Two-Party ECDSA from Hash Proof Systems and Efficient Instantiations

Guilhem Castagnos[1], Dario Catalano[2], Fabien Laguillaumie[3],
Federico Savasta[2,4], and Ida Tucker[3]

[1] Université de Bordeaux, INRIA, CNRS, IMB UMR 5251, F-33405 Talence, France.
[2] Università di Catania, Italy.
[3] Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France.
[4] Scuola Superiore di Catania, Italy

Abstract. ECDSA is a widely adopted digital signature standard. Unfortunately, efficient distributed variants of this primitive are notoriously hard to achieve and known solutions often require expensive zero knowledge proofs to deal with malicious adversaries. For the two party case, Lindell [Lin17] recently managed to get an efficient solution which, to achieve simulation-based security, relies on an interactive, non standard, assumption on Paillier's cryptosystem. In this paper we generalize Lindell's solution using hash proof systems. The main advantage of our generic method is that it results in a simulation-based security proof without resorting to any interactive assumptions.

Moving to concrete constructions, we show how to instantiate our framework using class groups of imaginary quadratic fields. Our implementations show that the practical impact of dropping such interactive assumptions is minimal. Indeed, while for 128-bit security our scheme is marginally slower than Lindell's, for 256-bit security it turns out to be better both in key generation and signing time. Moreover, in terms of communication cost, our implementation significantly reduces both the number of rounds and the transmitted bits without exception.

1 Introduction

Threshold cryptography [Des88,DF90,GJKR96,SG98,Sho00,Boy86,CH89,MR04] allows n users to share a common key in such a way that any subset of t parties can use this key to decrypt or sign, while any coalition of less than t can do nothing. The key feature of this paradigm is that it allows to use the shared key without explicitly reconstructing it in the clear. This means a subset of t parties have to actively participate in the protocol whenever the secret key is used.

Applications of threshold cryptography range from contexts where many signers need to agree to sign one common document to distributed scenarios where sensitive documents should become accessible only 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. Recent years have seen renewed interest in the field (e.g. [GGN16,Lin17,GG18,DKLs18,LN18,GG18,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]). Finally, some of the schemes developed twenty years ago are not as efficient as current applications want them to be. This is the case, for instance, for ECDSA/DSA signatures. Indeed, while for many other schemes fast threshold variants are known (e.g. RSA decryption/signing and ECIES decryption) constructing efficient threshold variant of these signatures proved to be much harder. The main reason for this unfair distribution seems to result from the inversion step that requires one to compute $k^{-1} \bmod q$ from an unknown k. To explain why this is the case, let us first briefly recall how ECDSA actually works[5]. The public key is an elliptic curve point Q and the signing key is x, such that $Q \leftarrow xP$, where P is a generator of the elliptic curve group of points of order q. To sign a message m one first hashes it using some suitable hash function H and then proceeds according to the following algorithm

1. Choose k random in $\mathbf{Z}/q\mathbf{Z}$
2. Compute $R \leftarrow kP$
3. Let $r \leftarrow r_x \bmod q$ where $R = (r_x, r_y)$
4. Set $s \leftarrow k^{-1}(H(m) + rx) \bmod q$
5. Output (r, s)

Now, the natural approach to make the above algorithm distributed would be to share x additively among the participants and then start a multiparty computation protocol to produce the signature. In the two party case, this means that players start with shares x_1 and x_2 such that $Q = (x_1 + x_2)P$. The players can then proceed by generating random shares k_1, k_2 such that $R = (k_1 + k_2)P$. At this point, however, it is not clear how to compute, efficiently, shares k'_1, k'_2 such that $k'_1 + k'_2 = k'^{-1} \bmod q$.

