF.KeyGen}$ and sets $\mathsf{hk} \in K_{\mathsf{hk}}$ as the secret key, and the associated public key is $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$

Enc The encryption algorithm takes in input a plaintext message m in $\mathbf{Z}/q\mathbf{Z}$ and a public key hp output by the key generation. m is encrypted by sampling a random pair $(u, w) \in \mathsf{R}$ and computing $\mathsf{Enc}(\mathsf{hp}, m) \leftarrow (u, \mathsf{projhash}_{\mathsf{hp}}(u, w)f^m)$. We sometimes use the notation $\mathsf{Enc}((u, w); (\mathsf{hp}, m))$ to specify the randomness used in the encryption algorithm

Dec The decryption algorithm takes in input a ciphertext $(u, e) \in \mathcal{X} \times \Pi$ and a secret key hk. To decrypt $c = (u, e)$ it computes:

$$\mathsf{Dec}(\mathsf{hk}, (u, e)) \leftarrow \log_f(e \cdot \mathsf{hash}_{\mathsf{hk}}(u)^{-1})$$

If $e \cdot \mathsf{hash}_{\mathsf{hk}}(u)^{-1} \notin F = \langle f \rangle$, the decryption algorithm returns the special error symbol $\perp$.

Theorem 3.2.1. The scheme is semantically secure under chosen plaintext attacks assuming both the smoothness of the HPS and the hardness of the underlying subset membership problem.

Homomorphic properties. Given encryptions (u_1, e_1) and (u_2, e_2) of respectively m_1 and m_2, and an integer a, we require that there exist two procedures $\mathsf{EvalSum}$ and $\mathsf{EvalScal}$ such that

$$\mathsf{Dec}(\mathsf{hk}, \mathsf{EvalSum}(\mathsf{hp}, (u_1, e_1), (u_2, e_2))) = m_1 + m_2$$
$$\mathsf{Dec}(\mathsf{hk}, \mathsf{EvalScal}(\mathsf{hp}, (u_1, e_1), a)) = a \cdot m_1$$

which is the case if the projective hash family is homomorphic. This definition is in line with Definition 1.2.4.

Invalid ciphertexts. In the description of the HPS based encryption scheme, it is important to note that the decryption of (u, e) ends with a valid message $m \neq \perp$ if $e \cdot \mathsf{hash}_{\mathsf{hk}}(u)^{-1} \in F$, i.e. if it is possible to compute the discrete logarithm in base F of $e \cdot \mathsf{hash}_{\mathsf{hk}}(u)^{-1}$. Indeed, if before the computation of the logarithm the element $M := e \cdot \mathsf{hash}_{\mathsf{hk}}(u)^{-1}$ does not belong to F, then it does not exist an exponent α such that $M = f^\alpha$. However, even if the encryption of a plaintext m is done taking a couple $(u, w) \in \mathsf{R}$, i.e. $u \in \mathcal{L}$, it is possible to encrypt m in a decryptable ciphertext using the secret key hk and $u \in \mathcal{X} \backslash \mathcal{L}$ from the private evaluation procedure. From this observation, we define the notion of *invalid* ciphertexts as these will be useful in our security proofs.

Definition 3.2.9 (Invalid ciphertexts). A ciphertext is said to be *invalid* if it is of the form $(u, e) := (u, \mathsf{hash}_{\mathsf{hk}}(u)f^m)$ where $u \in \mathcal{X} \backslash \mathcal{L}$, i.e. if it comes from an element which is not in the language.

Notice that one can compute an invalid ciphertext given the secret hashing key hk, but not the public projection key hp, since $u \notin \mathcal{L}$ and then it does not exist a witness w for u. Moreover, the decryption algorithm applied to (u, e) with secret key hk recovers a valid message m, then an invalid ciphertext is indistinguishable of a valid one under the hardness of the subset membership problem.

Homomorphic properties over invalid ciphertexts. It is easy to verify that homomorphic operations hold even if a ciphertext is invalid, whether this be between two invalid ciphertexts or between a valid and invalid ciphertext. This is true since the homomorphic properties we required of the PHF hold over the whole group $\mathcal{X}$ (and not only in $\mathcal{L}$).

3.2.5 ECDSA friendly Projective Hash Families

We defined HPS, PHF and part of the properties we need to build our two-party signature scheme. We said that the security of the resulting protocol relies on several assumptions, where two of them are the δ_s−smoothness and the $\delta_{\mathcal{L}}$−hard subset membership problem, and the encryption scheme we consider must be linearly homomorphic with respect to both group elements and keys. The last step to complete the description of the properties a PHF has to satisfy for our construction is looking at how to adapt it to ECDSA. In addition, in this section we formalise new properties for PHFs which contribute to the clarity of our security proofs. A fundamental property we require is the *decomposability* of element in $\mathcal{X}$. To understand the decomposability property, it is enough to think that in general in a group $\mathcal{X}$ there is not a way to represent elements in a specific form. For example, suppose that $x \in \mathcal{X} = G \times H$, where $\times$ denotes the direct product of two subgroups G and H of $\mathcal{X}$. It is clear that $x = a \cdot b$, where $a \in G$ and $b \in H$ in a unique way. Then, if the group has a known specific form, even its elements have a known specific form.

(Υ, F)-Decomposability. We informally introduced the idea of a decomposable PHF at the beginning of this subsection. In this paragraph we formalize this property. The decomposability property states that the domain $\mathcal{X}$ of hash is the direct product of the language $\mathcal{L}$ and some cyclic subgroup of $\mathcal{X}$. Since – given the projection key – one can publicly compute hash values over elements in $\mathcal{L}$, decomposability allows us to have a clear separation between the part of a given hash value which is predictable (whose preimage is in $\mathcal{L}$), and the part which appears random. Though the definition is new, many well known PHFs arising from groups satisfy this property (e.g. the original DDH and DCR based PHFs of [CS02]).

Definition 3.2.10 (Decomposability). Let $\mathcal{SM} := (\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R})$ be a subgroup membership problem, and consider the associated projective hash family PHF, such that the co-domain Π of hash is a finite Abelian group which contains a cyclic subgroup F. We say that PHF is (Υ, F)-decomposable if there exists $\Upsilon \in \mathcal{X}$ s.t.:

- $\mathcal{X}$ is the direct product of $\mathcal{L}$ and $\langle \Upsilon \rangle$;
- $\forall \mathsf{hk} \in K_{\mathsf{hk}}$, $\mathsf{hash}_{\mathsf{hk}}(\Upsilon) \in F$.

Remark 13. In what follows F is a cyclic subgroup of Π, generated by f and of prime order q, and the PHFs we consider are homomorphic and key homomorphic. For $\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$, if $\mathsf{hash}_{\mathsf{hk}}(\Upsilon) = 1$ smoothness does not hold, hence we assume this is not the case. Throughout the rest of the chapter, we denote Ψ the considered generator of K_{hk}, which satisfies $\mathsf{hash}_{\Psi}(\Upsilon) = f$. Consequently, for any $\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$, where $\mathsf{hk} = c \cdot \Psi$ (for some $c \in \mathbf{Z}$), and for any $y = \Upsilon^b$ (for some $b \in \mathbf{Z}$), one has $\mathsf{hash}_{\mathsf{hk}}(y) = f^{bc}$.

Finally, the third security assumption[3] we require is the difficulty of the *double encoding problem.* Unlike smoothness and the hard subset membership problem, the double encoding problem is not a standard property of PHF, but it is introduced in our context. A definition of this problem and the motivation behind its difficulty are given in the following two paragraphs.

The Double Encoding Problem. A two-party signing protocol involves two parties that interactively compute the signature of a message. What can happen is, essentially, that the protocol aborts or ends successfully. Anyway, a party can behave maliciously to obtain information leaked by the interactive protocol. To ensure security of our protocol, we need a notion which deals with these leaked information revealed by the situation in which the protocol concludes. To understand better which situation can occur, in the overall protocol some of the steps are:

- P_1 sends an encryption c_1 of its secret share x_1 of what will be the secret signing key, along with the elliptic curve point $Q_1 := x_1 P$ to P_2.
- P_2 sends a ciphertext c_2 (which should be homomorphically computed from c_1) in response to P_1.

If at the end of the last step the protocol stops, the HPS's smoothness would suffice to prove the security of the protocol, since the encrypted value is perfectly masked. The next step for P_1 is decrypting the received value, i.e. c_2 which comes from P_2. This is the last step to compute the overall signature. The problem we focus on the successful ending or abort of the protocol. Indeed, a malicious P_2 would send an encryption c_2 that reveals one bit of information from the fact the protocol aborts or not. It is thus necessary to ensure that a corrupted P_2 can not devise malformed ciphertexts which allows it to distinguish real and ideal executions. Indeed, a distinction between real and ideal executions could be caused by a different ending in the two executions, one successful and the other aborting. The idea behind the double encoding assumption is motivated by a potential point of failure in the security proof. Indeed, in the security proof there is an hard case to cover where it is possible to distinguish between real and ideal executions without assuming the hardness of the double encoding. Fortunately, under the hardness of the double encoding assumption, the proof works. As said above, next paragraph is dedicated on the hardness of this assumption, where its validity is motivated.
Having given a motivation behind the necessity of a new definition, we can formally define it. To this end, we require that – given a one way function (OWF) evaluated in $x \in \mathbf{Z}/q\mathbf{Z}$ (in our protocol this is the elliptic curve point $Q := xP$) – no polynomial time adversary can produce two invalid encryptions of x. Though the following assumption may seem quite ad-hoc, in the following paragraph we motivate that intuitively it seems a least as hard as inverting the one way function.

Definition 3.2.11 (Double encoding assumption [CCL+19]). Let $\lambda \in \mathbf{N}$ be a security parameter and q be a λ-bit prime. Consider a collection of one way functions sampled from an efficient algorithm Gen_{OW}, such that for $h \hookleftarrow \mathsf{Gen}_{OW}(1^\lambda, q)$, h has input space $\mathbf{Z}/q\mathbf{Z}$ (and arbitrary output space). Let $\mathsf{Gen}_{\mathcal{SM}}$ be a subset membership problem generator

[3]The other two assumption are the smoothness and the hard subset membership.

such that the resulting projective hash family PHF is (Υ, F)-decomposable, for F of prime order q. The *double encoding* (DE) problem is $\delta_{\mathsf{DE}}(\lambda)$-hard for ($\mathsf{Gen}_{\mathcal{SM}}$, Gen_{OW}) if, given $(\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R}) \hookleftarrow \mathsf{Gen}_{\mathcal{SM}}(1^\lambda, q)$, $h \leftarrow \mathsf{Gen}_{OW}(1^\lambda, q)$, and $y := h(x)$ for a randomly sampled $x \in \mathbf{Z}/q\mathbf{Z}$, no algorithm $\mathcal{A}$ running in time polynomial in λ can output two invalid encryptions of x, with probability greater than $\delta_{\mathsf{DE}}(\lambda)$. More precisely,

$$\begin{aligned}\delta_{\mathsf{DE}}(\lambda) \geqslant \Pr\Big[&u_1, u_2, u_2u_1^{-1} \in \mathcal{X} \setminus \mathcal{L} \text{ and } \mathsf{hp} = \mathsf{projkg}(\mathsf{hk}) : \\ &\mathcal{SM} \hookleftarrow \mathsf{Gen}_{\mathcal{SM}}(1^\lambda, q), h \hookleftarrow \mathsf{Gen}_{OW}(1^\lambda, q), x \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z},\ y \leftarrow h(x), \\ &(\mathsf{hp}, (u_1, \mathsf{hash}_{\mathsf{hk}}(u_1)f^x), (u_2, \mathsf{hash}_{\mathsf{hk}}(u_2)f^x)) \leftarrow \mathcal{A}(h, \mathcal{SM}, y)\Big].\end{aligned}$$

The DE assumption holds if for any λ-bit prime q, δ_{DE} is negligible in λ.

On the hardness of the double encoding problem. In this paragraph we motivate the hardness of our new assumption. We premise that breaking the double encoding assumption seems to be a similar problem to inverting a one-way function. We start giving a look at the relation between an HPS and a OWF. If the HPS and the OWF arise from independent structures, it seems unlikely that one could solve the DE problem without breaking the one wayness of h, and subsequently computing two invalid encodings of x. Even if they arise from the same structures, it is not clear how one problem implies the other. Of course we need to pay attention on the considered OWF, indeed if for example the OWF were the mapping of x to f^x, the DE problem would be easy. However, since in our applications we specifically require that it is easy to compute discrete logarithm in the group of encoded messages, i.e. computing x from $f^x \in F$, that mapping is *not* one way. The intuition about the hardness of the double encoding assumption can be given by the foolowing two examples. Consider two PHF instantiations which are relevant for our purposes. The first is build from the DCR assumption (cf. [CS02]), while the other from class group cryptography (and the HSM assumption[4]). Let us first recall the definition of a subgroup decomposition problem.

Definition 3.2.12. Consider a finite abelian group G, and subgroups G_1 and G_2 such that G is the direct product of G_1 and G_2. An algorithm $\mathcal{A}$ solves the subgroup decomposition (SD) problem in (G, G_1, G_2) if, given input $x \hookleftarrow G$, $\mathcal{A}$ outputs $y \in G_1, z \in G_2$ such that $x = yz$.

In PHFs arising from DCR and HSM, one has $K_{\mathsf{hk}} = \mathbf{Z}$, and for a hashing key $\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$, and x in the finite abelian group $\mathcal{X}$, one has $\mathsf{hash}_{\mathsf{hk}}(x) = x^{\mathsf{hk}}$. This implies that the output space of the hashing algorithm is $\Pi := \mathcal{X} = \mathcal{L} \times \langle\Upsilon\rangle$, and in fact $\Upsilon = f$ and $\langle\Upsilon\rangle = F$. Furthermore computing x from f^x can be done efficiently. Clearly these PHFs are homomorphic and key homomorphic (multiplicative homomorphic with respect to the base with the same key, additive homomorphic in the exponent with respect to the keys).

We want to prove that for both these PHFs, the problem of inverting the OWF can be reduced to the hardness of solving the SD problem in $(\mathcal{X}, \mathcal{L}, F)$ and the hardness of solving the DE problem. We can cope with the two different PHFs in a similar way, with the difference that for the HSM based PHF, the order of f is a prime q, while for the DCR based PHF, the order of f is an RSA integer N. As a consequence, when building two-party ECDSA from DCR, the order q of the one way function's input space and the order

[4]See Section 3.4 or Chapter 2

N of f are different, where $N \gg q$. Hence we modify slightly the assumption, so that $\mathcal{A}$ must output $(\mathsf{hp}, (u_1, \mathsf{hash}_{\mathsf{hk}}(u_1)f^x), (u_2, \mathsf{hash}_{\mathsf{hk}}(u_2)f^x))$, with $x \in \mathbf{Z}$ and $0 \leq x \leq q-1$ (this suffices to instantiate our generic construction of Section 3.3).

The following lemma proves the statement about the reduction of the problem of inverting OWF to a solution of the SD problem and of the DE problem.

Lemma 3.2.2 ([CCL+19])**.** Consider PHFs arising from DCR and from HSM (Section 3.4). Further consider a one way function h. Suppose there exists a PPT algorithm $\mathcal{B}_1$ solving the DE problem with non negligible probability; and a PPT algorithm $\mathcal{B}_2$ solving the SD problem with non negligible probability; then one can build a PPT algorithm breaking the one wayness of h with non negligible probability.

Proof. Consider $h \hookleftarrow \mathsf{Gen}_{OW}(1^\lambda, q)$, an adversary $\mathcal{A}$ attempting to invert h, and algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ as described in the lemma. $\mathcal{A}$ gets as input a value $y := h(x)$ for $x \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$. $\mathcal{A}$ runs $\mathcal{SM} = (\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R}) \leftarrow \mathsf{Gen}_{\mathcal{SM}}(1^\lambda, q)$, and sends $(h, \mathcal{SM}, y)$ to $\mathcal{B}_1$. With significant probability $\mathcal{B}_1$ outputs $(\mathsf{hp}, (u_1, u_1^{\mathsf{hk}} f^x), (u_2, u_2^{\mathsf{hk}} f^x))$ where $u_1, u_2, u_2 u_1^{-1} \in \mathcal{X} \backslash \mathcal{L}$ and $\mathsf{hp} = \mathsf{projkg}(\mathsf{hk})$.

HSM There exist unique values $z_1, z_2 \in \mathcal{L}$ and $b_1, b_2 \in \mathbf{Z}/q\mathbf{Z}$ such that $u_1 = z_1 f^{b_1}$ and $u_2 = z_2 f^{b_2}$. Let us denote $e_1 := u_1^{\mathsf{hk}} f^x = z_1^{\mathsf{hk}} f^{b_1 \mathsf{hk} + x}$ and $e_2 := z_2^{\mathsf{hk}} f^{b_2 \mathsf{hk} + x}$. $\mathcal{A}$ calls upon $\mathcal{B}_2$ four times, with inputs u_1, u_2, e_1 and e_2 respectively (these inputs can be re-randomized, but for simplicity we omit this level of detail), to obtain $z_1, z_2 \in \mathcal{L}$; $f^{b_1}, f^{b_2} \in F$; $z_1^{\mathsf{hk}}, z_2^{\mathsf{hk}} \in \mathcal{L}$; and $f^{b_1 \mathsf{hk} + x}, f^{b_2 \mathsf{hk} + x}$. Now $\mathcal{A}$ can efficiently compute $(b_1 \bmod q)$, $(b_2 \bmod q)$, $(b_1 \mathsf{hk} + x \bmod q)$ and $(b_2 \mathsf{hk} + x \bmod q)$. Since $u_2 u_1^{-1} \in \mathcal{X} \backslash \mathcal{L}$, $b_1 \neq b_2 \bmod q$, and so there exists a unique solution for $(x \bmod q)$ which $\mathcal{A}$ can efficiently compute from the aforementioned equations, thereby breaking the one wayness of h.

DCR There exist unique values $z_1, z_2 \in \mathcal{L}$ and $b_1, b_2 \in \mathbf{Z}/N\mathbf{Z}$ such that $u_1 = z_1 f^{b_1}$ and $u_2 = z_2 f^{b_2}$. Let us denote $e_1 := u_1^{\mathsf{hk}} f^x = z_1^{\mathsf{hk}} f^{b_1 \mathsf{hk} + x}$ and $e_2 := z_2^{\mathsf{hk}} f^{b_2 \mathsf{hk} + x}$. $\mathcal{A}$ calls upon $\mathcal{B}_2$ four times, with inputs u_1, u_2, e_1 and e_2 respectively (as we have seen in the previous item for HSM), to obtain $z_1, z_2 \in \mathcal{L}$; $f^{b_1}, f^{b_2} \in F$; $z_1^{\mathsf{hk}}, z_2^{\mathsf{hk}} \in \mathcal{L}$; and $f^{b_1 \mathsf{hk} + x}, f^{b_2 \mathsf{hk} + x}$. Now $\mathcal{A}$ can efficiently compute $(b_1 \bmod N)$, $(b_2 \bmod N)$, $(b_1 \mathsf{hk} + x \bmod N)$ and $(b_2 \mathsf{hk} + x \bmod N)$. Since $u_2 u_1^{-1} \in \mathcal{X} \backslash \mathcal{L}$, $b_1 \neq b_2 \bmod N$, and so there exists a unique solution for $(x \bmod q)$ which $\mathcal{A}$ can efficiently compute from the aforementioned equations, thereby breaking the one wayness of h.

□

Note that for the DCR based PHF there exists a trapdoor which renders the SD problem easy, which can be efficiently computed when generating the subset membership problem instance. Thus if the HPS arises from DCR, the DE problem is at least as hard as inverting the one way function.

For our HPS from the HSM assumption (resulting from class group cryptography) in Subsection 3.4.1, best known algorithms for solving the SD problem are sub-exponential, whereas for computing discrete logarithms in elliptic curves (which is the OWF we will consider in our construction) there currently exist only exponential algorithms. Consequently for this application the DE problem must have an exponential complexity.

ECDSA-friendly HPS. We conclude this section with the definition of an ECDSA-friendly HPS. We defined all the properties an HPS has to satisfy to be compatible with the ECDSA scheme. Essentially, the notion of an ECDSA-friendly HPS is a HPS which meets all of the aforementioned properties. This notion suffices to ensure simulation based security in the protocol of Section 3.3.

Definition 3.2.13 ($(\Upsilon, \mathcal{F}, \delta_s, \delta_{\mathcal{L}}, \delta_{\mathsf{DE}})$-ECDSA-friendly HPS, [CCL+19]). Let $\mathcal{X}, \Pi$ and F be groups such that F is a cyclic subgroup of Π of prime order q, generated by f, and such that there exists an efficient isomorphism from $(\mathbf{Z}/q\mathbf{Z}, +)$ to $(F, \cdot)$, mapping $m \in \mathbf{Z}/q\mathbf{Z}$ to f^m, whose inverse $\log_f$ is also efficiently computable. Let $\mathsf{exp}_{\mathbb{G}}$ be the function which to $x \in \mathbf{Z}/q\mathbf{Z}$ maps the elliptic curve point xP. An $(\Upsilon, F, \delta_s, \delta_{\mathcal{L}}, \delta_{\mathsf{DE}})$-ECDSA-friendly hash proof system is a hash proof system which associates to a $\delta_{\mathcal{L}}-$hard subset membership problem a homomorphically extended projective hash family $\mathsf{PHF} := (\{\mathsf{hash}_{\mathsf{hk}}\}_{\mathsf{hk} \in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \mathcal{L}, \Pi, K_{\mathsf{hp}}, K'_{\mathsf{hp}}, \mathsf{projkg})$ which is (Υ, F)-decomposable, δ_s-smooth over $\mathcal{X}$ on F, and such that the DE problem is δ_{DE}-hard for $(\mathsf{PHF}, \mathsf{exp}_{\mathbb{G}})$.

3.2.6 Zero-Knowledge Proofs

Proofs of knowledge. We use the $\mathcal{F}_{\mathsf{zk}}$, $\mathcal{F}_{\mathsf{com\text{-}zk}}$ hybrid model. Ideal zero-knowledge functionalities are used for the relations below, where the parameters of the elliptic curve $(\mathbb{G}, P, q)$ are implicit public inputs. We did not present them in the section dedicated to the model and the functionalities (Section 3.1) because one of the relation, $\mathsf{R}_{\mathsf{HPS-DL}}$, requires the background on HPS from Section 3.2 and for clarity it is better to put the relations together. The relations we are interested in are:

1. $\mathsf{R}_{\mathsf{DL}} := \{(Q, w) | Q = wP\}$, proves the knowledge of the discrete log of an elliptic curve point.

2. $\mathsf{R}_{\mathsf{HPS-DL}} := \{(\mathsf{hp}, (c_1, c_2), Q_1); (x_1, w) | (c_1, c_2) = \mathsf{Enc}((u, w); (\mathsf{hp}, x_1)) \wedge (c_1, w) \in \mathsf{R} \wedge Q_1 = x_1 P\}$, proves the knowledge of the randomness used for encryption, and of the value x_1 which is both encrypted in the ciphertext (c_1, c_2) and the discrete log of the elliptic curve point Q_1.

The functionalities $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R}_{\mathsf{DL}}}$, $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R}_{\mathsf{DL}}}$ can be instantiated using Schnorr proofs [Sch91]. For the $\mathsf{R}_{\mathsf{HPS-DL}}$ proof, Lindell in [Lin17] uses a proof of language membership as opposed to a proof of knowledge. Though his technique is quite generic, it cannot be used in our setting. Indeed, his approach requires that the ciphertext be *valid*, which means that the element c must be decryptable. As Lindell uses Paillier's encryption scheme, the surjectivity of the scheme implies that any element of $(\mathbf{Z}/N^2\mathbf{Z})^\times$ is a valid ciphertext. This is not the case for a HPS-based encryption scheme, since it incorporates redundancy so that any pair in $\mathcal{X} \times \Pi$ is not a valid ciphertext.

For our instantiations, we will introduce specific and efficient proofs. From the reasoning done in Section 3.2, using ECDSA-friendly HPS we need not to prove that $x_1 \in \mathbf{Z}/q\mathbf{Z}$, since both the message space of our encryption scheme and the elliptic curve group $\mathbb{G}$ are of order q. This is an advantage compared with [Lin17] because we do not need a range proof for x_1.

3.3 Two-Party ECDSA Signing Protocol with Simulation-Based Security

In this Section we provide our generic construction for two-party ECDSA signing resulting from hash proof systems. The scheme is depicted in Figure 3.6. Next in Theorem 3.3.1, we present a proof that the protocol is secure in the ideal/real paradigm. As already pointed, we must argue the indistinguishability of an adversary's view – corrupting either party P_1 or P_2 – in real and simulated executions. Before going deeper in the description is necessary to focus on a particular aspect of the simulation in the proof. In Cramer-Shoup like encryption schemes (resulting from HPSs as described in paragraph **Resulting Encryption Scheme** from Subsubsection 3.2.3), the chosen plaintext attack indistinguishability of ciphertexts allows for the simulator in the security game to sample the secret hashing key hk, and compute the resulting projection key hp. Thus hk is *known* to the simulator. Indeed here, in order to prove indistinguishability, the simulator first replaces the random masking element $u \in \mathcal{L}$ in the original encryption scheme with an element sampled outside the language $u' \in \mathcal{X} \backslash \mathcal{L}$. Note that in order to perform this change the simulator *must* know the secret hashing key, since a private evaluation is the way to compute the encryption of an element outside the language. Under the hardness of the subset membership problem this change goes unnoticed to any polynomial time adversary. Then to guarantee the indistinguishability of the resulting encryption scheme we rely on the smoothness of the projective hash family which allows one to replace the plaintext value by some random element from the plaintext space. This observation is of particular importance and we insist on this point since in Lindell's protocol [Lin17], many issues arise from the use of Paillier's cryptosystem, for which the indistinguishability of ciphertexts no longer holds if the simulator knows the secret key. In particular this implies that in Lindell's game based proof, instead of letting the simulator use the Paillier secret key to decrypt the incoming ciphertext (and check the corrupted party P_2 did not send a different ciphertext c than that prescribed by the protocol), the simulator *guesses* when the adversary may have cheated by simulating an abort with a probability depending on the number of issued signatures. This results in a proof of security which is not tight (the reduction implies a factor of the order of the inverse of the number of the issued signatures).

Lindell presents two proofs, a game-based one and a simulation-based one. The technique adopted by Lindell in its game based proof suffices for a game-based definition, but surely it is not enough for simulation-based security definitions. In the latter case, [Lin17] presents a proof which proves that their protocol is secure using simulation, but in order to prove security of the protocol it required the introduction of a new rather non-standard and interactive assumption called $\mathsf{Paillier\text{-}EC\ assumption}$, which we report here for completeness. $\mathsf{Paillier\text{-}EC}$ assumption is defined via the experiment in Figure 3.5, where $\mathcal{O}$ is an oracle such that $1 \leftarrow \mathcal{O}_{c^\star}(c', \alpha, \beta)$ if and only if $\mathsf{Paillier.Dec}(1^\lambda, \mathsf{sk}, c') = \alpha + \beta\omega_{b^\star} \mod q$, where $c' = \mathsf{Paillier.Enc}(1^\lambda, \mathsf{pk}, \omega_{b^\star})$, and $\mathcal{O}$ stops after the first time it returns 0. The $\mathsf{Paillier\text{-}EC\ assumption}$ is hard if for every probabilistic polynomial-time adversary $\mathcal{A}$ there exists a negligible function ν such that $\Pr[\mathbf{Exp}_{\mathcal{A}}(1^\lambda) = 1] \le \frac{1}{2} + \nu(n)$.

In our context, the framework we have chosen to adopt helps us to avoid such an interac-

```
Experiment $\mathbf{Exp}_{\mathcal{A}}(1^\lambda)$
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Paillier.KeyGen}(1^\lambda)$
  $(\omega_0, \omega_1) \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, $Q \leftarrow \omega_0 P$
  $b^\star \xleftarrow{\$} \{0,1\}$, $c^\star \leftarrow \mathsf{Paillier.Enc}(1^\lambda, \mathsf{pk}, \omega_{b^\star})$
  $b \leftarrow \mathcal{A}^{\mathcal{O}_{c^\star}(\cdot,\cdot,\cdot)}(\mathsf{pk}, c^\star, Q)$
  if $b = b^\star$ then return $1$
    else return $0$
```

Figure 3.5: Paillier-EC assumption from [Lin17]

tive assumption. Moreover, should one write a game based proof for our construction, the security loss present in [Lin17] would not appear. Finally we note that the correctness of our protocol follows from the correctness of the underlying public key encryption scheme and from the fact the hash function is linearly homomorphic for any public key in the efficiently recognisable space K'_{hp}.

3.3.1 The Two-Party ECDSA protocol

We present in this subsection our two-party ECDSA protocol. The protocol is depicted in figure 3.6. We give a description of the it below.

IKeyGen In the interactive key generation subprotocol, initially player P_1 chooses a share $x_1 \in \mathbf{Z}/q\mathbf{Z}$ for the secret signing key and the associated public share $Q_1 = x_1 P$. Then P_1 commits to a proof of knowledge of the discrete logarithm of Q_1 and sends the commitment and Q_1 to P_2. After that P_2 chooses its share $x_2 \in \mathbf{Z}/q\mathbf{Z}$ and $Q_2 = x_2 P$, where x_2 is unrelated to x_1 for the hiding property of the commitment scheme. P_2 answers to P_1 with Q_2 and a proof of knowledge of the discrete logarithm of Q_2. Then P_1 checks the validity of the proof on P_2 values and sends the decommitment. Otherwise, it aborts. After that, P_1 sends an encryption of its share x_1 with a proof for the relation $\mathsf{R}_{\mathsf{HPS-DL}}$ and P_2 checks the validity of the proof. If it does not pass, P_2 aborts. In the end, both players can compute the public key $Q = x_1 Q_2$ for P_1 and $Q = x_2 Q_1$ for P_2. The encryption of x_1 will be used in the signing part by P_2 to compute homomorphic operation and computing its share of the signature[5].

ISign In the interactive signing subprotocol, P_1 and P_2 run a protocol for computing the ECDSA nonce R as done with Q. After that, P_2 computes homomorphic operations on $\mathsf{Enc}(x_1)$ to generate the part of the signature which is dependent from its private values. Finally, P_2 sends to P_1 the resulting ciphertext c_3, P_1 decrypts and complete the signature with its private values. If the signature does not pass the ECDSA verification algorithm, P_1 aborts.

[5]Sending an encryption of x_1 is an idea which comes from [Lin17].

P_1	$\mathsf{lKeyGen}(\mathbb{G}, P, q)$	P_2
$x_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$Q_1 \leftarrow x_1 P$	$\xrightarrow{(\mathsf{com\text{-}prove}, 1, Q_1, x_1)} \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R}_{\mathsf{DL}}} \xrightarrow{(\mathsf{proof\text{-}receipt}, 1)}$	
		$x_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$
P_1 aborts if $(\mathsf{proof}, 2, Q_2)$ not received.	$\xleftarrow{(\mathsf{proof}, 2, Q_2)} \mathcal{F}_{\mathsf{zk}}^{\mathsf{R}_{\mathsf{DL}}} \xleftarrow{(\mathsf{prove}, 2, Q_2, x_2)}$	$Q_2 \leftarrow x_2 P$
	$\xrightarrow{(\mathsf{decom\text{-}proof}, 1)} \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R}_{\mathsf{DL}}} \xrightarrow{(\mathsf{decom\text{-}proof}, 1, Q_1)}$	
$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$ $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$ Sample $(u, w) \in \mathsf{R}$		
$c_{\mathsf{key}} \leftarrow \mathsf{Enc}((u, w); (\mathsf{hp}, x_1))$	$\xrightarrow{(\mathsf{prove}, 3, (\mathsf{hp}, c_{\mathsf{key}}, Q_1), (x_1, w))} \mathcal{F}_{\mathsf{zk}}^{\mathsf{R}_{\mathsf{HPS-DL}}} \xrightarrow{(\mathsf{proof}, 3, (\mathsf{hp}, c_{\mathsf{key}}, Q_1))}$	P_2 aborts unless $(\mathsf{decom\text{-}proof}, 1, Q_1)$, $(\mathsf{proof}, 3,$ $(\mathsf{hp}, c_{\mathsf{key}}, Q_1))$ received and $\mathsf{hp} \in K'_{\mathsf{hp}}$.
$Q \leftarrow x_1 Q_2$		$Q \leftarrow x_2 Q_1$

P_1	$\mathsf{lSign}(m, sid)$	P_2
$k_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$R_1 \leftarrow k_1 P$	$\xrightarrow{(\mathsf{com\text{-}prove}, sid\|\|1, R_1, k_1)} \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R}_{\mathsf{DL}}} \xrightarrow{(\mathsf{proof\text{-}receipt}, sid\|\|1)}$	$k_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $R_2 \leftarrow k_2 P$
P_1 aborts if $(\mathsf{proof}, sid\|\|2, R_2)$ not received.	$\xleftarrow{(\mathsf{proof}, sid\|\|2, R_2)} \mathcal{F}_{\mathsf{zk}}^{\mathsf{R}_{\mathsf{DL}}} \xleftarrow{(\mathsf{prove}, sid\|\|2, R_2, k_2)}$	
	$\xrightarrow{(\mathsf{decom\text{-}proof}, sid\|\|1)} \mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R}_{\mathsf{DL}}} \xrightarrow{(\mathsf{decom\text{-}proof}, sid\|\|1, R_1)}$	P_2 aborts if $(\mathsf{decom\text{-}proof}, sid\|\|1, R_1)$ not received.
$R = (r_x, r_y) \leftarrow k_1 R_2$ $r \leftarrow r_x \mod q$		$m' \leftarrow H(m)$ $R = (r_x, r_y) \leftarrow k_2 R_1$ $r \leftarrow r_x \mod q$ $c_1 \leftarrow \mathsf{Enc}(\mathsf{hp}, k_2^{-1} \cdot m')$ $c_2 \leftarrow \mathsf{EvalScal}(\mathsf{hp}, c_{\mathsf{key}}, k_2^{-1} r x_2)$
$\alpha \leftarrow \mathsf{Dec}(\mathsf{hk}, c_3)$ $\hat{s} \leftarrow \alpha \cdot k_1^{-1}$ $s \leftarrow \min(\hat{s}, q - \hat{s})$ If not $\mathsf{Verif}(Q, m, (r, s))$ P_1 aborts else Return (r, s)	$\xleftarrow{c_3}$	$c_3 \leftarrow \mathsf{EvalSum}(\mathsf{hp}, c_1, c_2)$

Figure 3.6: Two-Party ECDSA Key Generation and Signing Protocols from HPSs

3.3.2 Simulation-based security of the Two-Party ECDSA scheme

Theorem 3.3.1. Assume HPS is a $(\Upsilon, F, \delta_s, \delta_{\mathcal{L}}, \delta_{\mathsf{DE}})-$ECDSA-friendly HPS; and that no polynomial time algorithm can compute discrete logarithms in $\mathbb{G}$ with probability greater than δ_{DL}; then the protocol of Figure 3.6 securely computes $\mathcal{F}_{ECDSA}$ in the $(\mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com\text{-}zk}})-$hybrid model in the presence of a malicious static adversary (under the ideal/real definition). Indeed there exists a simulator for the scheme such that no polynomial time adversary – having corrupted either P_1 or P_2 – can distinguish a real execution of the protocol from a simulated one with probability greater than $2\delta_{\mathcal{L}}+\delta_{\mathsf{DE}}+2\delta_{\mathsf{DL}}+1/q+\delta_s$.

Proof. In this proof, the simulator $\mathcal{S}$ only has access to an ideal functionality $\mathcal{F}_{ECDSA}$ for computing ECDSA signatures, so all it learns in the ideal world is the public key Q which it gets as output of the KeyGen phase from $\mathcal{F}_{ECDSA}$ and signatures (r, s) for messages m of its choice as output of the Sign phase. However in the real world, the adversary, having either corrupted P_1 or P_2 will also see all the interactions with the non corrupted party which lead to the computation of a signature. Thus $\mathcal{S}$ must be able to simulate $\mathcal{A}$'s view of these interactions, while only knowing the expected output. To this end $\mathcal{S}$ must set up with $\mathcal{A}$ the same public key Q that it received from $\mathcal{F}_{ECDSA}$, in order to be able to subsequently simulate interactively signing messages with $\mathcal{A}$, using the output of $\mathcal{F}_{ECDSA}$ from the Sign phase.

$\mathcal{S}$ simulates P_2 – Corrupted P_1: We first show that if an adversary $\mathcal{A}_1$ corrupts P_1, one can construct a simulator $\mathcal{S}$ s.t. the output distribution of $\mathcal{S}$ is indistinguishable from $\mathcal{A}_1$'s view in an interaction with an honest party P_2. The main difference here with the proof of [Lin17] arises from the fact we no longer use a ZKP from which $\mathcal{S}$ can extract the encryption scheme's secret key. Instead, $\mathcal{S}$ extracts the randomness used for encryption and the plaintext x_1 from the ZKPoK for $\mathsf{R}_{\mathsf{HPS-DL}}$, which allows it to recompute the ciphertext and verify it obtains the expected value c_{key}. Moreover since the message space of our encryption scheme is $\mathbf{Z}/q\mathbf{Z}$, if $\mathcal{A}_1$ does not cheat in the proofs (which is guaranteed by the $(\mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com\text{-}zk}})$-hybrid model), the obtained distributions are identical in the ideal and real executions (as opposed to statistically close as in [Lin17]).

Key Generation Phase

1. Given input $\mathsf{KeyGen}(\mathbb{G}, P, q)$, the simulator $\mathcal{S}$ sends $\mathsf{KeyGen}(\mathbb{G}, P, q)$ to the ideal functionality $\mathcal{F}_{ECDSA}$ and receives back a public key Q.

2. $\mathcal{S}$ invokes $\mathcal{A}_1$ on input $\mathsf{IKeyGen}(\mathbb{G}, P, q)$ and receives the commitment to a proof of knowledge of x_1 such that $Q_1 = x_1 P$ denoted $(\mathsf{com\text{-}prove}, 1, Q_1, x_1)$ as $\mathcal{A}_1$ intends to send to $\mathcal{F}^{\mathsf{R}_{\mathsf{DL}}}_{\mathsf{com\text{-}zk}}$, such that $\mathcal{S}$ can extract x_1 and Q_1.

3. Using the extracted value x_1, $\mathcal{S}$ verifies that $Q_1 = x_1 P$. If so, it computes $Q_2 \leftarrow x_1^{-1} Q$ (using the value Q received from $\mathcal{F}_{ECDSA}$); otherwise $\mathcal{S}$ samples a random Q_2 from $\mathbb{G}$.

4. $\mathcal{S}$ sends $(\mathsf{proof}, 2, Q_2)$ to $\mathcal{A}_1$ as if sent by $\mathcal{F}^{\mathsf{R}_{\mathsf{DL}}}_{\mathsf{zk}}$ thereby $\mathcal{S}$ simulating a proof of knowledge of x_2 s.t. $Q_2 = x_2 P$.

5. $\mathcal{S}$ receives $(\mathsf{decom-proof}, 1)$ as $\mathcal{A}_1$ intends to send to $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R_{DL}}}$ and simulates P_2 aborting if $Q_1 \neq x_1 P$. $\mathcal{S}$ also receives $(\mathsf{prove}, 3, (\mathsf{hp}, c_{\mathsf{key}}, Q_1), (x_1, w))$ as $\mathcal{A}_1$ intends to send to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{HPS-DL}}}$.

6. $\mathcal{S}$ computes u from w such that $(u, w) \in \mathsf{R}$, and using the extracted value x_1 verifies that $c_{\mathsf{key}} = \mathsf{Enc}((u, w); (\mathsf{hp}, x_1))$, and simulates P_2 aborting if not.

7. $\mathcal{S}$ sends $\mathsf{continue}$ to $\mathcal{F}_{ECDSA}$ for P_2 to receive output, and stores $(x_1, Q, c_{\mathsf{key}})$.

When taking $\mathcal{F}_{\mathsf{zk}}$ and $\mathcal{F}_{\mathsf{com-zk}}$ as ideal functionalities, the only difference between the real execution as ran by an honest P_2, and the ideal execution simulated by $\mathcal{S}$ is that in the former $Q_2 \leftarrow x_2 P$ where $x_2 \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$, whereas in the latter $Q_2 \leftarrow x_1^{-1} Q$, where Q is the public key returned by the ideal functionality $\mathcal{F}_{ECDSA}$. However since $\mathcal{F}_{ECDSA}$ samples Q uniformly at random from $\mathbb{G}$, the distribution of Q_2 in both cases is identical.

Signing Phase

1. Upon input $\mathsf{Sign}(sid, m)$, simulator $\mathcal{S}$ sends $\mathsf{Sign}(sid, m)$ to $\mathcal{F}_{ECDSA}$ and receives back a signature (r, s).

2. $\mathcal{S}$ computes the elliptic curve point $R = (r, r_y)$ using the ECDSA verification algorithm.

3. $\mathcal{S}$ invokes $\mathcal{A}_1$ with input $\mathsf{ISign}(sid, m)$ and simulates the first three interactions such that $\mathcal{A}_1$ computes R. The strategy is similar to that used to compute Q, in brief, it proceeds as follows:

   (a) $\mathcal{S}$ receives $(\mathsf{com\text{-}prove}, sid||1, R_1, k_1)$ from $\mathcal{A}_1$.

   (b) If $R_1 = k_1 P$ then $\mathcal{S}$ sets $R_2 \leftarrow k_1^{-1} R$; otherwise it chooses R_2 at random. $\mathcal{S}$ sends $(\mathsf{proof}, sid||2, R_2)$ to $\mathcal{A}_1$.

   (c) $\mathcal{S}$ receives $(\mathsf{decom\text{-}proof}, sid||1)$ from $\mathcal{A}_1$. If $R_1 \neq k_1 P$ then $\mathcal{S}$ simulates P_2 aborting and instructs the trusted party computing $\mathcal{F}_{ECDSA}$ to abort.

4. $\mathcal{S}$ computes $c_3 \leftarrow \mathsf{Enc}_{pk}(k_1 \cdot s \mod q)$, where s was received from $\mathcal{F}_{ECDSA}$, and sends c_3 to $\mathcal{A}_1$.

As with the computation of Q_2 in the key generation phase, R_2 is distributed identically in the real and ideal executions since R is randomly generated by $\mathcal{F}_{ECDSA}$. The zero-knowledge proofs and verifications are also identically distributed in the $\mathcal{F}_{\mathsf{zk}}$, $\mathcal{F}_{\mathsf{com\text{-}zk}}$-hybrid model. Thus, the only difference between a real execution and the simulation is the way that c_3 is computed. In the simulation it is an encryption of $k_1 \cdot s = k_1 \cdot k^{-1}(m' + r \cdot x) = k_2^{-1} \cdot (m' + r \cdot x) \mod q$, whereas in a real execution c_3 is computed from c_{key}, using the homomorphic properties of the encryption scheme. However, notice that as long as there exist (u, w) such that $c_{\mathsf{key}} = \mathsf{Enc}((u, w); (\mathsf{hp}, x_1))$ where $Q = x_1 P$ – which is guaranteed by the ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{HPS-DL}}}$ – and as long as the homomorphic operations hold – which is guaranteed for any hp in the efficiently verifiable ensemble K'_{hp} (cf. Subsection 3.2.2) – the c_3 obtained in the real scenario is also an encryption of $s' = k_2^{-1} \cdot (m' + r \cdot x) \mod q$. Thus c_3 is distributed identically in both cases.

This implies that the view of a corrupted P_1 is identical in the real and ideal executions of the protocol (in the $\mathcal{F}_{\mathsf{zk}}$, $\mathcal{F}_{\mathsf{com\text{-}zk}}$-hybrid model), *i.e.*, the simulator perfectly simulates the real environment, which completes the proof of this simulation case.

$\mathcal{S}$ simulates P_1 – Corrupted P_2: We now suppose an adversary $\mathcal{A}_2$ corrupts P_2 and describe the simulated execution of the protocol. We demonstrate via a sequence of games – where the first game is a real execution and the last game is a simulated execution – that both executions are indistinguishable. This proof methodology differs considerably to that of [Lin17] since the main differences between a real and simulated execution are due to the ciphertext c_{key}, so the indistinguishability of both executions reduces to the hardness of the hash proof system, the smoothness of the underlying projective hash family, and the hardness of the double encoding problem. We first describe an ideal execution of the protocol:

Key Generation Phase

1. Given input $\mathsf{KeyGen}(\mathbb{G}, P, q)$, the simulator $\mathcal{S}$ sends $\mathsf{KeyGen}(\mathbb{G}, P, q)$ to the functionality $\mathcal{F}_{ECDSA}$ and receives back Q.

2. $\mathcal{S}$ invokes $\mathcal{A}_2$ upon input $\mathsf{IKeyGen}(\mathbb{G}, P, q)$ and sends $(\mathsf{proof\text{-}receipt}, 1)$ as $\mathcal{A}_2$ expects to receive from $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R_{DL}}}$.

3. $\mathcal{S}$ receives $(\mathsf{prove}, 2, Q_2, x_2)$ as $\mathcal{A}_2$ intends to send to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$.

4. Using the extracted value x_2, $\mathcal{S}$ verifies that Q_2 is a non zero point on the curve and that $Q_2 = x_2 P$. If so it computes $Q_1 \leftarrow (x_2)^{-1} Q$ and sends $(\mathsf{decom\text{-}proof}, 1, Q_1)$ to $\mathcal{A}_2$ as it expects to receive from $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R_{DL}}}$. If not it simulates P_1 aborting and halts.

5. $\mathcal{S}$ samples $\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$ and computes $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$. It also samples $\tilde{x}_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $(u, w) \in \mathsf{R}$ and computes $c_{\mathsf{key}} \leftarrow \mathsf{Enc}((u, w); (\mathsf{hp}, \tilde{x}_1))$.

6. $\mathcal{S}$ sends $(\mathsf{proof}, 3, (\mathsf{hp}, c_{\mathsf{key}}, Q_1))$ to $\mathcal{A}_2$, as $\mathcal{A}_2$ expects to receive from $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{HPS\text{-}DL}}}$.

7. $\mathcal{S}$ sends $\mathsf{continue}$ to $\mathcal{F}_{ECDSA}$ for P_1 to receive output, and stores Q.

Signing Phase

1. Upon input $\mathsf{Sign}(sid, m)$, simulator $\mathcal{S}$ sends $\mathsf{Sign}(sid, m)$ to $\mathcal{F}_{ECDSA}$ and receives back a signature (r, s).

2. $\mathcal{S}$ computes the point $R = (r, r_y)$ using the ECDSA verification algorithm.

3. $\mathcal{S}$ invokes $\mathcal{A}_2$ with input $\mathsf{ISign}(sid, m)$ and sends $(\mathsf{proof\text{-}receipt}, sid||1)$ as $\mathcal{A}_2$ expects to receive from $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R_{DL}}}$.

4. $\mathcal{S}$ receives $(\mathsf{prove}, sid||2, R_2, k_2)$ as $\mathcal{A}_2$ intends to send to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R_{DL}}}$.

Game_0	Game_1
$Q \leftarrow x_1 x_2 P$	$Q \leftarrow x_1 x_2 P$
$\vdots$	$\vdots$
$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$	$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$
$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$	$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$
	Sample $(u, w) \in \mathsf{R}$
$c_{\mathsf{key}} \leftarrow \mathsf{Enc}(\mathsf{hp}, x_1)$	$c_{\mathsf{key}} \leftarrow (u, \mathsf{hash}_{\mathsf{hk}}(u) \cdot f^{x_1})$
Send c_{key} to $\mathcal{A}_2$	Send c_{key} to $\mathcal{A}_2$
$\vdots$	$\vdots$
$R \leftarrow k_1 k_2 P$, $r \leftarrow r_x \bmod q$	$R \leftarrow k_1 k_2 P$, $r \leftarrow r_x \bmod q$
$\vdots$	$\vdots$
Receive $c_3 = (u_3, e_3)$ from $\mathcal{A}_2$	Receive $c_3 = (u_3, e_3)$ from $\mathcal{A}_2$
Let $\alpha \leftarrow \log_f \left(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}\right)$	Let $\alpha \leftarrow \log_f \left(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}\right)$
$s \leftarrow \alpha \cdot k_1^{-1}$	$s \leftarrow \alpha \cdot k_1^{-1}$
If not $\mathsf{Verif}(Q, m, (r, s))$ then abort	If not $\mathsf{Verif}(Q, m, (r, s))$ then abort
else Return (r, s)	else Return (r, s)

5. Using the extracted value k_2, $\mathcal{S}$ verifies that R_2 is a non zero point and that $R_2 = k_2 P$. If so it computes $R_1 \leftarrow k_2^{-1} R$ and sends $(\mathsf{decom\text{-}proof}, sid||1, R_1)$ to $\mathcal{A}_2$ as it expects to receive from $\mathcal{F}_{\mathsf{com\text{-}zk}}^{\mathsf{R_{DL}}}$. If not it simulates P_1 aborting and instructs the trusted party computing $\mathcal{F}_{ECDSA}$ to abort.

6. $\mathcal{S}$ receives $c_3 = (u_3, e_3)$ from $\mathcal{A}_2$, which it can decrypt using hk, i.e.

$$\alpha \leftarrow \log_f \left(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}\right).$$

If $\alpha = k_2^{-1} \cdot (m' + r \cdot x_2 \cdot \tilde{x}_1) \bmod q$ then $\mathcal{S}$ sends $\mathsf{continue}$ to the trusted party $\mathcal{F}_{ECDSA}$, s.t. the honest party P_1 receives output. Otherwise it instructs $\mathcal{F}_{ECDSA}$ to abort.

We now describe the sequence of games. Game_0 is the real execution of the protocol from P_1's view, and we finish in Game_6 which is the ideal simulation described above. In the following intermediary games, only the differences in the steps performed by $\mathcal{S}$ are depicted.

Let us now demonstrate that each game step is indistinguishable from the view of $\mathcal{A}_2$. Intuitively, in Game_1 the simulator uses the secret hashing key hk instead of the public projection key hp to compute c_{key}. Though the values are computed differently, they are distributed identically, and are perfectly indistinguishable. Next in Game_2 we replace the first element of the ciphertext (in Game_1 this is $u \in \mathcal{L}$) with an element $u \in \mathcal{X} \backslash \mathcal{L}$. By the hardness of the subset membership problem Game_1 and Game_2 are indistinguishable. Next in Game_3 we switch to the ideal world, so Q and R are received from $\mathcal{F}_{ECDSA}$. The value x_1 such that $Q = x_1 x_2 P$ is unknown to $\mathcal{S}$ simulating P_1, and the value $\tilde{x}_1$ encrypted in c_{key} is sampled uniformly at random from $\mathbf{Z}/q\mathbf{Z}$, and is unrelated to Q. Proving indistinguishability between Game_2 and Game_3 is the most involved analysis of all our game steps. The smoothness of the PHF ensures that the ciphertext c_{key} follows identical distributions in both games from $\mathcal{A}_2$'s view; however difficulties arise due to the check performed by $\mathcal{S}$ on α after decrypting c_3. Indeed if $\mathcal{A}_2$ produces a ciphertext c_3 which passes the check in one game, but not in the other, clearly $\mathcal{A}_2$ can distinguish both

Game_2	Game_3
$Q \leftarrow x_1 x_2 P$	$Q \leftarrow \mathcal{F}_{ECDSA}$
	Extract x_2 from $(\mathsf{prove}, 2, Q_2, x_2)$
	$\tilde{x}_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$
$\vdots$	$\vdots$
$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$	$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$
$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$	$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$
$u \xleftarrow{\$} \mathcal{X} \backslash \mathcal{L}$	$u \xleftarrow{\$} \mathcal{X} \backslash \mathcal{L}$
$c_{\mathsf{key}} \leftarrow (u, \mathsf{hash}_{\mathsf{hk}}(u) \cdot f^{x_1})$	$c_{\mathsf{key}} \leftarrow (u, \mathsf{hash}_{\mathsf{hk}}(u) \cdot f^{\tilde{x}_1})$
Send c_{key} to $\mathcal{A}_2$	Send c_{key} to $\mathcal{A}_2$
$\vdots$	$\vdots$
$R \leftarrow k_1 k_2 P$,	$(r, s) \leftarrow \mathcal{F}_{ECDSA}$
$r \leftarrow r_x \bmod q$	$r \leftarrow r_x \bmod q$
	Extr. k_2 from $(\mathsf{prove}, sid\|\|2, R_2, k_2)$
$\vdots$	$\vdots$
Receive $c_3 = (u_3, e_3)$ from $\mathcal{A}_2$	Receive $c_3 = (u_3, e_3)$ from $\mathcal{A}_2$
$\vdots$	$\vdots$
Let $\alpha \leftarrow \log_f \left(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}\right)$	Let $\alpha \leftarrow \log_f \left(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}\right)$
$s \leftarrow \alpha \cdot k_1^{-1}$	(1) If $\alpha \neq k_2^{-1}(m' + r\tilde{x}_1 x_2)$ then
If not $\mathsf{Verif}(Q, m, (r, s))$ then abort	(2) If $(\alpha k_2)P \neq mP + rQ$ abort
else Return (r, s)	else Return (r, s)

Game_4	Game_5	Game_6
$Q \leftarrow \mathcal{F}_{ECDSA}$	$Q \leftarrow \mathcal{F}_{ECDSA}$	$Q \leftarrow \mathcal{F}_{ECDSA}$
Extract x_2 from $(\mathsf{prove}, 2, Q_2, x_2)$	Extract x_2 from $(\mathsf{prove}, 2, Q_2, x_2)$	Extract x_2 from $(\mathsf{prove}, 2, Q_2, x_2)$
$\tilde{x}_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$	$\tilde{x}_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$	$\tilde{x}_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$
$\vdots$	$\vdots$	$\vdots$
$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$	$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$	$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$
$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$	$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$	$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$
$u \xleftarrow{\$} \mathcal{X} \backslash \mathcal{L}$	Sample $(u, w) \in \mathsf{R}$	
$c_{\mathsf{key}} \leftarrow (u, \mathsf{hash}_{\mathsf{hk}}(u) \cdot f^{\tilde{x}_1})$	$c_{\mathsf{key}} \leftarrow (u, \mathsf{hash}_{\mathsf{hk}}(u) \cdot f^{\tilde{x}_1})$	$c_{\mathsf{key}} \leftarrow \mathsf{Enc}(\mathsf{hp}, \tilde{x}_1)$
Send c_{key} to $\mathcal{A}_2$	Send c_{key} to $\mathcal{A}_2$	Send c_{key} to $\mathcal{A}_2$
$\vdots$	$\vdots$	$\vdots$
$(r, s) \leftarrow \mathcal{F}_{ECDSA}$	$(r, s) \leftarrow \mathcal{F}_{ECDSA}$	$(r, s) \leftarrow \mathcal{F}_{ECDSA}$
$r \leftarrow r_x \bmod q$	$r \leftarrow r_x \bmod q$	$r \leftarrow r_x \bmod q$
Extr. k_2 from $(\mathsf{prove}, sid\|\|2, R_2, k_2)$	Extr. k_2 from $(\mathsf{prove}, sid\|\|2, R_2, k_2)$	Extr. k_2 from $(\mathsf{prove}, sid\|\|2, R_2, k_2)$
$\vdots$	$\vdots$	$\vdots$
Receive $c_3 = (u_3, e_3)$ from $\mathcal{A}_2$	Receive $c_3 = (u_3, e_3)$ from $\mathcal{A}_2$	Receive $c_3 = (u_3, e_3)$ from $\mathcal{A}_2$
$\vdots$	$\vdots$	$\vdots$
Let $\alpha \leftarrow \log_f \left(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}\right)$	Let $\alpha \leftarrow \log_f \left(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}\right)$	Let $\alpha \leftarrow \log_f \left(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}\right)$
If $\alpha \neq k_2^{-1}(m' + r\tilde{x}_1 x_2)$	If $\alpha \neq k_2^{-1}(m' + r\tilde{x}_1 x_2)$	If $\alpha \neq k_2^{-1}(m' + r\tilde{x}_1 x_2)$
then abort	then abort	then abort
Check (2) removed		

games. To deal with this, in Game_3 we introduce an additional check (2). Check (2) is performed using the elliptic curve point Q, and compares α to $k_2^{-1}(m' + rx_1x_2)$. On the other hand check (1) is performed using the randomly sampled $\tilde{x}_1$, and compares α to $k_2^{-1}(m' + r\tilde{x}_1x_2)$. This extra check allows us to ensure that if $\mathcal{A}_2$ can cause one game to abort, while the other does not, it has either broken the double encoding challenge, or fixes the value of $\tilde{x}_1$. Since from the smoothness of the PHF, $\tilde{x}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$ from $\mathcal{A}_2$'s view, this cannot occur with probability greater than $1/q + \delta_s$. So Game_2 and Game_3 are indistinguishable. In Game_4 we remove check (2), and demonstrate that if $\mathcal{A}_2$ could distinguish both games, one could use $\mathcal{A}_2$ to break the discrete logarithm problem in $\mathbb{G}$.

Next we use the hardness of the subset membership problem again to hop from Game_4 to Game_5, such that in the latter the first element of the ciphertext is once again in $\mathcal{L}$; and finally Game_5 and Game_6 are identical from an adversary's point of view since we simply use the public evaluation function of the hash function instead of the private one.

We denote E_i the event an algorithm interacting with $\mathcal{S}$ in Game_i outputs 1. Thus by demonstrating that $|\Pr[\mathsf{E}_0] - \Pr[\mathsf{E}_6]|$ is negligible, we demonstrate that, from $\mathcal{A}_2$'s view, the real and ideal executions are indistinguishable.

Game_0 to Game_1. The only difference here is the way c_{key} is computed, namely we use the secret hashing key hk instead of the public projection key hp and the witness w to compute c_{key}. Though the values are computed differently, they are identical from $\mathcal{A}_2$'s point of view:

$$|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_0]| = 0.$$

Game_1 to Game_2. Suppose that an algorithm $\mathcal{D}$ is able to distinguish, with non negligible advantage, between the distribution generated in Game_1 from that generated in Game_2. Then we can devise $\hat{\mathcal{S}}$ that uses $\mathcal{D}$ to break the hard subset membership assumption, *i.e.*, distinguish random elements of $\mathcal{L}$ from those of $\mathcal{X}\backslash\mathcal{L}$. The input of $\hat{\mathcal{S}}$ is a hard subset membership challenge x^* which is either an element in $\mathcal{L}$ or an element of $\mathcal{X}\backslash\mathcal{L}$. Precisely $\hat{\mathcal{S}}$ works as $\mathcal{S}$ would in Game_1, interacting with $\mathcal{D}$ instead of $\mathcal{A}_2$, the only difference being that instead of sampling $(u, w) \in \mathsf{R}$ it sets $u := x^*$ and computes $c_{\mathsf{key}} := (u, \mathsf{hash}_{\mathsf{hk}}(u) \cdot f^{x_1})$. When $\mathcal{D}$ returns a bit b (relative to Game_{b+1}), $\hat{\mathcal{S}}$ returns the same bit, where 0 represents the case $x^* \in \mathcal{L}$ and 1 represents the case $x^* \in \mathcal{X}\backslash\mathcal{L}$.

Analysis – Case $x^* \in \mathcal{L}$: There exists $w \in \mathcal{W}$ such that $(x^*, w) \in \mathsf{R}$ and $\mathsf{projhash}_{\mathsf{hp}}(x^*, w) = \mathsf{hash}_{\mathsf{hk}}(x^*)$. So $c_{\mathsf{key}} = (u, e)$ is an encryption of x_1 as computed in Game_1.
Case $x^* \in \mathcal{X}\backslash\mathcal{L}$: The ciphertext is $(x^*, \mathsf{hash}_{\mathsf{hk}}(x^*)f^{x_1})$, which is exactly the distribution obtained in Game_2. So the advantage of $\hat{\mathcal{S}}$ in breaking the hard subset membership assumption is at least that of $\mathcal{D}$ in distinguishing both games. Thus:

$$|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_1]| \leqslant \delta_{\mathcal{L}}.$$

Game_2 to Game_3. In Game_3 the points $Q = x_1x_2P$ and R come from the functionality $\mathcal{F}_{ECDSA}$, while in Game_2 they are computed as in the real protocol. As a result, the value $\tilde{x}_1$ encrypted in c_{key} is unrelated to x_1.

Let us denote $c_{\mathsf{key}} := (u, e)$, where $e = \mathsf{hash}_{\mathsf{hk}}(u)f^{\tilde{x}_1}$, the invalid ciphertext which the simulator sends to $\mathcal{A}_2$ in Game_3. Using the fact PHF is decomposable, and since $u \in \mathcal{X}\backslash\mathcal{L}$, we can write $u = zy$, for unique $z \in \mathcal{L}$ and $y \in \langle\Upsilon\rangle$. Recall that Ψ is a generator for K_{hk} such that that $\mathsf{hash}_{\Psi}(\Upsilon) = f$ (*cf.* Remark 13). We denote $b \in \mathbf{Z}/q\mathbf{Z}$ the unique value such that $\mathsf{hash}_{\Psi}(y) = f^b$. Note that since $u \notin \mathcal{L}$, it holds that $b \neq 0 \bmod q$. Now to demonstrate that Game_2 and Game_3 are indistinguishable from $\mathcal{A}_2$'s view, we start by considering a fixed $\mathsf{hk}' \in K_{\mathsf{hk}}$ satisfying the following equations:

$$\begin{cases} \mathsf{projkg}(\mathsf{hk}') = \mathsf{hp} = \mathsf{projkg}(\mathsf{hk}), \\ \mathsf{hash}_{\mathsf{hk}'}(y)f^{x_1} = \mathsf{hash}_{\mathsf{hk}}(y)f^{\tilde{x}_1}. \end{cases}$$

Note that the smoothness of PHF over $\mathcal{X}$ on F ensures that such a hk' exists (it is not necessarily unique). We now see that in Game_3, c_{key} is an invalid encryption of both x_1 and of $\tilde{x}_1$, for respective hashing keys hk' and hk, but for the same projection key hp, indeed:

$$\begin{aligned} c_{\mathsf{key}} &= (u, \mathsf{hash}_{\mathsf{hk}}(u)f^{\tilde{x}_1}) = (u, \mathsf{projhash}_{\mathsf{hp}}(z, w)\mathsf{hash}_{\mathsf{hk}}(y)f^{\tilde{x}_1}) \\ &= (u, \mathsf{projhash}_{\mathsf{hp}}(z, w)\mathsf{hash}_{\mathsf{hk}'}(y)f^{x_1}) = (u, \mathsf{hash}_{\mathsf{hk}'}(u)f^{x_1}). \end{aligned}$$

Let us denote γ and $\gamma' \in \mathbf{Z}$ the values such that $\mathsf{hk} = \gamma \cdot \Psi$ and $\mathsf{hk}' = \gamma' \cdot \Psi$, such that $\mathsf{hash}_{\mathsf{hk}}(\Upsilon) = f^{\gamma}$ and $\mathsf{hash}_{\mathsf{hk}'}(\Upsilon) = f^{\gamma'}$. Now since $\mathsf{hash}_{\Psi}(y) = f^b$, it holds that

$$b\gamma + \tilde{x}_1 = b\gamma' + x_1 \bmod q \quad \Leftrightarrow \quad \gamma' - \gamma = b^{-1}(\tilde{x}_1 - x_1) \bmod q. \tag{3.1}$$

The adversary $\mathcal{A}_2$ receives the ECDSA public key Q, the public projection key $\mathsf{hp} = \mathsf{projkg}(\mathsf{hk})$, and c_{key} from $\mathcal{S}$ (at this point its view is identical to its' view in Game_2). Then $\mathcal{A}_2$ computes $c_3 = (u_3, e_3)$, which it sends to $\mathcal{S}$. The difference between Game_2 and Game_3 appears now in how $\mathcal{S}$ attempts to decrypt c_3. In Game_2 it would have used hk', whereas in Game_3 it uses hk.

Notation. We denote α the random variable obtained by decrypting c_3 (received in Game_3) with decryption key hk; we denote α' the random variable obtained by decrypting c_3 (received in Game_3) with decryption key hk'; we introduce a hypothetical Game_3', which is exactly as Game_3, only one decrypts c_3 (received in Game_3) with decryption key hk', thus obtaining α', and check (1) of Game_3 is replaced by 'If $\alpha \neq k_2^{-1}(m' + rx_1x_2)$'. Since both tests of Game_3' are redundant, we only keep check (2).

Observation. The view of $\mathcal{A}_2$ in Game_2 and in Game_3' is identical. We demonstrate that the probability $\mathcal{A}_2$'s view differs when $\mathcal{S}$ uses α in Game_3 from when it uses α' in Game_3' is negligible. This allows us to conclude that $\mathcal{A}_2$ cannot distinguish Game_2 and Game_3 except with negligible probability.

Let us consider the ciphertext $c_3 = (u_3, e_3) \in \mathcal{X} \times \Pi$ sent by $\mathcal{A}_2$. By the decomposability of PHF we know there exist unique $z_3 \in \mathcal{L}$, $y_3 \in \langle\Upsilon\rangle$ such that $u_3 = z_3y_3$. Moreover there exists a unique $b_3 \in \mathbf{Z}/q\mathbf{Z}$ such that $\mathsf{hash}_{\Psi}(y_3) = f^{b_3}$. By the homomorphic properties of PHF the decryption algorithm applied to c_3 with decryption key hk (resp. hk') returns $\perp$ if $e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1} = e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(z_3)^{-1} \cdot \mathsf{hash}_{\mathsf{hk}}(y_3)^{-1} \notin F$ (resp.

$e_3 \cdot \mathsf{hash}_{\mathsf{hk}'}(u_3)^{-1} = e_3 \cdot \mathsf{hash}_{\mathsf{hk}'}(z_3)^{-1} \cdot \mathsf{hash}_{\mathsf{hk}'}(y_3)^{-1} \notin F$). However since $z_3 \in \mathcal{L}$, and $\mathsf{projkg}(\mathsf{hk}') = \mathsf{projkg}(\mathsf{hk})$, by correctness of PHF it holds that $\mathsf{hash}_{\mathsf{hk}'}(z_3) = \mathsf{hash}_{\mathsf{hk}}(z_3)$; while $\mathsf{hash}_{\mathsf{hk}'}(y_3) = f^{\gamma' \cdot b_3}$ and $\mathsf{hash}_{\mathsf{hk}}(y_3) = f^{\gamma \cdot b_3}$ live in F. Consequently the decryption algorithm applied to c_3 with decryption key hk returns $\perp$ if and only if it does so with decryption key hk' (i.e. $\alpha = \perp$ if and only if $\alpha' = \perp$). In this case Game_3 is identical to $\mathsf{Game}_3{}'$ from $\mathcal{A}_2$'s view ($\mathcal{S}$ aborts in both cases). We hereafter assume decryption does not fail, which allows us to adopt the following notation:

$$e_3 = \mathsf{hash}_{\mathsf{hk}}(z_3) f^{h_3} = \mathsf{hash}_{\mathsf{hk}'}(z_3) f^{h_3} \qquad \text{with } h_3 \in \mathbf{Z}/q\mathbf{Z}$$

We thus have:

$$\alpha := \log_f(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}) = h_3 - b_3 \cdot \gamma \bmod q$$
$$\alpha' := \log_f(e_3 \cdot \mathsf{hash}_{\mathsf{hk}'}(u_3)^{-1}) = h_3 - b_3 \cdot \gamma' \bmod q$$

such that, injecting Equation (3.1), one gets:

$$\alpha - \alpha' = b_3(\gamma' - \gamma) = b_3 b^{-1}(\tilde{x}_1 - x_1) \bmod q.$$

We now consider four cases:

1. ($\alpha = \alpha' \bmod q$). This case occurs if $b_3 = 0 \bmod q$, i.e. $u_3 \in \mathcal{L}$ and so u_3 is a valid ciphertext; or if $\tilde{x}_1 = x_1 \bmod q$. If this occurs Game_2 and Game_3 are identical from $\mathcal{A}_2$'s view. Note that this is the only case where all checks pass for both α and α'.

2. ($\alpha \neq \alpha' \bmod q$) but ($\alpha - \alpha' = k_2^{-1} r x_2 (\tilde{x}_1 - x_1) \bmod q$). This occurs if $b_3 = k_2^{-1} r x_2 b \bmod q$, i.e. $\mathcal{A}_2$ performed homomorphic operations on c_{key}, and the difference between α and α' is that expected by the simulator. This results in identical views from $\mathcal{A}_2$'s perspective since α causes check (1) to pass if and only if α' causes check (2) to pass:

   $$\alpha = k_2^{-1}(m' + r\tilde{x}_1 x_2) \Leftrightarrow \alpha' + k_2^{-1} r x_2(\tilde{x}_1 - x_1) = k_2^{-1}(m' + r\tilde{x}_1 x_2) \Leftrightarrow \alpha' = k_2^{-1}(m' + r x_2 x_1).$$

3. ($\alpha \neq \alpha' \bmod q$) and ($\alpha - \alpha' \neq k_2^{-1} r x_2 (\tilde{x}_1 - x_1) \bmod q$). We here consider three sub-cases:

   (a) Either both tests fail for α and test (2) fails for α'; i.e. $\alpha \neq k_2^{-1}(m' + r\tilde{x}_1 x_2) \bmod q$; and $\alpha, \alpha' \neq k_2^{-1}(m' + r x_1 x_2) \bmod q$. This results in identical views from $\mathcal{A}_2$'s perspective.

   (b) Either the check on α' passes. This means that:

   $$\alpha' = k_2^{-1}(m' + r x_1 x_2) \bmod q.$$

   Since $\alpha - \alpha' \neq k_2^{-1} r x_2 (\tilde{x}_1 - x_1) \bmod q$ necessarily check (1) on α fails; and since $\alpha \neq \alpha' \bmod q$ necessarily check (2) on α fails. Consequently if this sub-case occurs, $\mathcal{A}_2$'s view differs. We demonstrate that if the DE problem is hard, this case occurs with negligible probability.

   Suppose that an algorithm $\mathcal{B}$ is able to cause this case to occur with non negligible probability $\mathfrak{p}$. Then we can devise an algorithm $\hat{\mathcal{S}}$ which uses $\mathcal{B}$

to break the DE assumption for $(\mathsf{PHF}, \mathsf{exp}_{\mathbb{G}})$. Algorithm $\hat{\mathcal{S}}$ gets as input a DE challenge point $Q = xP$ and the description $\mathcal{SM}$ of a subset membership problem, and must output hp, $(u_1, \mathsf{hash}_{\mathsf{hk}'}(u_1)f^x)$ and $(u_2, \mathsf{hash}_{\mathsf{hk}'}(u_2)f^x)$ where $\mathsf{hp} = \mathsf{projkg}(\mathsf{hk}')$; $u_1, u_2 \in \mathcal{X}\backslash\mathcal{L}$; $u_1 \neq u_2$; and $u_1/u_2 \in \mathcal{X}\backslash\mathcal{L}$. Precisely $\hat{\mathcal{S}}$ works as $\mathcal{S}$ would in Game_3, interacting with $\mathcal{B}$ instead of $\mathcal{A}_2$, the only difference being that instead of using the ECDSA public key it receives from $\mathcal{F}_{ECDSA}$, $\hat{\mathcal{S}}$ uses the DE challenge Q. When $\mathcal{B}$ sends c_3 to $\hat{\mathcal{S}}$, $\hat{\mathcal{S}}$ computes $c_1 := \mathsf{EvalScal}(\mathsf{hp}, \mathsf{EvalSum}(\mathsf{hp}, c_3, -k_2^{-1}m'), k_2r^{-1})$. Finally $\hat{\mathcal{S}}$ computes the component-wise product $c_2 := c_{\mathsf{key}} \odot (1, f^{x_2})$ and outputs hp, c_1, c_2 to its' own DE challenger.
Analysis. Let us denote x_2, k_2 the values $\hat{\mathcal{S}}$ extracts from its interactions with $\mathcal{B}$. We further denote $x_1 := x\, x_2^{-1}$ (which is unknown to $\hat{\mathcal{S}}$). $\hat{\mathcal{S}}$ samples $\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$, and computes $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$. It then samples $\tilde{x}_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and computes $c_{\mathsf{key}} := (u, \mathsf{hash}_{\mathsf{hk}}(u)f^{\tilde{x}_1})$ which can be interpreted as $(u, \mathsf{hash}_{\mathsf{hk}'}(u)f^{x_1})$. Let us denote (u_2, e_2) the components of $c_2 = c_{\mathsf{key}} \odot (1, f^{x_2})$ such that $c_2 = (u_2, e_2) = (u, \mathsf{hash}_{\mathsf{hk}'}(u) \cdot f^x)$, where $u_2 \in \mathcal{X}\backslash\mathcal{L}$ by construction.
When $\hat{\mathcal{S}}$ receives c_3 from $\mathcal{B}$, with probability $\mathfrak{p}$, using decryption key hk', c_3 decrypts to $\alpha' = k_2^{-1}(m' + rx_1x_2) \bmod q$. $\mathcal{S}$ does not know hk', but using the homomorphic properties of the PKE, $\mathcal{S}$ computes $c_1 := (u_1, e_1) = (u_1, \mathsf{hash}_{\mathsf{hk}'}(u_1)f^x)$. Since we ruled out the case 1. (where $\alpha = \alpha' \bmod q$), necessarily $u_1 \in \mathcal{X}\backslash\mathcal{L}$. And since we ruled out the case 2. (where $\alpha - \alpha' = k_2^{-1}rx_2(\tilde{x}_1 - x_1) \bmod q$), necessarily $u_1/u_2 \in \mathcal{X}\backslash\mathcal{L}$. Thus with probability $\mathfrak{p}$, $\hat{\mathcal{S}}$ breaks the DE assumption, and consequently $\mathfrak{p} \leqslant \delta_{\mathsf{DE}}$, which concludes that this case occurs with probability $\leqslant \delta_{\mathsf{DE}}$.

(c) Else one of the checks on α passes.

i. If $(\alpha = k_2^{-1}(m' + rx_1x_2) \bmod q)$, then since $(\alpha \neq \alpha' \bmod q)$ necessarily check (2) on α' fails. However if this occurs, since $\mathcal{S}$ has extracted k_2, x_2 from the zero knowledge proofs, it can compute x_1 from α, thereby breaking the DL problem in $\mathbb{G}$. This occurs with probability $\leqslant \delta_{\mathsf{DL}}$.

ii. If $\alpha = k_2^{-1}(m' + r\tilde{x}_1x_2) \bmod q$, then since $\alpha - \alpha' \neq k_2^{-1}rx_2(\tilde{x}_1 - x_1) \bmod q$ necessarily check (2) on α' fails. Let us prove that information theoretically, this can not happen with probability greater than $1/q + \delta_s$. For clarity, we first recall the expression of c_{key} received by $\mathcal{A}_2$:

$$c_{\mathsf{key}} = (zy, \mathsf{projhash}_{\mathsf{hp}}(z)\mathsf{hash}_{\mathsf{hk}}(y)f^{\tilde{x}_1}) = (zy, \mathsf{projhash}_{\mathsf{hp}}(z)f^{(\tilde{x}_1 + b\gamma)})$$

where $z \in \mathcal{L}$, $y \in \langle \Upsilon \rangle$, and $b \in (\mathbf{Z}/q\mathbf{Z})^*$ are unique, and $\mathsf{hash}_{\Psi}(y) = f^b$. We also recall the expression of c_3, sent by $\mathcal{A}_2$ to $\mathcal{S}$. Since c_3 decrypts to α with decryption key hk, we can write:

$$c_3 = (z_3y_3, \mathsf{projhash}_{\mathsf{hp}}(z_3)f^{\alpha + b_3\gamma})$$

where $z_3 \in \mathcal{L}$, $y_3 \in \langle \Upsilon \rangle$, and $b_3 \in (\mathbf{Z}/q\mathbf{Z})^*$ are unique, and $\mathsf{hash}_{\Psi}(y_3) = f^{b_3}$. Let us denote $\pi_0 := \tilde{x}_1 + b\gamma \bmod q$, and $\pi_1 := \alpha + b_3\gamma \bmod q$. For this case to occur, it must hold that $\alpha = k_2^{-1}(m' + r\tilde{x}_1x_2) \bmod q$, so

$$\begin{aligned} & \pi_1 = k_2^{-1}(m' + r\tilde{x}_1x_2) + b_3\gamma \bmod q \\ \Leftrightarrow \quad & \tilde{x}_1 = (k_2\pi_1 - m' - k_2b_3\gamma)(x_2r)^{-1} \bmod q \end{aligned}$$

Substituting γ for $b^{-1}(\pi_0 - \tilde{x}_1)$ yields:

$$\begin{aligned} \tilde{x}_1 &= (k_2\pi_1 - m' - k_2 b_3 b^{-1}(\pi_0 - \tilde{x}_1))(x_2 r)^{-1} \bmod q \\ \Leftrightarrow \quad \tilde{x}_1(1 - k_2 b_3 (b x_2 r)^{-1}) &= (k_2\pi_1 - m' - k_2 b_3 b^{-1}\pi_0)(x_2 r)^{-1} \bmod q \end{aligned}$$

As we dealt with $b_3 = k_2^{-1} r x_2 b \bmod q$ in case 2, here $b_3 \neq k_2^{-1} r x_2 b \bmod q$, and $1 - k_2 b_3 (b x_2 r)^{-1}$ is invertible mod q so we can write:

$$\tilde{x}_1 = (k_2\pi_1 - m' - k_2 b_3 b^{-1}\pi_0)(x_2 r)^{-1}(1 - k_2 b_3 (b x_2 r)^{-1})^{-1} \bmod q, \quad (3.2)$$

where π_0, b are fixed by c_{key}; π_1, b_3 are fixed by c_3; and m', r, k_2, x_2 are also fixed from $\mathcal{A}_2$'s view. So given $\mathcal{A}_2$'s view and $\mathcal{A}_2$'s output c_3, all the terms on the right hand side of equation (3.2) are fixed. However, given Q, hp and c_{key} (which is all the information $\mathcal{A}_2$ gets prior to outputting c_3), the δ_s-smoothness of the projective hash family ensures that $\tilde{x}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. For the current case to occur, equation (3.2) must hold, thus from being given a view where $\tilde{x}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$, $\mathcal{A}_2$ has succeeded in fixing this random variable to be the exact value sampled by $\mathcal{S}$. This occurs with probability $\leqslant 1/q + \delta_s$.

Combining the above, we get that Game_2 and Game_3 differ from $\mathcal{A}_2$'s view if and only if we are in case 3. (b) or 3. (c), which occur with probability $\leqslant 1/q + \delta_s + \delta_{\mathsf{DE}} + \delta_{\mathsf{DL}}$. Thus:

$$|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_2]| \leqslant 1/q + \delta_s + \delta_{\mathsf{DE}} + \delta_{\mathsf{DL}}.$$

Game_3 to Game_4. In Game_4 check (2) is removed. Both games differ if and only if check (1) fails in both of them, while check (2) passes. If this happens $\mathcal{S}$ has decrypted c_3 to the value $\alpha = k_2^{-1}(m' + r x_1 x_2) \bmod q$. Since $\mathcal{S}$ has extracted k_2, x_2 from the simulated proofs of knowledge, r from the ECDSA signature it received and knows m', it can compute x_1 from α, thereby computing the discrete logarithm of point Q. Thus distinguishing these games reduces to the hardness of breaking the DL problem in $\mathbb{G}$. We conclude that:

$$|\Pr[\mathsf{E}_4] - \Pr[\mathsf{E}_3]| \leqslant \delta_{\mathsf{DL}}.$$

Game_4 to Game_5. The change here is exactly that between Game_1 and Game_2, thus both games are indistinguishable under the hardness of the subset membership problem and:

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_4]| \leqslant \delta_{\mathcal{L}}.$$

Game_5 to Game_6. The change here is exactly that between Game_0 and Game_1, thus both games are perfectly indistinguishable, even for an unbounded adversary, thus:

$$|\Pr[\mathsf{E}_6] - \Pr[\mathsf{E}_5]| = 0.$$

Real/Ideal executions. Putting together the above probabilities, we get that:

$$|\Pr[\mathsf{E}_6] - \Pr[\mathsf{E}_0]| \leqslant 2\delta_{\mathcal{L}} + \delta_{\mathsf{DE}} + 2\delta_{\mathsf{DL}} + 1/q + \delta_s,$$

and so, assuming the hardness of the subset membership problem, the smoothness of PHF, and the hardness of the DE problem for PHF, it holds that the real and ideal executions are computationally indistinguishable from $\mathcal{A}_2$'s view, which concludes the proof of the theorem. □

3.4 Instantiation from Class Group of Imaginary Quadratic Fields

In this section, we give an instantiation of a hash proof system with the required properties in order to apply the generic construction of the previous section. Our instantiation relies on the CL encryption scheme which we explained in detail in Chapter 2, Section 2.2. We recall the useful properties of this encryption scheme seeing how it satisfies the construction from HPS. CL is a linearly homomorphic scheme modulo a prime number p which can be chosen equal to the order of the Elliptic Curve group. We also saw that messages in CL are encoded in a cyclic subgroup F of the class group (implicitly defined by its discriminant Δ) of order q and with generator f. Furthermore, we also saw that in F it is easy to compute the discrete logarithm. In addition, from the result of [CLT18a], we discussed how to enhance the CL framework with their introduction of an hard subgroup membership assumption (HSM) applied to G and its subgroup G^q of order s.

A Hard Subgroup Membership Assumption In this paragraph we refer to previously defined definitions. Indeed, we have discussed about hard assumptions for CL in Chapter 2. We give only a brief recap of what we need for our instantiation from CL, previously discussed in Section 2.2. First, we need to define an HSM problem, and to do the task we refer to the algorithm GenGroup (see Definition 2.2.5), which consists in a pair of algorithm $(\mathsf{Gen}, \mathsf{Solve})$. The former Gen is a group generator that outputs a group with a subgroup where the discrete logarithm is easy. The latter Solve computes the discrete logarithm in the subgroup where it is easy. In Section 2.2 we presented a slightly different definition (see Definition 2.2.5) of the original GenGroup in [CLT18a], i.e. its version given in our [CCL$^+$19] GenGroup thought to take in input q. We use this last modification in our instantiation, motivated by the choice of q as the order of the subgroup F.

Remark 14. We already noted from the definition of GenGroup that it is easy to recognise valid encodings of $\widehat{G}$ while it will be not so for elements of $G \subset \widehat{G}$. This is an important difference with Paillier's encryption, and one of the reason why Lindell's L_{PDL} proof does not work in our setting[6].

We do not give details about the HSM assumption here, we have already presented it in Definition 2.2.6. From Definition 2.2.5, one has $G = F \times G^q$. Informally, the HSM assumption says that it is hard to distinguish the elements of G^q in G.

[6]See 3.2.6 for $\mathsf{R}_{\mathsf{HPS-CL}}$. L_{PDL} is the proof of membership used in [Lin17] with Paillier encryption and the elliptic point Q.

Class groups Our instantiation makes use of class groups of orders of imaginary quadratic fields (see Chapter 2) [7]. We discussed in details the construction of CL scheme, then we limit ourselves to recall only the necessary properties and results. Let q be a prime. We construct a fundamental discriminant $\Delta_K := -q \cdot \tilde{q}$ where $\tilde{q}$ is a prime such that $q \cdot \tilde{q} \equiv -1 \bmod 4$ and $(q/\tilde{q}) = -1$. We then consider the non-maximal order of discriminant $\Delta_q := q^2 \cdot \Delta_K$ and its class group $\widehat{G} := Cl(\Delta_q)$ whose order is $h(\Delta_q) = q \cdot h(\Delta_K)$. This number is known to satisfy the following inequality (see [Coh00, p. 295] for instance): $h(\Delta_K) < \frac{1}{\pi} \log |\Delta_K| \sqrt{|\Delta_K|}$ which is the bound we take for $\tilde{s}$ (a slightly better bound can be computed from the analytic class number formula). Elements of $\widehat{G}$ are classes of ideals of the order of discriminant Δ_q. Such classes can be represented by a unique reduced ideal. Moreover, ideals can be represented using the so-called two elements representation which correspond to their basis as a lattice of dimension two. Then, classes can be uniquely represented by two integers (a, b), $a, b < \sqrt{|\Delta_q|}$ and one can efficiently verify that this indeed defines an element of $\widehat{G}$ (by checking if $b^2 \equiv \Delta_q \bmod 4a$). The arithmetic in class groups (which corresponds to reduction and composition of quadratic forms) is very efficient: the algorithms have a quasi linear time complexity using fast arithmetic (see [Coh00]). We build a generator g_q of a cyclic subgroup of $q-$th powers of $\widehat{G}$, and denote $G^q := \langle g_q \rangle$. Then we build a generator f for the subgroup F of order q, and then set $g := f \cdot g_q$ as a generator of a cyclic subgroup G of $Cl(\Delta_q)$ of order $q \cdot s$, where s is unknown. Computing discrete logarithms is easy in F thanks to the following facts. We denote the surjection $\bar{\varphi}_q : Cl(\Delta_q) \longrightarrow Cl(\Delta_K)$. From [CL09, Lemma 1][8], its kernel is cyclic of order q and is generated by f represented by (q^2, q). Moreover, if $1 \leqslant m \leqslant q-1$ then, once reduced, f^m is of the form $(q^2, L(m)q)$ where $L(m)$ is the odd integer in $[-q, q]$ such that $L(m) \equiv 1/m \bmod q$, which gives the efficient algorithm to compute discrete logarithms in $\langle f \rangle$. The Gen algorithm can be found in Figure 2.1.

3.4.1 A Smooth Homomorphic Hash Proof System from HSM

We set $\mathcal{X} := G$ and $\mathcal{L} := G^q$ then $\mathcal{X} \cap \mathcal{L} = G^q$ and the HSM assumption states that it is hard to distinguish random elements of G from those of G^q. This clearly implies the hardness of the subset membership problem, *i.e.*, it is hard to distinguish random elements of $G \backslash G^q$ from those of G^q. We see below how the properties we discussed in Section 3.2 are satisfied by CL.

Let $\mathcal{D}$ be a distribution over the integers such that the distribution $\{g^w, w \hookleftarrow \mathcal{D}\}$ is at distance $\delta_{\mathcal{D}} \leq 2^{-\lambda}$ of the uniform distribution in G.

Associated projective hash family. Let PHF be the projective hash family associated to the above subset membership problem, the description of which specifies:

- A hash key space $K := \mathbf{Z}$.
- A keyed hash function, with input and output domain G, s.t., for $\mathsf{hk} \hookleftarrow \mathcal{D}$, and for $x \in G$, $\mathsf{hash}_{\mathsf{hk}}(x) := x^{\mathsf{hk}}$.

[7] We also refer the interested reader to [BH01] for background on this algebraic object and its early use in cryptography

[8] We recall it in Proposition 2.2.11

- An auxiliary function $\mathsf{projkg} : K \mapsto G^q$ such that for $\mathsf{hk} \in K$, $\mathsf{projkg}(\mathsf{hk}) := \mathsf{hash}_{\mathsf{hk}}(g_q) = g_q^{\mathsf{hk}}$. Notice that for a hash key hk, and for $x \in G^q$, the knowledge of $\mathsf{projkg}(\mathsf{hk})$ completely determines the value $\mathsf{hash}_{\mathsf{hk}}(x)$.

- An efficient public evaluation function, such that, for $x \in G^q$ with witness w such that $x = g_q^w$ one can efficiently compute $\mathsf{hash}_{\mathsf{hk}}(x) = \mathsf{projkg}(\mathsf{hk})^w = x^{\mathsf{hk}}$ knowing only the value of the auxiliary function $\mathsf{projkg}(\mathsf{hk})$ (but not hk).

Before proving the lemma about the smoothness, we need the following preliminar result about distributions.

Lemma 3.4.1. Let X be a discrete random variable at statistical distance ϵ from the uniform distribution over $\mathbf{Z}/ab\mathbf{Z}$ for positive integers a and b such that $\gcd(a, b) = 1$. And let X_a (resp. X_b) be the random variable defined as $X_a := X \mod a$ (resp. $X_b := X \mod b$). Then the random variables X_a and X_b are less than ϵ close to the uniform distributions in $\mathbf{Z}/a\mathbf{Z}$ and $\mathbf{Z}/b\mathbf{Z}$ respectively. Moreover, even knowing X_b, X_a remains at statistical distance less than 2ϵ of the uniform distribution in $\mathbf{Z}/a\mathbf{Z}$ (and vice versa).

Proof. Let $\mathcal{C}$ be an algorithm which takes as input a tuple $(a, b, x) \in \mathbf{N}^2 \times \mathbf{Z}/ab\mathbf{Z}$, which can either be a sample of the distribution:

$$\mathcal{U} := \{a, b, x | \gcd(a, b) = 1 \wedge x \xleftarrow{\$} \mathbf{Z}/ab\mathbf{Z}\}$$

or a sample of:

$$\mathcal{V} := \{a, b, x | \gcd(a, b) = 1 \ \wedge \ x \hookleftarrow \mathcal{D}\},$$

where $\mathcal{D}$ is a distribution at statistical distance ϵ of the uniform distribution over $\mathbf{Z}/ab\mathbf{Z}$, and outputs a bit. Since distributions $\mathcal{U}$ and $\mathcal{V}$ are at statistical distance ϵ, for any such algorithm $\mathcal{C}$, it holds that:

$$|\Pr[\mathcal{C}(\mathcal{U}) \to 1] - \Pr[\mathcal{C}(\mathcal{V}) \to 1]| \leq \epsilon.$$

We further denote $\mathcal{U}_{\mathcal{A}} := \{a, b, x_b, x_a | (a, b, x) \hookleftarrow \mathcal{V};\ x_b \leftarrow x \mod b;\ x_a \xleftarrow{\$} \mathbf{Z}/a\mathbf{Z}\}$ and $\mathcal{V}_{\mathcal{A}} := \{a, b, x_b, x_a\ |(a, b, x) \hookleftarrow \mathcal{V};\ x_b \leftarrow x \mod b;\ x_a \leftarrow x \mod a\}$. Consider any algorithm $\mathcal{A}$ which takes as input a sample (a, b, x_b, x_a^*) of either $\mathcal{U}_{\mathcal{A}}$ or $\mathcal{V}_{A}$, and outputs a bit β'. $\mathcal{A}$'s goal is to tell whether x_a^* is sampled uniformly at random from $\mathbf{Z}/a\mathbf{Z}$ or if $x_a^* \leftarrow x \mod a$. We demonstrate that if $\mathcal{A}$ has significant probability in distinguishing both input types, then $\mathcal{C}$ can use $\mathcal{A}$ to distinguish distributions $\mathcal{U}$ and $\mathcal{V}$. We describe the steps of $\mathcal{C}$ below:

$\underline{\mathcal{C}(a, b, x):}$

```
1. Set $x_b \leftarrow x \mod b$
2. Sample $\beta^* \xleftarrow{\$} \{0, 1\}$
3. If $\beta^* = 0$, then $x_a^* \xleftarrow{\$} \mathbf{Z}/a\mathbf{Z}$
4. Else if $\beta^* = 1$, then $x_a^* \leftarrow x \mod a$
5. $\beta' \leftarrow \mathcal{A}(a, b, x_b, x_a^*)$
6. If $\beta = \beta'$ return 1
7. Else return 0.
```

If $\mathcal{C}$ gets as input an element of $\mathcal{U}$ whatever the value of β^*, x_a^* follows the uniform distribution modulo a and is independent of x_b. So $\mathcal{A}$'s success probability in outputting β' equal to β^* is 1/2.

$$\Pr[\mathcal{A}(a, b, x_b, x_a^*) \to \beta^* | (a, b, x) \hookleftarrow \mathcal{U}] = 1/2$$

and so

$$\Pr[\mathcal{C}(\mathcal{U}) \to 1] = 1/2$$

On the other hand if $(a, b, x) \hookleftarrow \mathcal{V}$, then $\mathcal{C}$ outputs 1 if $\mathcal{A}$ guesses the correct bit β^* (when its inputs are either in $\mathcal{U}_\mathcal{A}$ or $\mathcal{V}_\mathcal{A}$ as expected).

$$\Pr[\mathcal{C}(\mathcal{V}) \to 1] = \Pr[\mathcal{A} \to \beta^* | (a, b, x) \hookleftarrow \mathcal{V}]$$

And so

$$\begin{aligned} |\Pr[\mathcal{C}(\mathcal{U}) \to 1] - \Pr[\mathcal{C}(\mathcal{V}) \to 1]| &= |\Pr[\mathcal{A} \to \beta^* | (a, b, x) \hookleftarrow \mathcal{V}] - 1/2| \\ &= 1/2 \cdot |\Pr[\mathcal{A}(\mathcal{U}_\mathcal{A}) \to 1] - \Pr[\mathcal{A}(\mathcal{V}_A) \to 1]|. \end{aligned}$$

Since distributions $\mathcal{U}$ and $\mathcal{V}$ are at statistical distance ϵ, it holds that $|\Pr[\mathcal{C}(\mathcal{U}) \to 1] - \Pr[\mathcal{C}(\mathcal{V}) \to 1]| \leq \epsilon$, and so for any algorithm $\mathcal{A}$ as above:

$$|\Pr[\mathcal{A}(\mathcal{U}_\mathcal{A}) \to 1] - \Pr[\mathcal{A}(\mathcal{V}_A) \to 1]| \leq 2\epsilon.$$

Thus the statistical distance between $\mathcal{U}_\mathcal{A}$ and $\mathcal{V}_\mathcal{A}$ is smaller than 2ϵ, which implies that even given $x \mod b$, the value of $x \mod a$ remains at negligible statistical distance 2ϵ of the uniform distribution modulo a, which concludes the proof. $\square$

Lemma 3.4.2 (Smoothness)**.** The projective hash family PHF is δ_s-smooth over G in F, with $\delta_s \leqslant 2\delta_\mathcal{D}$, *i.e.*, for any $x \in G \backslash G^q$, $\pi \leftarrow f^\gamma \in F \subset G$ where $\gamma \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $k \hookleftarrow \mathcal{D}$, the distributions $\mathcal{D}_1 := \{x, g_q^k, \pi \cdot x^k\}$ and $\mathcal{D}_2 := \{x, g_q^k, x^k\}$ are less than $2\delta_\mathcal{D}$-close.

Proof. For $x \in G \backslash G^q$, there exist $a \in \mathbf{Z}/s\mathbf{Z}$ and $b \in (\mathbf{Z}/q\mathbf{Z})^*$ such that $x = g_q^a f^b$. Thus we can write $\mathcal{D}_1 = \{g_q^a f^b, g_q^k, g_q^{a \cdot k} f^{b \cdot k + \gamma}\}$ and $\mathcal{D}_2 = \{g_q^a f^b, g_q^k, g_q^{a \cdot k} f^{b \cdot k}\}$. It remains to study the statistical distance of the third coordinates of the two distributions, given the two first coordinates, *i.e*, if ($a \bmod s$), ($b \bmod q$), and ($k \bmod s$) are fixed. This is the statistical between $X := b \cdot k + \gamma$ and $Y := b \cdot k$ in $\mathbf{Z}/q\mathbf{Z}$. Since γ is uniform in $\mathbf{Z}/q\mathbf{Z}$, X is the uniform distribution. As $\mathcal{D}$ is by definition at statistical distance $\delta_{\mathcal{D}}$ from the uniform distribution modulo $q \cdot s$, and $\gcd(q, s) = 1$, one can prove (cf. Lemma 3.4.1) that even knowing ($k \bmod s$), the distribution of ($k \bmod q$) is at distance less than $2\delta_{\mathcal{D}}$ from the uniform distribution over $\mathbf{Z}/q\mathbf{Z}$. As a result, the distance between X and Y is bounded by $2\delta_{\mathcal{D}}$, which concludes the proof. $\square$

Linearly homomorphic. For all $\mathsf{hk} \in \mathbf{Z}$, and $u_1, u_2 \in G$, $\mathsf{hash}_{\mathsf{hk}}(u_1) \cdot \mathsf{hash}_{\mathsf{hk}}(u_2) = u_1^{\mathsf{hk}} \cdot u_2^{\mathsf{hk}} = (u_1 \cdot u_2)^{\mathsf{hk}} = \mathsf{hash}_{\mathsf{hk}}(u_1 \cdot u_2)$. Thus $\mathsf{hash}_{\mathsf{hk}}$ is a homomorphism for each hk.

Key homomorphic. The hash key space $(\mathbf{Z}, +)$ is an Abelian group, $(G, \cdot)$ is a multiplicative finite Abelian group; and $\forall x \in G$ and $\forall \mathsf{hk}_0, \mathsf{hk}_1 \in \mathbf{Z}$, it holds that $\mathsf{hash}(\mathsf{hk}_0, x) \cdot \mathsf{hash}(\mathsf{hk}_1, x) = x^{\mathsf{hk}_0} \cdot x^{\mathsf{hk}_1} = x^{\mathsf{hk}_0 + \mathsf{hk}_1} = \mathsf{hash}(\mathsf{hk}_0 + \mathsf{hk}_1, x)$.

(f, F)-Decomposability. The group G is the direct product of G^q and $F = \langle f \rangle$. Moreover $\forall \mathsf{hk} \in \mathbf{Z}$, $\mathsf{hash}_{\mathsf{hk}}(f) = f^{\mathsf{hk}} \in F$. Thus we set $\Upsilon := f$, such that PHF is (f, F)-decomposable.

Resulting encryption scheme. A direct application of the resulting scheme in Section 3.2 can be instantiead with the $\mathsf{HSM} - \mathsf{CL}$ scheme in [CLT18a], which is linearly homomorphic modulo q and $\mathsf{ind} - \mathsf{cpa}$ under the HSM assumption (see Section 2.2). We already discussed this scheme and we refer to Figure 2.2 in Section 2.2. Note that here the secret key x (and the randomness r) is drawn with a distribution $\mathcal{D}_q$ such that $\{g_q^x, x \hookleftarrow \mathcal{D}_q\}$ is at distance less than $2^{-\lambda}$ from the uniform distribution in G^q, this does not change the view of the attacker. Let $S := 2^{\lambda-2} \cdot \tilde{s}$. In practice, we will use for $\mathcal{D}_q$ the uniform distribution on $\{0, \dots, S\}$.

The double encoding problem. In this context, the DE problem is δ_{DE}-hard for the one way function $\mathsf{exp}_{\mathbb{G}} : x \mapsto xP$ if for any PPT $\mathcal{A}$, it holds that:

$$\delta_{\mathsf{DE}} \geqslant \Pr\left[\begin{array}{c|l} u_1, u_2 \in G \setminus G^q, & \mathsf{pp}_{\mathbb{G}} := (\mathbb{G}, P, q) \\ u_2 \cdot u_1^{-1} \in G \setminus G^q & \mathsf{pp}_G := (\tilde{s}, f, g_q, G, F) \leftarrow \mathsf{Gen}(1^\lambda, q) \\ \text{and } \mathsf{hp} = g_q^{\mathsf{hk}} & x \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z},\ Q := x \cdot P \\ & (\mathsf{hp}, (u_1, u_1^{\mathsf{hk}} \cdot f^x), (u_2, u_2^{\mathsf{hk}} \cdot f^x)) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathbb{G}}, \mathsf{pp}_G, Q) \end{array}\right].$$

On the hardness of the DE problem for the $\mathsf{HSM\text{-}CL}$ encryption scheme. As explained in Lemma 3.2.2, breaking the DE problem in sub-exponential time would give a sub-exponential algorithm to compute discrete logarithms in elliptic curves (for which there exist only exponential algorithms).

3.4.2 A zero-knowledge proof for $\mathsf{R}_{\mathsf{CL-DL}}$

We describe here the ZKPoK for $\mathsf{R}_{\mathsf{HPS-DL}}$ used for our instantiation with the encryption scheme of Fig. 2.2 and denote it $\mathsf{R}_{\mathsf{CL-DL}}$. It relies on the Schnorr-like GPS (statistically)

Input : (r, x_1) and $(\mathsf{pk}, c_1, c_2, Q, P)$		Input : $(\mathsf{pk}, c_1, c_2, Q, P)$
`Repeat` ℓ `times`		
$r_1 \xleftarrow{\$} [0, A[$; $r_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$t_1 \leftarrow \mathsf{pk}^{r_1} f^{r_2}$; $t_2 \leftarrow r_2 P$; $t_3 \leftarrow g_q^{r_1}$	$\xrightarrow{t_1, t_2, t_3}$	
	$\xleftarrow{k}$	$k \xleftarrow{\$} \{0, 1\}$
$u_1 \leftarrow r_1 + kr$ in $\mathbf{Z}$		
$u_2 \leftarrow r_2 + kx_1 \bmod q$	$\xrightarrow{u_1, u_2}$	`Check` $u_1 \in [0, A + S[$
		$t_1 \cdot c_2^k = \mathsf{pk}^{u_1} \cdot f^{u_2}$
		$t_2 + [k]Q = [u_2]P$
		$t_3 \cdot c_1^k = g_q^{u_1}$

Figure 3.7: The zero-knowledge proof of knowledge for $\mathsf{R}_{\mathsf{CL-DL}}$

zero-knowledge identification scheme [GPS06] that we turn into a zero-knowledge proof of knowledge of the randomness used for encryption and of the discrete logarithm of an element on an elliptic curve, using a binary challenge. This proof is partly performed in a group of unknown order.

We denote $c_{\mathsf{key}} := (c_1, c_2)$. If c_{key} is a valid encryption of x_1 under public key pk it holds that $c_{\mathsf{key}} = (g_q^r, f^{x_1}\mathsf{pk}^r)$ for some $r \in \{0, \ldots, S\}$. The protocol in Figure 3.7 provides a ZKPoK for the following relation:

$$\mathsf{R}_{\mathsf{CL-DL}} := \{(\mathsf{pk}, (c_1, c_2), Q_1); (x_1, r) \ \mid \ c_1 = g_q^r \wedge c_2 = f^{x_1}\mathsf{pk}^r \wedge Q_1 = x_1 G\}.$$

The following theorem states the security of the zero-knowledge proof of knowledge for relation $\mathsf{R}_{\mathsf{CL-DL}}$.

Theorem 3.4.3. The protocol described in Figure 3.7 is a statistical zero-knowledge proof of knowledge with soundness $2^{-\ell}$, as long as ℓ is polynomial and $\ell S/A$ is negligible, where A is a positive integer.

Proof. We prove completeness, soundness and zero-knowledge. Completeness follows easily by observing that when $((\mathsf{pk}, (c_1, c_2), Q_1); (x_1, r)) \in \mathsf{R}_{\mathsf{CL-DL}}$, for any $k \in \{0, 1\}$ the values computed by an honest prover will indeed verify the four relations checked by the verifier. For soundness, the protocol is in fact special sound. Indeed notice that for committed values t_1, t_2, t_3, if a prover P^* can answer correctly for two different values of k, he must be able to answer to challenges 0 and 1 with u_1, u_2 and u_1', u_2', where u_1 and u_1' are smaller than $A + S - 1$, such that $u_2 P = u_2' P - Q$, $\mathsf{pk}^{u_1} f^{u_2} c_2 = \mathsf{pk}^{u_1'} f^{u_2'}$ and $g_q^{u_1} c_1 = g_q^{u_1'}$. Let $\sigma_1 \leftarrow u_1' - u_1$, $\sigma_2 \leftarrow u_2' - u_2 \mod q$; we obtain $g_q^{\sigma_1} = c_1$, $\sigma_2 P = Q$ and $\mathsf{pk}^{\sigma_1} f^{\sigma_2'} = c_2$. Thus P^* can easily compute $x_1 \leftarrow \sigma_2 \mod q$ and $r \leftarrow \sigma_1$ in $\mathbf{Z}$.

While this gives a soundness error of $1/2$, the soundness is amplified to $2^{-\ell}$ by repeating the protocol sequentially ℓ times.

For zero-knowledge, we must exhibit a simulator $\mathcal{S}$ which, given the code of some verifier V^*, produces a transcript indistinguishable from that which would be produced between V^* and an honest prover P (proving the knowledge of a tuple in $\mathsf{R}_{\mathsf{CL-DL}}$) without knowing the witnesses (x_1, r) for $(\mathsf{pk}, (c_1, c_2), Q_1)$ in the relation $\mathsf{R}_{\mathsf{CL-DL}}$.

The potentially malicious verifier may use an adaptive strategy to bias the choice of the challenges to learn information about (r, x_1). This implies that challenges may not be randomly chosen, which must be taken into account in the security proof.

We describe an expected polynomial time simulation of the communication between a prover P and a malicious verifier V^* for one round of the proof. Since the simulated round may not be the first round, we assume V^* has already obtained data, denoted by hist, from previous interactions with P. Then P sends the commitments t_1, t_2, t_3 and V^* chooses – possibly using hist and t_1, t_2, t_3 – the challenge $k(t_1, t_2, t_3, \mathsf{hist})$.

Description of the simulator: Consider the simulator $\mathcal{S}$ which simulates a given round of identification as follows:

1. $\mathcal{S}$ chooses random values $\bar{k} \in \{0, 1\}$, $\bar{u}_1 \in [S-1, A-1]$ and $\bar{u}_2 \in \mathbf{Z}/q\mathbf{Z}$.
2. $\mathcal{S}$ computes $\bar{t_1} \leftarrow \mathsf{pk}^{\bar{u}_1} f^{\bar{u}_2} / c_2^{\bar{k}}$; $\bar{t_2} \leftarrow [\bar{u}_2]P - [\bar{k}]Q$ and $\bar{t_3} \leftarrow g_q^{\bar{u}_1} / c_1^{\bar{k}}$, and sends $\bar{t_1}$, $\bar{t_2}$ and $\bar{t_3}$ to V^*.
3. $\mathcal{S}$ receives $k(\bar{t_1}, \bar{t_2}, \bar{t_3}, \mathsf{hist})$ from V^*.
4. If $k(\bar{t_1}, \bar{t_2}, \bar{t_3}, \mathsf{hist}) \neq \bar{k}$ then return to step 1, else return $(\bar{t_1}, \bar{t_2}, \bar{t_3}, \bar{k}, \bar{u}_1, \bar{u}_2)$.

To demonstrate that the proof is indeed zero-knowledge, we need to justify that the distribution output by the simulator is statistically close to that output in a real execution of the protocol, and that the simulation runs in expected polynomial time.

Intuitively, sampling the randomness r from a large enough distribution – i.e. as long as $S << A$ – ensures that the distribution of t_1, t_2, t_3 in a real execution is statistically close[9] to that in a simulated execution.