Starting from [MR04] two party ECDSA signature protocols started adopting a less common *multiplicative* sharing both for x and k. The basic idea of these constructions is very simple. Players start holding shares x_1, x_2 such that $Q = x_1 x_2 P = xP$. Whenever a new signature has to be generated they generate random k_1, k_2 such that $R = k_1 k_2 P = kP$. This immediately allows to get shares of the inverse k' as clearly $(k_1)^{-1}(k_2)^{-1} = (k_1 k_2)^{-1} \bmod q$. As a final ingredient, the parties use Paillier's homomorphic encryption to secretly add their shares and complete the signature. For instance, player P_1 computes $c_1 \leftarrow \mathsf{Enc}((k_1)^{-1} H(m))$ and $c_2 \leftarrow \mathsf{Enc}((k_1)^{-1} x_1 r)$. P_2 can then complete the signature, using the homomorphic properties of the scheme as follows

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

P_2 concludes the protocol by sending back c to P_1. Now, if P_1 also knows the decryption key, he can extract the signature $s \leftarrow k^{-1}(H(m) + xr))$ from c.

However, proving that each party followed the protocol correctly turns out to be hard. Initial attempts [MR04] addressed this via expensive zero knowledge proofs. More recently Lindell in [Lin17] managed to provide a much simpler and efficient protocol. The crucial idea of Lindell's protocol is the observation that, in the above two party ECDSA signing protocol, dishonest parties can create very little trouble. Indeed, if in a preliminary phase P_2 receives both Paillier's encryption key *and* an encryption $\mathsf{Enc}(x_1)$ of P_1's share of the secret signing key, essentially, all a corrupted P_1 can do is participate in the generation

[5] From now on we will focus on the elliptic curve variant of the scheme, as this is the most commonly used scheme in applications. We stress that our reasoning apply to the basic DSA case as well.

of $R \leftarrow k_1 k_2 P$. Notice however that the latter is just the well established Diffie-Hellman protocol for which very efficient and robust protocols 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.

Turning this nice intuition into a formal proof induces some caveats though. A first problem comes from the fact that Paillier's plaintexts space is $\mathbf{Z}/N\mathbf{Z}$ (N is a large composite) whereas ECDSA signatures live in $\mathbf{Z}/q\mathbf{Z}$ (q is prime). Thus to avoid inconsistencies one needs to make sure that N is taken large enough so that no wraparounds occur during the whole signature generation process. This also means 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).

A more subtle issue arises from the use of 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 to overcome this. The first one relies on a game-based definition and 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 looses a factor q_s). The second proof is simulation based, avoids the aborts, but requires the introduction of a new interactive non standard assumption regarding Paillier's encryption.

Thus, it is fair to say that, in spite of recent progress in the area, the following question remains open:

Is 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), does not require interactive assumptions and allows for a tight security reduction?[6]

1.1 Our contribution

In this paper we provide a positive answer to the question above. In this sense, our contribution is twofold.

First, we provide a generic construction for two-party ECDSA signing from hash proof systems (HPS). 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 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 new protocol allows for a proof of security that is both tight and does not require artificial interactive assumptions when proving simulation security. Indeed,

[6] 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].

in public key encryption (PKE) 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.

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.

Towards efficient solutions, we then show how to instantiate our (homomorphic) HPS construction using class groups of imaginary quadratic fields. Although the devastating attack from [CL09] shows that large families of protocols built over such groups are insecure, Castagnos and Laguillaumie [CL15] showed that, if carefully designed, discrete logarithm based cryptosystems within such groups are still possible and allow for very efficient solutions. Algorithms to compute discrete logarithms in such groups have been extensively studied since the 80's and the best ones known to date have a subexponential complexity[7] of $\mathcal{O}(L[1/2, o(1)])$ (compared to an $\mathcal{O}(L[1/3, o(1)])$ complexity for factorisation or discrete logarithm computation in finite fields). In [BH03], Bauer and Hamdy also showed that, for the specific case of imaginary quadratic fields, better complexities seem unlikely. Thus, the resulting schemes benefit from (asymptotically) shorter keys. Moreover, interest in the area has recently been renewed as it allows versatile and efficient solutions such as encryption switching protocols [CIL17], inner product functional encryption [CLT18] or verifiable delay functions [BBBF18,Wes19].

Concretely, the main feature of the Castagnos and Laguillaumie cryptosystem and its variants (CL from now on) is that they rely on the existence of groups with associated easy discrete log subgroups, for which hard decision problems can be defined. More precisely, in [CL15] 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. Informally, and deferring for later the necessary mathematical details, this allows to build a linearly homomorphic PKE 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!