The analysis of the above statistical distance Σ between the distribution of tuples output by the simulator and that of tuples output by a real execution of the protocol is quite tedeous and similar to that of [GPS06]. We do not provide the details here but applying their analysis to our setting allows us obtain the following bound:

$$\Sigma < \frac{8S}{A}.$$

Thus the real and simulated distributions are statistically indistinguishable if S/A is negligible.

Running time of the Simulator: We now need to ensure that the simulator runs in expected polynomial time. To see this, notice that step 3 outputs a tuple $(\bar{t_1}, \bar{t_2}, \bar{t_3}, \bar{k}, \bar{u}_1, \bar{u}_2)$ if $k(\bar{t_1}, \bar{t_2}, \bar{t_3}, \mathsf{hist}) = \bar{k}$. Since $\bar{k}$ is sampled at random from $\{0, 1\}$, the expected number of iterations of the loop is 2. Therefore the complexity of the simulation of all ℓ rounds is $O(\ell)$.

Thus if $\ell S/A$ is negligible and ℓ is polynomial, the protocol is statistically zero-knowledge. $\square$

[9]The distributions cannot be distinguished by any algorithm, even using an infinite computational power, but only accessing a polynomial number of triplets of both distributions

Further improvements The proof for $\mathsf{R}_{\mathsf{CL-DL}}$ in Figure 3.7, which uses binary challenges, has been improved in our work [CCL+20]. In particular, with the introduction of a trick based on the *lowest common multiple*, it is possible to obtain a new ZKPoK for a slightly different relation $\mathsf{R}_{\mathsf{CL-lcm}}$ which is more efficient than the previous one. Indeed, the new ZKPoK works with an extended challenge space, which is $\{0,1\}^{10}$ instead of $\{0,1\}$. As a consequence, the proof has to be repeated $\lambda/10$ times for a soundness of $2^{-\lambda}$, instead of λ times. Furthermore, in [CCL+20] we presented a ZKAoK for a slightly different relation $\tilde{\mathsf{R}}_{\mathsf{CL-DL}}$ which relies on two hard assumptions, the *strong root assumption* and the *low order assumption.* This ZKAoK is significantly more efficient than the ZKPoK, however it requires that the deterministic generator g_q is randomized. For several reasons we will discuss these improvements in Section 4.4. First, it is because Chapter 4 is completely dedicated to our work [CCL+20] and the introduction of the ZKAoK for $\tilde{\mathsf{R}}_{\mathsf{CL-DL}}$ is a consequence of one of the main contributions in that work, and second because we introduce and discuss the strong root assumption and the low order assumption for the first time in Chapter 4. Then, we decided to put the discussion on the new proof and the improvements on the proof for $\tilde{\mathsf{R}}_{\mathsf{CL-DL}}$ together in only one section.

3.4.3 Two-Party Distributed ECDSA Protocol from HSM

The protocol results from a direct application of Subsection 3.3 using the HPS defined in Subsection 3.4.1, an the $\mathsf{R}_{\mathsf{CL-DL}}$ proof of the previous subsection. Therefore we present the protocol in Figure 3.8 and state the following theorem.

Theorem 3.4.4. Assuming $\mathsf{GenGroup}$ is the generator of a HSM group with easy DL subgroup F, then the protocol in Figure 3.8 securely computes $\mathcal{F}_{ECDSA}$ in the $(\mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com-zk}})-$hybrid model in the presence of a malicious static adversary (under the ideal/real definition).

3.5 Efficiency comparison

In this section we compare an implementation of our protocol with Lindell's protocol of [Lin17]. For fair comparison, we both protocols are implemented with the Pari C Library ([PAR18]), as this library handles arithmetic in class groups, $\mathbf{Z}/n\mathbf{Z}$ and elliptic curves. In particular, in this library, exponentiations in $\mathbf{Z}/n\mathbf{Z}$ and in class groups both use the same sliding window method. The running times are measured on a single core of an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz (even if key generation can easily be parallelized). We do not implement commitments (this does not bias the comparison as they appear with equal weight in both schemes), and we only measure computation time and do not include communication (again this is fair as communication is similar).

As in [Lin17], we ran our implementation on the standard NIST curves P-256, P-384 and P-521, corresponding to levels of security 128, 192 and 256. For the encryption scheme, we start with a 112 bit security, as in [Lin17], but also study the case where its level of security matches the security of the elliptic curves.

Again as in [Lin17], we fixed the number of rounds in zero knowledge proofs to reach a statistical soundness error of 2^{-40}. For the distributions we also set the parameters to

P_1	$\mathsf{IKeyGen}(\mathbb{G}, P, q, \widehat{G}, g_q)$	P_2
$x_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $Q_1 \leftarrow x_1 P$		
	$\xrightarrow{(\mathsf{com\text{-}prove}, 1, Q_1, x_1)} \mathcal{F}^{\mathsf{R_{DL}}}_{\mathsf{zk-com}} \xrightarrow{(\mathsf{proof\text{-}receipt}, 1)}$	
P_1 aborts if $(\mathsf{proof}, 2, Q_2)$ not received.		$x_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $Q_2 \leftarrow x_1 P$
	$\xleftarrow{(\mathsf{proof}, 2, Q_2)} \mathcal{F}^{\mathsf{R_{DL}}}_{\mathsf{zk}} \xleftarrow{(\mathsf{prove}, 2, Q_2, x_2)}$	
	$\xrightarrow{(\mathsf{decom\text{-}proof}, 1)} \mathcal{F}^{\mathsf{R_{DL}}}_{\mathsf{zk-com}} \xrightarrow{(\mathsf{decom\text{-}proof}, 1, Q_1)}$	
$x, \rho \hookleftarrow \mathcal{D}_q$ $h \leftarrow g_q^x$		
$c_{key} = (c_{key,1}, c_{key,2})$ $= (g_q^\rho, h^\rho f^{x_1})$	$\xrightarrow{(\mathsf{prove}, 3, (h, c_{key}, Q_1), (x_1, \rho))} L_{\mathsf{CL-DL}} \xrightarrow{(\mathsf{proof}, 3, (h, c_{key}, Q_1))}$	P_2 aborts unless $(\mathsf{decom\text{-}proof}, 1, Q_1)$, $(\mathsf{proof}, 3, (h, c_{key}, Q_1))$ received and $h \in \widehat{G}$.
$Q \leftarrow x_1 Q_2$		$Q \leftarrow x_2 Q_1$

P_1	$\mathsf{ISign}(m, sid)$	P_2
$k_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $R_1 \leftarrow k_1 P$		
	$\xrightarrow{(\mathsf{com\text{-}prove}, \mathsf{sid}\|\|1, R_1, k_1)} \mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{zk-com}} \xrightarrow{(\mathsf{proof\text{-}receipt}, \mathsf{sid}\|\|1)}$	
		$k_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $R_2 \leftarrow k_2 P$
P_1 aborts if $(\mathsf{proof}, sid\|\|2, R_2)$ not received.	$\xleftarrow{(\mathsf{proof}, \mathsf{sid}\|\|2, R_2)} \mathcal{F}^{\mathsf{R_{DL}}}_{\mathsf{zk}} \xleftarrow{(\mathsf{prove}, \mathsf{sid}\|\|2, R_2, k_2)}$	
	$\xrightarrow{(\mathsf{decom\text{-}proof}, \mathsf{sid}\|\|1)} \mathcal{F}^{\mathsf{R_{DL}}}_{\mathsf{zk-com}} \xrightarrow{(\mathsf{decom\text{-}proof}, \mathsf{sid}\|\|1, R_1)}$	P_2 aborts if $(\mathsf{decom\text{-}proof}, sid\|\|1, R_1)$ not received. $m' \leftarrow H(m)$
$R = (r_x, r_y) \leftarrow k_1 R_2$ $r \leftarrow r_x \mod q$		$R = (r_x, r_y) \leftarrow k_2 R_1$ $r \leftarrow r_x \mod q$ $\tau \hookleftarrow \mathcal{D}_q$ $c_1 = (c_{1,1}, c_{1,2}) \leftarrow (g_q^\tau, h^\tau f^{k_2^{-1} m'})$ $c_2 = (c_{2,1}, c_{2,2}) \leftarrow (c_{key,1}^{k_2^{-1} r x_2}, c_{key,2}^{k_2^{-1} r x_2})$ $c_3 = (c_{3,1}, c_{3,2}) \leftarrow (c_{1,1} c_{2,1}, c_{1,2} c_{2,2})$
	$\xleftarrow{\quad c_3 \quad}$	
$\alpha \leftarrow \mathsf{Solve}(c_{3,2}/c_{3,1}^x)$ $\hat{s} \leftarrow \alpha \cdot k_1^{-1}$ $s \leftarrow \min(\hat{s}, q - \hat{s})$ If not $\mathsf{Verif}(Q, m, (r, s))$ P_1 aborts Else Return (r, s)		

Figure 3.8: Two-Party ECDSA from HSM

get statistical error of 2^{-40}. The zero knowledge proofs for R_{DL} are implemented with the Schnorr protocol.

In the following, we review the theoretical complexity and experimental results of both schemes, before comparing them. In terms of theoretical complexity, exponentiations in the encryption schemes dominate the computation as elliptic curve operations are much cheaper. Thus, we only count these exponentiations; we will see this results in an accurate prediction of experimental timings.

3.5.1 Lindell's Protocol with Paillier's Encryption Scheme

The key generation uses on average 360 Paillier exponentiations (of the form $r^N \bmod N^2$) but not all of them are full exponentiations. The signing phase uses only 2 Paillier exponentiations.
The timings corresponds to the mean of several experiments (30 to 1000 depending on the security level). The timings are quite stable other than the generation of the RSA modulus in the key generation. We use standard RSA integers (*i.e.*, not strong prime factors) as this would be too slow for high security levels. For example, for 256 bits security (15360 bits modulus), the generation of the modulus takes 95 seconds (mean of 30 experiments) with a standard deviation of 56s. For the rest of the protocol the experimental timings are roughly equal to the number of exponentiations multiplied by the cost of one exponentiation.
The results are summarized in Figures 3.9a and 3.9b. Timings are given in milliseconds and sizes in bits. The columns corresponds to the elliptic curve used for ECDSA, the security parameter in bits for the encryption scheme, the corresponding modulus bit size, the timings of one Paillier exponentiation, of the key generation and of the signing phase and the total communication in bits for two phases. Modulus sizes are set according to the NIST recommendations.
Note that for the first line, we use a 2048 bits modulus as in [Lin17] and we obtain a similar experimental result.

3.5.2 Our Protocol with $\mathsf{HSM}-\mathsf{CL}$ Encryption Scheme

The key generation uses a total of 160 class group exponentiations (of the form g_q^r in the class group of discriminant $\Delta_q = -q^3 \cdot \tilde{q}$). This corresponds to the 40 rounds of the $\mathsf{R}_{\mathsf{CL-DL}}$ zero-knowledge proof of knowledge of Figure 3.7. Note that exponentiations in $\langle f \rangle$ are almost free. Signing uses 3 class group exponentiations (one encryption and one decryption).
We use the same number of experiments as for Lindell's protocol. Here timings are very stable. Indeed during key generation, we only compute the public key $h \leftarrow g_q^x$ with one exponentiation, as the output of Gen (mainly the discriminant Δ_q of the class group and the generator g_q) is a common public parameter that only depends on the cardinality q of the elliptic curve. As a result this can be considered as an input of the protocol, as the same group can be used by all users. Moreover, doing this does not change the global result of the comparison with Lindell's protocol: the running time of Gen is dominated by the generation of $\tilde{q}$, a prime of size much smaller than the factor of the RSA modulus. So even if we add this running time in the Keygen column, this does not affect the results

Curve	Sec. Param.	Modulus	Expo. (ms)	Keygen (ms)	Signing (ms)
P-256	112	2048	**7**	**2 133**	**20**
P-256	128	3072	**22**	**6 340**	**49**
P-384	192	7680	214	65 986	**437**
P-521	256	15360	1196	429 965	2 415

(a) Lindell's Protocol with Paillier - Timing

Curve	Sec. Param.	Modulus	Keygen (b)	Signing (b)
P-256	112	2048	881 901	5 636
P-256	128	3072	1 317 101	7 684
P-384	192	7680	3 280 429	17 668
P-521	256	15360	6 549 402	33 832

(b) Lindell's Protocol with Paillier - Communication

Curve	Sec. Param.	Discriminant	Expo. (ms)	Keygen (ms)	Signing (ms)
P-256	112	1348	32	5 521	101
P-256	128	1827	55	9 350	170
P-384	192	3598	**212**	**35 491**	649
P-521	256	5971	**623**	**103 095**	**1 888**

(c) Our Protocol with $\mathsf{HSM} - \mathsf{CL}$ - Timing

Curve	Sec. Param.	Discriminant	Keygen (b)	Signing (b)
P-256	112	1348	**178 668**	**4 748**
P-256	128	1827	**227 526**	**5 706**
P-384	192	3598	**427 112**	**10 272**
P-521	256	5971	**688 498**	**16 078**

(d) Our Protocol with $\mathsf{HSM} - \mathsf{CL}$ - Communication

Figure 3.9: Experimental results (timings in ms, sizes in bits)

of our comparisons for any of the considered security levels.

The results are summarized in Figures 3.9c and 3.9d. Timings are given in milliseconds and sizes in bits. The columns correspond to the elliptic curve used for ECDSA, the security parameter in bits for the encryption scheme, the corresponding fundamental discriminant $\Delta_K = -q \cdot \tilde{q}$ bit size, the timings of one class group exponentiation, of the key generation and of the signing phase and the total communication in bits for two phases. The discriminant sizes are chosen according to [BJS10].

3.5.3 Comparison

Figure 3.9 shows that Lindell's protocol is faster for both key generation and signing for standard security levels for the encryption scheme (112 and 128 bits of security) while our solution remains of the same order of magnitude. However for high security levels, our solution becomes faster (in terms of key generation from a 192-bits security level and for both key generation and signing from a 256-bits security level).

In terms of communications, our solution outperforms the scheme of Lindell at all levels of security by a factor 5 to 10 for Keygen. For Signing, we gain 15% for basic security to a factor 2 at 256-bits security level. In terms of rounds, our protocol uses 126 rounds for Keygen and Lindell's protocol uses 175 rounds, so we get a 28% gain. Both protocol use 7 rounds for Signing.

This situation can be explained by the following facts. Firstly we use less than half the number of exponentiations in the key generation as we do not need a range proof: our message space is $\mathbf{Z}/q\mathbf{Z}$ as the CL encryption scheme is homomorphic modulo a prime. Secondly, with class groups of quadratic fields we can use lower parameters than with $\mathbf{Z}/n\mathbf{Z}$ (as said in Subsection 2.2.4.2, the best algorithm against the discrete logarithm problem in class groups has complexity $\mathcal{O}(L[1/2, o(1)])$ compared to an $\mathcal{O}(L[1/3, o(1)])$ for factoring). However, the group law is more complex in class groups. By comparing the Expo. time columns in the tables, we see that exponentiations in class groups are cheaper from the 192 bits level. So even if we use half as many exponentiations, the key generation for our solution only takes less time from that level (while being of the same order of magnitude below this level). For signing, we increase the cost by one exponentiation due to the Elgamal structure of the CL encryption scheme. However, one can note that we can pre process this encryption by computing (g_q^τ, h^τ) in an offline phase and computing $c_1 \leftarrow (g_q^\tau, h^\tau f^{k_2^{-1} m'})$ which results in only one multiplication in the online phase (cf. Subsection 3.4.3, Figure 3.8). As a result we will have only one exponentiation in the online signing for the decryption operation. The same holds for Lindell's protocol with Paillier. Using that both protocols take the same time for signing at the 192 bits level.

Increasing the number of rounds to obtain a 2^{-60} soundness error. This impacts only KeyGen, where the [Lin17] scheme and ours both use 40 iterations of ZK proofs to achieve a 2^{-40} soundness error. Lindell's protocol performs 9 exponentiations per iteration while ours performs 4. All timings will thus be multiplied by 3/2 to achieve a 2^{-60} soundness error, and indeed this is what we observe in practice. Complexity is linear in the number of iterations and the ratio between our timings and those of [Lin17] remains constant.

3.6 Instantiation of our Generic Construction Using DCR

We can instantiate the generic construction of Section 3.3 with the hash proof system built upon Paillier's decision composite residuosity assumption (DCR).

This yields the Elgamal Paillier encryption scheme of [CS03] that closely resembles the HSM − CL encryption scheme. However, the message space is $\mathbf{Z}/n\mathbf{Z}$ as in Lindell's protocol, so in addition to the $\mathsf{R}_{\mathsf{HPS-DL}}$ proof, P_1 has to prove that x_1 is an element of $\mathbf{Z}_q$ with a range proof. For the same reason, one must slightly adapt the double encoding problem, s.t. given input challenge the elliptic curve point $Q := xP$, no adversary can output two invalid encryptions of $x \in \mathbf{Z}$, with $x < q$ (otherwise the assumption does not rule out an adversary which outputs invalid encryptions of $x, x' \in \mathbf{Z}$ where $x \neq x' \bmod N$ but $x = x' \bmod q$). Moreover, this encryption scheme uses two exponentiations instead of one for Paillier. This being said a gain arrises from the fact that following the techniques of [CS03] one can make a sound proof for $\mathsf{R}_{\mathsf{HPS-DL}}$ in a single round by relying on the strong RSA assumption. This means that one should use safe primes that can be very costly to generate at high security level. However, for 112 and 128 bits of security this should give a competitive solution compared to Lindell's with a simulation based security relying on the hardness of classical problems, the DCR and the strong RSA assumptions.

Chapter 4

Bandwidth-efficient Threshold ECDSA from Class Groups

In Chapter 3 we proposed a two-party ECDSA protocol from the HPS framework and an efficient instatiation of it from Class Groups of Imaginary Quadratic Fields. An advantage of using CL is that the ciphertexts and the keys are shorter with respect to the security parameter, compared to Paillier cryptosystem. In a more general context where the signature task is done by a set of more than two parties, the cost of communication plays a crucial role. Motivated by the reduction of communication using CL, our starting idea in [CCL⁺20] was to see how CL encryption could improve the existing solutions for Threshold ECDSA.

Towards Threshold ECDSA from CL In this chapter we explain in details the main points and contributions of our work [CCL$^+$20]. We present new techniques to realize efficient threshold variants of the ECDSA signature scheme. Our protocol has better efficiency for what concerns bandwidth consumption compared to recent works (e.g. [GG18]) and it allows to consider any threshold t such that $t \leq n-1$. As already discussed in previous chapters, using CL allow us to avoid expensive range proofs (Paillier modulus vs ECDSA modulus). Indeed, from this idea, our main contribution in [CCL$^+$20] is a new variant of the Gennaro and Goldfeder protocol [GG18] that exploits the advantage of CL while retaining comparable overall (computational) efficiency.

Multiplicative to Additive sharing In our two party ECDSA we worked with multiplicative shares of ECDSA public key Q and nonce R. When thinking about a threshold variant of this scheme we face with the difficulty to compute both $R = k^{-1}P$ and a multiplication of the two secret values k, x. Gennaro and Goldfeder proposed a solution for this issue. These authors thought about a conversion about multiplicative shares of a value to additive shares of the same one. The idea is the following: consider two secrets $a = a_1 + \cdots + a_n$, $b = b_1 + \cdots + b_n$ additively shared among the parties (i.e. P_i holds a_i and b_i). The product ab can be seen as the sum of the products of each couple of addendum in a and in b, i.e. $ab = \sum_{i,j} a_i b_j$, and parties can compute a and b by computing additive shares of each $a_i b_j$. This can be achieved via a simple two party protocol, originally proposed by Gilboa [Gil99] in the setting of two party RSA key generation, which parties execute in a pairwise way. Slightly more in detail, this latter protocol relies on linearly homomorphic encryption and Gennaro and Goldfeder opted to implement it using Paillier's cryptosystem.

On inconsistencies of moduli We have already discussed differences between Paillier encryption and CL encryption regarding the issues that arise when used with ECDSA signature in Chapter 3. To fix the problem, we rely again on class groups based encryption. Before publishing our result [CCL$^+$20], known techniques to prove the validity of a CL ciphertext were that of [CCL$^+$19], which are rather inefficient as they all use binary challenges. This means that to get soundness error 2^{-t} the proof needs to be repeated t times. The lack of a more efficient technique introduces one of the main contributions of [CCL$^+$20].

A new efficient zero-knowledge Switching from Paillier to CL carry some issues. Using CL we saw it is required to prove the vailidty of a ciphertext. Second main contribution in our [CCL$^+$20], is the developing of new techniques that address exactly this issue. We presented indeed a new efficient zero knowledge argument of knowledge to prove that a ciphertext is well-formed. This result is not specific to ECDSA or our threshold setting considered, but is more general and it could be used in other kind of applications. For example, as we will see in Subsection 4.2.1 and already announced in Subsection 3.4.2, this ZKAoK and other techniques can be used to improve the efficiency of our two party protocol in Chapter 3. We will detail about this in Subsection 4.2.1.

New assumptions for CL Our constructions rely on two recently introduced assumptions on class groups. Informally, given a group $\widehat{G}$ the first one states that it is hard to find low order elements in $\widehat{G}$ (low order assumption) while the latter assumes that it is hard to find roots of random elements in $\widehat{G}$ (strong root assumption). Both these assumptions are believed to hold in class groups of imaginary quadratic fields ([BH01, DF02, BBHM02, Lip12]) and were recently used in, e.g. [BBF18, Pie19, Wes19b].

Resorting to these assumptions allows us to dramatically improve the efficiency of the (zero knowledge) arguments of knowledge needed by our protocols. Informally this can be explained as follows. In the class group setting, the order of the group $\widehat{G}$ is unknown (to all parties, even to those who set up the parameters). This is typically a bad thing when doing arguments of knowledge as, unless one restricts to binary challenges, it is not immediate how to argue the extractability of the witness.

In our proofs, we manage to prove that, no matter how big the challenge space is, either one can extract the witness or one can find a root for some given (random) element of the group, thus violating the strong root assumption. Our argument is actually more convoluted than that as, for technical reasons that won't be discussed here, we still need to make sure that no undetected low order elements are maliciously injected in the protocols (e.g. to extract unauthorized information). This is where the low order assumption comes into play and allows us to avoid hard to handle corner cases in our proofs. Challenges also arise from the fact that in order to reduce to the hardness of finding roots, our reduction should output e^{th} roots where e is not a power of two, since, as observed in concluding remarks of [CCL$^+$19] and also discussed in Subsection 2.2.2, computing square roots or finding elements of order 2 can be done efficiently in class groups knowing the factorization of the discriminant (which is public in our case).

We also provide in Section 4.4 a zero knowledge proof of knowledge (without computational assumptions) for groups of unknown order in order to improve our setup. That proof can also be of independent interest and actually improves the key generation of [CCL$^+$19] for two party ECDSA.

4.1 Preliminaries

Part of the tools required to understand the results on the construction of the Threshold ECDSA from CL are introduced in Chapter 1, other are specific of this chapter. For the former we only recall the references to the dedicated sections in this manuscript, for the latter we will give more details.

Zero-knowledge proofs. See Section 1.5

Computationally convincing proofs of knowledge. See Section 1.5

Threshold secret sharing. See Section 1.6

Feldman verifiable secret sharing. See Section 1.6

Equivocable Commitments Schemes. See Section 1.4

The elliptic curve digital signature algorithm See Subsection 1.3.1

$(t, n)-$**threshold EC-DSA.** See Subsection 1.3.1

4.1.1 Building blocks from Class Groups

An instantiation of the CL framework. We will rely again on the CL framework, but this time we need to consider a slightly different version of the algorithm for the task of generating an instance. We refer to the two-party case (see Section 3.4) for the framework and we explain how to modify it and why it is necessary.

After a run of the Gen algorithm, we obtain a deterministic generator $\hat{g}_q$ of G^q [1]. We will then consider an element g_q built as a random power of $\hat{g}_q$. This slightly changes the construction of [CCL$^+$19] (Section 3.4), in order to make a reduction to a strong root problem for the soundness of the argument of knowledge of Subsection 4.2.1. As usual, one can compute the upper bound $\tilde{s}$ for the order of $\hat{g}_q$, using $\frac{1}{\pi}\log|\Delta_K|\sqrt{|\Delta_K|}$ as an upper bound for $h(\Delta_K)$ or a slightly better bound from the analytic class number formula, as done for the two-party instantiation from Class Groups of the previous chapter. For our application the prime q will have at least 256 bits, in that case q is prime to $h(\Delta_K)$ except with negligible probability. Therefore q will be prime to the order of $\hat{g}_q$ which is a divisor of $h(\Delta_K)$, except for a negligible probability.

[1] $\hat{g}_q$ has the same role of g_q in the definition of Gen in Section 2.2. We use the "hat" notation because $\hat{g}_q$ is an intermediate step to compute the generator we need in the modified version.

Notation. The notation we use is the same as we saw in Subsection 2.2.4 when discussing $\mathsf{GenGroup}$. The only difference with the definition we presented there is that we need to modify the deterministic generation of the parameters to use our efficient ZKAoK. Then, we introduce new elements for the discussion here. For a random power g_q of $\hat{g}_q$ we will denote G^q the subgroup generated by g_q, $g = g_q f$ and G the subgroup generated by g. This time we do not consider a deterministic group G^q, but we use the same notation "G^q" to denote a subgroup G^q generated by a random power of the deterministic generator $\hat{g}_q$ output by Gen, and the same for G. We further denote $\widehat{G}^q$ the subgroup consisting of all q-th powers in $\widehat{G}$, and it's order $\widehat{s}$. It holds that $\widehat{G}$ is the direct product of $\widehat{G}^q$ and F. We denote $\varpi := \widehat{s}_d$ the group exponent of $\widehat{G}^q$, i.e. the least common multiple of the orders of its elements. Clearly, the order of any element in G^q divides ϖ. In the following the distribution $\mathcal{D}$ from which exponents are sampled is chosen to be close to uniform mod $q \cdot \tilde{s}$, where $\tilde{s}$ is an upper bound for $\widehat{s}$. This means that exponents sampled from $\mathcal{D}$ follow a distribution close to uniform mod q, and mod any divisor of $\widehat{s}$. In particular mod ϖ.

HSM **for a random generator** We have already discussed the definition of a HSM in Section 2.2. Anyway, we need to slightly modify the assumption to take in account a random power of the deterministic generator $\hat{g}_q$ and the hardness to recognize elements in $\langle g_q \rangle = \langle \hat{g}_q^t \rangle$ for some integer t, instead of $\langle \hat{g}_q \rangle$. For a random power g_q of $\hat{g}_q$ the HSM assumption states it is hard to distinguish the elements of G^q in G, where G^q and G are computed from the random power g_q.

Definition 4.1.1 (HSM assumption, [CCL+20] version). For $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ an output of Gen, g_q a random power of $\hat{g}_q$ and $g := g_q f$, we denote $\mathcal{D}$ (resp. $\mathcal{D}_q$) a distribution over the integers such that the distribution $\{g^x, x \hookleftarrow \mathcal{D}\}$ (resp. $\{\hat{g}_q^x, x \hookleftarrow \mathcal{D}_q\}$) is at distance less than $2^{-\lambda}$ from the uniform distribution in $\langle g \rangle$ (resp. in $\langle \hat{g}_q \rangle$). Let $\mathcal{A}$ be an adversary for the HSM problem, its advantage is defined as:

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda) := \Big| 2 \cdot \Pr\Big[b = b^\star : (\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q), t \hookleftarrow \mathcal{D}_q, g_q = \hat{g}_q^t, \\ x \hookleftarrow \mathcal{D}, x' \hookleftarrow \mathcal{D}_q, b \xleftarrow{\$} \{0,1\}, Z_0 \leftarrow g^x, Z_1 \leftarrow g_q^{x'}, \\ b^\star \leftarrow \mathcal{A}(q, \tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, Z_b, \mathsf{Solve}(.)) \Big] - 1 \Big|\end{aligned}$$

The HSM problem is said to be hard in G if for all probabilistic polynomial time algorithm $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}}(\lambda)$ is negligible.

A remark on the $\mathsf{HSM} - \mathsf{CL}$ **scheme** We introduced the $\mathsf{HSM} - \mathsf{CL}$ in Subsection 2.2.4 and we refer to it for details. The original scheme is depicted in Figure 2.2. We use the output of $\mathsf{Gen}(1^\lambda, q)$ and as in Definition 4.1.1, we set $g_q = \hat{g}_q^t$ for $t \hookleftarrow \mathcal{D}_q$. The public parameters of the scheme are $\mathsf{pp} := (\tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, q)$. To instantiate $\mathcal{D}_q$, we set $\tilde{A} \geq \tilde{s} \cdot 2^{40}$ such that $\{g_q^r, r \hookleftarrow [\tilde{A}]\}$ is at distance less than 2^{-40} from the uniform distribution in G^q. Conversely to the original scheme of [CLT18a], we do not sample secret keys from $\mathcal{D}_q$. This is due to the way we use the encryption scheme in the signature protocol of Section 4.2. For security to hold, we need secret keys to be sampled from a distribution $\mathcal{D}$ such that $\{(g_q f)^r, r \hookleftarrow \mathcal{D}\}$ is at distance less than $2^{-\lambda}$ of the uniform distribution in $G = F \times G^q$. The plaintext space is $\mathbf{Z}/q\mathbf{Z}$.

We recall a lemma about the distribution followed by the secret keys from [CCL⁺19] (see Lemma 3.4.1 for the proof). In our specific case the lemma can be written as:

Lemma 4.1.1. Let $\mathcal{D}$ be a distribution which is δ-close to $\mathcal{U}(\mathbf{Z}/\widehat{s}q\mathbf{Z})$. For any $x \in G \backslash G^q$, $\pi \leftarrow f^\gamma \in F$ where $\gamma \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and $k \hookleftarrow \mathcal{D}$, the distributions $\mathcal{D}_1 := \{x, (k \bmod \varpi), \pi \cdot x^k\}$ and $\mathcal{D}_2 := \{x, (k \bmod \varpi), x^k\}$ are 2δ-close.

We can also write the smoothness property as below. This is somewhat an abuse of denotation, since smoothness usually refers to the projective hash family underlying an encryption scheme as we saw in Section 3.2.

Definition 4.1.2 (Smoothness). The CL encryption scheme of Figure 2.2 is said to be δ_s-smooth if the distribution $\mathcal{D}$ from which secret keys are sampled is δ-close to $\mathcal{U}(\mathbf{Z}/\widehat{s}q\mathbf{Z})$, where $\delta_s = 2\delta$.

Finally, the notion of invalid ciphertexts is the same in Subsection 3.2.4.

4.1.2 Algorithmic assumptions

We here provide further definitions for the algorithmic assumptions on which the security of our protocol relies. As for the two-party construction (Chapter 3), we need the HSM assumption guaranteeing the $\mathsf{ind\text{-}cpa}$-security of the linearly homomorphic encryption scheme. We also use two additional assumptions: one which states that it is hard to find low order elements in the group $\widehat{G}$, and one which states that it is hard to find roots in $\widehat{G}$ of random elements of the subgroup $\langle \hat{g}_q \rangle$. These assumptions allow us to significantly improve the efficiency of the ZKAoK needed in our protocol. Indeed, as the order of the group we work in is unknown, we cannot (unless challenges are binary as done for our proof for $\mathsf{R}_{\mathsf{CL-DL}}$ in Subsection 3.4.2) immediately extract the witness from two answers corresponding to two different challenges of a given statement. However we show in the ZKAoK of Section 4.2.1 that whatever the challenge space, if one cannot extract the witness, then one can break at least one of these two assumptions. Consequently these assumptions allow us to significantly increase the challenge space of our proofs, and reduce the number of rounds in the protocol to achieve a satisfying soundness, which yields an improvement both in terms of bandwidth and of computational complexity.

Using such assumptions in the context of generalized Schnorr Proofs in groups of unknown order is not novel (*cf.* e.g. [DF02, CKY09]). We adapt these techniques for our specific subgroups of a class group of an imaginary quadratic field, and state them with respect to Gen.

Definition 4.1.3 (Low order assumption). Consider a security parameter $\lambda \in \mathbf{N}$, and $\gamma \in \mathbf{N}$. The γ-*low order problem* (LOP_γ) is $(t(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for Gen if, given the output of Gen, no algorithm $\mathcal{A}$ running in time $\leq t(\lambda)$ can output a γ-low order element in $\widehat{G}$ with probability greater than $\epsilon_{\mathsf{LO}}(\lambda)$. More precisely,

$$\begin{aligned}\epsilon_{\mathsf{LO}}(\lambda) := \Pr[\mu^d = 1, 1 \neq \mu \in \widehat{G}, 1 < d < \gamma \ :& \\ (\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda, q); (\mu, d) &\overset{\$}{\leftarrow} \mathcal{A}(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)].\end{aligned}$$

The γ-low order assumption holds if $t = \mathsf{poly}(\lambda)$, and ϵ_{LO} is negligible in λ.

We now define a strong root assumption for class groups. This can be seen as a generalisation of the strong RSA assumption. We specialise this assumption for class groups where computing square roots is easy knowing the factorisation of the discriminant, and tailor it to our needs by considering challenges in a subgroup.

Definition 4.1.4 (Strong root assumption for Class Groups). Consider a security parameter $\lambda \in \mathbf{N}$, and let $\mathcal{A}$ be a probabilistic algorithm. We run Gen on input $(1^\lambda, q)$ to get $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ and we give this output and a random $Y \in \langle \hat{g}_q \rangle$ as an input to $\mathcal{A}$. We say that $\mathcal{A}$ solves the strong root problem for class groups (SRP) if $\mathcal{A}$ outputs a positive integer $e \neq 2^k$ for all k and $X \in \widehat{G}$, s.t. $Y = X^e$. In particular, the SRP is $(t(\lambda), \epsilon_{\mathsf{SR}}(\lambda))$-secure for Gen if any adversary $\mathcal{A}$, running in time $\leq t(\lambda)$, solves the SRP with probability at most $\epsilon_{\mathsf{SR}}(\lambda)$.

On the hardness of these assumptions in class groups. For our applications, we will use the strong root assumption and the low order assumption in the context of class groups. These assumptions are not completely novel in this setting: Damgård and Fujisaki ([DF02]) explicitly consider variants of these assumptions in this context. Then, Lipmaa used a strong root assumption in class groups to build accumulators without trusted setup in [Lip12]. Recently, an interactive variant of the strong root assumption was used, still in the context of class groups, by Wesolowski to build verifiable delay functions without trusted setup. Furthermore, the low order assumption is also used to implement Pietrzak's verifiable delay functions with class groups (see [BBF18, Pie19]). In the following, we advocate the hardness of these assumptions in the context of class groups.

The root problem and its hardness was considered in [BH01, BBHM02] in the context of class groups to design signature schemes. It is similar to the RSA problem: the adversary is not allowed to choose the exponent e. These works compare the hardness of this problem with the problem of computing the group order and conclude that there is no better known method to compute a solution to the root problem than to compute the order of the group.

The strong root assumption is a generalisation of the strong RSA assumption. Again, the best known algorithm to solve this problem is to compute the order of the group to be able to invert exponents. For strong RSA this means factoring the modulus. For the strong root problem in class groups, this means computing the class number, and best known algorithms for this problem have worst complexity than those to factor integers.

Remark 15. Note that in Definition 4.1.4, we presented the definition of the strong root assumption in the context of class groups specialized for exponents e which are not powers of 2. This is motivated by the fact that one can compute square roots in polynomial time in class groups of quadratic fields, knowing the factorisation of the discriminant (which is public in our setting). We discussed about this task in Subsection 2.2.1 (for details on the algorithm and its analysis, see [Lag80]).

Concerning the low order assumption, we need the $\gamma-$low order problem to be hard in $\widehat{G}$, where γ can be up to 2^{128}. As we discussed in Subsection 2.2.2, in our instantiation the discriminant is chosen such that the $2-$Sylow subgroup is isomorphic to $\mathbf{Z}/2\mathbf{Z}$. It is well known that the element of order 2 can be computed from the (known) factorisation of

Δ_q. However, we work with the odd part, which is the group of squares in this context, so we do not take this element into account.

Let us see that the proportion of such elements of low order is very low in the odd part. From the Cohen Lenstra heuristics (see Subsubection 2.2.2.2), the odd part of a class group $Cl(\Delta)$ of an imaginary quadratic field is cyclic with probability 97.75%. In [HS06], extending the Cohen Lenstra heuristics, it is conjectured that the probability an integer d divides the order $h(\Delta)$ of $Cl(\Delta)$ is less than:

$$\frac{1}{d} + \frac{1}{d \log d}.$$

As a consequence, if the odd part of $Cl(\Delta)$ is cyclic then the expected number of elements of order less than γ is less than

$$\sum_{d \leqslant \gamma} \left(\frac{1}{d} + \frac{1}{d \log d} \right) \varphi(d),$$

which can be bounded above by 2γ. For 128 bits of security, our class number will have around 913 bits, so the proportion of elements of order less than 2^{128} is less than 2^{-784}.

Moreover, if the odd part of the class group is non cyclic, it is very likely that it is of the form $\mathbf{Z}/n_1\mathbf{Z} \oplus \mathbf{Z}/n_2\mathbf{Z}$ where $n_2 | n_1$ and n_2 is very small. Still from the Cohen Lenstra heuristics, the probability that the $p-$rank (the number of cyclic factors in the $p-$Sylow subgroup) of the odd part is equal to r is equal to

$$\frac{\eta_\infty(p)}{p^{r^2} \eta_r(p)^2} \quad \text{where} \quad \eta_r(p) = \prod_{k=1}^{r} (1 - p^{-k}).$$

If we have two cyclic factors, and $p | n_2$, then the $p-$rank is 2. If $p > 2^{20}$ the probability of having a $p-$rank equal to 2 is less than 2^{-80}. Similarly, we cannot have many small cyclic components: the $3-$rank is 6 with probability less than 2^{-83}. Actually, we know only 3 class groups of such 3 ranks [Que87].

There have been intense efforts on the construction of families of discriminants such that there exist elements of a given small order p or with a given $p-$rank. However, these families are very sparse and will be reached by our generation algorithm of the discriminant only with negligible probability. The basic idea of these constructions is to build a discriminant Δ in order to obtain solutions of a Diophantine equation that gives m and the representation of a non principal ideal I of norm m such that I^p is principal, and I has order p in $Cl(\Delta)$ (see eg [Bue76] or [Bel04] for more references).

Solving such a norm equation for a fixed discriminant has been mentioned as a starting point for an attack in [BBF18] combined with the Coppersmith's method, but no concrete advances on the problem have been proposed.

4.2 Threshold EC-DSA protocol

We here provide a construction for (t, n)-threshold ECDSA signing from the CL framework. Security – which does not degrade with the number of signatures queried by the adversary in the tu-cma game (*cf.* Definition 1.3.3) – relies on the assumptions and tools introduced in Section 4.1.

As in many previous works on multiparty ECDSA (*e.g.* [MR01, Lin17, GG18], which we briefly discussed in the introduction chapter), we use a linearly homomorphic encryption scheme. This enables parties to perform operations collaboratively while keeping their inputs secret. Explicitly a party P_i sends a ciphertext encrypting its secret share (under its own public key) to party P_j, P_j then performs homomorphic operations on this ciphertext (using its own secret share), and sends the resulting ciphertext back to P_i – intuitively P_i should learn nothing more about the operations performed by P_j than that revealed by decrypting the ciphertext it receives. To ensure this, P_i must prove to P_j that the ciphertext it first sent is 'well formed'. To this end in Subsection 4.2.1, we provide an efficient zero-knowledge argument of knowledge of the plaintext and of the randomness used to compute a CL ciphertext. This ZKAoK is essential to secure our protocol against malicious adversaries. Next, in Subsection 4.2.2 we explain how parties interactively set up the public parameters of the CL encryption scheme, so that the assumptions underlying the ZKAoK hold. Though – for clarity – we describe this interactive set up as a separate protocol, it can be done in parallel to the $\mathsf{IKeyGen}$ protocol of threshold ECDSA, thereby only increasing by one the number of rounds of the threshold signing protocol. Finally, in Subsection 4.2.3 we present our (t, n)-threshold ECDSA signing protocol, whose security will be demonstrated in Section 4.3.

4.2.1 ZKAoK ensuring a CL ciphertext is well formed

Consider a prover P having computed an encryption of $a \in \mathbf{Z}/q\mathbf{Z}$ with randomness $r \xleftarrow{\$} [\tilde{A}]$, i.e. $\mathbf{c} := (c_1, c_2)$ with $c_1 := g_q^r$, $c_2 := \mathsf{pk}^r f^a$. We present a zero knowledge argument of knowledge for the following relation:

$$\begin{aligned}\mathsf{R}_{\mathsf{Enc}} := \{(\mathsf{pk}, \mathbf{c}); (a, (\rho_0, \rho_1)) \mid \mathsf{pk} \in \widehat{G};\ a \in \mathbf{Z}/q\mathbf{Z}; \rho_0 \in \mathbf{N},\\ \rho_1 \in [-\tilde{A}C(2^{40}+1)\cdot 2^{\rho_0}, \tilde{A}C(2^{40}+1)\cdot 2^{\rho_0}];\ c_1 = g_q^{2^{-\rho_0}\cdot\rho_1} \wedge c_2 = \mathsf{pk}^{2^{-\rho_0}\cdot\rho_1} f^a\}.\end{aligned}$$

Though the relation $\mathsf{R}_{\mathsf{Enc}}$ ensures a ciphertext is well formed, and allows to extract the plaintext, it does not allow to extract an *integer* value of the encryption randomness r. However, for our applications this will be sufficient. For example, this proves that $c_1 \in \langle g_q \rangle$ and one can also recompute in polynomial time c_1 from ρ_0 and ρ_1, by computing ρ_0 successive square roots of g_q and taking the result to the power ρ_1. The interactive protocol is given in Figure 4.1. Note that the honest prover knows and uses the integer value of the randomness r in this protocol. We denote $\mathcal{C}$ the challenge set, and $C := |\mathcal{C}|$. The only constraint on C is that the C-low order assumption holds. We further assume that $C < q$, which is not a restriction for our applications.

Theorem 4.2.1. If the strong root assumption is $(t'(\lambda), \epsilon_{\mathsf{S}R}(\lambda))$-secure for Gen, and the C-low order assumption is $(t'(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for Gen, denoting $\epsilon := \max(\epsilon_{\mathsf{S}R}(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$, then the interactive protocol of Figure 4.1 is a computationally convincing proof of knowledge for $\mathsf{R}_{\mathsf{Enc}}$ with knowledge error $\kappa(\lambda)$, time bound $t(\lambda)$ and failure probability $\nu(\lambda)$, where $\nu(\lambda) = 3\epsilon$, $t(\lambda) < t'(\lambda)/448$ and $\kappa(\lambda) = \max(4/C, 448t(\lambda)/t'(\lambda))$. If r is an integer in $[\tilde{s}\cdot 2^{40}]$ (it is so when the prover is honest), the protocol is honest verifier statistical zero-knowledge.

Proof. We prove the properties of soundness, completeness and (honest verifier) zero-knowledge.

Setup:

1. $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q)$.
2. Let $\tilde{A} := \tilde{s} \cdot 2^{40}$, sample $t \xleftarrow{\$} [\tilde{A}]$ and let $g_q := \hat{g}_q^t$.

Prover $(\mathsf{pk}, \mathbf{c}), a \in \mathbf{Z}/q\mathbf{Z}$; $r \in [\tilde{A}]$ s.t. $c_1 = g_q^r \wedge c_2 = \mathsf{pk}^r f^a$		Verifier $(\mathsf{pk}, \mathbf{c})$
$r_1 \xleftarrow{\$} [2^{40}\tilde{A}C]$ $r_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $t_1 := g_q^{r_1}$ $t_2 := \mathsf{pk}^{r_1} f^{r_2}$	$\xrightarrow{t_1, t_2}$	Check that $\mathsf{pk}, c_1, c_2 \in \widehat{G}$
	$\xleftarrow{k}$	$k \xleftarrow{\$} \mathcal{C}$
$u_1 := r_1 + kr \in \mathbf{Z}$ $u_2 := r_2 + ka \in \mathbf{Z}/q\mathbf{Z}$	$\xrightarrow{u_1, u_2}$	Check $u_1 \in [\tilde{A}C(2^{40}+1)]$ and $u_2 \in \mathbf{Z}/q\mathbf{Z}$ and $g_q^{u_1} = t_1 c_1^k$ and $\mathsf{pk}^{u_1} f^{u_2} = t_2 (c_2)^k$

Figure 4.1: Zero-knowledge argument of knowledge for $\mathsf{R}_{\mathsf{Enc}}$.

Soundness. Let us analyse to what extent the protocol of Figure 4.1 satisfies the notion of soundness defined in Def. 1.5.4, in particular for which knowledge error functions $\kappa()$ is the definition satisfied. Accordingly, let $\kappa()$ be any knowledge error function, such that $\kappa(\lambda) \geq 4/C$ for all λ. We then must define an extractor M. Let a PT prover P^* be given and let view be any view P^* may have after having produced $(\mathsf{pk}, \mathbf{c})$. Now, it can be shown that since there are C different challenges, then if $\epsilon_{\mathsf{view},P} > \kappa(\lambda) \geq 4/C$, standard rewinding techniques allow us to obtain in expected PT a situation where, for given (t_1, t_2), P^* has correctly answered two different values k and k'. We call u_1, u_2 and u_1', u_2' the corresponding answers, so we get:

- $g_q^{u_1} = t_1 \cdot c_1^k$ and $g_q^{u_1'} = t_1 \cdot c_1^{k'}$ s.t. $g_q^{u_1 - u_1'} = c_1^{k-k'}$,
- $\mathsf{pk}^{u_1} f^{u_2} = t_2 \cdot c_2^k$ and $\mathsf{pk}^{u_1'} f^{u_2'} = t_2 \cdot c_2^{k'}$ s.t. $\mathsf{pk}^{u_1 - u_1'} f^{u_2 - u_2'} = c_2^{k-k'}$.

Let Rewind be a (probabilistic) procedure that creates $k, k', u_1, u_2, u_1', u_2'$ in this way. A concrete algorithm for Rewind is given in [DF02, Appendix A]. It runs in expected time $56/\epsilon_{\mathsf{view},P}$, counting the time to do the protocol once with P^* as one step. Denote $d := \gcd(k - k', u_1 - u_1')$ and

$$\nu_1 := g_q^{(u_1 - u_1')/d} c_1^{-(k-k')/d} \quad \text{and} \quad \nu_2 := \mathsf{pk}^{(u_1 - u_1')/d} f^{(u_2 - u_2')/d} c_2^{-(k-k')/d}.$$

Clearly $\nu_1^d = \nu_2^d = 1$. We suppose in the following that $\nu_1 = \nu_2 = 1$. Further suppose that $e := (k - k')/d = 2^{\rho_0}$ for some $\rho_0 \in \mathbf{N}$. We have

$$g_q^{(u_1 - u_1')/d} = c_1^{2^{\rho_0}} \quad \text{and} \quad \mathsf{pk}^{(u_1 - u_1')/d} f^{(u_2 - u_2')/d} = c_2^{2^{\rho_0}}.$$

Now, as the verifier checks that $\mathsf{pk}, c_1, c_2 \in \widehat{G}$, as by definition $g_q, f \in \widehat{G}$, and as the order of $\widehat{G}$ is odd, we get that

$$g_q^{2^{-\rho_0}(u_1 - u_1')/d} = c_1 \quad \text{and} \quad \mathsf{pk}^{2^{-\rho_0}(u_1 - u_1')/d} f^{2^{-\rho_0}(u_2 - u_2')/d} = c_2.$$

The check of V on the size of u_1, u_1' implies that $\rho_1 := (u_1 - u_1')/d$ is in the required interval. One can now easily verify that P^* knows $((\mathsf{pk}, \mathbf{c}); ((u_2 - u_2')/d \mod q, (\rho_0, \rho_1))) \in \mathsf{R}_{\mathsf{Enc}}$, so from values $k, k', u_1, u_2, u_1', u_2'$ one can extract a witness for the statement. Note that in our applications $C < q$, so as $k \neq k' < C$ and as q is prime, d, which is a divisor of $k - k'$, is invertible mod q.

A set of values $k, k', u_1, u_2, u_1', u_2'$ is said to be *bad* if it does not enable extraction, i.e., one of the following holds

1. $\nu_1 \neq 1$ or $\nu_2 \neq 1$
2. $\nu_1 = \nu_2 = 1$, but e *is not* a power of 2.

The extractor M repeats calling Rewind (for the same $(\mathsf{pk}, \mathbf{c})$) until it gets a set of good values. We will analyse knowledge soundness with this M and the polynomial $p(\lambda)$ from the definition set to the constant of 112. We start with a lemma that gives an exact bound on the security.

Lemma 4.2.2. Let $\mathcal{R}$, (P, V), $\kappa()$, M and $p()$ be as defined above. Given any prover P^*, there exists an algorithm $\mathcal{A}(P^*)$ that solves either the strong root problem for class groups with input $(\widehat{G}, \widehat{G}^q, g_q)$, or the low order problem in $\widehat{G}$ with probability $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)/3$, and runs in time $448 \cdot t_{P^*}(k)/\kappa(\lambda)$ where $t_{P^*}(k)$ denotes the running time of P^*.

Proof. $\mathcal{A}$ does the following: receive $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q)$ as an input and accordingly set the public parameters for the CL encryption scheme as: $(\tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, q)$ as described in Section 4.1.1. Send $(\tilde{s}, f, \hat{g}_q, g_q, \widehat{G}, F, q)$ to P^*, and hope to get a set of bad values. However, if Rewind runs more than $448/\kappa(\lambda)$ times with P^*, we abort and stop. If we obtained a set of bad values, we attempt to compute a root of g_q as described below.

Clearly $\mathcal{A}$ runs in time $448 \cdot t_{P^*}(k)/\kappa(\lambda)$. We now look at its' success probability. Note that the distribution of $(\tilde{s}, f, g_q, \widehat{G}, G, F, G^q)$ that P^* receives here is exactly the same as in a real execution of the protocol. Hence the probability of producing a view for which M fails, is exactly $\mathsf{Adv}_{\kappa,M,p}(P^*, \lambda)$. Note also that given any view view where M fails, it must be the case that the values produced by Rewind are bad with probability of at least $1/2$. If this was not the case, then M could expect to find a witness for $(\mathsf{pk}, \mathbf{c})$ after calling Rewind twice, which takes expected time $\frac{112}{\epsilon_{\mathsf{view},P^*}} \leq \frac{p(\lambda)}{\epsilon_{\mathsf{view},P^*} - \kappa(\lambda)}$ which would contradict the fact M fails on view. So let E be the event that M fails on view and Rewind has returned a set of bad values.

Claim 1. Given that E occurs, we can solve either the root problem or the $C-$low order problem in $\widehat{G}$.

Rewind returns a set of bad values $k, k', u_1, u_2, u_1', u_2'$ s.t. $g_q^{u_1 - u_1'} = c_1^{k-k'}$ and $\mathsf{pk}^{u_1 - u_1'} f^{u_2 - u_2'} = c_2^{k-k'}$. Denoting ν_1, ν_2, d and e as above; two cases may occur:

1. $\nu_1 \neq 1$ or $\nu_2 \neq 1$. As we have seen, $\nu_1^d = \nu_2^d = 1$. And since $d|(k - k') < C$, and one can check that ν_1 and ν_2 are elements of $\widehat{G}$ (as c_1, pk, c_2 are checked to be in $\widehat{G}$) this solves the C-low order problem in $\widehat{G}$.

2. $\nu_1 = \nu_2 = 1$ and e *is not* a power of 2. Choose γ and δ s.t. $\gamma(k-k')+\delta(u_1-u_1') = d$. Then $g_q^d = g_q^{\gamma(k-k')+\delta(u_1-u_1')} = (g_q^\gamma c_1^\delta)^{k-k'}$.
   Now let:
$$\mu := (g_q^\gamma c_1^\delta)^{\frac{(k-k')}{d}} g_q^{-1}.$$
   Clearly $\mu^d = 1$, so since $d < C$, if $\mu \neq 1$, we again have a solution to the C-low order problem in $\widehat{G}$. Now suppose that $\mu = 1$. We can rewrite the above as:
$$g_q = (g_q^\gamma c_1^\delta)^{(k-k')/d},$$
   which gives a solution for the SRP with input g_q, which is $e = (k-k')/d$ and $X := g_q^\gamma c_1^\delta$, s.t. $g_q = X^e$, $e > 1$ and e is not a power of 2. The claim above now follows.

Summarizing, we therefore have that for every view view where M fails, the values produced by Rewind are bad with probability $\geq 1/2$, and so running Rewind will fail to solve either the strong root problem or the low order problem with probability at most 1/2. The expected number of executions of P^* needed to run Rewind is at most $56/\epsilon_{\mathsf{view},P^*} \leq 56/\kappa(\lambda)$. Thus Rewind is allowed to run for at least 8 times its expected running time, and so by the Markov rule it will run for longer with probability at most 1/8. As a result, for every view where M fails, $\mathcal{A}$ fails to solve either the strong root problem or the low order problem with probability at most $1/8 + 1 \cdot 1/2 = 9/16$.

Since the probability that view makes M fails is $\mathsf{Adv}_{\kappa,M,p}(P^*,\lambda)$, the success probability of $\mathcal{A}(P^*)$ is at least $\mathsf{Adv}_{\kappa,M,p}(P^*,\lambda)\cdot 7/16 \geq \mathsf{Adv}_{\kappa,M,p}(P^*,\lambda)/3$. This finishes the proof. □

Completeness. If P knows $r \in [\tilde{A}]$ and $a \in \mathbf{Z}/q\mathbf{Z}$ s.t. $(\mathsf{pk},\mathbf{c});(a,r) \in \mathsf{R}_{\mathsf{Enc}}$ and both parties follow the protocol, one has $u_1 \in [\tilde{A}C(2^{40}+1)]$ and $u_2 \in \mathbf{Z}/q\mathbf{Z}$; $\mathsf{pk}^{u_1} f^{u_2} = \mathsf{pk}^{r_1+k\cdot r} f^{r_2+k\cdot a} = \mathsf{pk}^{r_1} f^{r_2} (\mathsf{pk}^r f^a)^k = t_2 c_2^k$; and $g_q^{u_1} = g_q^{r_1+k\cdot r} = t_1 c_1^k$.

Honest verifier zero-knowledge. Given pk, $\mathbf{c} = (c_1, c_2)$ a simulator can sample $k \xleftarrow{\$} [C[$, $u_1 \xleftarrow{\$} [\tilde{A}C(2^{40}+1)]$ and $u_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, compute $t_2 := \mathsf{pk}^{u_1} f^{u_2} c_2^{-k}$ and $t_1 := g_q^{u_1} c_1^{-k}$ such that the transcript $(\mathsf{pk}, \mathbf{c}, t_2, t_1, k, u_1, u_2)$ is indistinguishable from a transcript produced by a real execution of the protocol. □

4.2.2 Interactive set up for the CL encryption scheme

Generating a random generator g_q. In order to use the above ZKAoK it must hold that g_q is a random element of the subgroup $\langle \hat{g}_q \rangle$ where $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q)$. Precisely if a malicious prover P^* could break the soundness of the ZKAoK, an adversary $\mathcal{S}$ trying to break the SRP, given input a random g_q, should be able to feed this input to P^*, and use P^* to solve it's own challenge. Consequently, as the ZKAoK will be used peer-to-peer by all parties in the threshold ECDSA protocol, they will collaboratively generate – in the interactive $\mathsf{IKeyGen}$ – the public parameters $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$, and a common g_q which is random to each party. We call this interactive sub-protocol ISetup, since it allows parties to collaboratively set up the public parameters for the CL encryption scheme. ISetup algorithm is shown in 4.2. All parties then use this g_q to compute their public keys and as a basis for the CL encryption scheme. As pointed in Section 4.1.1, discussed in

Section 2.2, the generation of $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ is deterministic from a pair of primes $\tilde{q}$ and q. To refer to this deterministic setup, we can use the notation $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(\tilde{q}, q)$, which use $(q, \tilde{q})$ to identify the output. We first define the functionality computed by ISetup, running in two steps.

Definition 4.2.1. For a number of parties n, ISetup consists of the following interactive protocols:

Step 1 $\langle k; \dots; k\rangle \rightarrow \langle \tilde{q}\rangle$ or $\langle \bot \rangle$ where $\bot$ is the error output, signifying the parties may abort the protocol, and $\tilde{q}$ is a random k bit prime.

Step 2 $\langle (\tilde{q}, q); \dots; (\tilde{q}, q)\rangle \rightarrow \langle (\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_1); \dots; (\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_n)\rangle$ or $\langle \bot \rangle$ where $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(\tilde{q}, q)$, and values $t_1, \dots, t_n \in [2^{40}\tilde{s}]$ constitute additive shares of t such that $g_q = \hat{g}_q^t$.

For n parties to collaboratively run ISetup, they proceed as depicted in Figure 4.2, performing the following steps:

Step 1 – *Generation of random public prime $\tilde{q}$ of bit-size k.*

1. Each P_i samples a random $r_i \overset{\$}{\leftarrow} \{0,1\}^k$, computes $(\mathsf{c}_i, \mathsf{d}_i) \leftarrow \mathsf{Com}(r_i)$ and broadcasts c_i.
2. After receiving $\{\mathsf{c}_j\}_{j\neq i}$, each P_i broadcasts d_i thus revealing r_i.
3. All players compute the common output $\tilde{q} := \mathsf{next\text{-}prime}(\bigoplus_{j=1}^{n} r_j)$.

Step 2 – *Generation of g_q.*

1. From $\tilde{q}$, (and the order of the elliptic curve q) all parties can use the deterministic set up of [CL15, CCL+19] (see Section 2.2), which sets a generator $\hat{g}_q$.
2. Next each player P_i performs the following steps:
   (a) Sample a random $t_i \overset{\$}{\leftarrow} [2^{40}\tilde{s}]$; compute $g_i := \hat{g}_q^{t_i}$; $(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i) \leftarrow \mathsf{Com}(g_i)$, and broadcast $\tilde{\mathsf{c}}_i$.
   (b) Receive $\{\tilde{\mathsf{c}}_j\}_{j\neq i}$. Broadcast $\tilde{\mathsf{d}}_i$ thus revealing g_i.
   (c) Perform a ZKPoK of t_i such that $g_i = \hat{g}_q^{t_i}$.If a proof fails, abort.
3. Each party computes $g_q := \prod_{j=1}^{n} g_j = \hat{g}_q^{\sum t_j}$, and has output $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q, t_i)$.

Theorem 4.2.3 states the security of the interactive protocol ISetup of Figure 4.2.

Theorem 4.2.3. If the commitment scheme is non-malleable and equivocal; and the proofs $\pi_i^{\langle \hat{g}_q\rangle-\mathsf{DL}}$ are zero knowledge proofs of knowledge of discrete logarithm in $\langle \hat{g}_q\rangle$, then the protocol of Figure 4.2 securely computes ISetup with abort, in the presence of a malicious adversary corrupting any $t < n$ parties, with point-to-point channels.

$$
\begin{array}{ccc}
P_i & \mathsf{ISetup}(k) & \text{All players } \{P_j\}_{j\neq i} \\
\hline
r_i \xleftarrow{\$} \{0,1\}^k & & \\
[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(r_i) & \xRightarrow{\mathsf{c}_i} & \\
 & \xRightarrow{\mathsf{d}_i} & r_i \leftarrow \mathsf{Open}(\mathsf{c}_i, \mathsf{d}_i) \\
\tilde{q} := \mathsf{next\text{-}prime}(\bigoplus_{j=1}^n r_j) & & \\
\hline
\text{Compute } \hat{g}_q \text{ from } q, \tilde{q} & & \\
t_i \xleftarrow{\$} [2^{40}\tilde{s}] \text{ and } g_i \leftarrow \hat{g}_q^{t_i} & & \\
(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i) \leftarrow \mathsf{Com}(g_i) & \xRightarrow{\tilde{\mathsf{c}}_i} & \\
 & \xRightarrow{\tilde{\mathsf{d}}_i} & g_i \leftarrow \mathsf{Open}(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i) \\
\pi_i^{\langle \hat{g}_q \rangle - \mathsf{DL}} := \mathsf{ZKPoK}_{g_i}\{(t_i) : g_i = \hat{g}_q^{t_i}\} & \xleftrightarrow{\pi_i} & \text{if a proof fails } \mathsf{abort} \\
g_q \leftarrow \prod_{j=1}^n \hat{g}_q^{t_j} = \prod_{j=1}^n g_j & &
\end{array}
$$

Figure 4.2: Threshold CL setup used in IKeyGen

Proof. We here demonstrate that for each execution of ISetup (*cf.* Figure 4.2), which interactively sets the public parameters of the CL framework, our reduction for the strong root problem can program the outputs $\tilde{q}$ and g_q if the reduction controls at least one uncorrupted player.

Indeed consider an adversary $\mathcal{S}$ for the SRP for generator Gen. $\mathcal{S}$ gets as input a description of $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F)$ output by $\mathsf{Gen}(1^\lambda, q)$, which includes $\tilde{q}$ and the order of the elliptic curve q, and a random element $Y \in \langle \hat{g}_q \rangle$. $\mathcal{S}$ must simulate Step 1 so that all players output the same $\tilde{q}$ as $\mathcal{S}$ received in the description of G. Next $\mathcal{S}$ must simulate Step 2 so that each player P_i outputs $\tilde{s}, f, \hat{g}_q, \widehat{G}, F, g_q = Y$ – of which $\mathcal{S}$ must find a root – and some $t_i \in [\tilde{s} \cdot 2^{40}]$. We describe $\mathcal{S}$ simulating P_1 against all the other (potentially corrupted) parties, since all parties play symmetric roles, this is without loss of generality.

Simulating step 1 — Generation of $\tilde{q}$.

1. $\mathcal{S}$ samples $r_1 \xleftarrow{\$} \{0,1\}^k$, computes $(\mathsf{c}_1, \mathsf{d}_1) := \mathsf{Com}(r_1)$ and broadcasts c_1.

2. $\mathcal{S}$ broadcasts d_1, revealing r_1, and receives $\{r_j\}_{j>1}$.

3. $\mathcal{S}$ samples r_1' uniformly at random in $\{0,1\}^k$, subject to the condition $\tilde{q} = \mathsf{next\ prime}(r_1' \oplus \bigoplus_{j=2}^n r_j)$. Then $\mathcal{S}$ computes an equivocated decommitment d_1' which opens to r_1', rewinds the adversary to item 2. and broadcasts d_1' instead of d_1.

4. All players compute the common output $\tilde{q} := \mathsf{next\ prime}(r_1' \oplus \bigoplus_{j=2}^n r_j)$.

Simulating step 2 — Generation of $Y = g_q$.

1. From $\tilde{q}$ and q all parties use the deterministic set up of [CCL+19] to set generator $\hat{g}_q$.

2. $\mathcal{S}$ (simulating P_1) does the following:

(a) Sample $t_1 \xleftarrow{\$} [2^{40}\tilde{s}]$; compute $g_1 := \hat{g}_q^{t_1}$; $(\tilde{\mathsf{c}}_1, \tilde{\mathsf{d}}_1) = \mathsf{Com}(g_1)$, and broadcast $\tilde{\mathsf{c}}_1$.

(b) Receive $\{\tilde{\mathsf{c}}_j\}_{j\neq 1}$. Broadcast $\tilde{\mathsf{d}}_1$ thus revealing g_1.

(c) Perform a ZKPoK for $\pi_1^{\langle \hat{g}_q \rangle - \mathsf{DL}} := \mathsf{ZKPoK}_{g_1}\{(t_1) : g_1 = \hat{g}_q^{t_1}\}$.

(d) Receive $\{\tilde{\mathsf{c}}_j\}_{j\neq 1}$, recover $g_j \leftarrow \mathsf{Open}(\tilde{\mathsf{c}}_j, \tilde{\mathsf{d}}_j)$ for each j.

(e) Let $h := \prod_{j=2}^n g_j$. Compute $g_1' := Y \cdot h^{-1}$ and an equivocated decommitment d_1' which opens to g_1', rewind the adversary to 2. (b) and broadcast d_1' instead of $\tilde{\mathsf{d}}_1$. In 2. (b) simulates the ZKPoK.

3. If all the proofs are correct, the protocol goes along with $g_q := g_1' h = Y$.

Lemma 4.2.4. If the commitment scheme is non-malleable and equivocal; and the proofs $\pi_i^{\langle \hat{g}_q \rangle - \mathsf{DL}}$ are zero knowledge proofs of knowledge then, a simulated execution of steps 1 and 2 above is – from the view of (potentially corrupted) parties $P_2, \dots, P_n$ – indistinguishable from a real execution. Moreover when the simulation – on input (G, g_q), where G is computed deterministically from a prime $\tilde{q}$ – does not abort, all parties output $\tilde{q}$ in step 1, and g_q in step 2.

Proof. Step 1: The only difference between real and simulated protocols is the way r_1 is computed. In the simulation $\mathcal{S}$ does not know r_1, but it chooses a r_1' such that $\tilde{q} = \mathsf{next\ prime}(r_1' \oplus \bigoplus_{j=2}^n r_j)$. Let $R = \bigoplus_{j=2}^n r_j$ and $H_q = \{x \in \{0,1\}^k : \mathsf{prev\text{-}prime}(\tilde{q}) \leq x \oplus R \leq \tilde{q} - 1\}$ be the set of all the elements x such that $\tilde{q} = \mathsf{next\ prime}(x \oplus R)$. Since r_1 belongs to the set H_q, and it has been chosen uniformly at random, as long as r_1' is chosen uniformly at random in the same set, the real and simulated executions are indistinguishable.
Step 2: The only difference is in point 2.(e), where the simulator computes g_1' instead of using g_1. Since g_1 and $Y \cdot h^{-1}$ follow the same distribution, real and simulated executions are indistinguishable.

Moreover, we observe that the simulation can fail in three points: in step 1 if someone refuses to decommit after rewinding and in step 2, if some $\pi_i^{\langle \hat{g}_q \rangle - \mathsf{DL}}$ fails or if someone refuses to decommit after rewinding. Since the commitment scheme is non-malleable and equivocal, in Step 1 the simulator can rewind and equivocate the commitment to r_1, and if there are not aborts, all parties decommit to their correct values. As a consequence, all parties output $\tilde{q}$ at the end of Step 1. In step 2, all parties compute the correct $\hat{g}_q$ using $\tilde{q}$ from the deterministic setup of CL, if not there is an abort caused by the soundness of the proof $\pi_i^{\langle \hat{g}_q \rangle - \mathsf{DL}}$ corresponding to the corrupted P_i. Finally, if no abort has occurred, in step 2, point e), the simulator can equivocate the decommitment to g_1 and all parties decommit to the correct values thanks to the non-malleability of the scheme. If no party refuses to decommit after rewinding, the protocol ends with $g_q = Y$ (and $\tilde{q}$ from step 1). □

Remark 16. The randomness of $\tilde{q}$ is not crucial to the security of the ECDSA protocol: conversely to RSA prime factors, here $\tilde{q}$ is public. However traditionally, class group based crypto uses random discriminants; we provide a distributed version of the setup of [CL15] – recalled in Section 2.2 – in which the prime $\tilde{q}$ is random. In our ISetup algorithm, the output of $\mathsf{next\text{-}prime}$ is biased. To patch this, for the same complexity, parties could jointly generate a seed for a prime pseudo-random generator to generate $\tilde{q}$; such a source of randomness would be sufficient in this context.

P_i	$\mathsf{IKeyGen}(\mathbb{G}, P, q)$	All players $\{P_j\}_{j \neq i}$
$u_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(u_i P)$		
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}(1^\lambda)$	$\xRightarrow{\mathsf{pk}_i \text{ and } \mathsf{kgc}_i}$	
	$\xRightarrow{\mathsf{kgd}_i}$	
Perform (t, n)-VSS share of u_i:		$Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$
$p_i(X) = u_i + \sum_{k=1}^{t} a_{i,k} X^k \mod q$		s.t. $Q_i = u_i P$
Denote $\{\sigma_{i,j} := p_i(j)\}_{j \in [n]}$		$Q = \sum_{i=1}^{n} Q_i$
and $\{V_{i,k} := a_{i,k} P\}_{k \in [t]}$	$\xrightarrow{\text{Send } \sigma_{i,j} \text{ to } P_j}$	
	$\xRightarrow{\{V_{i,k}\}_{k \in [t]}}$	
$\{\sigma_{k,i}\}_k$ are additive shares of $x_i := \sum_{k \in [n]} p_k(i)$		
where $\{x_i\}_{i \in [n]}$ are (t, n) Shamir shares of x.		
$\pi_i^{\mathsf{kg}} := \mathsf{ZKPoK}_{X_i}\{(x_i) : X_i = x_i P\}$	$\overset{\pi_i^{\mathsf{kg}}}{\longleftrightarrow}$	

Figure 4.3: Threshold Key Generation

4.2.3 Resulting threshold ECDSA protocol

We now describe the overall protocol. Participants run on input $(\mathbb{G}, q, P)$ used by the ECDSA signature scheme. In Figure 4.3, and in phases 1, 3, 4, 5 of Figure 4.4, all players perform the same operations (on their respective inputs) w.r.t. all other parties, so we only describe the actions of some party P_i. In particular if P_i broadcasts some value v_i, implicitly P_i receives v_j broadcast by P_j for all $j \in [n]$, $j \neq i$. Broadcasts from P_i to all other players are denoted by double arrows, whereas peer-to-peer communications are denoted by single arrows.

On the other hand, Phase 2 of Figure 4.4 is performed by all pairs of players $\{(P_i, P_j)\}_{i \neq j}$. Each player will thus perform $(n-1)$ times the set of instructions on the left (performed by P_i on the figure) and $(n-1)$ times those on the right hand side of the figure (performed by P_j).

4.2.3.1 Key generation.

We assume that prior to the interactive key generation protocol $\mathsf{IKeyGen}$, all parties run the ISetup protocol of Subsection 4.2.2 s.t. they output a common random generator g_q. Each party uses this g_q to generate its' CL encryption key pair, and to verify the ZKAoK in the ISign protocol. Although $\mathsf{IKeyGen}$ and ISetup are here described as two separate protocols, they can be ran in parallel. Consequently, in practice the number of rounds in $\mathsf{IKeyGen}$ increases by 1 broadcast per party if the ZK proofs are made non interactive, and by 2 broadcasts if it is performed interactively between players.
The $\mathsf{IKeyGen}$ protocol (also depicted in Fig 4.3) proceeds as follows:

1. Each P_i samples a random $u_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$; computes $[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(u_i P)$ and generates a pair of keys $(\mathsf{sk}_i, \mathsf{pk}_i)$ for the CL encryption scheme. Each P_i broadcasts $(\mathsf{pk}_i, \mathsf{kgc}_i)$.

2. Each P_i broadcasts kgd_i. Let $Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$. Party P_i performs a (t, n) Feldman-VSS of u_i, with Q_i as the free term in the exponent. The ECDSA public key is set to $Q = \sum_{i=1}^{n} Q_i$. Each player adds the private shares received during the n Feldman VSS protocols. The resulting values x_i are a (t, n) Shamir's secret sharing of the secret signing key x. Observe that all parties know $\{X_i := x_i \cdot P\}_{i\in[n]}$. Indeed,

$$X_i = x_i \cdot P = \sum_{k\in[n]} p_k(i) \cdot P = \left(\sum_{k\in[n]} \left(u_k + \sum_{j=1}^{t} a_{k,j} \cdot i^j\right)\right) \cdot P$$
$$= \sum_{k\in[n]} \left(u_k \cdot P + \sum_{j=1}^{t} a_{k,j} \cdot i^j \cdot P\right) = \sum_{k\in[n]} \left(Q_k + \sum_{j=1}^{t} i^j \cdot V_{k,j}\right) = Q + \sum_{j=1}^{t} i^j \cdot V_{k,j}$$

where each addendum is public.

3. Each P_i proves in ZK that he knows x_i using Schnorr's protocol [Sch91].

4.2.3.2 Signing.

The signature generation protocol runs on input m and the output of the $\mathsf{IKeyGen}$ protocol of Fig 4.3. We denote $S \subseteq [n]$ the subset of players which collaborate to sign m. Assuming $|S| = t$ one can convert the (t, n) shares $\{x_i\}_{i\in[n]}$ of x into (t, t) shares $\{w_i\}_{i\in S}$ of x using the appropriate Lagrangian coefficients. Since the $X_i = x_i \cdot P$ and Lagrangian coefficients are public values, all parties can compute $\{W_i := w_i \cdot P\}_{i\in S}$. We here describe the steps of the algorithm. A global view of the interactions is also provided in Figure 4.4.

Phase 1: Each party P_i samples $k_i, \gamma_i \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and $r_i \overset{\$}{\leftarrow} [\tilde{A}]$ uniformly at random. It computes $c_{k_i} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, k_i; r_i)$, a ZKAoK π_i that the ciphertext is well formed, and $[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(\gamma_i P)$. Each P_i broadcasts $(\mathsf{c}_i, c_{k_i}, \pi_i)$.

Phase 2: *Intuition: denoting $k := \sum_{i\in S} k_i$ and $\gamma := \sum_{i\in S} \gamma_i$ it holds that $k\gamma = \sum_{i,j\in S} k_j\gamma_i$ and $kx = \sum_{i,j\in S} k_j w_i$. The aim of Phase 2 is to convert the multiplicative shares k_j and γ_i of $(k_j\gamma_i)$ (resp. k_j and w_i of $(k_j w_i)$) into additive shares $\alpha_{j,i} + \beta_{j,i} = k_j\gamma_i$ (resp. $\mu_{j,i} + \nu_{j,i} = k_j w_i$).* Phase 2 is performed peer-to-peer between each pair $\{(P_i, P_j)\}_{i\neq j}$, s.t. at the end of the phase, P_i knows $\{\alpha_{i,j}, \beta_{j,i}, \mu_{i,j}, \nu_{j,i}\}_{j\in S, j\neq i}$. Each peer-to-peer interaction proceeds as follows:

(a) P_i samples $\beta_{j,i}, \nu_{j,i} \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$, and computes $B_{j,i} := \nu_{j,i} \cdot P$. It uses the homomorphic properties of the encryption scheme and the ciphertext c_{k_j} broadcast by P_j in Phase 1 to compute $c_{k_j\gamma_i}$ and $c_{k_j w_i}$: encryptions under pk_j of $k_j\gamma_i - \beta_{j,i}$ and $k_j w_i - \nu_{j,i}$ respectively.

(b) P_i sends $(c_{k_j\gamma_i}, c_{k_j w_i}, B_{j,i})$ to P_j, who decrypts both ciphertexts to recover respectively $\alpha_{j,i}$ and $\mu_{j,i}$.

(c) Since W_i is public, P_j verifies that P_i used the same share w_i as that used to compute the public key Q by checking $\mu_{j,i} \cdot P + B_{j,i}$. If the check fails, P_j aborts.

P_i computes $\delta_i := k_i\gamma_i + \sum_{j\neq i}(\alpha_{i,j} + \beta_{j,i})$ and $\sigma_i := k_i w_i + \sum_{j\neq i}(\mu_{i,j} + \nu_{j,i})$.

ISign

P_i	**Phase 1**	All players $\{P_j\}_{j\neq i}$
$k_i, \gamma_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$r_i \xleftarrow{\$} [\tilde{A}]$		
$c_{k_i} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, k_i; r_i)$		
$[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(\gamma_i P)$	$\xRightarrow{\mathsf{c}_i, c_{k_i}}$	if a proof fails, abort
$\pi_i := \mathsf{ZKAoK}_{\mathsf{pk}_i, c_{k_i}}\{(k_i, r_i) : ((\mathsf{pk}_i, c_{k_i}); (k_i, r_i)) \in \mathsf{R}_{\mathsf{Enc}}\}$	$\xleftrightarrow{\pi_i}$	
P_i	**Phase 2**	P_j
$\beta_{j,i}, \nu_{j,i} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$B_{j,i} := \nu_{j,i} \cdot P$		
$c_{\beta_{j,i}} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, -\beta_{j,i})$		
$c_{\nu_{j,i}} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, -\nu_{j,i})$		
$c_{k_j\gamma_i} \leftarrow \mathsf{EvalAdd}(\mathsf{EvalScal}(c_{k_j}, \gamma_i), c_{\beta_{j,i}})$		
$c_{k_j w_i} \leftarrow \mathsf{EvalAdd}(\mathsf{EvalScal}(c_{k_j}, w_i), c_{\nu_{j,i}})$	$\xrightarrow{c_{k_j\gamma_i}, c_{k_j w_i}, B_{j,i}}$	
		$\alpha_{j,i} \leftarrow \mathsf{Dec}(\mathsf{sk}_j, c_{k_j\gamma_i})$
		$\mu_{j,i} \leftarrow \mathsf{Dec}(\mathsf{sk}_j, c_{k_j w_i})$
		If $\mu_{j,i} \cdot P + B_{j,i} \neq k_j \cdot W_i$ then abort
$\delta_i := k_i\gamma_i + \sum_{j\neq i}(\alpha_{i,j} + \beta_{j,i})$		
$\sigma_i := k_i w_i + \sum_{j\neq i}(\mu_{i,j} + \nu_{j,i})$		
P_i	**Phase 3**	All players $\{P_j\}_{j\neq i}$
	$\xRightarrow{\delta_i}$	$\delta = \sum_{i\in S} \delta_i = k\gamma$
P_i	**Phase 4**	All players $\{P_j\}_{j\neq i}$
	$\xRightarrow{\mathsf{d}_i}$	$\Gamma_i := \mathsf{Open}(\mathsf{c}_i, \mathsf{d}_i) = \gamma_i P$
$\pi_i^\gamma := \mathsf{ZKPoK}_{\Gamma_i}\{(\gamma_i) : \Gamma_i = \gamma_i P\}$	$\xleftrightarrow{\pi_i^\gamma}$	if a proof fails, abort
		$R := \delta^{-1}(\sum_{i\in S} \Gamma_i)$ and $r := H'(R)$
P_i	**Phase 5**	All players $\{P_j\}_{j\neq i}$
$s_i := mk_i + r\sigma_i$		
$\ell_i, \rho_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$V_i := s_i R + \ell_i P$ and $A_i := \rho_i P$		
$[\widehat{\mathsf{c}}_i, \widehat{\mathsf{d}}_i] \leftarrow \mathsf{Com}(V_i, A_i)$	$\xRightarrow{\widehat{\mathsf{c}}_i}$	
$\widehat{\pi}_i := \mathsf{ZKPoK}_{(V_i, A_i)}\{(s_i, \ell_i, \rho_i) : V_i = s_i R + \ell_i P \wedge A_i = \rho_i P\}$	$\xRightarrow{\widehat{\mathsf{d}}_i}$	
	$\xleftrightarrow{\widehat{\pi}_i}$	if a proof fails, abort
		$V := -mP - rQ + \sum_{i\in S} V_i$
$U_i := \rho_i V$ and $T_i := \ell_i A$		and $A := \sum_{i\in S} A_i$
$[\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i] \leftarrow \mathsf{Com}(U_i, T_i)$	$\xRightarrow{\tilde{\mathsf{c}}_i}$	
	$\xRightarrow{\tilde{\mathsf{d}}_i}$	if $\sum_{i\in S} T_i \neq \sum_{i\in S} U_i$ then abort.
	$\xRightarrow{s_i}$	$s := \sum_{i\in S} s_i$,
		if (r, s) is not a valid signature, abort,
		else return (r, s).

Figure 4.4: Threshold signature protocol

Phase 3: Each P_i broadcasts δ_i. All players compute $\delta := \sum_{i \in S} \delta_i$.

Phase 4: (a) Each P_i broadcasts d_i which decommits to Γ_i.

(b) Each P_i proves knowledge of γ_i s.t. $\Gamma_i = \gamma_i P$. All players compute $R := \delta^{-1}(\sum_{i \in S} \Gamma_i) = k^{-1} \cdot P$ and $r := H'(R) \in \mathbf{Z}/q\mathbf{Z}$.

Phase 5: (a) Each P_i computes $s_i = k_i m + \sigma_i r$, samples $\ell_i, \rho_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ uniformly at random, computes $V_i := s_i R + \ell_i P$; $A_i := \rho_i P$; and $[\widehat{\mathsf{c}}_i, \widehat{\mathsf{d}}_i] \leftarrow \mathsf{Com}(V_i, A_i)$. Each P_i broadcasts $\widehat{\mathsf{c}}_i$.

(b) Each party P_i decommits by broadcasting $\widehat{\mathsf{d}}_i$ along with a NIZKPoK of (s_i, ℓ_i, ρ_i) s.t. $(V_i = s_i R + \ell_i P) \wedge (A_i = \rho_i P)$. It checks all the proofs it gets from other parties. If a proof fails P_i aborts.

(c) All parties compute $V := -mP - rQ + \sum_{i \in S} V_i$, $A := \sum_{i \in S} A_i$. Each party P_i computes $U_i := \rho_i V$, $T_i := \ell_i A$ and the commitment $[\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i] \leftarrow \mathsf{Com}(U_i, T_i)$. It then broadcasts $\tilde{\mathsf{c}}_i$.

(d) Each P_i decommits to (U_i, T_i) by broadcasting $\tilde{\mathsf{d}}_i$.

(e) All players check $\sum_{i \in S} T_i = \sum_{i \in S} A_i$. If the check fails they abort.

(f) Each P_i broadcasts s_i s.t. all players can compute $s := \sum_{i \in S} s_i$. They check that (r, s) is a valid ECDSA signature, if so, they output (r, s), otherwise they abort the protocol.

4.3 Security

The security proof is a reduction to the unforgeability of standard ECDSA. We demonstrate that if there exists a PPT algorithm $\mathcal{A}$ which breaks the threshold ECDSA protocol of Figure 4.3 and 4.4, then we can construct a forger $\mathcal{S}$ which uses $\mathcal{A}$ to break the unforgeability of standard ECDSA. To this end $\mathcal{S}$ must simulate the environment of $\mathcal{A}$, so that $\mathcal{A}$'s view of its interactions with $\mathcal{S}$ are indistinguishable from $\mathcal{A}$'s view in a real execution of the protocol. Precisely, we show that if an adversary $\mathcal{A}$ corrupts $\{P_j\}_{j>1}$, one can construct a forger $\mathcal{S}$ simulating P_1 s.t. the output distribution of $\mathcal{S}$ is indistinguishable from $\mathcal{A}$'s view in an interaction with an honest party P_1 (all players play symmetric roles in the protocol so it is sufficient to provide a simulation for P_1). $\mathcal{S}$ gets as input an ECDSA public key Q, and has access to a signing oracle for messages of its choice. After this query phase, $\mathcal{S}$ must output a forgery, i.e. a signature σ for a message m of its choice, which it did not receive from the oracle.

Simulating the key generation protocol

On input a public key $Q := x \cdot P$, the forger $\mathcal{S}$ must set up in its simulation with $\mathcal{A}$ this same public key Q (w/o knowing x). This will allow $\mathcal{S}$ to subsequently simulate interactively signing messages with $\mathcal{A}$, using the output of its' (standard) ECDSA signing oracle.

The main differences with the proof of [GG18] arise from the fact $\mathcal{S}$ knows it's own decryption key sk_1, but does not extract that of other players. As in [CCL+19], the encryption scheme we use results from hash proof systems, whose security is statistical, thus the fact $\mathcal{S}$ uses its' secret key does not compromise security, and we can still reduce the security of the protocol to the smoothness of the CL scheme. However as we do not prove knowledge of secret keys associated to public keys in the key generation protocol, $\mathcal{S}$ can not extract the decryption keys of corrupted players. The simulation is described below.

Simulating P_1 in $\mathsf{IKeyGen}$

1. $\mathcal{S}$ receives a public key Q from it's ECDSA challenger.

2. Repeat the following steps (by rewinding $\mathcal{A}$) until $\mathcal{A}$ sends correct decommitments for $P_2, \dots, P_n$ on both iterations.

3. $\mathcal{S}$ selects a random value $u_1 \in \mathbf{Z}/q\mathbf{Z}$, computes $[\mathsf{kgc}_1, \mathsf{kgd}_1] \leftarrow \mathsf{Com}(u_1 P)$ and broadcasts kgc_1. $\mathcal{S}$ receives $\{\mathsf{kgc}_j\}_{j\in[n], j\neq 1}$.

4. $\mathcal{S}$ broadcasts kgd_1 and receives $\{\mathsf{kgd}_j\}_{j\in[n], j\neq 1}$. For $i \in [n]$, let $Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$ be the revealed commitment value of each party. Each player performs a (t, n) Feldman-VSS of the value Q_i, with Q_i as the free term in the exponent.

5. $\mathcal{S}$ samples a CL encryption key pair $(\mathsf{pk}_1, \mathsf{sk}_1) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$.

6. $\mathcal{S}$ broadcasts pk_1 and receives the public keys $\{\mathsf{pk}_j\}_{j\in[n], j\neq 1}$.

7. $\mathcal{S}$ rewinds $\mathcal{A}$ to the decommitment step and

   - equivocates P_1's commitment to $\widehat{\mathsf{kgd}}$ so that the committed value revealed is now $\widehat{Q}_1 := Q - \sum_{j=2}^n Q_j$.
   - simulates the Feldman-VSS with free term $\widehat{Q}_1$.

8. $\mathcal{A}$ will broadcast the decommitments $\{\widehat{\mathsf{kgd}}_j\}_{j\in[n], j\neq 1}$. Let $\{\widehat{Q}_j\}_{j=2\dots n}$ be the committed value revealed by $\mathcal{A}$ at this point (this could be $\perp$ if $\mathcal{A}$ refuses to decommit).

9. All players compute the public signing key $\widehat{Q} := \sum_{i=1}^n \widehat{Q}_i$. If any $Q_i = \perp$ in the previous step, then $\widehat{Q} := \perp$.

10. Each player P_i adds the private shares it received during the n Feldman VSS protocols to obtain x_i (such that the x_i are a (t, n) Shamir's secret sharing of the secret key $x = \sum_i u_i$). Note that due to the free term in the exponent, the values $X_i := x_i \cdot P$ are public.

11. $\mathcal{S}$ simulates the ZKPoK that it knows x_1 corresponding to X_1, and for $j \in [n]$, $j \neq 1$, $\mathcal{S}$ receives from $\mathcal{A}$ a Schnorr ZKPoK of x_j such that $X_j := x_j \cdot P$. $\mathcal{S}$ can extract the values $\{x_j\}_{j\in[n], j\neq 1}$ from these ZKPoK.

Simulating the signature generation

On input m, $\mathcal{S}$ must simulate the interactive signature protocol from $\mathcal{A}$'s view.

We define $\tilde{k_i} := \mathsf{Dec}(\mathsf{sk}_i, c_{k_i})$, which $\mathcal{S}$ can extract from the proofs Π, and $\tilde{k} := \sum_{i\in S} \tilde{k_i}$. Let $k \in \mathbf{Z}/q\mathbf{Z}$ denote the value s.t. $R := k^{-1} \cdot P$ in Phase 4 of the signing protocol. Notice that if any of the players mess up the computation of R by revealing wrong shares δ_i, we may have $k \neq \tilde{k} \mod q$. As in [GG18], we distinguish two types of executions of the protocol: an execution where $\tilde{k} = k \mod q$ is said to be *semi-correct*, whereas an execution where $\tilde{k} \neq k \mod q$ is *non semi-correct*. Both executions will be simulated differently. At the end of Phase 4, when both simulations diverge, $\mathcal{S}$ knows k and $\tilde{k}$, so it can detect if it is in a semi-correct execution or not and choose how to simulate P_1.

We point out that $\mathcal{S}$ does not know the secret share w_1 of x associated with P_1, but it knows the shares $\{w_j\}_{j\in S, j\neq 1}$ of all the other players. Indeed $\mathcal{S}$ can compute these from the values $\{x_j\}_{j\in[n], j\neq 1}$ extracted during key generation. It also knows $W_1 = w_1 \cdot P$ from the key generation protocol. Moreover $\mathcal{S}$ knows the encryption keys $\{\mathsf{pk}_j\}_{j\in S}$ of all players, and it's own decryption key sk_1.

In the following simulation $\mathcal{S}$ aborts whenever $\mathcal{A}$ refuses to decommit any of the committed values, fails a ZK proof, or if the signature (r, s) does not verify.

Simulating P_1 in ISign

Phase 1: As in a real execution, $\mathcal{S}$ samples $k_1, \gamma_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $r_1 \xleftarrow{\$} [\tilde{A}]$ uniformly at random. It computes $c_{k_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_1, k_1; r_1)$, the associated ZKAoK π_1, and $[\mathsf{c}_1, \mathsf{d}_1] \leftarrow \mathsf{Com}(\gamma_1 P)$. It broadcasts $(\mathsf{c}_1, c_{k_1}, \pi_1)$ before receiving $\{\mathsf{c}_j, c_{k_j}, \pi_j\}_{j\in S, j\neq 1}$ from $\mathcal{A}$. $\mathcal{S}$ checks the proofs are valid and extracts the encrypted values $\{k_j\}_{j\in S, j\neq 1}$ from which it computes $\tilde{k} := \sum_{i\in S} k_i$.

Phase 2:
(a) For $j \in S, j \neq 1$, $\mathcal{S}$ computes $\beta_{j,1}$, $c_{k_j\gamma_1}$ as in a real execution of the protocol, however since it only knows $W_1 = w_1 P$ (but not w_1), it samples a random $\mu_{j,1} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and sets $c_{k_j w_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mu_{j,1})$, and $B_{j,1} := k_j \cdot W_1 - \mu_{j,1} \cdot P$. $\mathcal{S}$ then sends $(c_{k_j\gamma_1}, c_{k_j w_1}, B_{j,1})$ to P_j.

(b) When it receives $(c_{k_1\gamma_i}, c_{k_1 w_j}, B_{1,j})$ from P_j, it decrypts as in a real execution of the protocol to obtain $\alpha_{1,j}$ and $\mu_{1,j}$

(c) $\mathcal{S}$ verifies that $\mu_{1,j} P + B_{1,j} = k_1 W_j$. If so, since $\mathcal{S}$ also knows k_1 and w_j, it computes $\nu_{1,j} = k_1 w_j - \mu_{1,j} \mod q$

$\mathcal{S}$ computes $\delta_1 := k_1\gamma_1 + \sum_{k\neq 1} \alpha_{1,k} + \sum_{k\neq 1} \beta_{k,1}$. However $\mathcal{S}$ cannot compute σ_1 since it does not know w_1, but it can compute

$$\begin{aligned}\sum_{i>1} \sigma_i &= \sum_{i>1} (k_i w_i + \sum_{j\neq i} \mu_{i,j} + \nu_{j,i}) = \sum_{i>1}\sum_{j\neq i} (\mu_{i,j} + \nu_{j,i}) + \sum_{i>1} k_i w_i \\ &= \sum_{i>1} (\mu_{i,1} + \nu_{1,i}) + \sum_{i>1; j>1} k_i w_j\end{aligned}$$

since it knows all the values $\{k_j\}_{j\in S}$, $\{w_j\}_{j\in S, j\neq 1}$, it chooses the random values $\mu_{i,1}$ and it can compute all of the shares $\nu_{1,j} = k_1 w_j - \mu_{1,j} \mod q$.

Phase 3: $\mathcal{S}$ broadcasts δ_1 and receives all the $\{\delta_j\}_{j\in S, j\neq 1}$ from $\mathcal{A}$. Let $\delta := \sum_{i\in S} \delta_i$.

Phase 4: (a) $\mathcal{S}$ broadcasts d_1 which decommits to Γ_1, and $\mathcal{A}$ reveals $\{\mathsf{d}_j\}_{j\in S, j\neq 1}$ which decommit to $\{\Gamma_j\}_{j\in S, j>1}$.

(b) $\mathcal{S}$ proves knowledge of γ_1 s.t. $\Gamma_1 = \gamma_1 P$, and for $j \in S, j \neq 1$, receives the PoK of γ_j s.t. $\Gamma_j = \gamma_j P$. $\mathcal{S}$ extracts $\{\gamma_j\}_{j\in S, j\neq 1}$ from which it computes $\gamma := \sum_{i\in S} \gamma_i \mod q$ and $k := \delta \cdot \gamma^{-1} \mod q$.

(c) If $k = \tilde{k} \mod q$ (semi-correct execution), $\mathcal{S}$ proceeds as follows:

- $\mathcal{S}$ requests a signature (r, s) for m from its ECDSA signing oracle.
- $\mathcal{S}$ computes $R := s^{-1}(m \cdot P + r \cdot Q) \in \mathbb{G}$ (note that $r = H'(R) \in \mathbf{Z}/q\mathbf{Z}$).
- $\mathcal{S}$ rewinds $\mathcal{A}$ to the decommitment step at Phase 4. (a) and equivocates P_1's commitment to open to $\widehat{\Gamma}_1 := \delta \cdot R - \sum_{i>1} \Gamma_i$. It also simulates the proof of knowledge of $\widehat{\gamma}_1$ s.t. $\widehat{\Gamma}_1 = \widehat{\gamma}_1 P$. Note that $\delta^{-1}(\widehat{\Gamma}_1 + \sum_{i>1} \Gamma_i) = R$.

Phase 5: Now $\mathcal{S}$ knows $\sum_{j\in S, j\neq 1} s_j$ held by $\mathcal{A}$ since $s_j = k_j m + \sigma_j r$.

– $\mathcal{S}$ computes s_1 held by P_1 as $s_1 := s - \sum_{j\in S, j\neq 1} s_j$.

– $\mathcal{S}$ continues the steps of Phase 5 as in a real execution.

(d) Else $k \neq \tilde{k} \mod q$ (non-semi-correct), and $\mathcal{S}$ proceeds as follows:

- $\mathcal{S}$ computes $R := \delta^{-1}(\sum_{i\in S} \Gamma_i) = k \cdot P$ and $r := H'(R) \in \mathbf{Z}/q\mathbf{Z}$.
- Phase 5: $\mathcal{S}$ does the following
  - sample a random $\tilde{s}_1 \xleftarrow{\$} Zq$.
  - sample $\ell_1, \rho_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, compute $V_1 := s_1 R + \ell_1 P$; $A_1 := \rho_1 P$; $[\widehat{\mathsf{c}}_1, \widehat{\mathsf{d}}_1] \leftarrow \mathsf{Com}(V_1, A_1)$ and send $\widehat{\mathsf{c}}_1$ to $\mathcal{A}$.
  - receive $\{\widehat{\mathsf{c}}_j\}_{j\neq 1}$ and decommit by broadcasting $\widehat{\mathsf{d}}_1$. Prove knowledge of (s_1, ℓ_1, ρ_1) s.t. $(V_1 = s_1 R + \ell_1 P) \wedge (A_1 = \rho_1 P)$.
  - For $j \in S$, $j \neq 1$, $\mathcal{S}$ receive $\widehat{\mathsf{d}}_j$ and the ZKPoK of (s_j, ℓ_j, ρ_j) s.t. $V_j = s_j R + \ell_j P \wedge A_j = \rho_j P$.
  - Compute $V := -mP - rQ + \sum_{i\in S} V_i$, $A := \sum_{i\in S} A_1$, $T_1 := \ell_1 A$ and sample a random $U_1 \xleftarrow{\$} \mathbb{G}$.
  - Compute $[\tilde{\mathsf{c}}_1, \tilde{\mathsf{d}}_1] \leftarrow \mathsf{Com}(U_1, T_1)$ and send $\tilde{\mathsf{c}}_1$ to $\mathcal{A}$. Upon receiving $\{\tilde{\mathsf{c}}_j\}_{j\neq 1}$ from $\mathcal{A}$, broadcast $\tilde{\mathsf{d}}_1$ and receive the $\{\tilde{\mathsf{d}}_j\}_{j\neq 1}$.
  - Now since $\sum_{i\in S} T_1 \neq \sum_{i\in S} U_1$ both $\mathcal{A}$ and $\mathcal{S}$ abort.

The simulation of a semi-correct execution

Lemma 4.3.1. Assuming the strong root assumption and the C-low order assumption hold for Gen; the CL encryption scheme is δ_s-smooth ; and the commitment scheme is non-malleable and equivocable; then on input m the simulation either outputs a valid signature (r, s) or aborts, and is computationally indistinguishable from a semi-correct real execution.

Proof. The differences between the real and simulated views are the following:

1. $\mathcal{S}$ does not know w_1. So for $j > 1$ it cannot compute $c_{k_j w_1}$ as in a real execution of the protocol. However under the strong root and C-low order assumption in $\widehat{G}$, $\mathcal{S}$ can extract k_j from proof π_j in Phase 1. It then samples a random $\mu_{j,1} \in \mathbf{Z}/q\mathbf{Z}$,

computes $B_{j,1} := k_j \cdot W_1 - \mu_{j,1} \cdot P$, and $c_{k_j w_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mu_{j,1})$. The resulting view of $\mathcal{A}$ is identical to an honestly generated one since both in real and simulated executions $\mu_{j,1}$ is uniformly distributed in $\mathbf{Z}/q\mathbf{Z}$, while $B_{j,1}$ follows the uniform distribution in $\mathbb{G}$ and passes the check $B_{j,1} + \mu_{j,1} \cdot P = k_j \cdot W_1$ performed by $\mathcal{A}$. Moreover c_{k_j} was proven to be a valid ciphertext, so ciphertexts computed using homomorphic operations over c_{k_j} and fresh ciphertexts computed with pk_j follow identical distributions from $\mathcal{A}$'s view.

2. $\mathcal{S}$ computes $\widehat{\Gamma}_1 := \delta \cdot R - \sum_{i>1} \Gamma_i$, and equivocates its commitment c_1 s.t. d_1 decommits to $\widehat{\Gamma}_1$. Let us denote $\widehat{\gamma}_1 \in \mathbf{Z}/q\mathbf{Z}$ the value s.t. $\widehat{\Gamma}_1 = \widehat{\gamma}_1 P$, where $\widehat{\gamma}_1$ is unknown to $\mathcal{S}$, but the forger can simulate the ZKPoK of $\widehat{\gamma}_1$.
Let us further denote $\widehat{k} \in \mathbf{Z}/q\mathbf{Z}$ the randomness (unknown to $\mathcal{S}$) used by its' signing oracle to produce (r, s). It holds that $\delta = \widehat{k}(\widehat{\gamma}_1 + \sum_{j \in S, j>1} \gamma_j)$. Finally, let us denote $\widehat{k}_1 := \widehat{k} - \sum_{j \in S, j>1} k_j$.
Since δ was made public in Phase 3, by decommiting to $\widehat{\Gamma}_1 = \widehat{\gamma}_1 P$ instead of $\Gamma_1 = \gamma_1 P$, $\mathcal{S}$ is implicitly using $\widehat{k}_1 \neq k_1$, even though $\mathcal{A}$ received an encryption of k_1 in Phase 1. However, from the smoothness of the CL scheme, and the hardness of the HSM problem, this change is unnoticeable to $\mathcal{A}$.

Claim 2. If the CL encryption scheme is δ_s-smooth and the HSM problem is δ_{HSM}-hard, then no probabilistic polynomial time adversary $\mathcal{A}$ – interacting with $\mathcal{S}$ – can notice the value of k_1 in the computation of R being replaced by the (implicit) value $\widehat{k}$ with probability greater than $2\delta_{\mathsf{HSM}} + 3/q + 4\delta_s$.

Proof. To see this consider the following sequence of games. We denote E_i the probability $\mathcal{A}$ outputs 1 in $\mathsf{Game_i}$.
$\mathsf{Game_0}$ *to* $\mathsf{Game_1}$. $\mathcal{S}$ uses the secret key sk_1 instead of the public key pk_1 and r_1 to compute $c_{k_1} \leftarrow (u_1, u_1^{\mathsf{sk}_1} f^{k_1})$ where $u_1 = g_q^{r_1}$. Both games are perfectly indistinguishable from $\mathcal{A}$'s view:

$$|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_0]| = 0.$$

$\mathsf{Game_1}$ *to* $\mathsf{Game_2}$. In $\mathsf{Game_2}$ one replaces the first element of c_{k_1} (in $\mathsf{Game_1}$ this is $u_1 \in G^q$) with $\tilde{u}_1 \in G \backslash G^q$. There exists a unique $r_1 \in \mathbf{Z}/s\mathbf{Z}$ and $b_1 \in \mathbf{Z}/q\mathbf{Z}$ such that $\tilde{u}_1 = g_q^{r_1} f^{b_1}$. And $c_{k_1} = (\tilde{u}_1, \tilde{u}_1^{\mathsf{sk}_1} f^{k_1})$. Under the δ_{HSM}-hardness of HSM both games are indistinguishable:

$$|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_1]| \leqslant \delta_{\mathsf{HSM}}.$$

$\mathsf{Game_2}$ *to* $\mathsf{Game_3}$. In $\mathsf{Game_3}$ the points $Q = x \cdot P$ and $R = \widehat{k}^{-1} \cdot P$ come from the ECDSA oracle, while in $\mathsf{Game_2}$ they are computed as in the real protocol. As a result, the value k_1 encrypted in c_{k_1} is unrelated to $\widehat{k}$. Let us denote $\widehat{k}_1 := \widehat{k} - \sum_{j>1} k_j$, this is the value that – if used by $\mathcal{S}$ instead of k_1 – would lead to the joint computation of $R = \widehat{k}^{-1} P$.

To demonstrate that $\mathsf{Game_2}$ and $\mathsf{Game_3}$ are indistinguishable from $\mathcal{A}$'s view, we start by considering a fixed $\widehat{\mathsf{sk}_1} \in \mathbf{Z}$ satisfying the following equations:

$$\begin{cases} \widehat{\mathsf{sk}_1} \equiv \mathsf{sk}_1 \bmod \varpi, \\ \widehat{\mathsf{sk}_1} \equiv \mathsf{sk}_1 + {b_1}^{-1}(k_1 - \widehat{k}_1) \bmod q, \end{cases}$$

where ϖ is the group exponent of $\widehat{G}$ (cf. Subsubection 4.1.1), such that the order s of g_q divides ϖ. Note that the smoothness of the CL encryption scheme ensures that such a $\widehat{\mathsf{sk}_1}$ exists (it is not necessarily unique). We can now see that in Game_3, c_{k_1} is an invalid encryption of both $\widehat{k}_1$ and of k_1, for respective secret keys $\widehat{\mathsf{sk}_1}$ and sk_1, but for the same public key pk_1, indeed:

$$\begin{aligned} c_{k_1} &= (\tilde{u}_1, \tilde{u}_1^{\mathsf{sk}_1} f^{k_1}) = (g_q^{r_1} f^{b_1}, (g_q^{r_1} f^{b_1})^{\mathsf{sk}_1} \cdot f^{k_1}) \\ &= (g_q^{r_1} f^{b_1}, \mathsf{pk}_1^{r_1} f^{\widehat{\mathsf{sk}_1} \cdot b_1 + \widehat{k}_1}) = (\tilde{u}_1, \tilde{u}_1^{\widehat{\mathsf{sk}_1}} f^{\widehat{k}_1}). \end{aligned}$$

Adversary $\mathcal{A}$ receives the point Q, the encryption key $\mathsf{pk}_1 = g_q^{\mathsf{sk}_1}$, and c_{k_1} from $\mathcal{S}$ (at this point $\mathcal{A}$ view is identical to that in Game_2). Now $\mathcal{A}$ corrupting P_j computes $c_{k_1\gamma_j}$ which we denote $c_\alpha = (u_\alpha, e_\alpha)$, and $c_{k_1 w_j}$ which we denote $c_\mu = (u_\mu, e_\mu)$. $\mathcal{A}$ then sends c_α and c_μ to $\mathcal{S}$. The difference between Game_2 and Game_3 appears now in how $\mathcal{S}$ attempts to decrypt c_α and c_μ. In Game_2 it would have used $\widehat{\mathsf{sk}_1}$, whereas in Game_3 it uses sk_1.

Notation. We denote α (resp. μ) the random variable obtained by decrypting c_α (resp. c_μ) (received in Game_3) with decryption key sk_1; we denote α' (resp. μ') the random variable obtained by decrypting c_α (resp. c_μ) (received in Game_3) with decryption key $\widehat{\mathsf{sk}_1}$; we introduce a hypothetical $\mathsf{Game}_3{}'$, which is exactly as Game_3, only one decrypts c_α (resp. c_μ) (received in Game_3) with decryption key $\widehat{\mathsf{sk}_1}$, thus obtaining α' (resp. μ'). Moreover in Game $3'$ the check performed on the curve is 'If $\mu' \cdot P + B_{1,j} \neq \widehat{k}_1 \cdot W_j$ then abort'.

Observation. The view of $\mathcal{A}$ in Game_2 and in $\mathsf{Game}_3{}'$ is identical. By demonstrating that the probability $\mathcal{A}$'s view differs when $\mathcal{S}$ uses α, μ in Game_3 from when it uses α', μ' in $\mathsf{Game}_3{}'$ is negligible, we can conclude that $\mathcal{A}$ cannot distinguish Game_2 and Game_3 except with negligible probability.

The smoothness of the CL encryption scheme tells us that given pk_1, which fixes (sk_1 mod s), the value of (sk_1 mod q) remains δ-close to the uniform distribution modulo q. In particular this ensures that $\mathcal{A}$'s view of α and α' are δ-close. Indeed, $\mathcal{A}$ receives an invalid encryption of k_1, which information theoretically masks k_1. At this point $\mathcal{A}$'s view of k_1 is that of a random variable δ-close to the uniform distribution modulo q. $\mathcal{A}$ then computes c_α which it sends to $\mathcal{S}$. Finally $\mathcal{A}$ receives either (a one way function of) k_1, or (a one way function of) some random value which is unrelated to k_1, and must decide which it received.

For μ and μ', the indistinguishability of $\mathcal{A}$'s view of both random variables is a little more delicate, since $\mathcal{A}$ gets additional information from the check on the curve performed by $\mathcal{S}$, namely in Game_3 if $\mu \cdot P + B_{1,j} \neq k_1 \cdot W_j$ the simulator aborts. We call the output of this check test. And in $\mathsf{Game}_3{}'$, if $\mu' \cdot P + B_{1,j} \neq \widehat{k}_1 \cdot W_j$ the simulator aborts. We call the output of this check test'. Notice that if $\mathsf{test} = \mathsf{test}'$, both games are δ_s-close from $\mathcal{A}$'s view (the only change is in the ciphertext c_{k_1}). Let us bound the probability $\mathfrak{p}$ that $\mathsf{test} \neq \mathsf{test}'$. This will allow us to conclude that

$$|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_2]| \leq \mathfrak{p} + \delta_s.$$

Let us consider the ciphertext $c_\mu = (u_\mu, e_\mu) \in \widehat{G} \times \widehat{G}$ sent by $\mathcal{A}$. There exist unique $z_\mu \in \widehat{G}^q$, $y_\mu \in F$ such that $u_\mu = z_\mu y_\mu$. Moreover there exists a unique $b_\mu \in \mathbf{Z}/q\mathbf{Z}$ such that $y_\mu = f^{b_\mu}$.