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. As for Lindell's case, 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. While, in our setting we can achieve this without resorting to expensive range proofs, difficulties arise from the fact that (1) we work with groups of unknown order and (2) we cannot assume that all ciphertexts are valid (*i.e.*, actually encrypt a message)[8]. 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. We believe that it should be possible to enlarge the challenge space

[7] $L[\alpha, c]$ denotes $L_{\alpha,c}(x) := \exp(c \log(x)^{\alpha} \log(\log(x))^{1-\alpha})$

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

using techniques similar to those [CKY09] adapted to work in the context of class groups. Exploring the actual feasibility of this idea is left as a future work. Clearly, advances in this direction would lead to substantial efficiency improvements.

As final contribution, we propose a C implementation of our protocol[9]. Our results show that our improved security guarantees come almost at no additional cost. Indeed, while our scheme is slightly slower (by a factor 1.5 for key generation and 3.5 for signing) for 128-bit security level, we are actually better for larger parameters: for 256-bit security, we are more efficient both in terms of key generation and signing time (by respective factors of 4.2 and 1.3). Intuitively, this behavior is due to the fact that our interactive key generation requires fewer exponentiations than that of Lindell's protocol (160 vs. 360), but an exponentiation in a class group is more expensive than an exponentiation in $\mathbf{Z}/n\mathbf{Z}$. The effect of the $L_{1/2}$ complexity and the fewer number of exponentiations starts at 192 bit of security. In terms of bandwidth, our protocol dramatically improves the communication cost by *factors varying from 5 (112 bit security) to 10 (256 bit security)* for key generation, and from 1.2 to 2.1 for signatures. It reduces the number of rounds from 175 (in Lindell's protocol) to 126 for the key generation process (the two signatures have the same number of rounds). We refer to Section 5 for precise implementation considerations and timings.

As a final remark, our HPS methods also allow a concrete implementation based on Paillier's decisional composite residuosity assumption, competitive with Lindell's for 112 and 128 bits of security as detailed in Section 6.

Differences with the the published version. We point out that the original proof of our main protocol, as given in [CCL$^+$19], was incomplete. Very informally, the issue was related to the fact that one game hop cannot be justified using a purely information theoretically argument, as implicitly assumed in [CCL$^+$19]. To fix this we need to assume an additional (and new) property from the underlying hash proof system. This property, relies on the intractability of what we call *double encoding assumption*, a (non interactive) assumption that we discuss and formalize in section 3.4.

2 Preliminaries

Notations. For 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 in the set B. In an interactive protocol IP, between parties P_1 and P_2, we denote by $\mathsf{IP}\langle x_1; x_2\rangle \to \langle y_1; y_2\rangle$ the joint execution of parties $\{P_i\}_{i\in\{1,2\}}$ in the protocol, with respective inputs x_i, and where P_i's private output at the end of the execution is y_i.

The 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 $x \overset{\$}{\leftarrow} \mathbf{Z}/q\mathbf{Z}$ is the secret signing key and $Q \leftarrow xP$ is the public verification key.

[9] We also re-implemented Lindell's protocol to ensure a fair comparison

$\mathsf{Sign}(x, m) \to (r, s)$ 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 $R \leftarrow kP$; denote $R = (r_x, r_y)$ and let $r \leftarrow r_x \mod q$. If $r = 0$ chose another k.
3. Compute $s \leftarrow k^{-1}(m' + rx) \mod q$

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

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 $\mathscr{L}$ 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 Appendix I.

Simulation-based security and ideal functionalities. In order to prove a protocol is secure, one must first define what *secure* means. Basically, the Ideal/Real paradigm is to imagine what properties one would have in an ideal world; then if a real world (constructed) protocol has similar properties it is considered secure. We consider static adversaries, that choose which parties are corrupted before the protocol begins. For a detailed explanation of the simulation paradigm we refer the reader to [Lin16].

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. We give the intuition behind these ideal functionalities with the example of ZKPoK. We 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 $\mathscr{A}$ (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 $\mathscr{F}$. If some protocol satisfies the above property regarding this functionality, we call it secure.

Formally, we denote $\mathscr{F}\langle x_1; x_2\rangle \to \langle y_1; y_2\rangle$ the joint execution of the parties via the functionality $\mathscr{F}$, 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 $\mathscr{F}_{\mathsf{zk}}$, is defined for a relation R by $\mathscr{F}_{\mathsf{zk}}\langle(x, w); \emptyset\rangle \to \langle\emptyset; (x, \mathsf{R}(x, w))\rangle$, where $\emptyset$ is the empty output, signifying that the first party receives no output (cf. Fig. 1).

The *ideal commitment functionality*, denoted $\mathscr{F}_{\mathsf{com}}$, is depicted in Fig. 2. We also use an ideal functionality $\mathscr{F}^{\mathsf{R}}_{\mathsf{com-zk}}$ for *commitments to NIZK proofs* for a relation R (cf. Fig. 3). Essentially, this is a commitment functionality, where the committed value is a NIZK proof.

- 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}

Fig. 1: The $\mathcal{F}_{\mathsf{zk}}^{\mathsf{R}}$ functionality

- Upon receiving (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}.

Fig. 2: The $\mathcal{F}_{\mathsf{com}}$ functionality

The ideal functionality for two-party ECDSA. The ideal functionality $\mathcal{F}_{ECDSA}$ (cf. Fig. 4) consists of two functions: a key generation function, called once, and a signing function, called an arbitrary number of times with the generated keys.

- 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}

Fig. 3: The $\mathcal{F}_{\mathsf{com-zk}}^{\mathsf{R}}$ functionality

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 $H_q : \{0,1\}^* \to \{0,1\}^{\lfloor \log |q| \rfloor}$, 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).)

Fig. 4: The $\mathcal{F}_{ECDSA}$ functionality

3 Two-Party ECDSA from Hash Proof Systems

In this section we provide a generic construction for two-party ECDSA signing from hash proof systems (HPS) which we prove secure in the simulation based model. Throughout the section we consider the group of points of an elliptic curve $\mathbb{G}$ of order q, generated by P. In Subsection 3.1, we first recall the HPS framework from [CS02], before defining basic properties required for our construction in Subsection 3.2. 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. Such HPSs define linearly homomorphic encryption schemes as described in Subsection 3.3. In Subsection 3.4, we summarise all the properties required to build 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 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 and proving its security, we describe zero-knowledge proofs (ZKP) related to the aforementioned HPSs, and justify that they fulfil the $\mathcal{F}_{\mathsf{com}}/\mathcal{F}_{\mathsf{com\text{-}zk}}$ hybrid model.

3.1 Background on Hash Proof Systems

Hash proof systems were introduced in [CS02] as a generalisation of the techniques used in [CS98] to design chosen ciphertext secure PKE schemes. Consider a set of words $\mathcal{X}$, an NP language $\mathcal{L} \subset \mathcal{X}$ such that $\mathcal{L} := \{x \in \mathcal{X} \mid w \in \mathcal{W} : (x, w) \in \mathsf{R}\}$ where R is the relation defining the language, $\mathcal{L}$ is the language of true statements in $\mathcal{X}$, and for $(x, w) \in \mathsf{R}$, $w \in \mathcal{W}$ is 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 algorithm which on input a parameter 1^λ, outputs the description $(\mathcal{X}, \mathcal{L}, \mathcal{W}, \mathsf{R})$ of a subgroup membership problem.

A HPS associates a projective hash family (PHF) to such a subset membership problem. The PHF defines 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}, and a public projection key $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$ in projection key space K_{hp} (where $\mathsf{projkg} : K_{\mathsf{hk}} \mapsto K_{\mathsf{hp}}$ is an efficient auxiliary function). The secret hashing key hk defines a hash function $\mathsf{hash}_{\mathsf{hk}} : \mathcal{X} \mapsto \Pi$, and hp allows for the (public) evaluation of the hash function on words $x \in \mathcal{L}$, i.e. $\mathsf{projhash}_{\mathsf{hp}}(x, w) = \mathsf{hash}_{\mathsf{hk}}(x)$ for $(x, w) \in \mathsf{R}$. A projective hash family PHF is thus defined by $\mathsf{PHF} := (\{\mathsf{hash}_{\mathsf{hk}}\}_{\mathsf{hk} \in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \mathcal{L}, \Pi, K_{\mathsf{hp}}, \mathsf{projkg})$.