Since $\mathsf{sk}_1 = \widehat{\mathsf{sk}}_1 \bmod \varpi$, $\mu = \perp$ if and only if $\mu' = \perp$, and this occurs when $e_\mu \cdot z_\mu^{-\mathsf{sk}_1} = e_\mu \cdot z_\mu^{-\widehat{\mathsf{sk}}_1} \notin F$. In this case Game_3 is identical to $\mathsf{Game}_3{}'$ from $\mathcal{A}$'s view ($\mathcal{S}$ aborts in both cases). We hereafter assume decryption does not fail, which allows us to adopt the following notation $e_\mu = z_\mu^{\mathsf{sk}_1} f^{h_\mu} = z_\mu^{\widehat{\mathsf{sk}}_1} f^{h_\mu}$ with $h_\mu \in \mathbf{Z}/q\mathbf{Z}$. We thus have:

$$\mu := \log_f\left(\frac{e_\mu}{u_\mu^{\mathsf{sk}_1}}\right) = h_\mu - b_\mu \mathsf{sk}_1 \bmod q,$$

$$\mu' := \log_f\left(\frac{e_\mu}{u_\mu^{\widehat{\mathsf{sk}}_1}}\right) = h_\mu - b_\mu \widehat{\mathsf{sk}}_1 \bmod q$$

Thus we have

$$\mu - \mu' \equiv b_\mu(\widehat{\mathsf{sk}}_1 - \mathsf{sk}_1) \equiv b_\mu {b_1}^{-1}(k_1 - \widehat{k}_1) \bmod q.$$

We consider three cases:

(a) $\mu = \mu' \bmod q$. This may happen for two reasons:

  i. If $k_1 \equiv \widehat{k}_1 \bmod q$, then Game_2 and Game_3 are identical.

  ii. Else $b_\mu = 0 \bmod q$, i.e. c_μ is a valid ciphertext. Since we ruled out $k_1 \equiv \widehat{k}_1 \bmod q$ in the previous case, if test=true, necessarily test'=false, and vis versa. Both cases being symmetric, we consider the case test=true. From $\mathcal{A}$'s view, before outputting c_μ the only fixed information relative to k_1 is that contained $c_{k_1} = (g_q^{r_1} f^{b_1}, (g_q^{r_1} f^{b_1})^{\mathsf{sk}_1} f^{k_1})$. This fixes $\pi_0 := b_1 \cdot \mathsf{sk}_1 + k_1 \bmod q$. However from $\mathcal{A}$'s view, given pk_1, the random variable sk_1 follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. Thus k_1 also follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. Now suppose $\mathcal{A}$ returns $c_\mu = (z_\mu, z_\mu^{\mathsf{sk}_1} f^\mu)$ where $z_\mu \in \widehat{G}^q$. If $\mathsf{test} = \mathsf{true}$, then $\mu \cdot P + B_{1,j} = k_1 W_j$, and $\mathcal{A}$ has fixed the correct value of k_1, this occurs with probability $\leqslant 1/q + \delta_s$.

(b) $\mu \not\equiv \mu' \bmod q$ but $\mu - \mu' = w_j(k_1 - \widehat{k}_1) \bmod q$, i.e. $b_\mu = w_j b_1 \bmod q$. This results in $\mathcal{S}$ aborting on μ' in Game_2 if and only if $\mathcal{S}$ aborts on μ in Game_3. This occurs if the adversary performs homomorphic operations on c_{k_1}, and the difference between the random variables is that expected by $\mathcal{S}$. Indeed:

$$\mu = k_1 w_j - \nu_{1,j} \Leftrightarrow \mu' + w_j(k_1 - \widehat{k}_1) = k_1 w_j - \nu_{1,j} \Leftrightarrow \mu' = \widehat{k}_1 w_j - \nu_{1,j}.$$

(c) ($\mu \not\equiv \mu' \bmod q$) and ($\mu - \mu' \not\equiv w_j(k_1 - \widehat{k}_1) \bmod q$). We here consider three sub-cases:

  i. Either $\mathsf{test} = \mathsf{test}' = \mathsf{false}$; this results in identical views for $\mathcal{A}$.

  ii. Either $\mathsf{test}' = \mathsf{true}$; this means that:

$$\mu' = \widehat{k}_1 w_j - \nu_{1,j} \bmod q.$$

Now since $\mu-\mu' \neq w_j(k_1-\widehat{k}_1) \bmod q$ necessarily $\mathsf{test} = \mathsf{false}$. Consequently if this event occurs, $\mathcal{A}$'s view differs. Let us prove that information theoretically, this can not happen with probability greater than $1/q + \delta_s$. For clarity, we first recall the expression of c_{k_1} received by $\mathcal{A}$:

$$c_{k_1} = (g_q^{r_1} f^{b_1}, \mathsf{pk}_1^{r_1} f^{\widehat{\mathsf{sk}}_1 b_1 + \widehat{k}_1})$$

where $b_1 \neq 0 \bmod q$. We also recall the expression of c_μ, sent by $\mathcal{A}$ to $\mathcal{S}$. Since c_μ decrypts to μ' with decryption key $\widehat{\mathsf{sk}}_1$, we can write:

$$c_\mu = (z_\mu f^{b_\mu}, z_\mu^{\widehat{\mathsf{sk}}_1} f^{\mu' + b_\mu \widehat{\mathsf{sk}}_1}).$$

Let us denote $\pi_0 := \widehat{\mathsf{sk}}_1 b_1 + \widehat{k}_1 \bmod q$ and $\pi_1 := \mu' + b_\mu \widehat{\mathsf{sk}}_1$. For this case to occur, it must hold that $\mu' = \widehat{k}_1 w_j - \nu_{1,j} \bmod q$, so

$$\pi_1 = \widehat{k}_1 w_j - \nu_{1,j} + b_\mu \widehat{\mathsf{sk}}_1 \bmod q.$$

Substituting $\widehat{\mathsf{sk}}_1$ for $(\pi_0 - \widehat{k}_1) b_1^{-1}$ yields:

$$\begin{aligned} \pi_1 &= \widehat{k}_1 w_j - \nu_{1,j} + b_\mu b_1^{-1}(\pi_0 - \widehat{k}_1) \bmod q \\ \Leftrightarrow \pi_1 + \nu_{1,j} - b_\mu b_1^{-1} \pi_0 &= \widehat{k}_1 (w_j - b_\mu b_1^{-1}) \bmod q \end{aligned}$$

As we dealt with $b_\mu = w_j b_1 \bmod q$ in case (b), here $w_j - b_\mu b_1^{-1}$ is invertible mod q so we can write:

$$\widehat{k}_1 = (\pi_1 + \nu_{1,j} - b_\mu b_1^{-1} \pi_0)(w_j - b_\mu b_1^{-1})^{-1} \bmod q \tag{4.1}$$

where π_0, b_1 are fixed by c_{k_1}; π_1, b_μ are fixed by c_μ; w_j is fixed by W_j; and $\nu_{1,j}$ is fixed by $B_{1,j}$. So given $\mathcal{A}$'s view and $\mathcal{A}$'s output ($B_{1,j}$ and c_μ), all the terms on the right hand side of Eq. 4.1 are fixed. However, given pk_1, c_{k_1} and W_j (which is all the relevant information $\mathcal{A}$ gets prior to outputting c_μ), the δ_s-smoothness of the projective hash family ensures that $\widehat{k}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. If the current case occurs, Eq. 4.1 must hold, thus from being given a view where $\widehat{k}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$, $\mathcal{A}$ succeeds in fixing this random variable to be the exact value used by $\mathcal{S}$. This occurs with probability $\leqslant 1/q + \delta_s$.

iii. Else $\mathsf{test} = \mathsf{true}$; this means that $\mu = k_1 w_j - \nu_{1,j} \bmod q$. Since ($\mu - \mu' \neq w_j(k_1 - \widehat{k}_1) \bmod q$) necessarily test' fails, and $\mathcal{A}$'s view differs. Reasoning as in the previous case, but setting $\pi_0 := \mathsf{sk}_1 b_1 + k_1 \bmod q$ and $\pi_1 := \mu + b_\mu \mathsf{sk}_1$, one demonstrates that this case occurs with probability $\leqslant 1/q + \delta_s$.

Combining the above, we get that $\mathsf{test}' \neq \mathsf{test}$ if and only if we are in case (a) ii. (c) ii. or (c) iii., which occurs with probability $\leqslant 3(1/q + \delta_s)$. Thus:

$$|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_2]| \leqslant 3/q + 4\delta_s.$$

Game_3 *to* Game_4. In Game_4, the first element u_1 of c_{k_1} is once again sampled in G^q. Both games are indistinguishable under the hardness of HSM and:

$$|\Pr[\mathsf{E}_4] - \Pr[\mathsf{E}_3]| \leq \delta_{\mathsf{HSM}}.$$

Game_4 *to* Game_5. In Game_5 $\mathcal{S}$ uses the public key pk_1 to encrypt k_1. The change here is exactly that between Game_0 and Game_1, both games are perfectly indistinguishable, and:

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_4]| = 0.$$

Real/Ideal executions. Putting together the above probabilities, we get that:

$$|\Pr[\mathsf{E}_6] - \Pr[\mathsf{E}_0]| \leq 2\delta_{\mathsf{HSM}} + 3/q + 4\delta,$$

which concludes the proof of the claim. □

3. We now tackle the third and last difference between the real and simulated executions of the signature protocol. Justifying that this difference is unnoticeable to the adversary will allow us to conclude the proof of Lemma 4.3.1. Notice that $\mathcal{S}$ does not know σ_1, and thus cannot compute s_1 as in a real execution. Instead it computes $s_1 = s - \sum_{j\in S, j\neq 1} s_j = s - \sum_{j\in S, j\neq 1}(k_j m + \sigma_j r)$ where (implicitly) $s = \widehat{k}(m + rx)$. So $s_1 = \widehat{k}_1 m + r(\widehat{k}x - \sum_{j\in S, j\neq 1}\sigma_j)$, and $\mathcal{S}$ is implicitly setting $\widehat{\sigma}_1 := \widehat{k}x - \sum_{j\in S, j\neq 1}\sigma_j$ s.t. $\widehat{k}x = \widehat{\sigma}_1 + \sum_{j\in S, j\neq 1}\sigma_j$.
   We note that, since the real execution is semi correct, the correct shares of k for the adversary are the k_i that the simulator knows and $R = \widehat{k}P = (\widehat{k_1} + \sum_{j\in S, j\neq 1} k_j)$. Therefore the value s_1 computed by $\mathcal{S}$ is consistent with a correct share for P_1 for a valid signature (r, s), which makes Phase 5 indistinguishable from the real execution to the adversary.

   In particular, observe that if none of the parties aborted during Phase 2, the output shares are correct. So if $\mathcal{A}$ here uses the values $\{\sigma_j\}_{j\in S, j>1}$ as computed in a real execution of the protocol, it expects the signature generation protocol to output a valid signature. And indeed with $\mathcal{S}$'s choice of $\widehat{\sigma}_1$ and $\widehat{k}_1$, the protocol will terminate, outputting the valid signature (r, s) it received from its signing oracle. Conversely, if $\mathcal{A}$ attempts to cheat in Phase 5 by using a different set of σ_j's than those prescribed by the protocol, the check $\sum_{i\in S} T_i = \sum_{i\in S} U_i$ will fail, and all parties abort, as in a real execution of the protocol.

□

Non semi-correct executions

Lemma 4.3.2. Assuming the strong root assumption and the C-low order assumption hold for Gen; the DDH assumption holds in $\mathbb{G}$; and the commitment scheme is non-malleable and equivocable; then the simulation is computationally indistinguishable from a non-semi-correct real execution.

Proof. We construct three games between the simulator $\mathcal{S}$ (running P_1) and the adversary $\mathcal{A}$ (running all other players). In G_0, $\mathcal{S}$ runs the real protocol. The only change between G_0 and G_1 is that in G_1, $\mathcal{S}$ chooses U_1 as a random group element. In G_2 the simulator $\mathcal{S}$ runs the simulation described in Section 4.3.

Indistinguishability of G_0 and G_1. We prove that if there exists an adversary $\mathcal{A}_0$ distinguishing games G_0 and G_1, $\mathcal{A}_0$ can be used to break the DDH assumption in $\widehat{G}$. Let $\tilde{A} = a{\cdot}P$, $\tilde{B} = b{\cdot}P$, $\tilde{C} = c{\cdot}P$ be the DDH challenge where $c = ab$ or c is random in $\mathbf{Z}_q$. The DDH distinguisher $\mathcal{S}_0$ runs $\mathcal{A}_0$, simulating the key generation phase s.t. $Q = \tilde{B}$. It does so by rewinding $\mathcal{A}_0$ in step 7 of the IKeyGen simulation and changing the decommitment of P_1 to $Q_1 := \tilde{B} - \sum_{j\in[n],j\neq 1} Q_j$. $\mathcal{S}_0$ also extracts the values $\{x_j\}_{j\in[n],j\neq 1}$ chosen by $\mathcal{A}_0$ from the ZKPoK of step 11 of the IKeyGen simulation. Note that at this point $Q = \tilde{B}$ and $\mathcal{S}_0$ knows x_i and the decryption key sk_1 matching pk_1, but not b and therefore not x_1.

Next $\mathcal{S}_0$ runs the signature generation protocol for a non-semi-correct execution. Recall that $S \subseteq [n]$ denotes the subset of players collaborating in ISign. Denoting $t := |S|$, the (t, n) shares $\{x_i\}_{i\in[n]}$ are converted into (t, t) shares $\{w_i\}_{i\in S}$ as per the protocol. Thus $b = \sum_{i\in S} w_i$ where $\mathcal{S}_0$ knows $\{w_j\}_{j\in S,j\neq 1}$ but not w_1. We denote $w_A := \sum_{j\in S,j\neq 1} w_j$ (which is known to $\mathcal{S}_0$) s.t. $w_1 = b - w_A$. $\mathcal{S}_0$ runs the protocol normally for Phases $1, 2, 3, 4$. It extracts the values $\{\gamma_j\}_{j\in S,j\neq 1}$ from the proof of knowledge in Phase 4, and knows γ_1 since it ran P_1 normally. Therefore $\mathcal{S}_0$ knows k such that $R = k^{-1} \cdot P$ since $k = (\sum_i \gamma_i)^{-1}\delta \mod q$. It also knows k_1 (chosen normally according to the protocol) and $\{k_j\}_{j\in S,j\neq 1}$ which it can extract from the proofs in Phase 1.

Before moving to the simulation of Phase 5, let's look at Phase 2 of the protocol for the computation of the shares σ_i. We note that since $\mathcal{S}_0$ knows sk_1 it also knows all the shares $\mu_{1,j}$ since it can decrypt the ciphertext $c_{k_1 w_j}$ it receives from P_j. However $\mathcal{S}_0$ does not know w_1 therefore it sends the encryption of a random $\mu_{j,1}$ to P_j and sets (implicitly) $\nu_{j,1} = k_j w_1 - \mu_{j,1}$. At the end the share σ_1 held by P_1 is

$$\sigma_1 = k_1 w_1 + \sum_{j\in S,j\neq 1} (\mu_{1,j} + \nu_{j,1}) = \tilde{k} w_1 + \sum_{j\in S,j\neq 1} (\mu_{1,j} - \mu_{j,1}) \text{ where } \tilde{k} = \sum_{i\in S} k_i.$$

Recall that since this is a non-semi-correct execution $\tilde{k} \neq k$ where $R = k^{-1} \cdot P$. Since $w_1 = b - w_A$ we have $\sigma_1 = \tilde{k}b + \mu_1$ where $\mu_1 = \sum_{j\in S,j\neq 1}(\mu_{1,j} - \mu_{j,1}) - \tilde{k}w_A$ with $\mu_1, \tilde{k}$ known to $\mathcal{S}_0$. This allows $\mathcal{S}_0$ to compute the correct value $\sigma_1 \cdot P = \tilde{k}\tilde{B} + \mu_1 \cdot P$ and therefore the correct value of $s_1 \cdot R$ as:

$$\begin{aligned} s_1 \cdot R &= (k_1 m + r\sigma_1) \cdot R = k^{-1}(k_1 m + r\sigma_1) \cdot P \\ &= k^{-1}(k_1 m + r\mu_1) \cdot P + k^{-1}(\tilde{k}r) \cdot \tilde{B} = \hat{\mu}_1 \cdot P + \hat{\beta}_1 \cdot \tilde{B} \end{aligned}$$

where $\hat{\mu}_1 = k^{-1}(k_1 m + r\mu_1)$ and $\hat{\beta}_1 = k^{-1}\tilde{k}r$ are known to $\mathcal{S}_0$.

In the simulation of Phase 5, $\mathcal{S}_0$ selects a random ℓ_1 and sets $V_1 := s_1 \cdot R + \ell_1 \cdot P$, $A_1 = \rho_1 \cdot P = \tilde{A} = a \cdot P$. It simulates the ZK proof (since it does not know ρ_1 or s_1). It extracts s_i, ℓ_i, ρ_i from $\mathcal{A}_0$'s proofs s.t. $V_i = s_i \cdot R + \ell_i \cdot P = k^{-1}s_i \cdot P + \ell_i \cdot P$ and $A_i = \rho_i \cdot P$. Let $s_A = \sum_{j\in S,j\neq 1} k^{-1}s_j$. Note that, substituting the above relations (and setting $\ell = \sum_{i\in S} \ell_i$), we have: $V = -m{\cdot}P - r{\cdot}Q + \sum_{i\in S} V_i = \ell{\cdot}P + s_1{\cdot}R + (s_A - m){\cdot}P - r{\cdot}Q$. Moreover $Q = \tilde{B}$ so $-r \cdot Q = -r \cdot \tilde{B}$, and:

$$V = \ell \cdot P + \hat{\mu}_1 \cdot P + \hat{\beta}_1 \cdot \tilde{B} + (s_A - m) \cdot P - r \cdot \tilde{B} = (\ell + \theta) \cdot P + \kappa \cdot \tilde{B}$$

where $\mathcal{S}_0$ knows $\theta = \hat{\mu}_1 + s_A - m$ and $\kappa = \hat{\beta}_1 - r$. Note that for executions that are not semi-correct $\kappa \neq 0$.

Next $\mathcal{S}_0$ computes $T_1 := \ell_1 \cdot A$ (correctly), but computes U_1 as $U_1 := (\ell+\theta)\cdot\tilde{A}+\kappa\cdot\tilde{C}$, using this U_1 it continues as per the real protocol and aborts on the check $\sum_{i\in S} T_i = \sum_{i\in S} U_i$.

Observe that when $\tilde{C} = ab\cdot P$, by our choice of $a = \rho_1$ and $b = x$, we have that $U_1 = (\ell+\theta)\rho_1 \cdot P + \kappa\cdot\rho_1\tilde{B} = \rho_1\cdot V$ as in Game G_0. However when $\tilde{C}$ is a random group element, U_1 is uniformly distributed as in G_1. Therefore under the DDH assumption G_0 and G_1 are indistinguishable.

Indistinguishability of G_1 and G_2. In G_2, $\mathcal{S}$ broadcasts a random $\tilde{V}_1 = \tilde{s}_1\cdot R+\ell_1\cdot P$. This is indistinguishable from the correct $V_1 = s_1\cdot R+\ell_1\cdot P$ thanks to the mask $\ell_1\cdot P$ which (under the DDH assumption) is computationally indistinguishable from a random value, since the adversary only knows A_1. To be precise, let $\tilde{A} = (a-\delta)\cdot P, \tilde{B} = b\cdot P$ and $\tilde{C} = ab\cdot P$ be the DDH challenge where δ is either 0 or random in $\mathbb{Z}_q$. The simulator proceeds as in G_0 (i.e. the regular protocol) until Phase 5. In Phase 5 $\mathcal{S}_0$ broadcasts $V_1 = \tilde{s}_1\cdot R+\tilde{A}$ and $A_1 = \tilde{B}$. It simulates the ZKPoK (it does not know ℓ_1 or ρ_1), and extracts s_i, ℓ_i, ρ_i from the adversary s.t. $V_i = s_i\cdot R+\ell_i\cdot P = k^{-1}s_i\cdot P+\ell_i\cdot P$ and $A_i = \rho_i\cdot P$.

Next $\mathcal{S}_0$ samples a random U_1 and sets $T_1 := \tilde{C}+\sum_{j\in S, j\neq 1}\rho_j\cdot\tilde{A}$ before aborting. Note that when $\tilde{A} = a\cdot P$, we implicitly set $a = \ell_1$ and $b = \rho_1$ and have $V_1 = s_1\cdot R+\ell_1\cdot P$ and $T_1 = \ell_1\cdot A$ as in Game G_1. However when $\tilde{A} = a\cdot P-\delta\cdot P$ with a random δ, then this is equivalent to having $V_1 = \tilde{s}_1\cdot R+\ell_1\cdot P$ and $T_1 = \ell_1\cdot A$ with a randomly distributed $\tilde{s}_1$ as in Game G_2. Therefore under the DDH assumption G_1 and G_2 are indistinguishable. $\square$

Concluding the proof

As mentioned at the beginning of Section 4.3 in the simulation of the signing subprotocol, the forger $\mathcal{S}$ simulating $\mathcal{A}$'s environment can detect whether we are in a semi-correct-execution or not, i.e. whether $\mathcal{A}$ decides to be malicious and terminate the protocol with an invalid signature. Consequently $\mathcal{S}$ always knows how to simulate $\mathcal{A}$'s view and all simulations are indistinguishable of real executions of the protocol. Moreover if $\mathcal{A}$, having corrupted up to t parties in the threshold ECDSA protocol, outputs a forgery, since $\mathcal{S}$ set up with $\mathcal{A}$ the same public key Q as it received from its' ECDSA challenger, $\mathcal{S}$ can use this signature as its own forgery, thus breaking the existential unforgeability of standard ECDSA.

Denoting $\mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi,\mathcal{A}}$, $\mathcal{A}$'s advantage in breaking the existential unforgeability of our threshold protocol, and $\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa},\mathcal{S}}$ the forger $\mathcal{S}$'s advantage in breaking the existential unforgeability of standard ECDSA, from Lemmas 4.3.1 and 4.3.2 it holds that if the DDH assumption holds in $\mathbb{G}$; the strong root assumption and the C-low order assumption hold for Gen; the CL encryption scheme is $\mathsf{ind\text{-}cpa}$-secure; and the commitment scheme is non-malleable and equivocable then: $|\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa},\mathcal{S}} - \mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi,\mathcal{A}}| \leq \mathsf{negl}(\lambda)$. Under the security of the ECDSA signature scheme, $\mathsf{Adv}^{\mathsf{eu\text{-}cma}}_{\mathsf{ecdsa},\mathcal{S}}$ must be negligible, which implies that $\mathsf{Adv}^{\mathsf{tu\text{-}cma}}_{\Pi,\mathcal{A}}$ should too, thus contradicting the assumption that $\mathcal{A}$ has non-negligible advantage of forging a signature for our protocol. We can thus state the following theorem, which captures the security of the protocol.

Theorem 4.3.3. Assuming standard ECDSA is an existentially unforgeable signature scheme; the DDH assumption holds in $\mathbb{G}$; the strong root assumption and the C-low order

$$\begin{array}{ccc}
P(x, h := g_q^x) & \text{public: } g_q & V(h) \\
\hline
r \xleftarrow{\$} [0, \tilde{s} \cdot 2^{90}] & & \\
t := g_q^r & \xrightarrow{\quad t \quad} & \\
 & \xleftarrow{\quad k \quad} & k \xleftarrow{\$} \{0,1\}^{10} \\
u := r + kx \in \mathbf{Z} & \xrightarrow{\quad u \quad} & \\
 & & \text{Check } g_q^u = t \cdot h^k \\
\hline
\end{array}$$

Figure 4.5: ZKPoK of z s.t. $h^y = g_q^z$ where $y = \text{lcm}(1, 2, 3, \ldots, 2^{10})$

assumption hold for Gen; the CL encryption scheme is $\mathsf{ind\text{-}cpa}$-secure; and the commitment scheme is non-malleable and equivocable, then the (t, n)-threshold ECDSA protocol of Figure 4.3 and 4.4 is an existentially unforgeable threshold signature scheme.

4.4 Further improvements

4.4.1 An improved ZKPoK which kills low order elements.

We here provide a proof of knowledge of discrete logarithm in a group of unknown order. Traditionally, if one wants to perform such a proof, the challenge set must be binary, which implies expensive protocols as the proof must be repeated many times to achieve a satisfying (non computational) soundness error. Here using what we call the *lowest common multiple* trick, we are able to significantly increase the challenge set, and thereby reduce the number of repetitions required of the proof. We first present the resulting proof, before providing two applications: one for the $\mathsf{CL.ISetup}$ protocol of Section 4.2.2, and another for our two party ECDSA protocol in [CCL+19], which we presented in Subsection 3.4.2. Throughout this subsection we denote $y := \text{lcm}(1, 2, 3, \ldots, 2^{10})$.

The lowest common multiple trick. For a given statement h, the proof does not actually prove knowledge of the DL of h, but rather of h^y. Precisely, the protocol of Figure 4.5 is a zero knowledge proof of knowledge for the following relation:

$$\mathsf{R}_{\mathsf{lcm-DL}} := \{(h, g_q); z \mid h^y = g_q^z\}.$$

Correctness. If $h = g_q^x$, then $g_q^u = g_q^{r+kx} = g_q^r \cdot (g_q^x)^k = t \cdot h^k$ and V accepts.
Special soundness. Suppose that for a committed value t, prover P^* can answer correctly for two different challenges k_1 and k_2. We call u_1 and u_2 the two answers. Let $k := k_1 - k_2$ and $u := u_1 - u_2$, then since $g_q^{u_1} = t \cdot h^{k_1}$ and $g_q^{u_2} = t \cdot h^{k_2}$, it holds that $g_q^u = h^k$. By the choice of the challenge set, y/k is an integer and so $(g_q^u)^{y/k} = (h^k)^{y/k} = h^y$. Denoting $z := uy/k$, P^* can compute z such that $g_q^z = h^y$, so if P can convince V for two different challenge values, then P^* can compute a z satisfying the relation.
Zero knowledge. Given h a simulator can sample $k \xleftarrow{\$} \{0,1\}^{10}$ and $u \xleftarrow{\$} [0, \tilde{s} \cdot (2^{90} + k)]$, compute $t := g_q^u \cdot h^{-k}$, such that distribution of the resulting transcript (h, t, k, u) is statistically close to those produced by a real execution of the protocol (this holds since

an honest prover samples x from $[\tilde{s} \cdot 2^{40}]$, the challenge space is of size 2^{10} and r is sampled from a set of size $\tilde{s} \cdot 2^{90}$, which thus statistically hides kx).

Application to the CL interactive set up. In the ISetup protocol of Section 4.2.2, in Step 2. 2. (c) each P_i computes $\pi_i^{\langle \hat{g}_q \rangle - \mathsf{DL}} := \mathsf{ZKPoK}_{g_i}\{(t_i) : g_i = \widehat{g}_q^{t_i}\}$. In fact it suffices for them to compute $\mathsf{ZKPoK}_{g_i}\{(z_i) : g_i^y = \widehat{g}_q^{z_i}\}$, where $y := \mathrm{lcm}(1, 2, 3, \dots, 2^{10})$ using the lcm trick. Then in Step 2. 3. all players compute $g_q := (\prod_{j=1}^n g_j)^y$. The resulting g_q has the required properties to be plugged into the $\mathsf{IKeyGen}$ protocol. We use this modified interactive set up for our efficiency comparisons of Section 4.5.

Application to the [CCL+19] interactive key generation. In Subsection 3.4.2 we presented a ZKPoK for the relation $\mathsf{R}_{\mathsf{CL-DL}}$ which is at the basis of our two party ECDSA protocol. That interactive proof uses binary challenges, consequently in order to achieve a satisfying soundness error of $2^{-\lambda}$, the proof must be repeated λ times. Using the lcm trick one can divide by 10 this number of rounds, though we obtain a ZKPoK for the following relation:

$$\mathsf{R}_{\mathsf{CL-lcm}} := \{(\mathsf{pk}, (c_1, c_2), Q); (x, z) \mid c_1^y = g_q^z \wedge c_2^y = f^{x \cdot y} \mathsf{pk}^z \wedge Q = xP\}.$$

In our two-party protocol this ZKPoK is computed by Alice, who sends this proof to Bob such that he is convinced her ciphertext $\mathbf{c} = (c_1, c_2)$ is well formed. Bob then performs some homomorphic operations on $\mathbf{c}$ and sends the result back to Alice. Now since with the proof based on the lcm trick, Bob is only convinced that $\mathbf{c}^y$ is a valid ciphertext, Bob raises $\mathbf{c}$ to the power y before performing his homomorphic operations[2]. When Alice decrypts she multiplies the decrypted value by $y^{-1} \mod q$ (this last step is much more efficient than Bob's exponentiation).

Remark 17. The size of the challenge set $\mathcal{C}$ from which k is sampled determines the number of times the protocol needs to be repeated in order to achieve a reasonable soundness error. Consequently it is desirable to take $\mathcal{C}$ as large as possible. However, at the end of the protocol, V is only convinced that h^y is well formed, where $y = \mathrm{lcm}(1, \dots, |\mathcal{C}|)$. So if V wants to perform operations on h which are returned to P, without risking leaking any information to P, V must raise h to the power y before proceeding. When plugged into our two-party ECDSA protocol this entails raising a ciphertext to the power y at the end of the key generation phase. So $|\mathcal{C}|$ must be chosen small enough for this exponentiation to take reasonable time. Hence we set $\mathcal{C} := \{0, 1\}^{10}$, and $y = \mathrm{lcm}(1, \dots, 2^{10})$, which is a 1479 bits integer, so exponentiating to the power y remains efficient. To achieve a soundness error of $2^{-\lambda}$ the protocol must be repeated $\lambda/10$ times.

4.4.2 Assuming a standardised group

If we assume a standardised set up process, which allowed to provide a description of $\widehat{G}$, of the subgroups F and G_q and of a *random* generator g_q of G_q, one could completely omit the interactive set up phase for the CL encryption scheme and have all parties use

[2]For correctness Bob also needs to multiply the signed message m' by $y \mod q$, during the signature algorithm.

the output of this standardised process. This significantly improves the $\mathsf{IKeyGen}$ protocol, as mentioned in Section 4.5.

Furthermore, assuming such a set up, we can replace the most expensive ZKPoK in [CCL+19] by an argument of knowledge using similar techniques to those of Section 4.2.1, and relying on the strong root and low order assumptions in $\widehat{G}$. We detail this improvement in the following paragraph.

Efficient ZKAoK for our two party protocol ([CCL+19]). Consider again the relation $\mathsf{R}_{\mathsf{CL-DL}}$ from Subsection 3.4.2. Using similar techniques to those proposed in Subsection 4.2.1, and relying on the strong root and low order assumptions in $\widehat{G}$, we describe in Figure 4.6 a much more efficient ZKAoK, which can be plugged into our overall two-party protocol so as to further improve its' overall computational and communication costs. Though the relation $\tilde{\mathsf{R}}_{\mathsf{CL-DL}}$ proved by our proof system is slightly different, it is sufficient for application in our work [CCL+19]. Precisely, our ZKAoK proves knowledge of a witness for the following relation:

$$\tilde{\mathsf{R}}_{\mathsf{CL-DL}} := \{(\mathsf{pk}, (c_1, c_2), Q); (x, (\rho_0, \rho_1)) \mid c_1 = g_q^{2^{-\rho_0 \cdot \rho_1}} \wedge c_2 = f^x \mathsf{pk}^{2^{-\rho_0 \cdot \rho_1}} \wedge Q = xP\}.$$

We emphasise that in order for security of this proof to hold, g_q must be a random generator of G^q. Since the set up described in [CCL+19] outputs a deterministic g_q, in order to plug the following proof in their protocol, we need to assume some standardised set up process as mentioned in Section 4.4.2. Theorem 4.4.1 states the security of the ZKAoK for $\tilde{\mathsf{R}}_{\mathsf{CL-DL}}$.

Theorem 4.4.1. Let $\mathcal{C}$ be the challenge set for the interactive protocol of Fig. 4.6, and $C := |\mathcal{C}|$. If the strong root assumption is $(t'(\lambda), \epsilon_{\mathsf{S}R}(\lambda))$-secure for Gen, and the C-low order assumption is $(t'(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for Gen, denoting $\epsilon := \max(\epsilon_{\mathsf{S}R}(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$, then the interactive protocol of Fig. 4.6 is a computationally convincing proof of knowledge for $\tilde{\mathsf{R}}_{\mathsf{CL-DL}}$ with knowledge error κ, time bound t and failure probability $\nu(\lambda)$, where $\nu(\lambda) = 3\epsilon$, $t(\lambda) < t'(\lambda)/448$ and $\kappa(\lambda) = \max(4/C, 448t(\lambda)/t'(\lambda))$. If $r, x \in [\tilde{s} \cdot 2^{40}]$ (it is so when the prover is honest), the protocol is honest verifier statistical zero-knowledge.

Proof. Completeness. If P knows $r \in [\tilde{A}]$ and $x \in \mathbf{Z}/q\mathbf{Z}$ s.t. $c_1 = g_q^r$, $c_2 = f^x \mathsf{pk}^r$ and $Q = xP$, and if both parties follow the protocol, one has $u_1 \in [\tilde{A}C(2^{40}+1)]$ and $u_2 \in \mathbf{Z}/q\mathbf{Z}$; $\mathsf{pk}^{u_1} f^{u_2} = \mathsf{pk}^{r_1 + k\cdot r} f^{r_2 + k \cdot x} = \mathsf{pk}^{r_1} f^{r_2} (\mathsf{pk}^r f^x)^k = t_2 \cdot (c_2)^k$; $u_2 \cdot P = (r_2 + k \cdot x)P = T + k \cdot Q$; and $g_q^{u_1} = g_q^{r_1 + k\cdot r} = t_1 \cdot (c_1)^k$.
Honest verifier zero-knowledge. Given pk, $\mathbf{c} = (c_1, c_2)$ and Q a simulator can sample $k \xleftarrow{\$} [C[$, $u_1 \xleftarrow{\$} [\tilde{A}C(2^{40}+1)]$ and $u_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, compute $t_1 := g_q^{u_1} \cdot (c_1)^{-k}$, $t_2 := \mathsf{pk}^{u_1} \cdot f^{u_2} \cdot (c_2)^{-k}$ and $T := u_2 \cdot P - k \cdot Q$ such that the transcript $(t_1, t_2, T, k, u_1, u_2)$ is indistinguishable from a transcript produced by a real execution of the protocol where V runs on input $(\mathsf{pk}, c_1, c_2, Q, P)$.
Computational soundness. Let us analyse for which knowledge error functions $\kappa()$ the protocol of Figure 4.6 satisfies the notion of soundness defined in Definition 1.5.4. Accordingly, let $\kappa()$ be any knowledge error function, such that $\kappa(\lambda) \geq 4/C$ for all λ. As in proof of Theorem 4.2.1, consider a malicious prover P^*. Since there are C different challenges, if $\epsilon_{\mathsf{view},P} > \kappa(\lambda) \geq 4/C$, one can obtain in expected PT a situation where, for given (t_1, t_2, T), P^* has correctly answered two different challenges k and k'. Let Rewind

Setup:

1. $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q)$.

2. Let $\tilde{A} := \tilde{s} \cdot 2^{40}$, sample $t \xleftarrow{\$} [\tilde{A}]$ and let $g_q := \hat{g}_q^t$.

Input : (r, x) and $(\mathsf{pk}, c_1, c_2, Q, P)$		Input : $(\mathsf{pk}, c_1, c_2, Q, P)$
$r_1 \xleftarrow{\$} [\tilde{A} \cdot C \cdot 2^{40}]$		Check that $\mathsf{pk}, c_1, c_2 \in \widehat{G}$
$r_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$t_1 := g_q^{r_1}$		
$t_2 := \mathsf{pk}^{r_1} f^{r_2}$		
$T := r_2 P$	$\xrightarrow{t_1, t_2, T}$	
	$\xleftarrow{k}$	$k \xleftarrow{\$} \mathcal{C}$
$u_1 := r_1 + k \cdot r \in \mathbf{Z}$		
$u_2 := r_2 + k \cdot x \in \mathbf{Z}/q\mathbf{Z}$	$\xrightarrow{u_1, u_2}$	`Check` $u_1 \in [\tilde{A}C(2^{40}+1)]$; $u_2 \in \mathbf{Z}/q\mathbf{Z}$
		and $g_q^{u_1} = t_1 \cdot (c_1)^k$
		and $T + k \cdot Q = u_2 \cdot P$
		and $\mathsf{pk}^{u_1} f^{u_2} = t_2 \cdot (c_2)^k$

Figure 4.6: Zero-knowledge argument of knowledge for $\tilde{\mathsf{R}}_{\mathsf{CL-DL}}$.

be a (probabilistic) procedure that creates $k, k', u_1, u_2, u_1', u_2'$ in this way. We call u_1, u_2 and u_1', u_2' the corresponding answers, so we get:

$$g_q^{u_1 - u_1'} = c_1^{k-k'}; \quad \mathsf{pk}^{u_1-u_1'} f^{u_2-u_2'} = c_2^{k-k'}; \quad \text{and} \quad (u_2 - u_2')P = (k - k')Q.$$

The rest of the proof is similar to the proof of Theorem. 4.2.1: from these equations one can extract a witness for $\tilde{\mathsf{R}}_{\mathsf{CL-DL}}$ otherwise one shows that there exists an algorithm that solves either the strong root problem for class groups or the low order problem.

□

Remark 18. In our two-party protocol ([CCL+19]), when proving the security of the overall two party ECDSA protocol, the simulator must simulate the above proof of knowledge without knowing (r, x), to a malicious adversary. Consequently to be used in our protocol, the above ZKAoK must be secure against malicious adversaries.

In order to attain security against malicious verifiers V^*, which may deviate from the protocol, a simulator simulating P chooses a random $k_P \in \mathcal{C}$, computes t_1, t_2 and T as in the proof of zero-knowledge against honest verifiers, and sends them to V^*, *hoping* that the challenge k_V chosen by V^* will be s.t. $k_V = k_P$. If so the simulated view of V^* is indistinguishable from V^*'s view in a real execution, if not S rewinds V^* and starts again until $k_V = k_P$. Consequently for the simulation to run in polynomial time we cannot chose $\mathcal{C}$ arbitrarily big. In practice one could take $\mathcal{C} := [2^{40}]$ and repeat the protocol $\lambda/40$ times to achieve a satisfying soundness error of $2^{-\lambda}$. We emphasise that this is still considerably better than the proof used in our two-party protocol for which $\mathcal{C} = \{0, 1\}$.

4.5 Efficiency comparisons

In this section, we analyse the theoretical complexity of our protocol by counting the number of exponentiations and communication of group elements. We compare the communication cost of our protocol to that of [GG18, LN18] for the standard NIST curves P-256, P-384 and P-521, corresponding to levels of security 128, 192 and 256. For the encryption scheme, we start with a 112 bit security, as in [GG18, LN18]' implementations, but also study the case where its level of security matches the security of the elliptic curves.

We chose to compare our work to best performing protocols using similar construction techniques (from homomorphic encryption) which achieve the same functionality, i.e. (t, n)-threshold ECDSA for any t s.t. $n \geq t + 1$. We do not provide a comparison to [DKLs18], [DKLs19] as they use OT which leads to protocols with a much higher communication cost. Similarly, and as noted in [DKO$^+$19] a direct comparison to [DKO$^+$19, SA19] is difficult as they rely on preprocessing to achieve efficient signing, which is a level of optimisation we have not considered. We don't compare to [GGN16, BGG17] as [GG18] is already faster and cheaper in terms of communication complexity.

The computed communication cost is for our protocol as described in Section 4.2, and as such is provably secure. Conversely the implementation which [GG18] provided omits a number of range proofs present in their described protocol. Though this substantially improves the efficiency of their scheme, they themselves note that removing these proofs creates an attack which leaks information on the secret signing key shared among the servers. They conjecture this information is limited enough for the protocol to remain secure, however since no formal analysis is performed, the resulting scheme is not proven secure. For a fair comparison we estimate the communication cost and timings of both their secure protocol and the stripped down version. In terms of bandwidth we outperform even their stripped down protocol.

In both protocols, when possible zero knowledge proofs are performed non interactively, replacing the challenge by a hash value, whose size depends on the security parameter λ. We note that our interactive set up for the CL encryption scheme uses a ZKPoK where challenges are of size 10bits (using the lcm trick), it must thus be repeated $\lambda/10$ times. We note however that the PoK of integer factorization used in the key generation of [GG18] has similar issues.

For non-malleable equivocable commitments, we use a cryptographic hash function H and define the commitment to x as $h = H(x, r)$, for a uniformly chosen r of length λ and assume that H behaves as a random oracle.

The comm. cost comparison is done by counting the number of bits that are both sent and received by a given party throughout the protocol[3]. In terms of timings, we count the number of exponentiations in the class group (for our protocol), the bit size of the exponent, and multiply this by 3/2 of the cost of a multiplication in the group. We compare this to an equivalent computation for [GG18], where we count exponentiations modulo N and N^2, the bit size of the exponent, and multiply this by 3/2 of the cost of a multiplication modulo N (resp. N^2). We do not count exponentiations and multiplications over the group of points of the elliptic curve as these are very cheap compared to the

[3]Broadcasting one element is counted as sending one element.

aforementioned computations, furthermore both protocols essentially perform identical operations on the curve.

The [LN18] protocol with Paillier encryption. We use the figures Lindell et al. provide in [LN18, Tab. 1] to compare our protocol to theirs. We note that – to their advantage – their key generation should include additional costs which are not counted in our figures (e.g. local Paillier key generation, verification of the ZKP of correctness of the Paillier key). The resulting costs are given in Tab.4.7a

The [GG18] protocol with Paillier encryption. The main cost in their key generation protocol is the ZKPoK of integer factorization, which is instantiated using [PS00, Thm. 8]. Precisely each prover commits to K values mod N, the challenge lives mod B, the final opening is an element of size A, where, as prescribed by Poupard and Stern, we take $\log(A) = \log(N)$, $\log(B) = \lambda$ and $K = \frac{\lambda+\log(|N|)}{log(C)}$ where $C := 2^{60}$ is chosen s.t. Floyd's cycle-finding algorithm is efficient in a space of size smaller than C. For their signature protocol, the cost of the ZK Proofs used in the MtA protocol are counted using [GG18, Appendix A].

The results are summarized in Fig. 4.7b. Since the range proofs (omitted in the stripped down version) only occur in the signing protocol, the timings and comm. cost of their interactive key generation is identical in both settings, we thus only provide these figures once. The comm. cost of each protocol is given in Bytes. The columns correspond to the elliptic curve used for EC-DSA, the security parameter λ in bits for the encryption scheme, the corresponding bit size of the modulus N, the timings of one Paillier exponentiation, of the key generation and of the signing phase and the total comm. in bytes for each interactive protocol. Modulus sizes are set according to the NIST recommendations.

Our protocol with the CL encryption scheme. For key generation we take into account the interactive key generation for the CL encryption scheme, which is done in parallel with $\mathsf{IKeyGen}$ s.t. the number of rounds of $\mathsf{IKeyGen}$ increases by only one broadcast per player. In $\mathsf{IKeyGen}$, each party performs 2 class group exponentiations of $\log(\tilde{s}) + 40$ bits (where $\tilde{s} \approx \sqrt{q \cdot \tilde{q}}$), to compute generators g_i and public keys pk_i, and $\lambda/10 \times n$ exponentiations of $\log(\tilde{s}) + 90$ bits for the proofs and checks in the ISetup sub-protocol.

Note that exponentiations in $\langle f \rangle$ are almost free. Signing uses $2 + 10t$ exponentiations of $\log(\tilde{s}) + 40$ bits (for computing ciphertexts and homomorphic operations), $2(t + 1)$ of $\log(\tilde{s}) + 80 + \lambda$ (for the ZKAoK) and $2t$ exponentiations of size q (for homomorphic scalar multiplication of ciphertexts).

The results for our protocols are summarized in Fig. 4.7c. The columns correspond to the elliptic curve used for EC-DSA, the security parameter λ in bits for the encryption scheme, the corresponding fundamental discriminant $\Delta_K = -q \cdot \tilde{q}$ bit size, the timings of one class group exponentiation (for an exponent of $\lambda + 40$ bits, i.e. that used for encryption), of the key generation and of the signing phase and the total comm. in bytes for $\mathsf{IKeyGen}$ and ISign. The discriminant sizes are chosen according to [BJS10].

Curve	λ (bits)	N (bits)	Mult. (ms)	IKeyGen (ms)	ISign (ms)	IKeyGen (Bytes)	ISign (Bytes)
P-256	112	2048	0.0023	$> 52n + 52$	$99t$	$> 6\,336(n-1)$	$16\,064t$
P-256	128	3072	0.0048	$> 162n + 162$	$310t$	$> 9\,152(n-1)$	$22\,208t$
P-384	192	7680	0.0186	$> 1\,571n + 1571$	$3\,000t$	$> 22\,176(n-1)$	$51\,744t$
P-521	256	15360	0.0519	$> 8\,769n + 8\,769$	$16\,741t$	$> 43\,672(n-1)$	$99\,845t$

(a) [LN18]'s secure t out of n protocol.

				Provably secure (with range proofs)				Stripped down	
Curve	λ (bits)	N (bits)	Mult. (ms)	IKeyGen (ms)	ISign (ms)	IKeyGen (Bytes)	ISign (Bytes)	ISign (ms)	ISign (Bytes)
P-256	112	2048	0.0023	$64n + 7$	$140t$	$32(n+t) + 9\,990n - 64$	$23\,308t + 588$	$28t$	$4\,932t + 588$
P-256	128	3072	0.0048	$293n + 22$	$428t$	$32(n+t) + 21\,392n - 64$	$33\,568t + 608$	$88t$	$7\,008t + 608$
P-384	192	7680	0.0186	$7\,017n + 214$	$4\,071t$	$48(n+t) + 128088n - 96$	$81\,072t + 912$	$857t$	$16\,656t + 912$
P-521	256	15360	0.0519	$77\,725n + 1196$	$22\,528t$	$65(n+t) + 503\,591n - 130$	$159\,391t + 1\,232$	$4783t$	$32\,470t + 1\,231$

(b) [GG18]'s t out of n protocol.

Curve	λ (bits)	Δ_K (bits)	Mult. (ms)	IKeyGen (ms)	ISign (ms)	IKeyGen (Bytes)	ISign (Bytes)
P-256	112	1348	0.029	$366n + 62$	$430t + 137$	$\mathbf{32(n+t) + 2951n - 64}$	$\mathbf{3\,670t + 1\,747}$
P-256	128	1827	0.038	$744n + 109$	$730t + 237$	$\mathbf{32(n+t) + 4\,297n - 64}$	$\mathbf{4\,455t + 2\,052}$
P-384	192	3598	0.077	$4\,145n + 424$	$\mathbf{2\,780t + 903}$	$\mathbf{48(n+t) + 10\,851n - 96}$	$\mathbf{8\,022t + 3\,560}$
P-521	256	5971	0.137	$16\,432n + 1\,243$	$\mathbf{8\,011t + 2,608}$	$\mathbf{65(n+t) + 22\,942n - 130}$	$\mathbf{12\,576t + 5\,433}$

(c) Our secure t out of n protocol – With an interactive CL setup.

Figure 4.7: Comparative sizes (in bits), timings (in ms) & comm. cost (in Bytes)

Rounds. In terms of the number of rounds, we perform identically to [LN18]. Our IKeyGen requires 5 rounds (only 4 assuming a standardised set up), compared to 4 in [GG18]. Our signing protocol requires 8 rounds as opposed to 9 in [GG18].

Comparison. Fig. 4.7 shows that the protocols of [LN18, GG18] are faster for both key generation and signing for standard security levels for the encryption scheme (112 and 128 bits of security) while our solution remains of the same order of magnitude. However for high security levels, our signing protocol is fastest from a 192-bits security level.

In terms of communications, our solution outperforms the other two protocols for all levels of security, factors vary according to the number of users n and the desired threshold t. In terms of rounds, both our protocols use the same number of rounds as Lindell's. For key generation we use one more than [GG18], whereas for signing we use one less.

This situation can be explained by the following facts. Firstly with class groups of quadratic fields we can use lower parameters than with $\mathbf{Z}/n\mathbf{Z}$ (as said in Subsubsection 2.2.4.2, the best algorithm against the discrete logarithm problem in class groups has complexity $\mathcal{O}(L[1/2, o(1)])$ compared to an $\mathcal{O}(L[1/3, o(1)])$ for factoring). However, the group law is more complex in class groups, indeed exponentiations in class groups are cheaper than those modulo N^2 from the 192 bits level. So even if removing range proofs allows us to drastically reduce the number of exponentiations, our solution only takes less time from that level (while being of the same order of magnitude below this level).

We note that assuming a standardized set up for CL (as mentioned in Section 4.4.2), one would reduce the bandwidth consumption of IKeyGen by a factor varying from 6 to 16 (for increasing levels of security). Moreover in terms of timings, the only exponentiation in the class group would be each party computing its own ciphertext, and so the only operations linear in the number of users n would be on the curve (or integers modulo q), which are extremely efficient.

Chapter 5

Improvements on Threshold ECDSA

In Chapter 4 we proposed an efficient instantiation based on class groups of a full $(t,n)-$threshold ECDSA protocol involving n players and with any threshold $t \leq n-1$. Recently, Canetti et al. [CGG$^+$20] departs from previous threshold realizations in that it offers several additional features. They propose a new threshold ECDSA scheme which is proven secure against adaptive adversaries in the UC model, which achieves accountability (if some parties causes an abort during the generation of the signature, at least one misbehaving player is recognized) and uses an interesting online/offline optimization which allows to perform the heaviest part of the computation in a pre-signing step, i.e. before the message to be signed is known. Furthermore, their protocol allows for non interactive online signing (assuming the offline preprocessing) and it is proactive. Compared to our proposal in [CCL$^+$20], they are considerably heavier, especially in terms of overall communication cost. In practice, looking at the bandwidth consumption, having a lighter communication cost when coping with more parties is preferable. Improving our protocol from Chapter 4 with the new features discussed above brings to a new enhanced scheme that at the same time inherits part of the bandwidth efficiency from our threshold ECDSA from Chapter 4.
In this chapter we present and discuss this improved protocol, i.e. our work [CCL$^+$21]. In that work, we departs from the construction in Chapter 4, and we present a new construction which satisfy the additional properties just defined and which is proven secure against a static adversary. Finally, we prove security against adaptive adversary for the specific case $t = n - 1$. The motivation behind this choice and more details about the properties we want to satisfy are given in paragraphs below. In the technical part, we omit the details about the construction of our basic threshold construction from [CCL$^+$20] since already explained in Chapter 4 and we refer to that chapter.

Online/Offline Before [CGG$^+$20] was public, the authors published their result in two separate works ([GG20],[CMP20]), then they proposed a unique result. Our construction follows some of the ideas initially proposed in [GG20] as the online/offline and the accountability properties and we adapted them to our threshold construction. In a similar fashion, we can split our protocol in a offline phase and in a online one. Informally, shares of k^{-1} and of kx are computed by players in the offline phase. Once the message m is available and known by signers, they can compute their shares of signature $s_i = k_iH(m) + r(kx)_i$ locally from public known information, i.e. non interactively, and finally they broadcast their own s_i. Having received the shares, they can compute the signature of m. The online phase consists only in broadcasting s_i, which is an improvement on our basic threshold scheme since it avoids its long interactive signing. Another substantial difference with our basic threshold protocol is in the role of the nonce R. Indeed, since R is made public before the message is chosen, to guarantee the security of the scheme we need a modification of the definition of security of ECDSA which takes in account this fact. This modified ECDSA assumption is called *enhanced existential unforgeability* ECDSA, it is presented in Definition 1.3.4 and it is necessary in online/offline setting. This definition was independently proposed by the authors of [CGG$^+$20], and considered in that work[1].