3.2 Basic Properties

δ_s-smoothness. The standard *smoothness* property of a PHF requires that for any $x \notin \mathcal{L}$, the value $\mathsf{hash}_{\mathsf{hk}}(x)$ be uniformly distributed knowing hp. In this work messages will be encoded in a subgroup $\mathcal{F}$ of Π of order q, generated by f. Indeed, for integration with ECDSA it

must hold that the group in which the message is encoded has order q, since the message space is dictated by the order of the elliptic curve group $\mathbb{G}$. In some instantiations $\mathcal{F} = \Pi$, but $\mathcal{F}$ may also be a strict subgroup of Π. For $m \in \mathbf{Z}/q\mathbf{Z}$ one encodes m in $\mathcal{F}$ as f^m, this induces an efficient isomorphism. The inverse isomorphism $\log_f : f^m \mapsto m$ must also be efficiently computable.

If $\mathcal{F} \subsetneq \Pi$, we require smoothness over $\mathcal{X}$ on $\mathcal{F}$ [CS02, Subsection 8.2.4]. A projective hash family is δ_s-smooth over $\mathcal{X}$ on $\mathcal{F}$ if for any $x \in \mathcal{X}\backslash\mathcal{L}$, a random $\pi \in \mathcal{F}$ 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. 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.

Linearly homomorphic PHF. In order for the homomorphic operations performed by P_2 to hold in the two party ECDSA protocol, we require that the projective hash family also be homomorphic as defined in [HO09].

Definition 1 ([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.

This clearly implies that for $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$ the public projective hash function is linearly homomorphic with respect to elements $u_1, u_2 \in \mathcal{L}$.

Homomorphically extended PHF. Note 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 2 (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. Finally for our security proofs we also need projective hash families which are linearly homomorphic w.r.t. the hashing keys.

Definition 3. *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 1. 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. The interested reader can verify 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].

3.3 Resulting Encryption Scheme

We use the standard chosen plaintext attack secure encryption scheme which results from a HPS [CS02]. The key generation algorithm simply runs $\mathsf{PHF.KeyGen}$ and sets $\mathsf{hk} \in K_{\mathsf{hk}}$ as the secret key, and the associated public key is $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$. Encryption of a plaintext message m in $\mathbf{Z}/q\mathbf{Z}$ is done by sampling a random pair $(u, w) \in \mathsf{R}$ and computing $\mathsf{Enc}(\mathsf{hp}, m) \leftarrow (u, \mathsf{projhash}_{\mathsf{hp}}(u, w) f^m)$. To specify the randomness used in the encryption algorithm, we sometimes use the notation $\mathsf{Enc}((u, w); (\mathsf{hp}, m))$. To decrypt a ciphertext $(u, e) \in \mathcal{X} \times \Pi$ with secret key hk do: $\mathsf{Dec}(\mathsf{hk}, (u, e)) \leftarrow \log_f(e \cdot \mathsf{hash}_{\mathsf{hk}}(u)^{-1})$. Note that if $e \cdot \mathsf{hash}_{\mathsf{hk}}(u)^{-1} \notin \mathcal{F} = \langle f \rangle$, the decryption algorithm returns the special error symbol $\perp$.

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 $\mathsf{Dec}(\mathsf{hk}, \mathsf{EvalSum}(\mathsf{hp}, (u_1, e_1), (u_2, e_2))) = m_1 + m_2$ and $\mathsf{Dec}(\mathsf{hk}, \mathsf{EvalScal}(\mathsf{hp}, (u_1, e_1), a)) = a \cdot m_1$; which is the case if the projective hash family is homomorphic.

Invalid ciphertexts. We define the notion of invalid ciphertexts as these will be useful in our security proofs. A ciphertext is said to be *invalid* if it is of the form $(u, e) := (u, \mathsf{hash}_{\mathsf{hk}}(u) f^m)$ where $u \in \mathcal{X} \backslash \mathcal{L}$. Note that one can compute such a ciphertext using the secret hashing key hk, but not the public projection key hp; that the decryption algorithm applied to (u, e) with secret key hk recovers m; and that 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 of 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 $\mathcal{L}$).