Identifying aborts Our improved protocol from [CCL$^+$21] is proven secure against a corruption of a majority of the players and also relies on aborts. This means that the protocol ends prematurely if a malicious behaviour of some player is detected. Such protocols are susceptible to "denial of service" attacks, allowing even a single malicious party to force the protocol to abort. An additional part of our protocol is an identification sub-protocol which is run after an abort occurs to identify at least one of the misbehaving parties which caused the abort. Anyway, it can result in a challenging task, and as a result it is seems to be not possible to exclude any cheater from future computation of the signature. Looking at public operations done by a single party, e.g. proving in zero-knowledge some calculation to everyone, it could be easy in some cases to detect a cheater if a proof fails. However, looking at previous solutions in [GG18] or in [CCL$^+$20], in those protocols some specific operations done by parties are not publicly verifiable, as for example the computation of a common R. In cases where it is not possible to publicly verify operations which brought to abort, it is quite hard to detect a cheater since there are not enough information about who did not follow instructions. An easy way to identify players that behave maliciously would be to force them to prove in zero knowledge that they followed the protocol correctly. A negative consequence of this solution is that it typically induces significant communication overhead, thus making the resulting protocol impractical. One of the improvements present in [CGG$^+$20] and in our enhanced protocol, is the ability of identifying cheaters and then excluding them from future executions. [GG20] follows a different approach specifically tailored to their online/offline solution. In strategy followed by [GG20] it is possble to detect misbehaving parties checking shares of the signature from public values for the online phase, and revealing some private values in the offline phase observing that there is not a security issue since the signature is not yet generated. In our setting a technical problem prevents the resulting proof to go through. Informally this has to do with the fact that the security of our protocol crucially relies on some properties of hash proof systems.[2] The technical complication in the proof arises when the simulator needs to switch from valid ciphertexts to invalid ones (see Subsection 3.2.4 for the definition of invalid ciphertext) that perfectly hide the underlying plaintext. This step is crucial for the proof in [CCL$^+$20] to go through

[1] Actually, the definition comes from the two independent works [GG20],[CMP20]. The authors of both papers then decided to present a unique work, which is [CGG$^+$20].

[2] The connection with hash proof systems comes from the fact that the underlying linearly homomorphic encryption in [CCL$^+$20] scheme is the one resulting from hash proof systems when using CL as underlying building block. We saw this link in Section 3.4.

but becomes problematic here as the simulator cannot provide the randomness used to create the ciphertexts, as no valid ciphertexts exist anymore. Indeed, an invalid ciphertext of the CL encryption scheme has the form (u, e) where $u \in G \setminus G^q$. Then, $u = g_q^{\alpha} f^{\beta}$ and it does not exist an integer r such that $u = g_q^r$ (if such r exists, then $G^q \cap F \neq \{1\}$, which is absurd by construction). Another relevant point is that revealing the randomness used and the message permits to verify the decryption of a ciphertext by re-encryption. If the ciphertext is a fresh one, we incur in the simulation problem above, then suppose the randomness is not known as when the ciphertext is computed from homomorphic operations. With CL we cannot extract the randomness used even knowing the secret key. The reason is that in the class group the order of the group is unknown even to who set the parameters, while in Paillier encryption it can be extracted using the secret key. To overcome this problem, we designed a new zero-knowledge proofs that manage to let the simulator complete the proof without compromising the overall efficiency of the protocol. The new zero-knowledge proves that a ciphertext decrypts to some value or not, without the knowledge of randomness.
This stronger model is referred to as secure multi-party computation with *identifiable aborts* (ID-MPC) [IOZ14, CL14]. It ensures that all honest parties learn the identity culprit if some party causes an abort.

Proactivity Our basic protocol can also easily be modified to achieve proactive security. Informally, proactivity is the property to resist against an attack which can compromise all the parties, corrupting part of them at any time period. It is usual to consider adversaries that can corrupt at most t parties during the entire lifetime of the execution, but this is not considered too realistic when a protocol is run for a long time. The idea of proactivization is born with the introduction of *mobile adversary* in [OY91]. A mobile adversary represents a more realistic situation where the adversary $\mathcal{A}$ is able to corrupt at most $t < n$ new parties during a time period called *epoch*, where n is the total number of parties. An epoch is defined as the time period between the beginning of a key refreshing and the end of the next key refreshing. The security of a proactive threshold signature scheme is based on the security of the scheme against at most t corruption during an epoch. Our key refreshing protocol has the advantage to be very efficient, much more efficient than the corresponding protocol from [CGG$^+$20], when considering a number of players one would expect in real applications. protocol. To understand why ours has better efficiency, it is enough to see that the role of a key refreshing protocol is to refresh the used keys, where these key come from two different settings (CL for us, Paillier for [CGG$^+$20]). When using Paillier encryption scheme, parties should generate an (RSA) modulus N together with a proof that it has been constructed correctly, where this proof is known to be very expensive. In our case it is enough to generate a couple of the form $(h_i = g^{\mathsf{sk}})$ where g is a public, fixed parameter. As we saw in previous chapter, we do not prove that the key is constructed correctly. In a concrete comparison, e.g. for $n = 5$ and a 112-bit level of security, the total data sent and received between players in our key refresh protocol is *15 times less* than theirs (28 KBytes as opposed to 420 KBytes). Our solutions reduce the bandwidth consumption of [CGG$^+$20] by up to a factor 10. We compare the communication cost of our signature protocol to those of Canetti et al. for the standard NIST curve P-256 corresponding to a 128 bit security level. Regarding encryption, we start with a 112 bit security, as in their implementations, but also study the case where its level of security matches that of the elliptic curve. In both cases, our comparison shows that our signing protocol is an order of magnitude more efficient. More details about the efficiency are given in the final section of this chapters.

A $(n-1, n)-$threshold adaptive secure proposal Our most basic construction works for any $t < n$ and it is proved secure against static adversaries. Next, we prove security of our protocol against adaptive corruptions for the special case when $t = n-1$. That choice of $t = n-1$ is motivated by the difficulties to cope with an adaptive adversary, which are argued in Subsection 5.3.2. However, even the whole work of Canetti *et al.* focuses on the n-out-of-n[3] case and does not explicitly consider more general $(t, n)-$thresholds. In [CGG+20], the authors suggest to apply the tecniques of [GG18] to extend their protocol to the general $t \leq n-1$ case, however in Subsection 5.3.2 we look at difficulties in proving our protocol secure against an adaptive adversary for $t < n$. How to extend to more general threshold $t < n$ is left as a future work and then not considered in this thesis document. In brief, both the protocol achieve security against adaptive adversary for the case $t = n-1$, i.e. when the adversary has the power to corrupt all the parties except one at each time period.

Adversary model. The kind of adversary we consider is active (or malicious) and it is computationally bounded. Furthermore, we consider two settings, static and adaptive. For an informal description, see Section 1.1.1.

Communication model As for [CCL+20] and [GG18], we consider a broadcast channel and peer-to-peer channels between parties.

On composability. Our results are proven secure in the Random Oracle Model (see Section 1.5). We implicitly use the Fiat-Shamir transform to convert our honest-verifier sigma protocols into extractable non-interactive proofs. Extractability in the Fiat-Shamir paradigm relies on rewinding, which in the UC setting results in the running time of the simulator blowing up exponentially. One way to avoid the blow-up while allowing for extractability is to run proofs interactively, in which case they must be secure against malicious verifiers. Such an approach would lead to a large increase in the number of rounds of our protocol. Hence, motivated by efficiency, the definitions we adopt are game-based and do not guarantee composability.

Building from class groups Because of our new threshold proposal is an extension of the previous one from Chapter 4, the choice of the parameters of CL are essentially the same. We then refer to Subsection 4.1.1 for the setup of CL

Remark 19. Dobson *et al.* [DGS20] suggested a new formula to estimate security in a setting where a large set of users (say a billion) share common parameters. In such a setting, public parameters may be targeted by an adversary. They suggest that in this context, an attack with 2^{λ} running time and a $2^{-\lambda}$ success probability equates to λ bits of security. This leads to estimates on sizes of the discriminant which are much larger than

[3]It is usual in literature to call n-out-of-n the case where the threshold is $t = n-1$. However, we use the term $(n-1, n)-$threshold to refer to this specific case.

traditionally used. However, this formula is not relevant for our application where there won't be billions of users. Here the standard definition (2^λ running time for a probability of success 1/2 or close to 1) seems more appropriate. As a side note, these estimates in the 'large set of users' context also apply to applications based on an RSA modulus N, as there exists an algorithm which efficiently factors N given the class number of discriminant $-N$ or $-4N$.

Hard assumptions No new assumptions with regards to the construction in Chapter 4 are required. We refer to Section 4.1.2 for the low order assumption and the strong root assumption.

Remark 20. Belabas *et al.* in [BKSW20] show that if the discriminant belongs to some class of weak primes, then computing small order elements is easy. They also demonstrate that one can easily construct discriminants together with a low order element in their class group without computing the class number. However the likelihood that such a discriminant is chosen at random is negligible. On the other hand, given a discriminant, it seems hard to prove that it is not of such a weak form. Though in our case the discriminant is not prime ($\Delta_K = -q\tilde{q}$), their ideas can be extended to this setting. Hence when relying on the low order assumption, particular attention must be paid in ensuring that discriminants are generated randomly (in our case $\tilde{q}$ must be so), and in particular that special primes are not chosen to meet specific optimization requirements.

Differences with the eprint version. The relevant differences between this chapter and the eprint version ([CCL$^+$21]) are:

- A little change in ISetup protocol which splits the ZK proofs of the knowledge of the discrete logarithm in the curve and in the class group in two separate proofs (it is only a little improvement in efficiency).
- A new proof for a slightly different $\mathsf{R}_{\mathsf{Dec}}$ similar to the $\mathsf{R}_{\mathsf{Enc}}$ relation from Subsection 4.2.1 which takes in account the difficulty of extracting an integer value of the randomness in a ciphertext. Section 5.2.1 is dedicated to the new $\mathsf{R}_{\mathsf{Dec}}$.
- Two small additional notes: *On the impossibility of re-encryption* at the end of Section 5.2 and *On the number of rewindings* in Subsection 5.3.2.

5.1 Improved Threshold ECDSA protocol

We here present our $(t, n)-$threshold ECDSA protocol with the addtional properties. It is divided into five sub-protocols for ease of readability. The first sets up public parameters for the CL encryption scheme, and a common random elliptic curve point $H \in \mathbb{G}$. We then present the key generation sub-protocol for IKeyGen and a key refresh sub-protocol IKeyRefresh. Next we provide our protocol for signing, this is divided into two sub-protocols: an offline protocol PreSign allowing to pre-compute a number of *pre-signatures* before knowing the message to be signed, and an online protocol ISign which takes as input shares of a pre-signature and a message, and outputs a signature. The protocol is similar to that one in Chapter 4, except for the key refresh subprotol which is new, and some additional steps required to identifying parties which cause an abort. Furthermore,

$$\begin{array}{ccc}
P_i & \mathsf{ISetup}(k, \mathsf{pp}_{\mathbb{G}}) & \text{All players } \{P_j\}_{j \neq i} \\
\hline
r_i \xleftarrow{\$} \{0,1\}^k & & \\
[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(r_i) & \xRightarrow{\mathsf{c}_i} & \\
 & \xRightarrow{\mathsf{d}_i} & r_i \leftarrow \mathsf{Open}(\mathsf{c}_i, \mathsf{d}_i) \\
\tilde{q} := \mathsf{next\text{-}prime}(\bigoplus_{j=1}^{n} r_j) & & \\
\hline
\text{Compute } \hat{g}_q \text{ from } q, \tilde{q} & & \\
t_i \xleftarrow{\$} [2^{40}\tilde{s}] \text{ and } g_i \leftarrow \hat{g}_q^{t_i} & & \\
h_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z} \text{ and } H_i \leftarrow h_i \cdot P & & \\
(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i) \leftarrow \mathsf{Com}((g_i, H_i)) & \xRightarrow{\tilde{\mathsf{c}}_i} & \\
 & \xRightarrow{\tilde{\mathsf{d}}_i} & (g_i, H_i) \leftarrow \mathsf{Open}(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i) \\
\pi_i^{\langle \hat{g}_q \rangle - \mathsf{DL}} := \mathsf{ZKPoK}_{g_i}\{t_i : g_i = \hat{g}_q^{t_i}\} & & \\
\pi_i^{\mathsf{DL}} := \mathsf{ZKPoK}_{H_i}\{h_i : H_i = h_i \cdot P\} & \xRightarrow{\pi_i^{\mathsf{DL}}, \tilde{\pi}_i} & \text{if a proof fails } \mathsf{abort} \\
g_q \leftarrow \prod_{j=1}^{n} \hat{g}_q^{t_j} = \prod_{j=1}^{n} g_j & & \\
\text{Erase all data other than } (g_q, H, \mathsf{pp}_{\mathbb{G}}) & &
\end{array}$$

Figure 5.1: Threshold CL setup used in IKeyGen

unlike the last Phase of the signature protocol of [CCL$^+$20] in Section 4.2 (Figures 4.2, 4.3 and 4.4), the computation of the signature from shares consists only in one step, because of preprocessed values and checks that make sure the operations done before putting together the shares of the signature were correct. This makes unncecessary to run the zero-knowledge proofs of Phase 5 (see Figure 4.4). The similarity between protocols in [CCL$^+$20] and in [CCL$^+$21] – which is the topic of this Chapter – is useful to understand the ideas behind the new protocol, and we refer to the detailed explanation of threshold protocol in Chapter 4. The procedure for identifying misbehaving players is given in Section 5.2.

Interactive Set Up Sub-Protocol. Our protocol requires a set up, as did that of [CCL$^+$20] in Chapter 4, to ensure that the generator g_q used by all parties in the CL encryption scheme is a *random* generator. We already discussed the necessity of a random generator in Subsection 4.1.1. Our ISetup protocol is that of [CCL$^+$20], with the slight difference that parties also set up a random elliptic curve point H which will be used in the pre-signing protocol. As usual, we denote $\mathsf{pp}_{\mathbb{G}} := (\mathbb{G}, P, q)$ the description of the elliptic curve used in ECDSA. For n parties to collaboratively run ISetup, they proceed as depicted in Figure 5.1.

Key Generation Sub-Protocol. After running the ISetup protocol of Figure 5.1, all parties possess $(g_q, H, \mathsf{pp}_{\mathbb{G}})$. All parties use this as input for the interactive key generation protocol IKeyGen. Note that in practice ISetup and IKeyGen would be ran in parallel, and are only here presented separately for ease of readability. As it is exactly the IKeyGen protocol of [CCL$^+$20], we do not detail the steps of the sub-protocol. Its description is given in Figure 4.3 in Chapter 4, we recall it in Figure 5.2 only for ease of readability.

P_i	$\mathsf{IKeyGen}(g_q, H, \mathsf{pp}_{\mathbb{G}})$	All players $\{P_j\}_{j \neq i}$
$u_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and $Q_i := u_i P$		
$[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(Q_i)$		
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}(1^\lambda)$	$\xRightarrow{\mathsf{pk}_i \text{ and } \mathsf{kgc}_i}$	
	$\xRightarrow{\mathsf{kgd}_i}$	$Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$
Perform (t, n)-VSS share of u_i:		$Q := \sum_{i=1}^n Q_i$
$p_i(X) = u_i + \sum_{k=1}^t a_{i,k} X^k \bmod q$		
Let $\{\sigma_{i,j} := p_i(j)\}_{j \in [n]}$ and $\{V_{i,k} := a_{i,k} P\}_{k \in [t]}$	$\xrightarrow{\text{Send } \sigma_{i,j} \text{ to } P_j}$	
	$\xRightarrow{\{V_{i,k}\}_{k \in [t]}}$	
$\{\sigma_{k,i}\}_k$ are additive shares of $x_i := \sum_{k \in [n]} p_k(i)$		
where $\{x_i\}_{i \in [n]}$ are (t, n) Shamir shares of x.		
$\pi_i^{\mathsf{kg}} := \mathsf{ZKPoK}_{X_i}\{(x_i) : X_i = x_i P\}$	$\xleftrightarrow{\pi_i^{\mathsf{kg}}}$	
Store (Q, u_i, x_i)		

Figure 5.2: Threshold Key Generation

Notice that in this work, as already said we use Fiat-Shamir transform for proofs, then π_i^{kg} is assumed non interactive.

Key Refresh Sub-Protocol. This protocol allows players to generate new shares of the ECDSA secret signing key x and public verification key Q. Each party P_i for $i \in [n]$ runs on input it's previous ECDSA key shares (u_i, Q_i) satisfying $Q_i = u_i P$; the public parameters $\mathsf{pp}_{\mathbb{G}} = (\mathbb{G}, P, q)$; the verification key Q; and the public parameters $\mathsf{pp}_{\mathsf{CL}}$ for the CL encryption scheme.

Note that upon key refresh all pre-signatures computed in the previous epoch are erased. This is crucial to ensure that – in our security proof – an adversary can not obtain signatures on two different messages for the same randomness R.

Furthermore, after each execution of Key Refresh, note that the shares u_i^{new} are $(n-1, n)-$additive shares of x. These can be converted into $(t, n)-$shares x_i of x via. a VSS as in $\mathsf{IKeyGen}$.

Offline Pre-Signature Sub-Protocol. We now present the Pre-Signing sub-protocol which pre-processes signatures before the messages are known. For $i \in [n]$, let x_i denote P_i's $(t, n)-$share of x output by the $\mathsf{IKeyGen}$ sub-protocol of Figure 5.2 (or the output of the latest key refresh, *cf.* Figure 5.3); and let $X_i := x_i P$. For each execution a set S of players (satisfying $|S| > t$) is chosen and the secret values $\{w_i\}_{i \in S}$ constitute a $(t, t)-$additive secret sharing of the secret signing key x. Each w_i is computed from x_i using Lagrangian coefficients. Furthermore the associated elliptic curve point $W_i := w_i P$ is known to all parties (as W_i can be computed from X_i). The offline Pre-Signing sub-protocol is depicted in Figure 5.4.

As in [CCL+20], Phase 2 of the Pre-Signing protocol is a peer-to-peer sub-protocol between each pair of players P_i and P_j, for $i, j \in S, j \neq i$. For private shares $\gamma_i, w_i \in \mathbf{Z}/q\mathbf{Z}$ owned by P_i and $k_j \in \mathbf{Z}/q\mathbf{Z}$ owned by P_j, it allows to convert multiplicative shares $k_j \gamma_i$

$P_i(u_i, Q_i, Q, \mathsf{pp}_{\mathbb{G}}, \mathsf{pp}_{\mathsf{CL}})$	Key Refresh	All players
$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}(1^\lambda)$	$\xrightarrow{\mathsf{pk}_i}$	
$v_{i,1}, \ldots, v_{i,n} \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ s.t $\sum_j v_{i,j} = 0$		
For $j \in [n]$ let $Q_{i,j} := v_{i,j} \cdot P$		
$\mathbf{Y}_i = \{Q_{i,j}\}_{j\in[n]}$		
Sample $\rho_j \overset{\$}{\leftarrow} [\tilde{A}]$		
$C_{i,j} := \mathsf{Enc}(\mathsf{pk}_j, v_{i,j}; \rho_j)$	$\xrightarrow{\{C_{i,j}\}_{j\in[n]}, \mathbf{Y}_i}$	if $\sum_{j\in[n]} Q_{i,j} \neq 0_{\mathbb{G}}$ then abort
$\pi^{\mathsf{kr}}_{i,j} := \mathsf{ZKAoK}_{C_{i,j}, Q_{i,j}, P}\{(v_{i,j}, \rho_j) :$		
$C_{i,j} = \mathsf{Enc}(\mathsf{pk}_j, v_{i,j}; \rho_j) \wedge Q_{i,j} = v_{i,j} \cdot P\}$	$\xrightarrow{\{\pi^{\mathsf{kr}}_{i,j}\}_{j\in[n]}}$	if a proof fails abort
Overwrite old shares with:		
$u_i^{\mathsf{new}} := u_i + \sum_{j\in[n]} \mathsf{Dec}(\mathsf{sk}_i, C_{j,i}) \mod q$		
$Q_i^{\mathsf{new}} := Q_i + \sum_{j\in[n]} Q_{j,i}$		
Erase previously computed pre-signatures		
and all Key Refresh data except $u_i^{\mathsf{new}}, Q, \{\mathsf{pk}_j\}_{j\in[n]}, \mathsf{sk}_i$		

Figure 5.3: Key Refresh

and $k_j w_i$ into additive shares $\alpha_{j,i}, \beta_{j,i}, \mu_{j,i}, \nu_{j,i} \in \mathbf{Z}/q\mathbf{Z}$ satisfying $\alpha_{j,i} + \beta_{j,i} = k_j\gamma_i \mod q$ and $\mu_{j,i} + \nu_{j,i} = k_j w_i \mod q$. Since a check on values $\mu_{j,i}, \nu_{j,i}$ ensures they are consistent with P_i's secret key share w_i, the sub-protocol computing $\mu_{j,i}, \nu_{j,i}$ is referred to as MtAwc (Multiplicative to Additive with check), whereas that computing $\alpha_{j,i}, \beta_{j,i}$ is referred to as MtA.

Online Signature Sub-Protocol. Our sub-protocol computing signature-shares once the message is known is depicted in Figure 5.5. This sub-protocol is executed between the same set S of players that interacted in the Pre-Signing sub-protocol of Figure 5.4, all running on input a message m to be signed and a precomputed pre-signature.

5.2 Identifying Aborts

In this section we show how to identify at least one misbehaving player if an abort occurs during an execution of the protocol. We follow a similar idea to that in [CGG+20] and in [CCL+20], adapting their techniques to take into account the fact we use a class group based encryption scheme. Indeed in some specific cases, to identify aborts, we require that parties prove that a ciphertext decrypts to a given value using their decryption key. To ensure this, a player must prove that the ciphertext it first sent decrypts to some message or to nothing (this happens if the ciphertext is not well-formed). To this end, we provide an efficient zero-knowledge argument of knowledge of some information related to the secret key used to decrypt a CL ciphertext. The proof we provide is specific to our considered encryption scheme, precisely, it is for the following relation:

$$\mathsf{R}_{\mathsf{Dec}} := \{(\mathsf{pk}, (c_1, c_2), M); (\sigma_0, \sigma_1); \mid c_1, c_2, M \in \widehat{G};\ \sigma_0 \in \mathbf{N},$$

PreSign

$P_i(w_i, Q, \mathsf{pp}_{\mathbb{G}}, \mathsf{pp}_{\mathsf{CL}}, \mathsf{sk}_i, \{\mathsf{pk}_j\}_{j\in S})$	**Phase 1**	All players $\{P_j\}_{j\neq i}$
$r_i \xleftarrow{\$} [\tilde{A}]$		
$k_i, \gamma_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$c_{k_i} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, k_i; r_i)$		
$[\mathsf{c}_i, \mathsf{d}_i] \leftarrow \mathsf{Com}(\gamma_i P)$	$\xRightarrow{\mathsf{c}_i, c_{k_i}}$	
$\pi_i := \mathsf{ZKAoK}_{\mathsf{pk}_i, c_{k_i}}\{(k_i, r_i) :$		
$((\mathsf{pk}_i, c_{k_i}); (k_i, r_i)) \in \mathsf{R}_{\mathsf{Enc}}\}$	$\xleftrightarrow{\pi_i}$	if a proof fails, abort
P_i	**Phase 2**	P_j
$\beta_{j,i}, \nu_{j,i} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$B_{j,i} := \nu_{j,i} \cdot P$		
$c_{\beta_{j,i}} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, -\beta_{j,i})$		
$c_{\nu_{j,i}} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, -\nu_{j,i})$		
$c_{k_j\gamma_i} \leftarrow \mathsf{EvalAdd}(\mathsf{EvalScal}(c_{k_j}, \gamma_i), c_{\beta_{j,i}})$		
$c_{k_j w_i} \leftarrow \mathsf{EvalAdd}(\mathsf{EvalScal}(c_{k_j}, w_i), c_{\nu_{j,i}})$	$\xrightarrow{c_{k_j\gamma_i}, c_{k_j w_i}, B_{j,i}}$	$\alpha_{j,i} \leftarrow \mathsf{Dec}(\mathsf{sk}_j, c_{k_j\gamma_i})$
		$\mu_{j,i} \leftarrow \mathsf{Dec}(\mathsf{sk}_j, c_{k_j w_i})$
		If $\mu_{j,i} \cdot P + B_{j,i} \neq k_j \cdot W_i$ then abort
$\delta_i := k_i\gamma_i + \sum_{j\neq i}(\alpha_{i,j} + \beta_{j,i})$		
$\sigma_i := k_i w_i + \sum_{j\neq i}(\mu_{i,j} + \nu_{j,i})$		
P_i	**Phase 3**	All players $\{P_j\}_{j\neq i}$
	$\xRightarrow{\delta_i}$	$\delta = \sum_{i\in S}\delta_i = k\gamma$
$\ell_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$T_i = \sigma_i \cdot P + \ell_i \cdot H$	$\xRightarrow{T_i}$	
$\widetilde{\pi}_i := \mathsf{ZKPoK}_{T_i}\{(\sigma_i, \ell_i) : T_i = \sigma_i \cdot P + \ell_i \cdot H \in \mathbb{G}\}$	$\xleftrightarrow{\widetilde{\pi}_i}$	if a proof fails, abort
P_i	**Phase 4**	All players $\{P_j\}_{j\neq i}$
	$\xRightarrow{\mathsf{d}_i}$	$\Gamma_i := \mathsf{Open}(\mathsf{c}_i, \mathsf{d}_i)$
$\pi_i^\gamma = \mathsf{ZKPoK}_{\Gamma_i}\{\gamma_i : \Gamma_i = \gamma_i \cdot P\}$	$\xleftrightarrow{\pi_i^\gamma}$	if a proof fails, abort
		$R := \delta^{-1}(\sum_{i\in S}\Gamma_i)$
		Let $R = (r_x, r_y)$ and $r := r_x \bmod q$
P_i	**Phase 5**	All players $\{P_j\}_{j\neq i}$
$\bar{R}_i = k_i \cdot R$	$\xRightarrow{\bar{R}_i}$	
$\pi_i' = \mathsf{ZKPoK}_{\mathsf{pk}_i, c_{k_i}, \bar{R}_i, R}\Big\{(k_i, r_i) :$		
$c_{k_i} = \mathsf{Enc}(\mathsf{pk}_i, k_i; r_i) \wedge \bar{R}_i = k_i \cdot R\Big\}$	$\xleftrightarrow{\pi_i'}$	if a proof fails, abort
		if $P \neq \sum_{i\in S}\bar{R}_i$ abort
Erase all data except for (ℓ_i, k_i, σ_i) and (Q, u_i, x_i)		
P_i	**Phase 6**	All players $\{P_j\}_{j\neq i}$
$S_i = \sigma_i \cdot R$		
$\pi_i'' = \mathsf{ZKAoK}_{S_i, T_i, R}\{(\sigma_i, \ell_i) : T_i = \sigma_i \cdot P + \ell_i \cdot H \wedge S_i = \sigma_i \cdot R\}$	$\xRightarrow{S_i}$	
	$\xleftrightarrow{\pi_i''}$	if a proof fails, abort
		if $Q \neq \sum_{i\in S} S_i$ abort
Erase all data except for: (Q, u_i, x_i) and pre-signature share (R, k_i, σ_i)		

Figure 5.4: Offline Threshold Pre-Signature Protocol

ISign

P_i	**Phase 7**	All players $\{P_j\}_{j \neq i}$
$s_i := mk_i + r\sigma_i$	$\xRightarrow{s_i}$	
		$s := \sum_{i \in S} s_i$,
Erase (R, k_i, σ_i)		if (r, s) is not a valid signature, abort, else return (r, s).

Figure 5.5: Online Threshold Signature Protocol

$$\sigma_1 \in [-\tilde{A}C(2^{40}+1)\cdot 2^{\sigma_0}, \tilde{A}C(2^{40}+1)\cdot 2^{\sigma_0}];\ c_2 \cdot M^{-1} = c_1^{2^{-\sigma_0}\sigma_1} \wedge \mathsf{pk} = g_q^{2^{-\sigma_0}\sigma_1}\}.$$

To understand why such a proof is necessary, let us compare Paillier and CL decryptions. With Paillier's cryptosystem, one can extract the encryption randomness from a ciphertext given the decryption key. So if an abort occurs in the protocol, players can publish both the plaintext and encryption randomness underlying ciphertexts which were encrypted using their public key. They can thereby convince other players that the ciphertext is an encryption of the announced plaintext by re-encryption. However, due of the unknown order of G^q, in CL it is not possible to efficiently compute the encryption randomness, even knowing the decryption key.
Consequently parties must prove ciphertexts decrypt to a given value (this may be $\perp$ if decryption fails) in another way.

5.2.1 Arguing knowledge of a decrypted message

In this subsection we present a proof for the relation $\mathsf{R}_{\mathsf{Dec}}$. The reason why we need such a proof is motivated by the fact that some ciphertexts in the resulting signature scheme are not fresh ciphertexts, but they are obtained from homomorphic operations. If a party P encrypts a message m in a ciphertext c, it is clear that P knows the randomness used and after publishing the message and the randomness everyone can check the validity of c by re-encryption. However, when a party P receives a homomorphically computed ciphertext c' from c, we saw it cannot compute the randomness from the ciphertext and neither the sender knows the randomness of the original c.
Consider a party P with encryption key pair $(\mathsf{pk}, \mathsf{sk})$ and suppose P has received a ciphertext $c = (c_1, c_2) \in \widehat{G}^2$, encrypted under pk. The following proof allows P to convince a verifier that decrypting c with some information of the secret key (sk if P is honest) yields m, which may either be $\perp$ if c fails to decrypt, or some value in $\mathbf{Z}/q\mathbf{Z}$.

Note that c may or may not be valid (i.e. $c_1 \in G^q$); this is irrelevant, indeed P himself may not know if it is the case, so there is no way P can prove it is so.

Let $M := c_2 \cdot c_1^{-\mathsf{sk}}$. Note that if $M = f^a$ for some $a \in \mathbf{Z}/q\mathbf{Z}$, the decryption algorithm returns a, otherwise it returns $\perp$. Observe that when P is proving that $\vec{c}$ decrypts to a given value, P can reveal M to all players, so everyone can compute $c_2 \cdot M^{-1}$. We present a ZKAoK for the relation $\mathsf{R}_{\mathsf{Dec}}$ and the interactive protocol is given in Figure 5.6.
This proof does not allow the extraction of an *integer* value of the secret key, however it is enough to prove that $\mathsf{pk} \in \langle g_q \rangle$ and that the exponent of c_1 w.r.t. to $c_2 \cdot M^{-1}$ and of pk in base g_q is the same when the decryption is $m \neq \perp$. Furthermore an honest prover

Setup:

1. $(\tilde{s}, f, \hat{g}_q, \widehat{G}, F) \leftarrow \mathsf{Gen}(1^\lambda, q)$.
2. Let $\tilde{A} := \tilde{s} \cdot 2^{40}$, sample $t \xleftarrow{\$} [\tilde{A}]$ and let $g_q := \hat{g}_q^t$.

Prover $(\mathsf{pk}, (c_1, c_2), M; \mathsf{sk})$		Verifier $(\mathsf{pk}, (c_1, c_2), M)$
$r \xleftarrow{\$} [2^{40}\tilde{A}C]$		
$t_1 := c_1^r$ and $t_2 := g_q^r$	$\xrightarrow{t_1, t_2}$	
		$k \xleftarrow{\$} \mathcal{C}$
	$\xleftarrow{k}$	
$u := r + k \cdot \mathsf{sk} \in \mathbf{Z}$		
	$\xrightarrow{u}$	Check $u \in [\tilde{A}C(2^{40}+1)]$ and $c_1^u = t_1 \cdot (c_2 \cdot M^{-1})^k$ and $g_q^u = t_2 \cdot \mathsf{pk}^k$

Figure 5.6: Zero-knowledge argument of knowledge for $\mathsf{R}_{\mathsf{Dec}}$.

knows and uses its secret key in the protocol.

The interactive protocol is given in Figure 5.6. We denote $\mathcal{C}$ the challenge set, and $C := |\mathcal{C}|$ as usual. The only constraint on C is that the C-low order assumption holds in $\widehat{G}$. The protocol is complete, honest verifier zero-knowledge, and sound under the assumption the strong root problem for class groups with input $(\widehat{G}, \widehat{G}^q, g_q)$, and the C-low order problem in $\widehat{G}$ are hard. The proof is essentially that of the ZKAoK for relation $\mathsf{R}_{\mathsf{Enc}}$ in Subsection 4.2.1 (from [CCL+20]) with very minor variations. Formally, the following theorem is valid for $\mathsf{R}_{\mathsf{Dec}}$.

Theorem 5.2.1. If the strong root assumption is $(t'(\lambda), \epsilon_{\mathsf{S}R}(\lambda))$-secure for Gen, and the C-low order assumption is $(t'(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$-secure for Gen, denoting $\epsilon := \max(\epsilon_{\mathsf{S}R}(\lambda), \epsilon_{\mathsf{LO}}(\lambda))$, then the interactive protocol of Figure 5.6 is a computationally convincing proof of knowledge for $\mathsf{R}_{\mathsf{Dec}}$ with knowledge error $\kappa(\lambda)$, time bound $t(\lambda)$ and failure probability $\nu(\lambda)$, where $\nu(\lambda) = 3\epsilon$, $t(\lambda) < t'(\lambda)/448$ and $\kappa(\lambda) = \max(4/C, 448t(\lambda)/t'(\lambda))$. If $r \in [\tilde{s} \cdot 2^{40}]$ (it is so when the prover is honest), the protocol is honest verifier statistical zero-knowledge.

Proof. The proof is very similar to that for $\mathsf{R}_{\mathsf{Enc}}$ in Subsection 4.2.1. □

5.2.2 Aborting and detection of misbehaving parties

As the authors do in [CGG+20], we suppose that all the messages are signed from the sender and as a result all the players know the identity of the sender when a message is published. This is important to identify bad behaviours. We also consider a local timeout in case of delays in sending. After this timeout a player which has not sent the requested message for a specific phase is considered corrupted. We can list all the possible situations in which the protocol ends with an abort, giving the corresponding solution to identify a misbehaving player. Of course, there can be more than one cheater player, however the aim of identifiable aborts is to detect at least one of them and exclude her from next executions of the protocol. We give below the list of cases in which the protocol aborts and a way to detect who caused the failure of the procedure.

Setup In the set up phase, an abort occurs if a player refuses to decommit, if a proof fails or if a signature on a published value fails. In each case all the players know who the faulty player is, since commitments and proofs are signed.

Key Generation Aborts in key generation may occur due to a player refusing to decommit, or a proof π_i^{kg} failing to verify. For such types of abort, the culprit is immediately identified. Aborts may also occur if a player complains that the share it received from a Feldman-VSS is inconsistent (i.e. does not verify correctly). In this case the player raising the complaint can publish the received private share and all players can check consistency. If the check passes and the sender's signature is valid, the misbehaving player is the receiver, otherwise it is the sender. After the misbehaving player is identified, the key generation protocol is re-ran with fresh randomness to establish a secure key.

Key Refresh In a key refresh, an aborts occurs if the sum of the points received from a player is not equal to $0_{\mathbb{G}}$ or a proof $\pi_{i,j}^{\mathsf{kr}}$ fails to verify. In both cases, the sender is detected as the misbehaving player because the proofs or the points are signed.

Pre-Signing and Signing These protocols may abort for various reasons. When a proof fails or some player refuses to decommit, others can immediately detect the cheater. However, in some cases it is not clear who led the protocol to an end. To have an idea of what can happens, notice that in the previous subprotocols each player sends values whom consistency is directly verifiable. But when it is necessary to compute a common value by putting together players' choices and this value is not consistent, it could be hard to detect which is the wrong share. In these specific cases players have to publish private data used in the computation. We will see that these published values do not reveal information about secret signature key shares. We hereafter list the problematic reasons these protocols may abort.

Problematic types of abort in Pre-Sign and Sign:

1. Phase 2. If a player cannot decrypt the message received (α or μ).
2. Phase 2. If the check on μ fails.
3. Phase 5. If $P \neq \sum \bar{R}_i$.
4. Phase 6. If $Q \neq \sum S_i$.
5. Phase 7. If the signature (r, s) is not valid for the message m.

From now, we refer to the items $\#\tau$ of the previous list as "type τ". Furthermore, notice that a type of abort is reached by the protocol if previous types in the list do not occur. We analyze types of abort.

For an abort of type 5, if the protocol reached this point, then $P = \sum \bar{R}_i$ and $Q = \sum S_i$. Furthermore, all the players know the shares $s_i = mk_i + r\sigma_i \mod q$ of the signature. To detect the cheater, it suffices to verify for which index $s_i \cdot R \neq m \cdot \bar{R}_i + r \cdot S_i$. For the remaining four types of abort a more involved discussion is required.

For aborts occurring in the Pre-Sign protocol, parties may be required to publish their private shares k_i, γ_i and other values depending on the considered case. We emphasize that this does not compromise the secret signing key. Indeed, these aborts occur before revealing the shares s_i of the signature, and as a result publishing k_i or γ_i does not give information on s or s_i, since they are not revealed yet. With this observation, we can present the strategy used to detect cheaters in these remaining cases.

Abort of type 1: Assume P_j is complaining that a ciphertext it received from P_i does not decrypt. Then P_j publishes the faulty ciphertext $c := (c_1, c_2)$ along with P_i's signature. Party P_j also reveals $M := c_2 \cdot c_1^{-\mathsf{sk}_j}$, and proves that $(\mathsf{pk}_j, (c_1, c_2), M) \in \mathsf{R}_{\mathsf{Dec}}$, using the proof of Subsection 5.2.1. If this proof fails, or if $M \in F$, then P_j is lying. Otherwise all parties are convinced that P_i is the cheater.

Abort of type 2: Assume P_j is complaining that $\mu_{j,i} \cdot P + B_{j,i} \neq k_j W_i$. Then P_j publishes:

1. k_j and proves that $(\mathsf{pk}_j, c_{k_j} = (c_{k_j,1}, c_{k_j,2}), f^{k_j}) \in \mathsf{R}_{\mathsf{Dec}}$, so that all the parties can check that k_j is the right decryption of c_{k_j}.

2. $\mu_{j,i}$, the ciphertext $\vec{c}_{k_j w_i} := (c_1, c_2)$ and the elliptic curve point $B_{j,i}$ along with P_i's signatures on the latter two elements. Party P_j also proves that $(\mathsf{pk}_j, (c_1, c_2), f^{\mu_{j,i}}) \in \mathsf{R}_{\mathsf{Dec}}$, using the proof of Subsection 5.2.1. If this proof fails then P_j is lying. Otherwise all parties are convinced that ciphertext $\vec{c}_{k_j w_i}$ received by P_j decrypts to $\mu_{j,i}$ using P_j's decryption key.

All parties now check that $\mu_{j,i} \cdot P + B_{j,i} \neq k_j W_i$; if so P_i is identified as the cheater, else it is P_j.

Abort of type 3 and 4: Since one can not identify the cheater directly from $P \neq \sum \bar{R}_i$ or from $Q \neq \sum S_i$, in these cases identifying a misbehaving player requires a more convoluted list of actions. To detect who misbehaved, let us consider the protocol steps leading up to an abort of type 3. Note that aborts of type 3 occur after proofs $\{\pi_i^{\mathsf{kg}}\}_{i\in[n]}$ and $\{\pi_i, \widetilde{\pi}_i, \pi_i^\gamma, \pi_i'\}_{i\in S}$ have been accepted, and also after proofs $\{\pi_i''\}_{i\in S}$ have been accepted for aborts of type 4.

The identification protocol uses new techniques, and requires the use of a zero-knowledge proof for $\mathsf{R}_{\mathsf{Dec}}$ to prove ciphertexts decrypt to a given value (e.g. the proof of Subsection 5.2.1). In order to prove that the players indeed ran the protocol correctly, it is necessary and sufficient to prove that for $i \in S$ all the following consistency items hold:

(i) The value k_i input to the MtA protocol is consistent with that input to the MtAwc protocol. This holds unconditionally since only a single encryption of k_i is broadcast in Phase 1.

(ii) The value w_i input to the MtAwc protocol is consistent with the public value $W_i = w_i \cdot P$ that is associated with player P_i. Under the soundness of the Schnorr ZKPoK used for π_i^{kg} in $\mathsf{IKeyGen}$, this item holds.

(iii) The value $\tilde{\gamma}_i$ input to the MtA protocol is consistent with $\Gamma_i = \gamma_i \cdot P$ that is decommitted to in Phase 4, i.e. $\gamma_i = \tilde{\gamma}_i$.

(iv) The value δ_i published in Phase 3 is consistent with the shares received during the MtA protocol. In particular, the following should hold:

$$\delta_i = k_i\gamma_i + \sum_{j\neq i} \alpha_{ij} + \sum_{j\neq i} \beta_{ji}.$$

(v) The value S_i published in Phase 6 is consistent with the shares received during the MtAwc protocol. In particular, it must hold that

$$S_i = \sigma_i \cdot R \quad \text{and} \quad \sigma_i = k_i w_i + \sum_{j\neq i} \mu_{ij} + \sum_{j\neq i} \nu_{ji}.$$

We now distinguish aborts of type 3 and 4.

Identification - Abort of type 3 in Phase 5. For this to occur, either consistency item (iii) or (iv) does not hold (since S_i has not been computed yet, item (v) is not relevant here). We hereafter explain how, for both of these inconsistency types, parties can identify a cheating player.

(iii) Each party P_j publishes (in order):

  (a) For $i \neq j$, $\beta_{i,j}$ and γ_j. All players then check that Γ_j revealed in Phase 4 satisfies $\Gamma_j = \gamma_j \cdot P$. If the check fails, P_j is identified as a cheater.

  (b) After all $\beta_{\ell,i}$ and γ_i, for $i \neq j$ and for $\ell \neq i$ have been published, i.e. after P_j sees all the βs and γs from other parties, P_j reveals k_j and performs a ZK proof that $(\mathsf{pk}_j, c_{k_j}, f^{k_j}) \in \mathsf{R}_{\mathsf{Dec}}$. If the proof is accepted, all parties are convinced c_{k_j} decrypts to k_j using P_j's decryption key; else P_j is identified as a cheater.

  (c) For $i \neq j$, reveal $\alpha_{j,i}$ and $c_{k_j\gamma_i}$. Then perform a ZK proof that $(\mathsf{pk}_j, c_{k_j\gamma_i}, f^{\alpha_{j,i}}) \in \mathsf{R}_{\mathsf{Dec}}$. If the proof is accepted, all parties are convinced $c_{k_j\gamma_i}$ decrypts to $\alpha_{j,i}$ using P_j's decryption key; else P_j is identified as a cheater.

Then for $i \neq j$, all players can compute:

$\widetilde{\beta}_{i,j} := k_i\gamma_j - \alpha_{i,j}$ and check if $\beta_{i,j} = \widetilde{\beta}_{i,j}$. If it is not true for some index i, then P_j is the cheater (for using $\tilde{\gamma}_j \neq \gamma_j$ in the MtA protocol).

We now argue that if all such checks pass, item (iii) holds, i.e., all players are convinced that $\tilde{\gamma}_j \neq \gamma_j$. Observe that if P_j used a different value $\tilde{\gamma}_j$ in the MtA than the γ_j published in step (a) satisfying $\Gamma_j = \gamma_j \cdot P$, then to have $\beta_{i,j} = \widetilde{\beta}_{i,j}$, party P_j must have predicted the value $k_i\gamma_j - \alpha_{i,j}$ without knowing $\alpha_{i,j}$ or k_i. Under the smoothness of the encryption scheme, the distribution followed by k_i is uniformly random in $\mathbf{Z}/q\mathbf{Z}$ from P_j's view in step (a), hence P_j (who only knows $\beta_{i,j} = k_i\tilde{\gamma}_j - \alpha_{i,j}$) cannot predict $\widetilde{\beta}_{i,j} = k_i\gamma_j - \alpha_{i,j}$ with probability significantly greater than $1/q$.

(iv) Now if consistency of (iii) holds, all parties can check that $\delta_j = k_j\gamma_j + \sum_{i\neq j}\alpha_{j,i} + \sum_{i\neq j}\beta_{i,j}$; if not P_j is identified as the cheater. If equality holds then consistency of (iv) holds (i.e. consistency of δ_j).

Identification - Abort of type 4 in Phase 6. Since this failure occurs in Phase 6, we know that consistency of (i), (ii), (iii) and (iv) hold. The abort must hence be due to (v) not holding. To detect a cheater, each P_j does the following:

1. Publish k_j and perform a ZK proof that $(\mathsf{pk}_j, c_{k_j}, f^{k_j}) \in \mathsf{R}_{\mathsf{Dec}}$. If the proof is accepted, all parties are convinced c_{k_j} decrypts to k_j using P_j's decryption key; else P_j is identified as a cheater.

2. For all $i \neq j$, publish $\mu_{j,i}$ and $c_{k_j w_i}$. Perform a ZK proof that $(\mathsf{pk}_j, c_{k_j w_i}, f^{\mu_{j,i}}) \in \mathsf{R}_{\mathsf{Dec}}$. If the proof is accepted, all parties are convinced $c_{k_j w_i}$ decrypts to $\mu_{j,i}$ using P_j's decryption key; else P_j is identified as a cheater.

3. For all $i \neq j$, publish the elliptic curve point $B_{j,i}$ received from P_i (and the signature). Notice that since checks of Phase 2 passed, for all i, j it holds that $\mu_{j,i} \cdot P + B_{j,i} = k_j \cdot W_i$ and so $B_{j,i} = \nu_{j,i} \cdot P$.

If the above checks passed, all parties can compute:

$$\Sigma_j = \sigma_j \cdot P := k_i W_i + \sum_{i\neq j}\mu_{j,i} \cdot P + \sum_{i\neq j} B_{i,j}.$$

Finally, P_j performs a ZK proof that the discrete log in base P of Σ_j is equal to the discrete log in base R of S_j. Precisely, let for public parameters $(\mathbb{G}, P, q)$, let $\mathsf{R}_{\mathsf{log}} := \{(R, S, T) \in \mathbb{G}^3; \sigma \mid R = \sigma \cdot S \wedge T = \sigma \cdot P\}$. Then P_j performs a ZKPoK π_j^{log} that $(R, S_j, \Sigma_j) \in \mathsf{R}_{\mathsf{log}}$; this can be done as described in [CP93]. If this proof fails, P_j is identified as a cheater.

Note that if none of these proofs fail, under the soundness of the aforementioned ZK proofs it holds that, for $i \in S$, consistency of (v) holds, and no aborts occur. This concludes the description of the identification procedure.

On the impossibility of re-ecnryption From the analysis of aborts of types 2 and 4 we notice that after revealing k_j, P_j run a proof that $(\mathsf{pk}_j, c_{k_j}, f^{k_j}) \in \mathsf{R}_{\mathsf{Dec}}$. Since c_{k_j} is a fresh ciphertext computed by P_j, it is clear that P_j knows the randomness r_j used to compute c_{k_j} and it seems that is sufficient that P_j reveals k_j and r_j and other parties can check the validity of c_{k_j} by re-encryption, instead of running a ZK for $\mathsf{R}_{\mathsf{Dec}}$. However, it is not possible to reveal the randomness used for security reasons as we will see from the simulation in the proof of security in Section 5.3. However, we give here the idea behind this fact. The proof is divided in games, and there is a switch between two games where a valid ciphertext is converted into an invalid one. From the definition of an invalid ciphertext, we know that the first component is an element of $\widehat{G}$ instead of G_q and then it does not exist an r such that $g_q^r = c_{k_j,1}$. As a consequence, revealing the randomness in c_{k_j} permits to distinguish between games, since the randomness as intended in CL exists only in the case of valid ciphertexts.

5.3 Security

To prove that our protocol is secure, we demonstrate that if there exists a PPT algorithm $\mathcal{A}$ which breaks enhanced unforgeability of the threshold ECDSA protocol of Fig. 5.2, 5.3, 5.4 and 5.5, then one can devise an algorithm $\mathcal{S}$ using $\mathcal{A}$ to break the enhanced unforgeability of centralised ECDSA. To this end $\mathcal{S}$ simulates the environment of $\mathcal{A}$, so that $\mathcal{A}$'s view of its interactions with $\mathcal{S}$ are indistinguishable from $\mathcal{A}$'s view in a real execution of the protocol.

In Subsection 5.3.1, we first prove our $(t, n)-$threshold protocol, for any $t < n$, secure against static corruptions. Next, in Subsection 5.3.2 we show that if all players participate in the signing algorithm ($t = n - 1$) one can easily adapt the proof for static corruptions to the adaptive case. We note that [CGG$^+$20] also only prove their protocol secure for $t = n-1$. They state that using techniques of Gennaro et al. [GG18] one can immediately derive a full threshold protocol. We emphasize that in order to build our $(t, n)-$protocol, for any $t < n$, *we do* use the techniques of [GG18], hence it may be the case that it is also secure against active adversaries. However, as we will detail in Subsection 5.3.2, it remains unclear to us how one can claim security against adaptive adversaries in this setting. Indeed, after proving the security of the protocol against static corruptions for any threshold $t < n$, we will present the proof of security against adaptive corruptions only for the special case $t = n - 1$ and we will explain the difficulties that occur for other values $t < n - 1$ of the threshold.

In both adversarial settings, $\mathcal{S}$ gets as input an ECDSA public key Q, where $Q = x \cdot P$, and can query oracles $\mathcal{O}^R$ and $\mathcal{O}^{\mathsf{Sign}(x,\cdot;\cdot)}$ of Definition 1.3.4. After this query phase, $\mathcal{S}$ must output a forgery, i.e. a signature s for a message m of its choice, which it did not receive from the oracle[4].

5.3.1 Security of the Full Threshold Protocol with Identifiable Aborts against Static Adversaries

As all players play symmetric roles in the protocol, it suffices to demonstrate that if $\mathcal{A}$ corrupts $\{P_j\}_{j>1}$, one can construct $\mathcal{S}$ simulating P_1 such that the output distribution of $\mathcal{S}$ is indistinguishable from $\mathcal{A}$'s view in an interaction with an honest party P_1. The simulation is similar to that one in Section 4.3, but takes in account the simulation of

[4]Recently, in [GS21] the authors provide an analysis of variants of ECDSA in the Generic Group Model, including the case of ECDSA with presignatures

the key refresh subprotocol, of the identification of misbehaving players in case of aborts and of the additional steps required – as we argued in Section 5.2 – to guarantee that the identification is possible. It is obvious that $\mathcal{S}$ must be able to simulate identification procedures even not knowing some of the values. This changes the way one defines semi correct executions with regard to the protocol in Chapter 4 (see proof of security in Section 4.3 for semi correct execution original definition). Another difference with our protocol in Chapter 4 is the way it is simulated the long interactive procedure after sharing the signature. Here, we do not have this interactive step. We see how $\mathcal{S}$ works in the following paragraphs.

Simulating the Key Generation Protocol On input $Q = x \cdot P$, the forger $\mathcal{S}$ must set up in its simulation with $\mathcal{A}$ this same verification key Q (without knowing x). This will allow $\mathcal{S}$ to subsequently simulate interactively signing messages with $\mathcal{A}$, using the output of its' (enhanced) ECDSA signing oracle.

The main differences with the proof of [GG20][5], arises from the fact $\mathcal{S}$ knows it's own decryption key sk_1, but does not extract that of other players. As in [CCL+20], the linearly homomorphic encryption scheme we use results from projective hash functions, whose security is statistical, thus the fact $\mathcal{S}$ uses its' secret key does not compromise security, and we can still reduce the security of the protocol to the smoothness of the CL scheme. However, we do not need to prove knowledge of secret keys associated to public keys in the key generation protocol, and then $\mathcal{S}$ can not extract the decryption keys of corrupted players. The simulation is described below.

Simulating Key Generation - Description of $\mathcal{S}$:

1. $\mathcal{S}$ receives a public key Q from it's ECDSA challenger.

2. $\mathcal{S}$ samples a CL encryption key pair $(\mathsf{pk}_1, \mathsf{sk}_1) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, and a random value $u_1 \in \mathbf{Z}/q\mathbf{Z}$. It computes $[\mathsf{kgc}_1, \mathsf{kgd}_1] \leftarrow \mathsf{Com}(u_1 P)$ and broadcasts pk_1 and kgc_1. In return, $\mathcal{S}$ receives the public keys $\{\mathsf{pk}_j\}_{j\in[n], j\neq 1}$ and commitments $\{\mathsf{kgc}_j\}_{j\in[n], j\neq 1}$.

3. $\mathcal{S}$ broadcasts kgd_1 and receives $\{\mathsf{kgd}_j\}_{j\in[n], j\neq 1}$. For $i \in [n]$, let $Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$ be the revealed commitment value of each party. Each player performs a (t, n) Feldman-VSS of the value Q_i, with Q_i as the free term in the exponent.

4. $\mathcal{S}$ rewinds $\mathcal{A}$ to the decommitment step and
   - equivocates P_1's commitment to $\widehat{\mathsf{kgd}}$ so that the committed value revealed is now $\widehat{Q}_1 := Q - \sum_{j=2}^{n} Q_j$.
   - simulates the Feldman-VSS with free term $\widehat{Q}_1$.

5. $\mathcal{A}$ will broadcast the decommitments $\{\widehat{\mathsf{kgd}}_j\}_{j\in[n], j\neq 1}$. Let $\{\widehat{Q}_j\}_{j=2\ldots n}$ be the committed value revealed by $\mathcal{A}$ at this point (this could be $\perp$ if $\mathcal{A}$ refuses to decommit).

6. All players compute the public signing key $\widehat{Q} := \sum_{i=1}^{n} \widehat{Q}_i$. If any $\widehat{Q}_i = \perp$, then $\widehat{Q} := \perp$.

[5]We recall from the introduction of this chapter, that for identifying aborts we follow the construction of [GG20] which composed to [CMP20] brought to [CGG+20]

7. Each player P_i adds the private shares it received during the n Feldman VSS protocols to obtain x_i (such that the x_i are a (t, n) Shamir's secret sharing of the secret key $x = \sum_i u_i$). Note that due to the free term in the exponent, the values $X_i := x_i \cdot P$ are public.

8. $\mathcal{S}$ simulates π_1^{kg} (the ZKPoK that it knows x_1 corresponding to X_1). Then, for $j \in [n]$, $j \neq 1$, $\mathcal{S}$ receives from $\mathcal{A}$ a ZKPoK of x_j satisfying $X_j := x_j \cdot P$; from which $\mathcal{S}$ can extract x_j.

Simulating the Key Refresh Protocol. For all honest players P_i, $\mathcal{S}$ runs the prescribed steps of the Key Refresh protocol.

Simulating Protocols Pre-Sign and Sign. After the key generation is over, the simulator must handle the signature queries issued by $\mathcal{A}$. Recall that $\mathcal{A}$ can issue two types of queries:

- oracle $\mathcal{O}^R$ to obtain a uniformly random point $R = (r_x, r_y)$ in $\mathbb{G}$:
- oracle $\mathcal{O}^{\mathsf{Sign}(\mathsf{sk}, m; R)}$ which on input a message m chosen by $\mathcal{A}$, returns a valid signature (r, s) for m where $r := r_x \bmod q$ if $R = (r_x, r_y)$ was queried to $\mathcal{O}^R$; else it returns $\perp$.

The simulator simulates P_1 in the threshold signature protocol on input R for the offline phase (Phases 1-6), and a correct signature (r, s) for m under the public key Q for the online phase (Phase 7). We stress that though the simulator knows the decryption key sk_1, and P_1's ECDSA "public key share" $W_1 = w_1 \cdot P$; it does not know the secret value w_1 associated with P_1. However it does know the shares w_j, $j > 1$ of all other players (extracted from the Schnorr proofs in the Key Generation phase).

The simulation of the Pre-Signing and Signing protocols is based on [GG20] and [CCL+20]. In the following simulation $\mathcal{S}$ aborts whenever the protocol is supposed to abort, i.e., whenever $\mathcal{A}$ refuses to decommit a committed value, a ZK proof fails, a check does not pass or if the signature (r, s) does not verify.

Simulating Pre-signing and Signing - Description of $\mathcal{S}$:

Phase 1: As in a real execution, $\mathcal{S}$ samples $k_1, \gamma_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, $r_1 \xleftarrow{\$} [\widetilde{A}]$ uniformly at random. It computes $c_{k_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_1, k_1; r_1)$, the associated ZKAoK π_1, and $[\mathsf{c}_1, \mathsf{d}_1] \leftarrow \mathsf{Com}(\gamma_1 P)$. It broadcasts $\mathsf{c}_1, c_{k_1}, \pi_1$ before receiving $\{\mathsf{c}_j, c_{k_j}, \pi_j\}_{j \in S, j \neq 1}$ from $\mathcal{A}$. $\mathcal{S}$ checks the proofs are valid and extracts the encrypted values $\{k_j\}_{j \in S, j \neq 1}$ from which it computes $k := \sum_{i \in S} k_i$.

Phase 2: Recall that during the regular run of the protocol, P_1 will engage in two MtA protocols and two MtAwc protocols with each other player $P_j, j \in S, j \neq 1$. $\mathcal{S}$ runs the protocol for P_1 as follows:

(a) *Initiator for MtA with* k_1 *and* γ_j: Since $\mathcal{S}$ knows k_1, it runs the protocol as would an honest P_1; it also decrypts $c_{k_1 \gamma_j}$ received from P_j thereby obtaining $\alpha_{1,j} \bmod q$.

(b) *Respondent for MtA with k_j and γ_1*: Since $\mathcal{S}$ knows γ_1, it runs the protocol as would an honest P_1. Recall that $\mathcal{S}$ extracted k_j from π_j in Phase 1, it also knows $\beta_{j,1}$ (as $\mathcal{S}$ chose it), hence $\mathcal{S}$ can compute P_j 's share $\alpha_{j,1} := k_j\gamma_1 - \beta_{j,1} \bmod q$.

(c) *Initiator for MtAwc with k_1 and w_j (note that the first message sent in this sub-protocol is common to all players in both MtA (item (a)) and MtAwc (item (c)))*: Since $\mathcal{S}$ knows k_1, it runs the protocol as would an honest P_1; decrypting $c_{k_1 w_j}$ to obtain $\mu_{1,j}$; and checking that $\mu_{1,j}P + B_{1,j} = k_1 W_j$. Furthermore, recall that $\mathcal{S}$ extracted x_j from π_j^{kg} in KeyGen, hence it knows w_j, so $\mathcal{S}$ can compute $\nu_{1,j} = k_1 w_j - \mu_{1,j} \bmod q$.

(d) *Respondent for MtAwc with k_j and w_1*: Here, $\mathcal{S}$ only knows $W_1 = w_1 \cdot P$ (but not w_1). So it samples a random $\mu_{j,1} \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and sets $c_{k_j w_1} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mu_{j,1})$, and $B_{j,1} := k_j \cdot W_1 - \mu_{j,1} \cdot P$. Finally $\mathcal{S}$ sends the ciphertexts and the point.

Note that at this point $\mathcal{S}$ knows:

- k_i for each $i \in S$, w_j for each $j \in S, j \neq 1$
- $\alpha_{1,j}, j \in S, j \neq 1$ as initiator for MtA, $\alpha_{j,1}, \beta_{j,1}, j \in S, j \neq 1$ as respondent for MtA. Indeed, $\alpha_{1,j}$ is a value decrypted by $\mathcal{S}$; $\beta_{j,1}$ is chosen by $\mathcal{S}$; and $\alpha_{j,1} = k_j\gamma_1 - \beta_{j,1}$, where $\mathcal{S}$ knows all the values on the right, and so can compute $\alpha_{j,1}$.
- $\mu_{1,j}, \nu_{1,j}, j \in S, j \neq 1$ as initiator for MtAwc, $\mu_{j,1}, \nu_{j,1} \cdot P = B_{j,1}, j \in S, j \neq 1$ as respondent for MtAwc. Indeed, $\mu_{1,j}$ is a value decrypted by $\mathcal{S}$; $\mu_{j,1}$ is chosen by $\mathcal{S}$ in (d); $\nu_{1,j} = k_1 w_j - \mu_{1,j}$, where $\mathcal{S}$ knows all the values on the right.

After all these sub-protocols, $\mathcal{S}$ computes $\delta_1 := k_1\gamma_1 + \sum_{j \in S, j\neq 1} \alpha_{1,j} + \sum_{j\in S, j \neq 1} \beta_{j,1}$. Note that $\mathcal{S}$ does not know the internal values from the MtA and MtAwc protocols executed by two players that are both controlled by the adversary. Hence $\mathcal{S}$ is not able to compute the values σ_j and δ_j for $j \in S, j \neq 1$. Furthermore $\mathcal{S}$ cannot compute σ_1 since it doesn't know the value w_1, but it can compute

$$\begin{aligned}\sigma_C := \sum_{i>1} \sigma_i = \sum_{i>1}(k_i w_i + \sum_{j \neq i} \mu_{i,j} + \sum_{j\neq i} \nu_{j,i}) = \sum_{i>1}\sum_{j\neq i}(\mu_{i,j} + \nu_{j,i}) + \sum_{i>1} k_i w_i \\ = \sum_{i>1}(\mu_{i,1} + \nu_{1,i}) + \sum_{i>1; j>1} k_i w_j\end{aligned}$$

since it knows all the values $\{k_j\}_{j\in S}$, $\{w_j\}_{j \in S, j\neq 1}$, it chooses the random values $\mu_{i,1}$ and it can compute all of the shares $\nu_{1,j} = k_1 w_j - \mu_{1,j} \mod q$.

Furthermore, as $\mathcal{S}$ knows W_1, it can compute $\Sigma_1 := \sigma_1 \cdot P = k_1 W_1 + \sum_{j \neq 1}(\mu_{1,j} + \nu_{j,1})P$.

Phase 3: $\mathcal{S}$ broadcasts δ_1 and receives all the $\{\delta_j\}_{j \in S, j\neq 1}$ from $\mathcal{A}$. Let $\widetilde{\delta} := \sum_{i \in S} \delta_i$. Next $\mathcal{S}$ samples a random $\ell_1 \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and broadcasts $T_1 := \Sigma_1 + \ell_1 \cdot H$. Note that

$\mathcal{S}$ does not know σ_1, so it simulates the ZK proof $\tilde{\pi}_1$. Then, from the proofs $\tilde{\pi}_j$ received from $\mathcal{A}$, $\mathcal{S}$ extracts the values $(\widehat{\sigma}_j, \ell_j)$ for $j \in S, j \neq 1$. We hereafter denote $\widehat{\sigma}_C := \sum_{j \in S, j \neq 1} \widehat{\sigma}_j$.

Phase 4: $\mathcal{S}$ broadcasts d_1 which decommits to Γ_1, and $\mathcal{A}$ reveals $\{\mathsf{d}_j\}_{j \in S, j \neq 1}$ which decommit to $\{\Gamma_j\}_{j \in S, j \neq 1}$. From the proofs π_j^{γ} on Γ_j, $\mathcal{S}$ can extract γ_j for each $j \in S, j \neq 1$; these are consistent with the values used in Phase 1 thanks to the binding property of the commitment scheme. Now $\mathcal{S}$ can compute

$$\delta = (\sum_{i \in S} k_i) \cdot (\sum_{i \in S} \gamma_i) = k\gamma,$$

where $\gamma = \sum_{i \in S} \gamma_i$. Note that $\mathcal{A}$ may have used different values $\widetilde{\gamma}_i$ in the MtA protocol than the γ_i committed to in Phase 1, hence we denote them with a tilde.

At this point $\mathcal{S}$ can detect if the values published so far by $\mathcal{A}$ are consistent; note that $\mathcal{S}$ will behave differently in Phases 5, 6 and 7 depending on this detection.

To detect inconsistencies, $\mathcal{S}$ first computes $\widetilde{R} := \widetilde{\delta}^{-1}(\sum_i \Gamma_i)$.

Then using the values k_i extracted in Phase 1, $\mathcal{S}$ checks that $\sum_i k_i \cdot \widetilde{R} = P$. Notice that if

$$\sum_i k_i \cdot \widetilde{R} = k \cdot \widetilde{R} = k\widetilde{\delta}^{-1} \sum_{i \in S} \gamma_i \cdot P = k\widetilde{\delta}^{-1}\gamma \cdot P = P \Rightarrow \widetilde{\delta} = k\gamma = \delta$$

The simulator can also detect if the values σ_j computed in Phase 2 are consistent with those used to compute points T_j in Phase 3; in particular $\mathcal{S}$ checks that $\widehat{\sigma}_C = \sigma_C$. We thus distinguish two types of executions: an execution is said to be *semi-correct* if

$$\sum_i k_i \widetilde{R} = P \qquad \text{and} \qquad \widehat{\sigma}_C = \sigma_C$$

which, as explained above, implies that $\delta = \widetilde{\delta}$ and $\widehat{\sigma}_C = \sigma_C$. If either of the above equalities do *not* hold, the execution is said to be *non semi-correct.*

Now $\mathcal{S}$ adapts its behaviour depending on the type of execution:

- **Semi-correct execution:**
  1. $\mathcal{S}$ invokes oracle $\mathcal{O}^R$ to obtain $R = (r_x, r_y)$.
  2. $\mathcal{S}$ sets $\widehat{\Gamma}_1 := \widetilde{\delta} \cdot R - \sum_{i \in S, i \neq 1} \Gamma_i$, so that $R = \widetilde{\delta}^{-1}\left(\widehat{\Gamma}_1 + \sum_{i \in S, i \neq 1} \Gamma_i\right)$. Then $\mathcal{S}$ rewinds $\mathcal{A}$ to the decommitment step in Phase 4, equivocates P_1's commitment so that it decommits to $\widehat{\Gamma}_1$ instead of Γ_1.
- **Non semi-correct execution:** $\mathcal{S}$ simply moves on to Phase 5.

Phase 5:
- **Semi-correct execution:** $\mathcal{S}$ publishes $\bar{R}_1 := P - \sum_{i \in S, i \neq 1} k_i \cdot R$ together with π_1': a simulated ZKP of consistency with $c_{k_1} = \mathsf{Enc}_1(k_1)$ (note that in this case $\bar{R}_1 \neq k_1 \cdot R$ due to the rewinding).
- **Non semi-correct execution:** $\mathcal{S}$ publishes $\bar{R}_1 := k_1 \cdot R$ together with π_1': a real ZKP of consistency with $c_{k_1} = \mathsf{Enc}_1(k_1)$ (this proof needn't be simulated).

Phase 6: – **Semi-correct execution:** $\mathcal{S}$ publishes $S_1 := Q - \sum_{j \in S, j \neq 1} \sigma_j R$ together with π_1'': a simulated ZKP of consistency with T_1 (again in this case the simulated $S_1 \neq \sigma_1 \cdot R$ due to the rewinding).

– **Non semi-correct execution:** $\mathcal{S}$ has P_1 publish $S_1 := \sigma_1 R$ together with π_1'': a real ZKP of consistency with T_1 (this proof needn't be simulated).

In a non semi-correct execution, at least one of the the adversary's proofs π_j' or π_j'' for some $j > 1$ will fail, and the protocol will abort.

Phase 7: $\mathcal{S}$ invokes the second oracle $\mathcal{O}^{\mathsf{Sign}(\mathsf{sk}, m; R)}$ with input m. In return, $\mathcal{S}$ receives the valid signature (r, s) on m, where $r = r_x \bmod q$, for some $R = (r_x, r_y)$ computed in a previous offline phase (in particular in one that was semi-correct, since it concluded successfully).

Now $\mathcal{S}$ knows $s_C = \sum_{j \in S, j \neq 1} s_j$ because $s_C = m \sum_{j \in S, j \neq 1} k_j + \sigma_C r$ where σ_C is as defined in the simulation of Phase 2 (Note: if $\mathcal{A}$ cheats in Phase 7 – denoting $\{\tilde{s}_i\}_{i>1}$ the values $\mathcal{S}$ receives from $\mathcal{A}$ in Phase 7, and $\tilde{s}_C := \sum_{i>1} \tilde{s}_i$ – it is possible that $s_C \neq \tilde{s}_C$). So $\mathcal{S}$ computes the share s_1 consistent with (r, s) and s_C as $s_1 := s - s_C$. Finally, $\mathcal{S}$ broadcasts this value s_1.

Simulating Identification.

Simulating Identification of aborts in Key Generation. If an abort occurs in the Key Generation protocol, $\mathcal{S}$ runs the identification protocol as would P_1 in a real execution. Furthermore, if some player P raises a compliant against P_1 (simulated by $\mathcal{S}$), then P is detected as a cheater since the simulation of Feldman VSS is done in such a way that corrupted players receive values which pass the verification check.

Simulating Identification of aborts in Pre-Sign and Sign. For any of the trivial types of abort allowing to immediately detect the faulty player, $\mathcal{S}$ has nothing to simulate. Consider the problematic types of abort listed on page 150:

- Abort of type 1: if $\mathcal{S}$ cannot decrypt another player's ciphertext, since $\mathcal{S}$ knows sk_1, it can run the proof for relation $\mathsf{R}_{\mathsf{Dec}}$ as would P_1.
- Abort of type 2: if $\mathcal{S}$ announces that the check on $\mu_{1,j}$ fails (for some $j > 1$), it runs the identification protocol as would P_1. Conversely, if some player P_j for $j > 1$ complains about the $\mu_{j,1}$ it received, observe that: if $\mu_{j,1}$ is the real decryption of $c_{k_j w_1}$ (which it must be if the proof for $\mathsf{R}_{\mathsf{Dec}}$ provided by P_j is valid), then since the point $B_{j,1}$ sent by $\mathcal{S}$ to P_j was computed as $B_{j,1} := k_j \cdot W_1 - \mu_{j,1}$, necessarily the equality test will pass. Observe that the value $\nu_{j,1}$ is not revealed; hence no other (corrupted) party can check that $\mathcal{S}$ knows $\nu_{j,1}$ such that $B_{j,1} = \nu_{j,1} \cdot P$. Hence the simulation ends correctly.
- Abort of type 3: $\mathcal{S}$ follows the real identification procedure (see page 152).
- Abort of type 4: $\mathcal{S}$ follows the real identification procedure, up until it needs to prove knowledge of σ_1 satisfying $S_1 = \sigma_1 \cdot R$. Since $\mathcal{S}$ does not know σ_1, it simulates the proof π_1^{log}.

- Abort of type 5: here an abort occurs if the computed signature (r, s) is not valid, as no extra values need to be published to identify the cheater, $\mathcal{S}$ has nothing to simulate.

  Let us denote $\{s_i\}_{i>1}$ the values computed during the pre-sign protocol (which are correct since no abort occurred), and let us denote $\{\tilde{s}_i\}_{i>1}$ the values that $\mathcal{A}$ broadcasts in Phase 7. Let us further denote $s_C := \sum_{i>1} s_i$ and $\tilde{s}_C := \sum_{i>1} \tilde{s}_i$. Now observe that – as long as $s_C = \tilde{s_C}$ – from the way $\mathcal{S}$ computes s_1 (i.e. $s_1 := s - s_C$), since the signature it receives from its oracle is valid for verification key Q, the computed signature is necessarily valid. Conversely, if $s_C \neq \tilde{s_C}$, $\mathcal{S}$ sets $s_1 := s - s_C$ and aborts. To identify the cheater, everyone checks $s_i \cdot R = m \cdot \bar{R}_i + r \cdot S_i$ for all i. From the way $\mathcal{S}$ computed $s_1 R$, it will not be identified as the cheater.

5.3.1.1 Indistinguishability of Real and Simulated Environments

We here argue that a static adversary $\mathcal{A}$ can not distinguish a real execution of the protocol – interacting with P_1 – from a simulated execution. We distinguish semi-correct and non semi-correct executions.

Semi-Correct Executions

Lemma 5.3.1 states the assumptions under which indistinguishability holds. Regarding the key generation, pre-signing and signing sub-protocols, the proof resembles that of Lemma 4.3 ([CCL+20]). Furthermore the simulator runs the key refresh sub-protocol as in a real execution, hence the simulation there is perfect.

Lemma 5.3.1. Assuming the strong root assumption and the C-low order assumption hold for Gen; the CL encryption scheme is δ_s-smooth; the HSM problem is δ_{HSM}-hard; and the commitment scheme is non-malleable and equivocable; then on input m the simulation either outputs a valid signature (r, s) or aborts, and is computationally indistinguishable from a semi-correct real execution.

The proof is very similar to that in [CCL+20], the main difference being that $\mathcal{S}$ now also simulates the identification procedure when the protocol aborts.

Indistinguishability of pre-signing and signing protocols. The differences between $\mathcal{A}$'s real and simulated view are the following:

1. $\mathcal{S}$ does not know w_1. So for each $j \in S, j \neq 1$ it cannot compute $c_{k_j w_1}$ as in a real execution of the protocol. However under the strong root and C-low order assumption in $\widehat{G}$, $\mathcal{S}$ can extract k_j from proof π_j in Phase 1 for each $j \in S, j \neq 1$. $\mathcal{S}$ needs to simulate P_1 as a respondent in MtAwc protocols, then it chooses a random $\mu_{j,1}$ and encrypt it as we have seen in Phase 2 simulation. The resulting view of $\mathcal{A}$ is identical to an honestly generated one since both in real and simulated executions $\mu_{j,1}$ is uniformly distributed in $\mathbf{Z}/q\mathbf{Z}$. Moreover c_{k_j} was proven to be a valid ciphertext, so ciphertexts computed using homomorphic operations over c_{k_j} and fresh ciphertexts computed with pk_j follow identical distributions from $\mathcal{A}$'s view.

2. $\mathcal{S}$ computes $\widehat{\Gamma}_1 := \widetilde{\delta} \cdot R - \sum_{i \in S, i \neq 1} \Gamma_i$, and equivocates its commitment c_1 s.t. d_1 decommits to $\widehat{\Gamma}_1$. Let us denote $\widehat{\gamma}_1 \in \mathbf{Z}/q\mathbf{Z}$ the value s.t. $\widehat{\Gamma}_1 = \widehat{\gamma}_1 P$, where $\widehat{\gamma}_1$ is unknown to $\mathcal{S}$, but the forger can simulate the ZKPoK of $\widehat{\gamma}_1$.
   Let us further denote $\widehat{k} \in \mathbf{Z}/q\mathbf{Z}$ the randomness (unknown to $\mathcal{S}$) used by its' signing oracle to produce R. It holds that $\widetilde{\delta} = \widehat{k}(\widehat{\gamma}_1 + \sum_{j \in S, j \neq 1} \gamma_j)$, where $\Gamma_i = \gamma_i \cdot P$, to distinguish them from the $\widetilde{\gamma}_i$s which are used in MtA protocols. Finally, let us denote $\widehat{k}_1 := \widehat{k} - \sum_{j \in S, j \neq 1} k_j$. $\mathcal{S}$ is implicitly using $\widehat{k}_1 \neq k_1$, even though $\mathcal{A}$ received an encryption of k_1 in Phase 1. However, from the smoothness of the CL scheme, and the hardness of the HSM problem, this change is unnoticeable to $\mathcal{A}$.

   Claim 3. If the CL encryption scheme is δ_s-smooth and the HSM problem is δ_{HSM}-hard, then no probabilistic polynomial time adversary $\mathcal{A}$ – interacting with $\mathcal{S}$ – can notice the value of k_1 in the computation of R being replaced by the (implicit) value $\widehat{k}$ with probability greater than $2\delta_{\mathsf{HSM}} + 3/q + 4\delta_s$.

   Proof. To see this consider the following sequence of games. We denote E_i the probability $\mathcal{A}$ outputs 1 in $\mathsf{Game_i}$.
   Game_0 *to* Game_1. $\mathcal{S}$ uses the secret key sk_1 instead of the public key pk_1 and r_1 to compute $c_{k_1} \leftarrow (u_1, u_1^{\mathsf{sk}_1} f^{k_1})$ where $u_1 = g_q^{r_1}$. The simulation of honest players uses the public key as usual. Both games are perfectly indistinguishable from $\mathcal{A}$'s view:

   $$|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_0]| = 0.$$

   Game_1 *to* Game_2. In Game_2 one replaces the first element of c_{k_1} (in Game_1 this is $u_1 \in G^q$) with $\widetilde{u}_1 \in G \backslash G^q$. There exists a unique $r_1 \in \mathbf{Z}/s\mathbf{Z}$ and $b_1 \in \mathbf{Z}/q\mathbf{Z}$ such that $\widetilde{u}_1 = g_q^{r_1} f^{b_1}$. And $c_{k_1} = (\widetilde{u}_1, \widetilde{u}_1^{\mathsf{sk}_1} f^{k_1})$. Under the δ_{HSM}-hardness of HSM both games are indistinguishable:

   $$|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_1]| \leqslant \delta_{\mathsf{HSM}}.$$

   Game_2 *to* Game_3. In Game_3 the points $Q = x \cdot P$ and $R = \widehat{k}^{-1} \cdot P$ come from the EC-DSA oracle, while in Game_2 they are computed as in the real protocol. As a result, the value k_1 encrypted in c_{k_1} is unrelated to $\widehat{k}$. Let us denote $\widehat{k}_1 := \widehat{k} - \sum_{j \in S, j \neq 1} k_j$, this is the value that – if used by $\mathcal{S}$ instead of k_1 – would lead to the joint computation of $R = \widehat{k}^{-1} P$.

   To demonstrate that Game_2 and Game_3 are indistinguishable from $\mathcal{A}$'s view, we start by considering a fixed $\widehat{\mathsf{sk}_1} \in \mathbf{Z}$ satisfying the following equations:

   $$\begin{cases} \widehat{\mathsf{sk}_1} \equiv \mathsf{sk}_1 \bmod \varpi, \\ \widehat{\mathsf{sk}_1} \equiv \mathsf{sk}_1 + b_1^{-1}(k_1 - \widehat{k}_1) \bmod q, \end{cases}$$

   where ϖ is the group exponent of $\widehat{G}$, such that the order s of g_q divides ϖ. Note that the smoothness of the CL encryption scheme ensures that such a $\widehat{\mathsf{sk}_1}$ exists (it is not necessarily unique). We can now see that in Game_3, c_{k_1} is an invalid encryption of both $\widehat{k}_1$ and of k_1, for respective secret keys $\widehat{\mathsf{sk}_1}$ and sk_1, but for the same public key pk_1, indeed:

   $$\begin{aligned} c_{k_1} &= (\widetilde{u}_1, \widetilde{u}_1^{\mathsf{sk}_1} f^{k_1}) = (g_q^{r_1} f^{b_1}, (g_q^{r_1} f^{b_1})^{\mathsf{sk}_1} \cdot f^{k_1}) \\ &= (g_q^{r_1} f^{b_1}, \mathsf{pk}_1^{r_1} f^{\widehat{\mathsf{sk}_1} \cdot b_1 + \widehat{k}_1}) = (\widetilde{u}_1, \widetilde{u}_1^{\widehat{\mathsf{sk}_1}} f^{\widehat{k}_1}). \end{aligned}$$

Adversary $\mathcal{A}$ receives the point Q, the encryption key $\mathsf{pk}_1 = g_q^{\mathsf{sk}_1}$, and c_{k_1} from $\mathcal{S}$ (at this point $\mathcal{A}$ view is identical to that in Game_2). Now $\mathcal{A}$ corrupting P_j computes $c_{k_1\gamma_j}$ which we denote $c_\alpha = (u_\alpha, e_\alpha)$, and $c_{k_1 w_j}$ which we denote $c_\mu = (u_\mu, e_\mu)$. $\mathcal{A}$ then sends c_α and c_μ to $\mathcal{S}$. The difference between Game_2 and Game_3 appears now in how $\mathcal{S}$ attempts to decrypt c_α and c_μ. In Game_2 it would have used $\widehat{\mathsf{sk}}_1$, whereas in Game_3 it uses sk_1.

Notation. We denote α (resp. μ) the random variable obtained by decrypting c_α (resp. c_μ) (received in Game_3) with decryption key sk_1; we denote α' (resp. μ') the random variable obtained by decrypting c_α (resp. c_μ) (received in Game_3) with decryption key $\widehat{\mathsf{sk}}_1$; we introduce a hypothetical $\mathsf{Game}_3{}'$, which is exactly as Game_3, only one decrypts c_α (resp. c_μ) (received in Game_3) with decryption key $\widehat{\mathsf{sk}}_1$, thus obtaining α' (resp. μ'). Moreover in Game $3'$ the check performed on the curve is 'If $\mu' \cdot P + B_{1,j} \neq \widehat{k}_1 \cdot W_j$ then abort'.

Observation. The view of $\mathcal{A}$ in Game_2 and in $\mathsf{Game}_3{}'$ is identical. By demonstrating that the probability $\mathcal{A}$'s view differs when $\mathcal{S}$ uses α, μ in Game_3 from when it uses α', μ' in $\mathsf{Game}_3{}'$ is negligible, we can conclude that $\mathcal{A}$ cannot distinguish Game_2 and Game_3 except with negligible probability.

The smoothness of the CL encryption scheme tells us that given pk_1, which fixes (sk_1 mod s), the value of (sk_1 mod q) remains δ-close to the uniform distribution modulo q. In particular this ensures that $\mathcal{A}$'s view of α and α' are δ-close. Indeed, $\mathcal{A}$ receives an invalid encryption of k_1, which information theoretically masks k_1. At this point $\mathcal{A}$'s view of k_1 is that of a random variable δ-close to the uniform distribution modulo q. $\mathcal{A}$ then computes c_α which it sends to $\mathcal{S}$. Finally $\mathcal{A}$ receives either (a one way function of) k_1, or (a one way function of) some random value which is unrelated to k_1, and must decide which it received. For μ and μ', the indistinguishability of $\mathcal{A}$'s view of both random variables is a little more delicate, since $\mathcal{A}$ gets additional information from the check on the curve performed by $\mathcal{S}$, namely in Game_3 if $\mu \cdot P + B_{1,j} \neq k_1 \cdot W_j$ the simulator aborts. We call the output of this check test. And in $\mathsf{Game}_3{}'$, if $\mu' \cdot P + B_{1,j} \neq \widehat{k}_1 \cdot W_j$ the simulator aborts. We call the output of this check test'. Notice that if $\mathsf{test} = \mathsf{test}'$, both games are δ_s-close from $\mathcal{A}$'s view (the only change is in the ciphertext c_{k_1}). Let us bound the probability $\mathfrak{p}$ that $\mathsf{test} \neq \mathsf{test}'$. This will allow us to conclude that

$$|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_2]| \leq \mathfrak{p} + \delta_s.$$

Let us consider the ciphertext $c_\mu = (u_\mu, e_\mu) \in \widehat{G} \times \widehat{G}$ sent by $\mathcal{A}$. There exist unique $z_\mu \in \widehat{G}^q$, $y_\mu \in F$ such that $u_\mu = z_\mu y_\mu$. Moreover there exists a unique $b_\mu \in \mathbf{Z}/q\mathbf{Z}$ such that $y_\mu = f^{b_\mu}$.

Since $\mathsf{sk}_1 = \widehat{\mathsf{sk}}_1 \bmod \varpi$, $\mu = \perp$ if and only if $\mu' = \perp$, and this occurs when $e_\mu \cdot z_\mu^{-\mathsf{sk}_1} = e_\mu \cdot z_\mu^{-\widehat{\mathsf{sk}}_1} \notin F$. In this case Game_3 is identical to $\mathsf{Game}_3{}'$ from $\mathcal{A}$'s view ($\mathcal{S}$ aborts in both cases). We hereafter assume decryption does not fail, which allows us to adopt the following notation $e_\mu = z_\mu^{\mathsf{sk}_1} f^{h_\mu} = z_\mu^{\widehat{\mathsf{sk}}_1} f^{h_\mu}$ with $h_\mu \in \mathbf{Z}/q\mathbf{Z}$. We thus have:

$$\mu := \log_f \left(\frac{e_\mu}{u_\mu^{\mathsf{sk}_1}} \right) = h_\mu - b_\mu \mathsf{sk}_1 \bmod q$$

$$\mu' := \log_f \left(\frac{e_\mu}{u_\mu^{\widehat{\mathsf{sk}}_1}} \right) = h_\mu - b_\mu \widehat{\mathsf{sk}}_1 \bmod q$$

Thus we have

$$\mu - \mu' \equiv b_\mu(\widehat{\mathsf{sk}}_1 - \mathsf{sk}_1) \equiv b_\mu {b_1}^{-1}(k_1 - \widehat{k}_1) \bmod q.$$

We consider three cases:

(a) $\mu = \mu' \bmod q$. This may happen for two reasons:

i. If $k_1 \equiv \widehat{k}_1 \bmod q$, then Game_2 and Game_3 are identical.

ii. Else $b_\mu = 0 \bmod q$, i.e. c_μ is a valid ciphertext. Since we ruled out $k_1 \equiv \widehat{k}_1 \bmod q$ in the previous case, if test=true, necessarily test'=false, and vis versa. Both cases being symmetric, we consider the case test=true. From $\mathcal{A}$'s view, before outputting c_μ the only fixed information relative to k_1 is that contained $c_{k_1} = (g_q^{r_1} f^{b_1}, (g_q^{r_1} f^{b_1})^{\mathsf{sk}_1} f^{k_1})$. This fixes $\pi_0 := b_1 \cdot \mathsf{sk}_1 + k_1 \bmod q$. However from $\mathcal{A}$'s view, given pk_1, the random variable sk_1 follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. Thus k_1 also follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. Now suppose $\mathcal{A}$ returns $c_\mu = (z_\mu, z_\mu^{\mathsf{sk}_1} f^\mu)$ where $z_\mu \in \widehat{G}^q$. If $\mathsf{test} = \mathsf{true}$, then $\mu \cdot P + B_{1,j} = k_1 W_j$, and $\mathcal{A}$ has fixed the correct value of k_1, this occurs with probability $\leqslant 1/q + \delta_s$.

(b) $\mu \not\equiv \mu' \bmod q$ but $\mu - \mu' = w_j(k_1 - \widehat{k}_1) \bmod q$, i.e. $b_\mu = w_j b_1 \bmod q$. This results in $\mathcal{S}$ aborting on μ' in Game_2 if and only if $\mathcal{S}$ aborts on μ in Game_3. This occurs if the adversary performs homomorphic operations on c_{k_1}, and the difference between the random variables is that expected by $\mathcal{S}$. Indeed:

$$\mu = k_1 w_j - \nu_{1,j} \Leftrightarrow \mu' + w_j(k_1 - \widehat{k}_1) = k_1 w_j - \nu_{1,j} \Leftrightarrow \mu' = \widehat{k}_1 w_j - \nu_{1,j}.$$

(c) ($\mu \not\equiv \mu' \bmod q$) and ($\mu - \mu' \not\equiv w_j(k_1 - \widehat{k}_1) \bmod q$). We here consider three sub-cases:

i. Either $\mathsf{test} = \mathsf{test}' = \mathsf{false}$; this results in identical views for $\mathcal{A}$.

ii. Either $\mathsf{test}' = \mathsf{true}$; this means that:

$$\mu' = \widehat{k}_1 w_j - \nu_{1,j} \bmod q.$$

Now since $\mu - \mu' \neq w_j(k_1 - \widehat{k}_1) \bmod q$ necessarily $\mathsf{test} = \mathsf{false}$. Consequently if this event occurs, $\mathcal{A}$'s view differs. Let us prove that information theoretically, this can not happen with probability greater than $1/q + \delta_s$. To this end we consider the distribution followed by the point

$P := (\mathsf{sk}_1, \widehat{\mathsf{sk}}_1, k_1, \widehat{k}_1) \in (\mathbf{Z}/q\mathbf{Z})^4$, conditioned on $\mathcal{A}$'s view. For clarity, we first recall the expression of c_{k_1} received by $\mathcal{A}$:

$$c_{k_1} = (g_q^{r_1} f^{b_1}, \mathsf{pk}_1^{r_1} f^{\widehat{\mathsf{sk}}_1 b_1 + \widehat{k}_1})$$

where $b_1 \neq 0 \bmod q$. We also recall the expression of c_μ, sent by $\mathcal{A}$ to $\mathcal{S}$. Since c_μ decrypts to μ' with decryption key $\widehat{\mathsf{sk}}_1$, we can write:

$$c_\mu = (z_\mu f^{b_\mu}, z_\mu^{\widehat{\mathsf{sk}}_1} f^{\mu' + b_\mu \widehat{\mathsf{sk}}_1}).$$

Let us denote $\pi_0 := \widehat{\mathsf{sk}}_1 b_1 + \widehat{k}_1 \bmod q$ and $\pi_1 := \mu' + b_\mu \widehat{\mathsf{sk}}_1$. For this case to occur, it must hold that $\mu' = \widehat{k}_1 w_j - \nu_{1,j} \bmod q$, so

$$\pi_1 = \widehat{k}_1 w_j - \nu_{1,j} + b_\mu \widehat{\mathsf{sk}}_1 \bmod q.$$

Substituting $\widehat{\mathsf{sk}}_1$ for $(\pi_0 - \widehat{k}_1) b_1^{-1}$ yields:

$$\begin{aligned} \pi_1 &= \widehat{k}_1 w_j - \nu_{1,j} + b_\mu b_1^{-1}(\pi_0 - \widehat{k}_1) \bmod q \\ \Leftrightarrow \pi_1 + \nu_{1,j} - b_\mu b_1^{-1} \pi_0 &= \widehat{k}_1 (w_j - b_\mu b_1^{-1}) \bmod q \end{aligned}$$

As we dealt with $b_\mu = w_j b_1 \bmod q$ in case (b), here $w_j - b_\mu b_1^{-1}$ is invertible mod q so we can write:

$$\widehat{k}_1 = (\pi_1 + \nu_{1,j} - b_\mu b_1^{-1} \pi_0)(w_j - b_\mu b_1^{-1})^{-1} \bmod q \tag{5.1}$$

where π_0, b_1 are fixed by c_{k_1}; π_1, b_μ are fixed by c_μ; w_j is fixed by W_j; and $\nu_{1,j}$ is fixed by $B_{1,j}$. So given $\mathcal{A}$'s view and $\mathcal{A}$'s output ($B_{1,j}$ and c_μ), all the terms on the right hand side of Eq. 5.1 are fixed. However, given pk_1, c_{k_1} and W_j (which is all the relevant information $\mathcal{A}$ gets prior to outputting c_μ), the δ_s-smoothness of the CL scheme ensures that $\widehat{k}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$. If the current case occurs, Eq. 5.1 must hold, thus from being given a view where $\widehat{k}_1$ follows a distribution δ_s-close to $\mathcal{U}(\mathbf{Z}/q\mathbf{Z})$, $\mathcal{A}$ succeeds in fixing this random variable to be the exact value used by $\mathcal{S}$. This occurs with probability $\leqslant 1/q + \delta_s$.

iii. Else $\mathsf{test} = \mathsf{true}$; this means that $\mu = k_1 w_j - \nu_{1,j} \bmod q$. Since $(\mu - \mu' \neq w_j(k_1 - \widehat{k}_1) \bmod q)$ necessarily test' fails, and $\mathcal{A}$'s view differs. Reasoning as in the previous case, but setting $\pi_0 := \mathsf{sk}_1 b_1 + k_1 \bmod q$ and $\pi_1 := \mu + b_\mu \mathsf{sk}_1$, one demonstrates that this case occurs with probability $\leqslant 1/q + \delta_s$.

Combining the above, we get that $\mathsf{test}' \neq \mathsf{test}$ if and only if we are in case (a) ii. (c) ii. or (c) iii., which occurs with probability $\leqslant 3(1/q + \delta_s)$. Thus:

$$|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_2]| \leqslant 3/q + 4\delta_s.$$

Game_3 *to* Game_4. In Game_4, the first element u_1 of c_{k_1} is once again sampled in G^q. Both games are indistinguishable under the hardness of HSM and:

$$|\Pr[\mathsf{E}_4] - \Pr[\mathsf{E}_3]| \leq \delta_{\mathsf{HSM}}.$$

Game_4 *to* Game_5. In Game_5 $\mathcal{S}$ uses the public key pk_1 to encrypt k_1. The change here is exactly that between Game_0 and Game_1, both games are perfectly indistinguishable, and:

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_4]| = 0.$$

Real/Ideal executions. Putting together the above probabilities, we get that:

$$|\Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_0]| \le 2\delta_{\mathsf{HSM}} + 3/q + 4\delta,$$

which concludes the proof of the claim. □

3. As a consequence of the different values k and $\widehat{k}$, there is also a difference in the values $k_1 \cdot R$ and $\widehat{k}_1 \cdot R = P - \sum_{i\in S, i\neq 1} k_i \cdot R$ after rewinding in phase 4. However, they follow the same distribution, and they can be distinguished if k_1 and $\widehat{k}_1$ are distinguishable in MtAwc protocols. As we have seen in point 2. this happens with negligible probability. Furthermore, since we are in a semi-correct execution, in the real protocol P_1 runs normally the zero-knowledge proof for the consistency between $k_1 \cdot R$ and c_{k_1}. In the simulated protocol, the simulator just simulates the proof for $c_{k_i 1}$ and $\hat{k}_1 \cdot R$. In each case the two worlds are indistinguishable.

4. The same reasoning in previous item can be applied to $S_1 = \sigma_1$ and $S_1 = Q - \sum_{i\in S, i\neq 1} S_i$

5. $\mathcal{S}$ does not know σ_1, and thus cannot compute s_1 as in a real execution. Instead it computes $s_1 = s - \sum_{j\in S, j\neq 1} s_j = s - \sum_{j\in S, j\neq 1}(k_j m + \sigma_j r)$ where (implicitly) $s = \widehat{k}(m + rx)$. So $s_1 = \widehat{k}m + r(\widehat{k}x - \sum_{j\in S, j\neq 1}\sigma_j)$, and $\mathcal{S}$ is implicitly setting $\widehat{\sigma}_1 := \widehat{k}x - \sum_{j\in S, j\neq 1}\sigma_j$ s.t. $\widehat{k}x = \widehat{\sigma}_1 + \sum_{j\in S, j\neq 1}\sigma_j$.
   We note that, since the real execution is semi correct, the correct shares of k for the adversary are the k_i that the simulator knows and $R = \widehat{k}^{-1}P = (\widehat{k_1} + \sum_{j\in S, j\neq 1} k_j)^{-1} \cdot P$. Therefore the value s_1 computed by $\mathcal{S}$ is consistent with a correct share for P_1 for a valid signature (r, s), which makes Phase 7 indistinguishable from the real execution to the adversary.

Indistinguishability of identification procedure. We just proved that except with negligible probability, a simulated execution results in an abort if and only if a real execution would (this must be true for real and simulated views of the adversary to be indistinguishable). Of course if no abort occurs, the simulation of the identification procedure is not an issue. Now assuming there is an abort, consider the problematic types of abort listed on page 150. For an abort of type 5, identifying the culprit is trivial, and there is no impact on the view $\mathcal{A}$ has of real and simulated executions. If an abort of type 1 or 2 occurs, in all of our considered game steps $\mathcal{S}$ can honestly perform the proof for relation $\mathsf{R}_{\mathsf{Dec}}$ as would P_1, hence $\mathcal{A}$'s view of these identification procedures is identical in all game steps, and therefore in real and simulated executions.

Finally, as we are here considering the simulation of semi-correct executions aborts of type 3 and 4 do not occur (indeed, the occurrence of such aborts means we are in a non-semi-correct executions). Hence any abort which may occur in a semi-correct execution is perfectly simulated.

Non Semi-Correct Executions

Lemma 5.3.2. If the strong root assumption and C-low order assumptions hold for Gen then the simulation is computationally indistinguishable from a non-semi-correct real execution.

Proof. In this case both real and simulated executions of the protocol abort before Phase 7; so in all situations where semi-correct and non semi-correct simulations differ, in a non semi-correct execution $\mathcal{S}$ follows the protocol as would P_1; hence the non semi-correct simulation of the pre-signing sub-protocol is indistinguishable from a non semi-correct real execution.
Identification procedures. Here aborts are either of type 3 or 4. For type 3, $\mathcal{S}$ follows the real identification procedure, hence the simulation is perfect. For type 4 the only difference is that $\mathcal{S}$ simulates π_j^{log}, so $\mathcal{A}$'s view is indistinguishable.

□

Concluding the proof.

The forger $\mathcal{S}$ simulating $\mathcal{A}$'s environment can detect whether we are in a semi-correct execution or not. Consequently $\mathcal{S}$ always knows how to simulate $\mathcal{A}$'s view and all simulations are indistinguishable from real executions of the protocol. Moreover if $\mathcal{A}$, having corrupted up to t parties in the threshold ECDSA protocol, outputs a forgery, since $\mathcal{S}$ set up with $\mathcal{A}$ the same public key Q it received from its' ECDSA challenger, and randomness R it received from $\mathcal{O}^R$, $\mathcal{S}$ can use this signature as its own forgery, thus breaking the enhanced existential unforgeability of centralised ECDSA. Hence the following theorem, which captures the protocol's security, follows from Lemmas 5.3.1 and 5.3.2.

Theorem 5.3.3. Assuming ECDSA is enhanced existentially unforgeable under chosen message attacks; the strong root and C-low order assumptions hold for Gen; the CL encryption scheme is $\mathsf{ind\text{-}cpa}$-secure; and the commitment scheme is non-malleable and equivocable, it holds that the (t, n)-threshold ECDSA protocol of Fig. 5.2-5.3-5.4-5.5 is enhanced existentially unforgeable against static adversaries.

5.3.2 Security Against Adaptive Adversaries

Our protocol can further be proved secure against adaptive corruptions in the specific case $t = n - 1$, i.e. all parties must participate in the signing phases (we leave the study of adaptive security for any $t \leq n - 1$ for future work).

Theorem 5.3.4. Assuming ECDSA is $\mathsf{e-eu-cma}$; the DL assumption holds in $\mathbb{G}$; the strong root and C-low order assumptions hold for Gen; the CL encryption scheme is $\mathsf{ind\text{-}cpa}$-secure; and the commitment scheme is non-malleable and equivocable, then the $(n-1, n)$-threshold ECDSA protocol of Figure 5.7-5.3-5.4-5.5 is $\mathsf{e-tu-cma}$ against adaptive corruptions.

As explained hereafter, in the specific case $t = n - 1$, the security proof very much resembles that against static adversaries. We present the details below.

Proof strategy. In the context of adaptive corruptions the adversary $\mathcal{A}$ can choose to corrupt players throughout the execution of the protocol. When such a corruption occurs, $\mathcal{A}$ is given the corrupted party's internal state: $\mathcal{A}$ learns the party's secret values, randomness, and any other information the party may have stored from previous interactions. Hence to ensure $\mathcal{A}$ is unable to distinguish between real and simulated executions, one must ensure that $\mathcal{A}$ can not detect any inconsistencies when it chooses to corrupt a new party P. If $\mathcal{A}$ corrupts a player which *does* reveal inconsistencies, the simulator $\mathcal{S}$ rewinds the protocol. One should thus minimise the number of players possessing inconsistent values, so as to reduce the number of rewinds. In particular, if a player P is simulated as an honest player following the real protocol, then it can only give consistent values to $\mathcal{A}$ if it is corrupted. A crucial point of using the CL encryption scheme is that $\mathcal{S}$ knows the decryption keys of honest players; so if an honest player is corrupted, $\mathcal{S}$ can give this secret key to $\mathcal{A}$, which is consistent with the encryption key. This is not immediate in a situation where the secret key is not known by $\mathcal{S}$. In our proof, $\mathcal{S}$ simulates the behaviour of each honest player, revealing the relevant internal states upon corruption; it also chooses a single *special player* among all the honest ones. For all honest players which are not special, $\mathcal{S}$ runs the protocol normally. On the other hand, the role of the special player is to fix values as did P_1 in the case of static corruptions (Subsection 5.3.1). This explains why the security proof against adaptive corruptions very much resembles that for the static case, with some adaptations to deal with the dynamic corruption of players. As hinted previously, if the special player is corrupted, $\mathcal{S}$ rewinds the protocol; this rewind goes back to the beginning of the previous key refresh, where $\mathcal{S}$ chooses a new special player. Since there is only one special player, $\mathcal{S}$ rewinds at most $n-1$ times. If such a switching of special players (SSP) occurs, we assume that, for the duration of the Key Refresh, both the previous special player, and at least one of the remaining $n-1$ players remain uncorrupted. Indeed while handing over the inconsistent values from the old special player to the new one, both players possess values that are inconsistent with publicly available information. However by the end of the Key Refresh in which the SSP occurs, only the new special player is inconsistent; and we can thereafter again handle $n-1$ corruptions.

We stress that since all precomputed pre-signatures are erased at every Key Refresh,[6] the aforementioned rewind does not introduce the risk that $\mathcal{A}$ may request the signature of two different messages with the same randomness.

Key Generation for $t = n - 1$. As suggested in [CGG+20], for $t = n - 1$, key generation can be simplified by using an additive sharing instead of a Feldman-VSS. This improves the protocols' communication cost, speed, and simplifies the security proof. This simplified Key Generation sub-protocol is depicted in Figure 5.7.

On the adaptive security of Feldman-VSS. We here give an idea why attaining security against adaptive adversaries for our full threshold protocol (any $t < n$) is considerably more challenging. As mentioned above, to guarantee adaptive security, $\mathcal{S}$ must be capable of providing a consistent internal state whenever $\mathcal{A}$ corrupts an honest

[6]In fact all randomness and data used in the previous refreshment phase is erased, except for the information that the protocol specifies should be used afterwards.

$$
\begin{array}{ccc}
P_i & \mathsf{IKeyGen}(\mathbb{G}, P, q) & \text{All players } \{P_j\}_{j \neq i} \\
\hline
w_i \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z};\ W_i \leftarrow w_i P & & \\
[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(W_i) & & \\
(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{CL.KeyGen}(1^\lambda) & \xRightarrow{\mathsf{pk}_i \text{ and } \mathsf{kgc}_i} & \\
 & \xRightarrow{\mathsf{kgd}_i} & Q = \sum_{i=1}^{n} W_i \\
\pi_i^{\mathsf{kg}} := \mathsf{ZKPoK}_{W_i}\{(w_i) : W_i = w_i P\} & \xRightarrow{\pi_i^{\mathsf{kg}}} & \\
\hline
\end{array}
$$

Figure 5.7: Key Generation protocol when $t = n - 1$

player. We point out that using Feldman-VSS causes issues in the simulation, since the simulator – simulating the special player P_* – computes a polynomial $p_*(X)$, for which it can give at most t consistent shares which pass the verification check. Indeed, $t + 1$ shares define in a unique way $p_*(X)$, which has an unknown degree zero coefficient. As a result, $\mathcal{S}$ will send t consistent shares and $n - t$ inconsistent shares with overwhelming probability. In the case of static corruptions this is not a problem since the consistent shares are given to the adversary, while the inconsistent ones are given to the honest players, that will not be corrupted. In contrast, if the adversary is adaptive, $\mathcal{S}$ does not know which players $\mathcal{A}$ will corrupt, so it sends the t consistent values to t random players. If the t players $\mathcal{A}$ chooses to corrupt do not coincide with the t players having received consistent values, then $\mathcal{A}$ has corrupted a player with an inconsistent internal state and hence distinguishes real and simulated executions. As explained in the previous *proof strategy* paragraph, each time an honest player is corrupted revealing inconsistent values, $\mathcal{S}$ rewinds the protocol. Therefore $\mathcal{S}$'s running time (which must be polynomial for security to hold) grows exponentially with $n - t$, reaching its maximum in $t = n/2$. For $t = n - 1$ (the setting we consider), there is only one inconsistent share, that of the special player, hence the number of potential rewinds is n in expectation, i.e. it remains reasonable.

On the number of rewindings. To understand why the number of rewindings is exponential with respect to the number of parties and the threshold, consider initially the case $t = n/2$. $\mathcal{S}$ can select $n/2$ players in $\binom{n}{n/2} = \frac{n!}{(n/2!)^2}$. Using Stirling approximation of the binomial coefficient, i.e. $n! \approx \sqrt{2\pi n} \cdot (\frac{n}{e})^n$, we have $\binom{n}{n/2} = \frac{n!}{((n/2)!)^2} \approx \sqrt{\frac{2}{\pi}} \cdot \frac{2^n}{\sqrt{n}} = \Theta\left(\frac{2^n}{\sqrt{n}}\right)$, which is esponential with respect to $n - t = n/2$. If we consider a more general case $t = n/\alpha$, $\alpha \in \left]\frac{n}{n-1}, n\right] \cap \mathbf{R}$, i.e. representing t as a fraction of n, the resulting approximation is $\frac{\alpha}{\sqrt{2\pi}} \frac{\alpha^n}{\sqrt{n}(\alpha-1)^{n-n/\alpha+1/2}}$. Just to give an example, if we consider $n = 128$ and $t = 4$ ($\Rightarrow \alpha = 32$), $t = 16$ ($\Rightarrow \alpha = 8$) and $t = 64$ ($\Rightarrow \alpha = 2$), we need in expectation about $2^{24}, 2^{66}$ and 2^{124} rewindings, respectively. Since we consider a threshold of $t = n - 1$ parties for our security against adaptive corruption, then $\binom{n}{t} = n$.[7]

[7]The calculation is easy to do manually, it is not necessary to use Stirling approximation.

Proof of Theorem 5.3.4

Notation. Before proving the theorem, let us introduce some notations. The sets of the indices of all players and all corrupted players are denoted $\mathbf{P}$ and $\mathbf{C}$ respectively. At the beginning of the experiment, the simulator randomly chooses an honest player P_* that is henceforth referred to as the special player. The set $\mathbf{H}$ contains indices of all honest players except P_*, while $\mathbf{NC}$ contains indices of all non corrupted players including P_*. Hence $\mathbf{H} = \mathbf{P} \setminus (\mathbf{C} \cup \{P_*\})$ and $\mathbf{NC} = \mathbf{P} \setminus \mathbf{C}$. The sets $\mathbf{H}, \mathbf{C}, \mathbf{NC}$ are dynamically updated with new corruptions throughout the protocol. In particular, $\mathbf{C}$ grows in size with the condition that $|\mathbf{C}| \leqslant n-1$, taking elements from $\mathbf{NC}$. Note that if P_* is corrupted, the simulator will rewind the protocol and choose a different special player. Clearly if some P is corrupted and $\mathbf{NC} \leftarrow \mathbf{NC} \setminus \{P\}$ then $\mathbf{H} \leftarrow \mathbf{H} \setminus \{P\}$. Finally, all values belonging to the special player P_* are indexed with the symbol $*$.

Simulating Key Generation.

1. $\mathcal{S}$ receives a public key Q from it's ECDSA challenger.
2. For $i \in \mathbf{NC}$, $\mathcal{S}$ samples $w_i \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and computes $[\mathsf{kgc}_i, \mathsf{kgd}_i] \leftarrow \mathsf{Com}(w_i P)$.
3. For $i \in \mathbf{NC}$, $\mathcal{S}$ samples CL encryption key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\lambda)$.
4. $\mathcal{S}$ broadcasts $\{\mathsf{kgc}_i\}_{i \in \mathbf{NC}}$ and $\{\mathsf{pk}_i\}_{i \in \mathbf{NC}}$, before receiving $\{\mathsf{kgc}_j\}_{j \in \mathbf{C}}$ and the public keys $\{\mathsf{pk}_j\}_{j \in \mathbf{C}}$ from $\mathcal{A}$.
5. $\mathcal{S}$ broadcasts $\{\mathsf{kgd}_i\}_{i \in \mathbf{NC}}$ and receives $\{\mathsf{kgd}_j\}_{j \in \mathbf{C}}$. For $i \in \mathbf{P}$, let $W_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$ be the revealed commitment value of each party.
6. $\mathcal{S}$ chooses a special player P_* and rewinds $\mathcal{A}$ to the decommitment step, so as to equivocate P_*'s commitment to $\widehat{\mathsf{kgd}}_*$ which decommits to $\widehat{W}_* := Q - \sum_{j \neq *} W_j$.
7. $\mathcal{S}$ simulates π_*^{kg} (the ZKPoK that it knows w_* corresponding to $\widehat{W}_*$) and honestly performs the proofs π_i^{kg} for $i \in \mathbf{H}$. Then, for $j \in \mathbf{C}$, $\mathcal{S}$ receives from $\mathcal{A}$ a ZKPoK of w_j satisfying $W_j := w_j \cdot P$; from which $\mathcal{S}$ can extract w_j.

Simulating Key Refresh. In the event of a normal Key Refresh (i.e. which is not due to $\mathcal{S}$ rewinding to switch special players), $\mathcal{S}$ simply runs the real Key Refresh sub-protocol for all players in $\mathbf{NC}$.

Switching Special Players in Key Refresh. As explained in the paragraph entitled *Proof Strategy* of Subsection 5.3.2, if at any point during the simulation $\mathcal{A}$ corrupts the special player P_*, then $\mathcal{S}$ rewinds the adversary and chooses a new special player P_*^{new} among the honest parties P_i for $i \in \mathbf{H}$. We will hereafter refer to this particular simulation of the Key Refresh protocol as *Key Refresh with special player switch* (KRSS). At the end of the KRSS, the previous special player P_* has consistent values, so that P_*^{new} is the unique inconsistent (i.e. special) player. Note that throughout KRSS, we assume P_* is not corrupted, and $\mathcal{A}$ can corrupt at most $n-2$ of the remaining $n-1$ players.

Without loss of generality we set $P_1 := P_*$ and $P_2 := P_*^{\mathsf{new}}$. For $i \in \mathbf{P}$ we denote $u_i \in \mathbf{Z}/q\mathbf{Z}$ the secret share of the ECDSA signing key x owned by P_i from the previous Key Refresh; and $Q_i := u_i P$. Recall that $\mathcal{S}$ does not know u_1. If a KRSS occurs, $\mathcal{S}$ simulates P_1 and P_2 in the following way (the simulation remains the same for other players):

- Sample $v_{1,1}, \dots, v_{1,n}$ and $v_{2,1}, \dots, v_{2,n}$ as per the protocol.
- Sample a random $\alpha \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ and let $\beta := v_{1,1} + v_{1,2} - \alpha$.
  Then set $Q_{1,1} := -Q_1 + \alpha P$, $Q_{1,2} := Q_1 + \beta \cdot P$ and for each $j \in \mathbf{P}, j > 2$ set $Q_{1,j} := v_{1j} \cdot P$.
- Compute $Q_{2,j} := v_{2,j} \cdot P$ for all $j \in \mathbf{P}, j \neq 2$ as in the real protocol.
- For $j \in \mathbf{P}$ compute ciphertexts $C_{1,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, v_{1,j})$ and $C_{2,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, v_{2,j})$ as per the protocol. Simulate proofs $\pi^{\mathsf{kr}}_{1,1}$ and $\pi^{\mathsf{kr}}_{1,2}$, but run all other proofs $\{\pi^{\mathsf{kr}}_{1,j}\}_{j\in\mathbf{P},j>2}$ and $\{\pi^{\mathsf{kr}}_{2,j}\}_{j\in\mathbf{P}}$ as in the real protocol.
- After having received all the $\{Q_{i,j}\}_{i\in\mathbf{P},i>2,j\in\mathbf{P}}$, $\mathcal{S}$ computes $Q_i^{\mathsf{new}} = Q_i + \sum_{j\in\mathbf{P}} Q_{j,i}$ for each $i \neq 2$. It then rewinds the sub-protocol and changes $Q_{2,2}$ to

$$Q_{2,2} := Q - \sum_{i\neq 2} Q_i^{\mathsf{new}} - Q_2 - \sum_{i\neq 2} Q_{i,2}^{\mathsf{new}}$$

  With this choice of $Q_{2,2}$, $Q_2^{\mathsf{new}} = Q_2 + \sum_i Q_{i,2}$ is such that $Q = \sum_{i\in\mathbf{P}} Q_i^{\mathsf{new}}$.
- Erase all values $v_{i,j}$ and $Q_{i,j}$

Notice that with this choice of Q_1^{new}, there are no inconsistencies for P_1 and it knows the discrete log of its' point. Furthermore, thanks to the values α and β the elliptic curve points computed in an unusual way are distributed as in a real execution of the protocol.

Simulating protocols Pre-Sign and Sign. After the key generation is over, the simulator must handle the signature queries issued by $\mathcal{A}$. Recall that $\mathcal{A}$ can issue two types of queries:

- oracle $\mathcal{O}^R$ to obtain a uniformly random point $R = (r_x, r_y)$ in $\mathbb{G}$:
- oracle $\mathcal{O}^{\mathsf{Sign}(\mathsf{sk},m;R)}$ which on input a message m chosen by $\mathcal{A}$, returns a valid signature (r, s) for m where $r := r_x \bmod q$ if $R = (r_x, r_y)$ was queried to $\mathcal{O}^R$; else it returns $\perp$.

The simulator simulates P_i for each $i \in \mathbf{NC}$ in the threshold signature protocol on input R for the offline phase (Phases 1-6), and a correct signature (r, s) for m under the public key Q for the online phase 7. We stress that though the simulator knows the decryption key sk_*, and P_*'s ECDSA public key share $W_* = w_* \cdot P$; it does not know w_*. However the simulator knows the shares w_i of all other players ($i \in \mathbf{P} \setminus \{*\}$) from the Schnorr proofs in Key Generation phase (for $i \in \mathbf{C}$) or because it computed them (for $i \in \mathbf{H}$).

The simulation of the Pre-Signing and Signing protocols is based on [CGG$^+$20] and [CCL$^+$20], with adaptations considering previously defined dynamic sets of players $(\mathbf{NC}, \mathbf{H}, \mathbf{C})$. For each execution all parties in $\mathbf{P}$ participate. This implies that $\{w_i\}_{i\in[n]}$ are long term secrets. In the following simulation $\mathcal{S}$ aborts whenever the protocol is supposed to abort, i.e., whenever $\mathcal{A}$ refuses to decommit a committed value, a ZK proof fails, a check does not pass or if the signature (r, s) does not verify.

Simulating Pre-signing and Signing - Description of $\mathcal{S}$: For all $i \in \mathbf{H}$, i.e. honest – but not special – players P_i, $\mathcal{S}$ just runs the protocol as would P_i in a real execution. Hence in the following phases we only describe how $\mathcal{S}$ simulates P_*.

Phase 1: $\mathcal{S}$ samples $k_*, \gamma_* \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$, $r_* \xleftarrow{\$} [\widetilde{A}]$ uniformly at random. It computes $c_{k_*} \leftarrow \mathsf{Enc}(\mathsf{pk}_*, k_*; r_*)$, the associated ZKAoK π_*, and $[\mathsf{c}_*, \mathsf{d}_*] \leftarrow \mathsf{Com}(\gamma_* P)$. It broadcasts $\mathsf{c}_*, c_{k_*}, \pi_*$ before receiving $\{\mathsf{c}_j, c_{k_j}, \pi_j\}_{j\in\mathbf{C}}$ from $\mathcal{A}$. $\mathcal{S}$ checks the proofs are valid and extracts the encrypted values $\{k_j\}_{j\in\mathbf{C}}$ from which it computes $k := \sum_{i\in\mathbf{P}} k_i$.

Phase 2: Recall that during the regular run of the protocol, P_* will engage in two MtA protocols and two MtAwc protocols with each other player $P_j, j \in \mathbf{P} \setminus \{*\}$ (the corrupted players and other honest players in $\mathbf{P}$). $\mathcal{S}$ runs the protocol for P_* as follows:

(a) *Initiator for MtA with* $k_i, i \in \mathbf{NC}$ *and* $\gamma_j, j \in \mathbf{P} \setminus \{i\}$: $\mathcal{S}$ runs the real sub-protocol, as it knows k_i. For $j \in \mathbf{P} \setminus \{*\}$, $\mathcal{S}$ decrypts the ciphertext received from P_j obtaining $\alpha_{*,j} \mod q$ (for $j \in \mathbf{NC} \subset \mathbf{P}$ it already knows the values, however P_j may be corrupted in this phase, so $\mathcal{S}$ runs the real protocol, even between non corrupted parties).

(b) *Respondent for MtA with* $k_j, j \in \mathbf{P} \setminus \{*\}$ *and* γ_*: $\mathcal{S}$ runs the real sub-protocol, as it knows γ_*.
Recall that $\mathcal{S}$ knows k_j from extraction in Phase 1, it also knows its own shares $\beta_{j,i}$ for $i \in \mathbf{NC}$, hence $\mathcal{S}$ can compute P_j's shares $\alpha_{j,i} = k_j\gamma_i - \beta_{j,i} \mod q$.

(c) *Initiator for MtAwc with* k_* *and* $w_j, j \in \mathbf{P} \setminus \{*\}$: $\mathcal{S}$ runs the real sub-protocol, as it knows k_i for $i \in \mathbf{NC}$. Notice that $\mathcal{S}$ chose w_i for $i \in \mathbf{H}$ as in the real protocol, while for $j \in \mathbf{C}$, $\mathcal{S}$ extracted w_j from π_j^{kg} in KeyGen. The only unknown share of x is the special player's w_*. For $j \in \mathbf{P}\setminus\{*\}$, $\mathcal{S}$ runs the real sub-protocol; decrypting $c_{k_* w_j}$ to obtain $\mu_{*,j}$; and checking that $\mu_{*,j}P + B_{*,j} = k_* W_j$. If so, since $\mathcal{S}$ also knows k_* and w_j, it computes $\nu_{*,j} = k_* w_j - \mu_{*,j} \mod q$.

(d) *Respondent for MtAwc with* $k_j, j \in \mathbf{P} \setminus \{*\}$ *and* w_*: $\mathcal{S}$ knows $W_* = w_* \cdot P$ but not w_*, so it samples a random $\mu_{j,*} \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ and sets $c_{k_j w_*} \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mu_{j,*})$, and $B_{j,*} := k_j \cdot W_* - \mu_{j,*} \cdot P$. Finally $\mathcal{S}$ sends the cipertexts and the point.

Note that at this point $\mathcal{S}$ knows:

- k_i for each $i \in \mathbf{P}$, w_j for each $j \in \mathbf{P} \setminus \{*\}$
- $\alpha_{i,j}, i \in \mathbf{NC}, j \in \mathbf{P} \setminus \{i\}$ as initiator for MtA, $\alpha_{j,i}, \beta_{j,i}, i \in \mathbf{NC}, j \in \mathbf{P} \setminus \{i\}$ as respondent for MtA

- $\mu_{i,j}, \nu_{i,j}, i \in \mathbf{NC}, j \in \mathbf{P}\backslash\{i\}$ as initiator for MtAwc, $\mu_{j,i}, \nu_{j,i}, i \in \mathbf{NC}, j \in \mathbf{P}\backslash\{i\}$ as respondent for MtAwc

$\mathcal{S}$ computes δ_* for the special player and δ_i for $i \in \mathbf{H}$ as per protocol.

Note that $\mathcal{S}$ does not know the internal values from the MtA and MtAwc protocols executed by two players that are both controlled by the adversary. Thus $\mathcal{S}$ is not able to compute the individual values σ_j and δ_j for $j \in \mathbf{C}$; nor can $\mathcal{S}$ compute σ_* since it doesn't know the value w_*. However $\mathcal{S}$ can compute:

$$\begin{aligned}
\sigma_{\mathbf{C}} = \sum_{i\in\mathbf{C}} \sigma_i = \sum_{i\in\mathbf{C}} (k_i w_i + \sum_{j\in\mathbf{P}\backslash\{i\}} \mu_{i,j} + \sum_{j\in\mathbf{P}\backslash\{i\}} \nu_{j,i}) \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{P}\backslash\{i\}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} k_i w_i \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{C}, j\neq i} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} k_i w_i \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{C}, j\neq i} (\mu_{i,j} + \nu_{i,j}) + \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} k_i w_i \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{C}, j\neq i} k_i w_j + \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} k_i w_i \\
&= \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{NC}} (\mu_{i,j} + \nu_{j,i}) + \sum_{i\in\mathbf{C}} \sum_{j\in\mathbf{C}} k_i w_j
\end{aligned}$$

since it knows all the values $\{k_j\}_{j\in\mathbf{P}}$, $\{w_j\}_{j\in\mathbf{P}, j\neq *}$, $\mu_{i,j}$ and $\nu_{j,i}$ in MtAwc with the honest players and $\mu_{i,*}$, $\nu_{*,i}$ from special player.
Furthermore, up until the moment $\sigma_{\mathbf{C}}$ is used to check whether the execution is semi-correct or not, every time a player P_i for some $i \in \mathbf{NC}$ is corrupted, $\mathcal{S}$ updates $\sigma_{\mathbf{C}} \leftarrow \sigma_{\mathbf{C}} + \sigma_i$. If the special player is corrupted, the simulator rewinds, and $\sigma_{\mathbf{C}}$ is recomputed.

Phase 3: $\mathcal{S}$ broadcasts δ_* and receives $\{\delta_j\}_{j\in\mathbf{C}}$ from $\mathcal{A}$. Let $\widetilde{\delta} := \sum_{i\in\mathbf{P}} \delta_i$. $\mathcal{S}$ broadcasts $T_* = \sigma_* \cdot P + \ell_* \cdot H$ ($\mathcal{S}$ can compute T_* since it knows $\sigma_* \cdot P$). As $\mathcal{S}$ does not know σ_*, it simulates the ZK proof $\widetilde{\pi}_*$. Next, $\mathcal{S}$ extracts values $\widehat{\sigma}_j, \widehat{\ell}_j$ for $j \in \mathbf{C}$ from the proofs $\widetilde{\pi}_j$ received from $\mathcal{A}$. Let $\widehat{\sigma}_{\mathbf{C}} := \sum_{j\in\mathbf{C}} \widehat{\sigma}_j$. Here again $\widehat{\sigma}_{\mathbf{C}}$ is updated to include $\widehat{\sigma}_i$ if a player P_i is adaptively corrupted for $i \in \mathbf{H}$.

Phase 4: $\mathcal{S}$ broadcasts d_* which decommits to Γ_*, and for all $j \in \mathbf{C}$, $\mathcal{A}$ reveals d_j which decommits to Γ_j. $\mathcal{S}$ honestly performs the ZK proof π^γ_*; and receives π^γ_j, from which $\mathcal{S}$ can extract γ_j. These are consistent with the values used in Phase 1 thanks to the binding property of the commitment scheme. Now $\mathcal{S}$ can compute

$$\delta = (\sum_{i\in\mathbf{P}} k_i) \cdot (\sum_{i\in\mathbf{P}} \gamma_i) = k\gamma, \quad \text{where} \quad \gamma = \sum_{i\in\mathbf{P}} \gamma_i \cdot P.$$

Note that $\mathcal{A}$ may have used different values $\widetilde{\gamma}_j$ in the MtA protocol than the γ_j extracted here, hence we denote them with a tilde. At this point $\mathcal{S}$ can detect if the values published so far by $\mathcal{A}$ are consistent (the sum of the γ_j, not each individual

γ_j); note that $\mathcal{S}$ will behave differently in Phases 5, 6 and 7 depending on this detection. To detect inconsistencies, $\mathcal{S}$ first computes

$$\widetilde{R} = \widetilde{\delta}^{-1} \cdot \sum_{i \in \mathbf{P}} \Gamma_i.$$

Then using the values $\{k_j\}_{j \in \mathbf{C}}$ extracted in Phase 1, and its own values $\{k_i\}_{i \in \mathbf{NC}}$, $\mathcal{S}$ checks if $\sum_{i \in \mathbf{P}} k_i \cdot \widetilde{R} = P$. If equality holds then $\widetilde{R} = k^{-1} \cdot P$ and $\widetilde{\delta} = k\gamma = \delta$.

The simulator can also detect if the values σ_j computed in Phase 2 are consistent with those used to compute points T_j in Phase 3; in particular $\mathcal{S}$ checks that $\widehat{\sigma}_{\mathbf{C}} = \sigma_{\mathbf{C}}$. We thus distinguish two types of executions: an execution is said to be *semi-correct* if

$$\sum_{i \in \mathbf{P}} k_i \widetilde{R} = P \qquad \text{and} \qquad \widehat{\sigma}_{\mathbf{C}} = \sigma_{\mathbf{C}}.$$

Conversely, if either of the above equalities do not hold, the execution is said to be *non semi-correct.*

Note that using EC points to check the consistency of δ and $\widetilde{\delta}$ avoids the need for proofs of affine transformation which were necessary in [CGG+20] to attain security against malicious adversaries.

Now $\mathcal{S}$ adapts its behaviour depending on the type of execution:

- **Semi-correct execution:**
  (a) $\mathcal{S}$ invokes oracle $\mathcal{O}^R$ to obtain $R = (r_x, r_y)$.
  (b) $\mathcal{S}$ sets $\widehat{\Gamma}_* := \widetilde{\delta} \cdot R - \sum_{i \in \mathbf{P}, i \neq *} \Gamma_i$, so that $R = \widetilde{\delta}^{-1} \left(\widehat{\Gamma}_* + \sum_{i \in \mathbf{P}, i \neq *} \Gamma_i \right)$. Then $\mathcal{S}$ rewinds $\mathcal{A}$ to the decommitment step in Phase 4, and equivocates P_*'s commitment so that it decommits to $\widehat{\Gamma}_*$ instead of Γ_*.
- **Non semi-correct execution:** $\mathcal{S}$ simply moves on to Phase 5.

Phase 5:
- **Semi-correct execution:** $\mathcal{S}$ publishes $\bar{R}_* = P - \sum_{i \in \mathbf{P} \setminus \{*\}} k_i \cdot R$ together with π'_*: a simulated ZKP of consistency with $c_{k_*} = \mathsf{Enc}(\mathsf{pk}_*, k_*; r_*)$ (note that in this case $\bar{R}_* \neq k_* \cdot R$ due to the rewinding).
- **Non semi-correct execution:** $\mathcal{S}$ publishes $\bar{R}_* := k_* \cdot R$ together with π'_*: a real ZKP of consistency with c_{k_*} (this needn't be simulated).

Phase 6:
- **Semi-correct execution:** $\mathcal{S}$ publishes $S_* := Q - \sum_{j \in \mathbf{P} \setminus \{*\}} \sigma_j R$ together with π''_*: a simulated ZKP of consistency with T_* (again in this case the simulated $S_* \neq \sigma_* \cdot R$ due to the rewinding).
- **Non semi-correct execution:** $\mathcal{S}$ publishes $S_* := \sigma_* R$ together with π''_*: a real ZKP of consistency with T_* (this needn't be simulated).

In a non semi-correct execution, at least one of the the adversary's proofs π'_j or π''_j for some $j \neq *$ will fail, and the protocol will abort.

Phase 7: $\mathcal{S}$ invokes the second oracle $\mathcal{O}^{\mathsf{Sign}(\mathsf{sk}, m; R)}$ with input m and R, where R was computed in one of the previous offline phases (in particular in one that was semi-correct, since it concluded successfully). In return, $\mathcal{S}$ receives the valid signature (r, s) on m, where $r = r_x \bmod q$.

At this point $\mathcal{S}$ knows $s_{\mathbf{C}} = \sum_{j \in \mathbf{C}} s_j$ (i.e., the summed value of all the s_j held by the corrupted players) because $s_{\mathbf{C}} = k_{\mathbf{C}} m + \sigma_{\mathbf{C}} r$ where $\sigma_{\mathbf{C}}$ is as defined in the simulation of Phase 2 and $k_{\mathbf{C}} = \sum_{j \in \mathbf{C}} k_j$. As in the static case, if $\mathcal{A}$ cheats in Phase 7 – denoting $\{\tilde{s}_i\}_{i \in \mathbf{C}}$ the values that $\mathcal{S}$ receives from $\mathcal{A}$ in Phase 7, and $\tilde{s}_{\mathbf{C}} := \sum_{i \in \mathbf{C}} \tilde{s}_i$ – it is possible that $s_{\mathbf{C}} \neq \tilde{s}_{\mathbf{C}}$. $\mathcal{S}$ also knows $s_{\mathbf{H}} = \sum_{i \in \mathbf{H}} s_i$ since it honestly ran the protocol for $i \in \mathbf{H}$. So $\mathcal{S}$ computes the share s_* consistent with (r, s) and $s_{\mathbf{H} \cup \mathbf{C}}$ as $s_* := s - s_{\mathbf{H} \cup \mathbf{C}}$. Finally, $\mathcal{S}$ broadcasts this value s_*.

Note on the dynamic sets. Since the set of honest and corrupted players may change throughout the protocol, if $\mathcal{S}$ has computed σ_i as an honest P_i, and P_i is subsequently corrupted, one can simply consider $i \in \mathbf{C}$, instead of $i \in \mathbf{H}$ and nothing changes. This is because once P_i is corrupted, it will be considered as malicious, with the difference that its σ_i was computed by $\mathcal{S}$ as opposed to being extracted. The proofs and checks of Phases 4, 5, and 6 ensure that σ_i does not change before Phase 7.

Simulating Identification of aborts in Key Generation – Description of $\mathcal{S}$. If an abort occurs in the Key Generation protocol, $\mathcal{S}$ runs the identification protocol as would an honest P_i for each $i \in \mathbf{NC}$ (i.e. as described in Section 5.2). Furthermore, if some player P raises a compliant against P_* (simulated by $\mathcal{S}$), then P is detected as a cheater since the simulation key generation is done in such a way that corrupted players receive values which pass the verification check.

Simulating Identification of aborts in Pre-Sign and Sign – Description of $\mathcal{S}$. For all $i \in \mathbf{H}$, i.e. honest – but not special – players P_i, $\mathcal{S}$ just runs the identification procedure as would P_i in a real execution. Hence in the following phases we only describe how $\mathcal{S}$ simulates P_*. Consider the problematic types of abort listed on page 150. For an abort of type 1, 3 or 5 occurs, $\mathcal{S}$ runs the real identification procedure.

If an abort of type 2 occurs due to $\mathcal{S}$ announcing that the check on $\mu_{*,j}$ fails (for some $j \in \mathbf{C}$), it runs the real identification procedure. Conversely, if some player P_j for $j \in \mathbf{C}$ complains about the $\mu_{j,*}$ it received, observe that: if $\mu_{j,*}$ is the real decryption of $c_{k_j w_*}$ (which it must be if the proof for $\mathsf{R}_{\mathsf{Dec}}$ provided by P_j is valid), then since the point $B_{j,*}$ sent by $\mathcal{S}$ to P_j was computed as $B_{j,*} := k_j \cdot W_* - \mu_{j,*}$, necessarily the equality test will pass. Observe that the value $\nu_{j,*}$ remain secret in this identification protocol; hence no other (corrupted) party can check that $\mathcal{S}$ knows $\nu_{j,*}$ such that $B_{j,*} = \nu_{j,*} \cdot P$, and the simulation remains undetected.

If an abort of type 4 occurs, $\mathcal{S}$ follows the real procedure for aborts up until it needs to prove knowledge of σ_* such that $S_* = \sigma_* \cdot R$. Since $\mathcal{S}$ does not know σ_*, it simulates the proof π^*_{log}.

5.3.2.1 Indistinguishability of real and simulated executions against adaptive adversaries

The simulation of a semi-correct execution

Lemma 5.3.5. Assuming the strong root and C-low order assumptions hold for Gen; the CL encryption scheme is δ_s-smooth; the HSM problem is δ_{HSM}-hard; and the commitment scheme is non-malleable and equivocable; then on input m the simulation either outputs a valid signature (r, s) or aborts, and is computationally indistinguishable from a real semi-correct execution.

The proof of Lemma 5.3.5 very much resembles that of Lemma 5.3.1. Hence many details are here omitted.

Proof. Since, in all considered protocols, $\mathcal{S}$ simulates parties P_i for $i \in \mathbf{H}$ by running the real protocol exactly as would P_i one only needs to prove that $\mathcal{S}$'s simulation of P_* is indistinguishable from a real execution.

Indistinguishability of identification procedure in semi-correct executions. This follows immediately from the static case; it suffices to replace P_1 with P_* in the relevant paragraph in proof of Lemma 5.3.1.

Indistinguishability of Key Generation and Key refresh. For Key Generation, indistinguishability follows immediately from the static case (replacing 1 with *). Regarding Key Refresh, as long as there is no switching of special player, the simulator runs the real protocol, and the simulation is perfect. Conversely, in a KRSS, all players which are not the old or new special player are consistent; as long as neither of these is corrupted during KRSS, the simulation is perfect. If the newly chosen special player is corrupted, $\mathcal{S}$ rewinds again. And it is assumed that during a KRSS, the old special player is not corrupted.

Indistinguishability of signature protocol in semi-correct executions. The differences between $\mathcal{A}$'s real and simulated views are the following:

1. $\mathcal{S}$ does not know w_* so it cannot compute $\{c_{k_j w_*}\}_{j \in \mathbf{P} \setminus \{*\}}$ as in a real execution of the protocol. However as in the static case (replacing '1' with '*'), $\mathcal{S}$ can extract k_j from π_j for each $j \in \mathbf{C}$ and it knows k_j for each $j \in \mathbf{H}$. It then computes the problematic ciphertexts as in the static case (*cf.* Lemma 5.3.1), and – as argued there – $\mathcal{A}$'s real and simulated view of these ciphertexts follow identical distributions.

2. $\mathcal{S}$ computes $\widehat{\Gamma}_* := \widetilde{\delta} \cdot R - \sum_{i \neq *} \Gamma_i$, and equivocates its commitment c_* s.t. d_* decommits to $\widehat{\Gamma}_*$. Once again, the proof that this change is not noticeable to $\mathcal{A}$ is identical to the static case (replacing '1' with '*', and the set S with all players $\mathbf{P}$). And using the same reasoning as in proof of Lemma 5.3.1, one can demonstrate that the following claim holds:

   Claim 4. If the CL encryption scheme is δ_s-smooth and the HSM problem is δ_{HSM}-hard, then no probabilistic polynomial time adversary $\mathcal{A}$ – interacting with $\mathcal{S}$ – can notice the value of k_* in the computation of R being replaced by the (implicit) value $\widehat{k}$ with probability greater than $2\delta_{\mathsf{HSM}} + 3/q + 4\delta_s$.

   Hence from the smoothness of the CL scheme, and the hardness of the HSM problem, this change is unnoticeable to $\mathcal{A}$.

3. Let us denote $\widehat{k} \in \mathbf{Z}/q\mathbf{Z}$ the randomness (unknown to $\mathcal{S}$) used by oracle $\mathcal{O}^R$ to produce R. With overwhelming probability, $k \neq \widehat{k}$. Hence there is also a difference in the values $k_* \cdot R$ and $\widehat{k}_* \cdot R = P - \sum_{i \neq *} k_i \cdot R$ after the rewind in phase 4. For the same reasons as discussed in proof of Lemma 5.3.1, item 3. (*i.e.* smoothness of encryption scheme and simulatability of the ZKP π_*), this change is indistinguishable to $\mathcal{A}$.

4. The same reasoning as in the previous item can be applied to $S_* = \sigma_* \cdot R$ and $S_* = Q - \sum_{i \neq *} S_i$.

5. $\mathcal{S}$ does not know σ_*, and thus cannot compute s_* as in a real execution. However, as in the static case, since we are in a semi-correct execution the value s_* computed by $\mathcal{S}$ is consistent with a correct share for P_* for a valid signature (r, s), which makes the simulation of Phase 7 indistinguishable from a real execution from $\mathcal{A}$'s view.

□

Non semi-correct executions

Once again, the proof of Lemma 5.3.6 is essentially identical to that in the static case, substituting P_1 for P_*.

Lemma 5.3.6. Assuming the strong root and C-low order assumptions hold for Gen; it holds that the view of the simulation, from an adaptive adversary's view, is computationally indistinguishable from a non-semi-correct real execution.

5.3.2.2 Concluding the proof

Combining Lemmas 5.3.5 and 5.3.6, it holds that if the strong root an C-low order assumptions hold for Gen; the CL encryption scheme is ind-cpa-secure and the commitment scheme is non malleable and equivocable, then the $(n-1, n)$ ECDSA protocol described in Figures 5.7, 5.3, 5.4 and 5.5 is enhanced threshold existentially unforgeable against adaptive adversaries.

5.4 Efficiency comparisons

We here compare the theoretical complexity of our protocol to that of [CGG+20] for the standard NIST curve P-256 corresponding to 128 security level. For the encryption scheme, we start with a 112 bit security as in [CGG+20], but also study the case where its level of security matches that of the elliptic curve.

The figures we provide count the number of group and ring elements which are both sent and received from a given party, including broadcasts; whereas the figures provided in [CGG+20, Fig 1] only include the data sent from one player to another. We focus on pre-signing and signing sub-protocols; these are the most critical as they will be most frequently executed.

We compute the communication costs for both protocols presented in [CGG+20]; one which benefits of only having three rounds, and their six round protocol which benefits

Protocol	Curve size	λ (bits)	Δ_K (bits)	N (bits)	Total Signing (KBytes)
Canetti et al.'s 6 rounds	256	112	-	2048	$31.3t + 1.0$
Canetti et al.'s 3 rounds	256	112	-	2048	$31.6t + 1.3$
Ours	256	112	1348	-	$\mathbf{3.4t + 2.0}$
Canetti et al.'s 6 rounds	256	128	-	3072	$45.1t + 1.5$
Canetti et al.'s 3 rounds	256	128	-	3072	$45.3t + 1.8$
Ours	256	128	1827	-	$\mathbf{4.1t + 2.3}$

Figure 5.8: Comparative sizes (in bits) & comm. cost (in Bytes)

of a more efficient identification procedure if an abort occurs. Regarding our work, computations are based on the sub-protocols described in Section 5.1. The resulting figures are provided in Fig. 5.8. For our choice on the size of the discriminant Δ_K defining the class group and the resulting number of bits required to represent elements, we refer to [CCL+20] (see Section 4.5). We further reduce the representation of class group elements by a factor 3/4 by relying on the simple yet elegant compression technique presented by Dobson et al in [DGS20]. Figure 5.8 clearly demonstrates the impressive efficiency gains we attain, reducing by a factor 10 the bandwidth consumption compared to [CGG+20].

Comparing Key Refresh. One of the main benefits of our protocol compared to that of Canetti *et al.* is the huge improvement provided by our Key Refresh protocol for reasonable numbers of users. Precisely, in [CGG+20] each player is required to generate a new Paillier's (RSA) modulus N together with a proof that this was constructed correctly. For a 112 bits level of security, in their n out of n Key Refresh protocol requires essentially $5n^2+n$ elements of size $|q|$ and n^2+163n ring elements (of size $2|N|$ to be sent between all players. Whereas our Key Refresh requires $3(n^2+n)$ elements of size $|q|$; $4n^2+5n$ group elements from the class group and n^2+n challenges, of size $|\Delta_K|/2+80+\lambda$. Concretely, for $n=5$, $\lambda=112$ this results in 420KBytes of data being transmitted in their protocol, as opposed to 28KBytes in ours; i.e. a reduction by a factor 15.

Conclusion

We presented and analyzed our series of works concerning distributed ECDSA for the specific two-party case and the more general threshold case. We are not the first ones that present simple and efficient solutions for distributed ECDSA, but we improved on the state of art for what concerns bandwidth consumption in general and efficiency when considering high levels of security (from 192-bit security for both two-party and threshold signing). The contribution of our works presented in this manuscript is twofold, since we did not only propose new schemes, but we also proposed solutions to several problems regarding proving statements about elements of the underlying encryption scheme, which is of independent interest and useful for future works built upon it, even in contexts different from ECDSA.
First, in Chapter 3, we discussed our two-party ECDSA scheme (in [CCL+19]) inspired by the idea of the two-party ECDSA scheme of [Lin17]. The solution we proposed is the first generic construction for two-party ECDSA signing which is build from hash proof systems which are homomorphic with respect to a prime number modulus. Compared with [Lin17], we proved that also our scheme has simulation-based security, but we do not need to rely on some interactive ad hoc assumption (as [Lin17] did) and the reduction of security is tight. That is a consequence of the structure of the underlying homomorphic encryption schemes based on hash proof systems (HPS). Furthermore, all the required properties of our general framework from HPS are satisfied by the Castagnos-Laguillaumie encryption scheme (CL) – which is built from class groups of imaginary quadratic fields – and we gave a concrete instatiantion of the two-party ECDSA scheme using CL. Indeed, we observed the compatibility of the linear homomorphic CL scheme – which work modulo a prime of our choice – and the structure of ECDSA.
In Chapter 4, inspired by the work of Gennaro and Goldfeder ([GG18]), we discussed our class group based instantiation of full threshold ECDSA (in [CCL+20]). Our solution presents a lower bandwidth consumption and it avoids expensive zero-knowledge range proof inherent to the usage of Paillier encryption with ECDSA. The usage of CL instead of Paillier completely changed the issues regarding efficiency in the threshold ECDSA scheme, even if the structure is similar. Indeed, we had to cope with the efficiency issues in the proofs of statements in the CL scheme. Finally, we were able to propose new efficient proofs regarding CL ciphertexts.
About the bandwidth consumption, our scheme rely on the shorter representation of ciphertexts in CL compared with the widely used Paillier encryption scheme in the context of linear homomorphic operations on ciphertexts. This choice puts a step stone in the design of bandwidth efficient constructions which rely on linear homomorphic encryption. At the same time, we also observed the pros and cons of using CL encryption together with other objects, as an Elliptic curve. Indeed, we can choose any prime number as the

size of the group of encoded messages where it is easy to compute discrete logarithms. This construction is independent from the context, but it presents issues in proving the validity of operations done. We analyzed them in this manuscript giving a description of the zero-knowledge protocols necessary to prove that things work. This completes the set of tools of independent interest useful to use class group cryptography in other situations. Furthermore in Chapter 2, i.e. the chapter dedicated to the mathematical background about class groups, we gave an explanation of the structural choice we have to pay attention to avoid destructive attacks. Putting together the previous points, in summary we propose a new line of work in distributed ECDSA using class groups. Furthermore, this direction is also motivated by the recent growing interest in class group cryptography (for example [YCX21][8],[DMZ+21],[XAX+21]).
In Chapter 5, taking inspiration from the recent work of Canetti *et al.* ([CGG+20]), we discussed our extended threshold ECDSA solution in [CCL+21]. This solution extends the previous one in [CCL+20]) to a new scheme which reduces the number of communication rounds required to check the validity of a signature and such that it guarantee practical properties of interest in applications. Indeed, in our improved scheme it is possible to identify bad behaviours and the scheme is proactive. About the technical aspect, we generalized Canetti *et al.* solution to class groups, and again we took in account the issues inherent to working with a group of unknown order. Indeed, to identify a misbehaving player with CL it is necessary to know if a ciphertext decrypts to a valid message (CL is not surjective). Differently from Paillier encryption where knowing the decryption key is enough to extract the randomness which brings to a certain ciphertext, with CL this is not possible. For solving this problem, we introduced a new ZK to prove that a ciphertext decrypts to a message or not. The efficiency and the bandwidth consumption of the resulting ECDSA scheme is comparable with our base threshold construction (in [CCL+20]) and it inherits the bandwidth efficiency pros.

A summary of the ZKs In our two-party proposal, we originally proposed a ZKPoK to prove the knowledge of an encrypted value which is also the discrete logarithm of a elliptic point. The main issue of our instantiation relies on the use of binary challenges in that ZKPoK, since it requires more repetitions to guarantee a certain level of soundness. A binary challenge space is necessary because we work in a cyclic subgroup of a group of unknown order, which implies that it is not possible for us to check if an element comes from the subgroup. We took inspiration from proposals to deal with generalized Schnorr proofs in groups of unknown order (for instance the framework of [CKY09], or [TW12]). If we consider the case of subgroups of $(\mathbf{Z}/n\mathbf{Z})^\times$, efficient solutions for this type of proofs extend the dimension of the challenge space, and they are build upon variants of the strong RSA assumption. In the case of class groups, informal proposals were given ([DF02]). As discussed deeply in Chapter 2, we saw that if the factorization of the discriminant (which is public in our case) is known, then computing square roots or finding elements of order 2 in the class group are easy tasks, i.e. we are able to compute them efficiently. In addition, [BBF18] suggests that there may be other approaches to find elements with low order in class groups.

[8]Authors in this work improve the state of our ZKP for well-formness of CL ciphertexts and of the two-party/threshold ECDSA protocols, but they prove security in the General Group Model since ECDSA is proved secure in this model and the authors exploit this fact to improve overall efficiency

Inspired by the framework of the work cited above ([CKY09]) for groups of unknown order and assuming two not novel assumptions – the low order assumption and the strong root assumption – we proposed a new zero-knowledge arguments of knowledge which improves dramatically the state of art of proofs in class groups. This new ZKAoKs are necessary to obtain a bandwidth efficient distributed ECDSA schemes and they also improve the ZKs in our two-party schemes extending the challenge space from size 2 to size 2^λ. Indeed, we presented in Chapters 4 and 5, new efficient proofs for proving the validity of a CL ciphertext, if a CL ciphertext decrypts to some value (since differently from Paillier, CL is not surjective) and we improved the efficiency of our proof with binary challenges assuming the new additional assumption and the state of the ZKPoK without relying on them using the least common multiple trick (which reduces the number of repetitions of the proof by a factor of 10).

Bibliography

[Atk90] O. Atkin. *Composition of binary quadratic forms.* 1990.

[BBBF18] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *CRYPTO 2018, Part I*, *LNCS* 10991, pages 757–788. Springer, Heidelberg, August 2018.

[BBF18] D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. `https://eprint.iacr.org/2018/712`.

[BBHM02] I. Biehl, J. Buchmann, S. Hamdy, and A. Meyer. A signature scheme based on the intractability of computing roots. *Designs, Codes and Cryptography*, 25(3):223–236, Mar 2002.

[BBL17] F. Benhamouda, F. Bourse, and H. Lipmaa. CCA-secure inner-product functional encryption from projective hash functions. In *PKC 2017, Part II*, *LNCS* 10175, pages 36–66. Springer, Heidelberg, March 2017.

[BCP03] E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *ASIACRYPT 2003*, *LNCS* 2894, pages 37–54. Springer, Heidelberg, November / December 2003.

[Bel04] K. Belabas. On quadratic fields with large 3-rank. *Mathematics of Computation*, 73(248):2061–2074, 2004.

[BGG17] D. Boneh, R. Gennaro, and S. Goldfeder. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In *LATINCRYPT 2017*, *LNCS* 11368, pages 352–377. Springer, Heidelberg, September 2017.

[BH01] J. Buchmann and S. Hamdy. A survey on IQ cryptography. In *Public Key Cryptography and Computational Number Theory*, pages 1–15. De Gruyter Proceedings in Mathematics, 2001.

[BJS10] J.-F. Biasse, M. J. Jacobson, and A. K. Silvester. Security estimates for quadratic field based cryptosystems. In *ACISP 10*, *LNCS* 6168, pages 233–247. Springer, Heidelberg, July 2010.

[BKSW20] K. Belabas, T. Kleinjung, A. Sanso, and B. Wesolowski. A note on the low order assumption in class group of an imaginary quadratic number fields. Cryptology ePrint Archive, Report 2020/1310, 2020. `https://eprint.iacr.org/2020/1310`.

[Boy86] C. Boyd. Digital multisignature. *Cryptography and Coding*, pages 241–246, 1986.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[Bro00] D. Brown. The exact security of ecdsa. December 2000.

[Bro02] D. R. L. Brown. Generic groups, collision resistance, and ECDSA. Cryptology ePrint Archive, Report 2002/026, 2002. `https://eprint.iacr.org/2002/026`.

[Bue76] D. A. Buell. Class groups of quadratic fields. *Mathematics of Computation*, 30(135):610–623, 1976.

[BV07] J. Buchmann and U. Vollmer. *Binary Quadratic Forms: An Algorithmic Approach*, *Algorithms and Computation in Mathematics* 20. Springer-Verlag, Berlin, Heidelberg, 1 edition, 2007.

[CCL$^+$19] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 191–221. Springer, Heidelberg, August 2019.

[CCL$^+$20] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA. In *PKC 2020, Part II*, *LNCS* 12111, pages 266–296. Springer, Heidelberg, May 2020.

[CCL$^+$21] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts, proactivity and adaptive security. Cryptology ePrint Archive, Report 2021/291, 2021. `https://eprint.iacr.org/2021/291`.

[CGG$^+$20] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020.

[CGH04] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004.

[CH89] R. A. Croft and S. P. Harris. Public-key cryptography and reusable shared secret. *Cryptography and Coding*, pages 189–201, 1989.

[CH94] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *CRYPTO'94*, *LNCS* 839, pages 425–438. Springer, Heidelberg, August 1994.

[CIL17] G. Castagnos, L. Imbert, and F. Laguillaumie. Encryption switching protocols revisited: Switching modulo p. In *CRYPTO 2017, Part I*, *LNCS* 10401, pages 255–287. Springer, Heidelberg, August 2017.

[CKY09] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized Schnorr proofs. In *EUROCRYPT 2009*, *LNCS* 5479, pages 425–442. Springer, Heidelberg, April 2009.

[CL84] H. Cohen and H. W. Lenstra Jr. Heuristics on class groups. In *Number Theory*, pages 26–36, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[CL09] G. Castagnos and F. Laguillaumie. On the security of cryptosystems with quadratic decryption: The nicest cryptanalysis. In *EUROCRYPT 2009*, *LNCS* 5479, pages 260–277. Springer, Heidelberg, April 2009.

[CL14] R. Cohen and Y. Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *ASIACRYPT 2014, Part II*, *LNCS* 8874, pages 466–485. Springer, Heidelberg, December 2014.

[CL15] G. Castagnos and F. Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA 2015*, *LNCS* 9048, pages 487–505. Springer, Heidelberg, April 2015.

[CLT18a] G. Castagnos, F. Laguillaumie, and I. Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 733–764. Springer, Heidelberg, December 2018.

[CLT18b] G. Castagnos, F. Laguillaumie, and I. Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. Cryptology ePrint Archive, Report 2018/791, 2018. `https://eprint.iacr.org/2018/791`.

[CMP20] R. Canetti, N. Makriyannis, and U. Peled. Uc non-interactive, proactive, threshold ecdsa. Cryptology ePrint Archive, Report 2020/492, 2020. `https://eprint.iacr.org/2020/492`.

[Coh00] H. Cohen. *A course in computational algebraic number theory.* Springer-Verlag, 2000.

[Cox14] D. A. Cox. *Primes of the Form $x^2 + ny^2$: Fermat, Class Field Theory, and Complex Multiplication*, *Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts* 119. John Wiley & Sons, 2nd edition, 2014.

[CP93] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO'92*, *LNCS* 740, pages 89–105. Springer, Heidelberg, August 1993.

[CS97] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO'97*, *LNCS* 1294, pages 410–424. Springer, Heidelberg, August 1997.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO'98*, *LNCS* 1462, pages 13–25. Springer, Heidelberg, August 1998.

[CS02] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT 2002*, *LNCS* 2332, pages 45–64. Springer, Heidelberg, April / May 2002.

[CS03] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO 2003*, *LNCS* 2729, pages 126–144. Springer, Heidelberg, August 2003.

[DDN00] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[Des88] Y. Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO'87*, *LNCS* 293, pages 120–127. Springer, Heidelberg, August 1988.

[DF90] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO'89*, *LNCS* 435, pages 307–315. Springer, Heidelberg, August 1990.

[DF02] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT 2002*, *LNCS* 2501, pages 125–142. Springer, Heidelberg, December 2002.

[DGS20] S. Dobson, S. D. Galbraith, and B. Smith. Trustless groups of unknown order with hyperelliptic curves. Cryptology ePrint Archive, Report 2020/196, 2020. `https://eprint.iacr.org/2020/196`.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DKLs18] J. Doerner, Y. Kondi, E. Lee, and a. shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society Press, May 2018.

[DKLs19] J. Doerner, Y. Kondi, E. Lee, and a. shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019.

[DKO$^+$19] A. P. K. Dalskov, M. Keller, C. Orlandi, K. Shrishak, and H. Shulman. Securing dnssec keys via threshold ecdsa from generic mpc. *IACR Cryptology ePrint Archive*, 2019:889, 2019.

[DMZ+21] Y. Deng, S. Ma, X. Zhang, H. Wang, X. Song, and X. Xie. Promise σ-protocol: How to construct efficient threshold ecdsa from encryptions based on class groups. In *Tibouchi M., Wang H. (eds) Advances in Cryptology – ASIACRYPT 2021. ASIACRYPT 2021*. Springer-Verlag, 2021.

[Fel87] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. of FOCS 87*, pages 427–437. IEEE Computer Society, 1987.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, *LNCS* 263, pages 186–194. Springer, Heidelberg, August 1987.

[GG18] R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *ACM CCS 2018*, pages 1179–1194. ACM Press, October 2018.

[GG20] R. Gennaro and S. Goldfeder. One round threshold ECDSA with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. `https://eprint.iacr.org/2020/540`.

[GGN16] R. Gennaro, S. Goldfeder, and A. Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *ACNS 16*, *LNCS* 9696, pages 156–174. Springer, Heidelberg, June 2016.

[Gil99] N. Gilboa. Two party RSA key generation. In *CRYPTO'99*, *LNCS* 1666, pages 116–129. Springer, Heidelberg, August 1999.

[GJKR96a] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *CRYPTO'96*, *LNCS* 1109, pages 157–172. Springer, Heidelberg, August 1996.

[GJKR96b] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *EUROCRYPT'96*, *LNCS* 1070, pages 354–371. Springer, Heidelberg, May 1996.

[GM82] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982.

[GM02] S. Goldwasser and D. Micciancio. *Complexity of Lattice Problems*. Kluwer Academic Publishers, USA, 2002.

[GMR88] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[Gol01] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, UK, 2001.

[Gol04] O. Goldreich. *Foundations of Cryptography: Basic Applications.* Cambridge University Press, Cambridge, UK, 2004.

[GPS06] M. Girault, G. Poupard, and J. Stern. On the fly authentication and signature schemes based on groups of unknown order. *Journal of Cryptology*, 19(4):463–487, October 2006.

[GS21] J. Groth and V. Shoup. On the security of ECDSA with additive key derivation and presignatures. Cryptology ePrint Archive, Report 2021/1330, 2021. `https://eprint.iacr.org/2021/1330`.

[HJPT98] D. Hühnlein, M. J. Jacobson Jr., S. Paulus, and T. Takagi. A cryptosystem based on non-maximal imaginary quadratic orders with fast decryption. In *EUROCRYPT'98*, *LNCS* 1403, pages 294–307. Springer, Heidelberg, May / June 1998.

[HL10] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions.* ISC. Springer, Heidelberg, 2010.

[HM00] S. Hamdy and B. Möller. Security of cryptosystems based on class groups of imaginary quadratic orders. In *ASIACRYPT 2000*, *LNCS* 1976, pages 234–247. Springer, Heidelberg, December 2000.

[HO09] B. Hemenway and R. Ostrovsky. Lossy trapdoor functions from smooth homomorphic hash proof systems. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:127, 01 2009.

[HS06] S. Hamdy and F. Saidak. Arithmetic properties of class numbers of imaginary quadratic fields. *JP Journal of Algebra, Number Theory and Application*, 6(1):129–148, 2006.

[IOZ14] Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO 2014, Part II*, *LNCS* 8617, pages 369–386. Springer, Heidelberg, August 2014.

[Jac00] M. J. Jacobson Jr. Computing discrete logarithms in quadratic orders. *Journal of Cryptology*, 13(4):473–492, September 2000.

[Kap78] P. Kaplan. Divisibilité par 8 du nombre des classes des corps quadratiques dont le $2-$groupe des classes est cyclique, et réciprocité biquadratique. *J. Math. Soc. Japan*, 25(4):574–733, 1978.

[KL14] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition.* Chapman & Hall/CRC, 2nd edition, 2014.

[Lag80] J. Lagarias. Worst-case complexity bounds for algorithms in the theory of integral quadratic forms. *Journal of Algorithms*, 1(2):142 – 186, 1980.

[Lin16] Y. Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. `https://eprint.iacr.org/2016/046`.

[Lin17] Y. Lindell. Fast secure two-party ECDSA signing. In *CRYPTO 2017, Part II*, *LNCS* 10402, pages 613–644. Springer, Heidelberg, August 2017.

[Lip12] H. Lipmaa. Secure accumulators from euclidean rings without trusted setup. In *ACNS 12*, *LNCS* 7341, pages 224–240. Springer, Heidelberg, June 2012.

[LN18] Y. Lindell and A. Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM CCS 2018*, pages 1837–1854. ACM Press, October 2018.

[MR01] P. D. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. In *CRYPTO 2001*, *LNCS* 2139, pages 137–154. Springer, Heidelberg, August 2001.

[MR04] P. D. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. *Int. J. Inf. Sec.*, 2(3-4):218–239, 2004.

[OY91] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *10th ACM PODC*, pages 51–59. ACM, August 1991.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, *LNCS* 1592, pages 223–238. Springer, Heidelberg, May 1999.

[PAR18] PARI Group, Univ. Bordeaux. *PARI/GP version* `2.11.1`, 2018. available from `http://pari.math.u-bordeaux.fr/`.

[Ped92] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO'91*, *LNCS* 576, pages 129–140. Springer, Heidelberg, August 1992.

[Pie19] K. Pietrzak. Simple verifiable delay functions. In *ITCS 2019*, pages 60:1–60:15. LIPIcs, January 2019.

[PR05] R. Pass and A. Rosen. Concurrent non-malleable commitments. In *46th FOCS*, pages 563–572. IEEE Computer Society Press, October 2005.

[PS00] G. Poupard and J. Stern. Short proofs of knowledge for factoring. In *PKC 2000*, *LNCS* 1751, pages 147–166. Springer, Heidelberg, January 2000.

[PT00] S. Paulus and T. Takagi. A new public-key cryptosystem over a quadratic order with quadratic decryption time. *Journal of Cryptology*, 13(2):263–272, March 2000.

[Que87] J. Quer. Corps quadratiques de 3-rang 6 et courbes elliptiques de rang 12. *C. R. Acad. Sci., Paris, Sér. I*, 305:215–218, 1987.

[SA19] N. P. Smart and Y. T. Alaoui. Distributing any elliptic curve based protocol: With an application to mixnets. *IACR Cryptology ePrint Archive*, 2019:768, 2019.

[Sch90] C.-P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, *LNCS* 435, pages 239–252. Springer, Heidelberg, August 1990.

[Sch91] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[Sep] Sepior. `http://www.sepior.com`.

[Ser] I. D. P. Services. `https://security.intuit.com/`.

[SG98] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *EUROCRYPT'98*, *LNCS* 1403, pages 1–16. Springer, Heidelberg, May / June 1998.

[Sha79] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT'97*, *LNCS* 1233, pages 256–266. Springer, Heidelberg, May 1997.

[Sho00] V. Shoup. Practical threshold signatures. In *EUROCRYPT 2000*, *LNCS* 1807, pages 207–220. Springer, Heidelberg, May 2000.

[TW12] B. Terelius and D. Wikström. Efficiency limitations of S-protocols for group homomorphisms revisited. In *SCN 12*, *LNCS* 7485, pages 461–476. Springer, Heidelberg, September 2012.

[Unb] Unboundtech. `https://www.unboundtech.com/`.

[Van92] S. Vanstone. Responses to nist's proposal. *Communications of the ACM*, 35:50–52, July 1992. (communicated by John Anderson).

[Wes19a] B. Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology – EUROCRYPT 2019*, pages 379–407, Cham, 2019. Springer International Publishing.

[Wes19b] B. Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT 2019, Part III*, *LNCS* 11478, pages 379–407. Springer, Heidelberg, May 2019.

[XAX$^+$21] H. Xue, M. H. Au, X. Xie, T. H. Yuen, and H. Cui. Efficient online-friendly two-party ecdsa signature. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 558–573, New York, NY, USA, 2021. Association for Computing Machinery.

[YCX21] T. H. Yuen, H. Cui, and X. Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In *PKC 2021, Part I*, *LNCS* 12710, pages 481–511. Springer, Heidelberg, May 2021.

Acknowledgment

I would like to thank several people which I talk with during my Ph.D. First of all, a thanks go to my two examiners Michel Abdalla and Manuel Barbosa for having read this manuscript and for having suggested to me relevant corrections. Then, I thank my supervisor Dario Catalano for having guided me during my Ph.D and for his suggestions. I also thank other people from the team I work with: Guilhem Castagnos, Fabien Laguillaumie and Ida Tucker, in particular for the hospitality in Lyon. A special thanks goes to Ida for her availability and for her great help both in the academic side and in the human side. If I forgot someone, do not worry since I chose to write only academic acknowledgement here, the complete version will be present only in the printed version of this manuscript.