3.4 ECDSA friendly Projective Hash Families

We here formalise new properties for PHFs which contribute to the clarity of our security proofs.

$(\Upsilon, \mathcal{F})$-Decomposability. We introduce the notion of a decomposable PHF, this 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 pre-image 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 4. *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 $\mathcal{F}$. We say that* PHF *is $(\Upsilon, \mathcal{F})$-decomposable if there exists $\Upsilon \in \mathcal{X}$ s.t.:*

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

Remark 2. In this work $\mathcal{F}$ 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 paper, 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 $\mathsf{hash}_{\mathsf{hk}}(y) = f^{bc}$.

The Double Encoding Problem. To ensure security of our protocol, we need a notion which deals with information leaked by the actual fact an interactive signing protocol concludes successfully or aborts. Indeed, in the overall protocol, P_1 sends an encryption c_1 of its secret share x_1, along with the elliptic curve point $Q_1 := x_1 P$ to P_2. Then P_2 sends another ciphertext c_2 (which should be homomorphically computed from c_1) back to P_1. If the protocol stopped here, the HPS's smoothness would suffice to prove the security of the protocol, since the encrypted value is perfectly masked. However P_1 uses the value decrypted from c_2 to compute the overall signature. The fact that the protocol concludes successfully or aborts may leak one bit of information to a malicious P_2. We must thus ensure that a corrupted P_2 can not devise malformed ciphertexts allowing it to distinguish real and ideal executions, by causing an execution to conclude successfully in one case, while it would abort in the other. 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 5 (Double encoding assumption). *Consider a security parameter $\lambda \in \mathbf{N}$, and a λ-bit prime q. Further consider a collection of one way functions sampled via. 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 such that the resulting projective hash family* PHF *is $(\Upsilon, \mathcal{F})$-decomposable, for $\mathcal{F}$ of prime order q. The* double encoding (DE) *problem is $\delta_{\mathsf{DE}}(\lambda)$-hard for $(\mathsf{Gen}_{\mathcal{SM}}, \mathsf{Gen}_{OW})$ 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,*

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

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. 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 their structures are the same, it is unclear how one could do this. Of course if the OWF were the mapping of x to f^x, the DE problem would be easy. However in our applications we specifically require that computing x from f^x be easy, and consequently this mapping is *not* one way. We back the intuition that this problem is hard by considering two PHF instantiations which are relevant for our purposes. One from the DCR assumption (cf. [CS02]) and the other from class group cryptography (cf. Section 4). Let us first recall the definition of a subgroup decomposition problem.

Definition 6. *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 (cf. [CS02]) and HSM (cf. Section 4), 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 $\langle \Upsilon \rangle = \mathcal{F}$. Furthermore computing x from f^x can be done efficiently. Clearly these PHFs are homomorphic and key homomorphic.

Note that though for the HSM based PHF, the order of f is a prime q, for the DCR based PHF, the order of f is an RSA integer N. This implies that when building two-party ECDSA from DCR, the order q of the one way function's input space and the order 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 3.6).

In the following lemma we demonstrate that for both these PHFs, one can reduce the problem of inverting the OWF to the hardness of solving the SD problem in $(\mathcal{X}, \mathcal{L}, \mathcal{F})$, and the hardness of solving the DE problem.

Lemma 1. *Consider PHFs arising from* DCR *(cf. [CS02]) and* HSM *(cf. Section 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 $u_1, u_2, u_2 u_1^{-1} \in \mathcal{X} \setminus \mathcal{L}$ and $\mathsf{hp} = \mathsf{projkg}(\mathsf{hk})$. There exist unique values $z_1, z_2 \in \mathcal{L}$ and $b_1, b_2 \in \mathbf{Z}/q\mathbf{Z}$ for HSM (resp. $b_1, b_2 \in \mathbf{Z}/N\mathbf{Z}$ for DCR) 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 $z_1, z_2 \in \mathcal{L}$; $f^{b_1}, f^{b_2} \in \mathcal{F}$; $z_1^{\mathsf{hk}}, z_2^{\mathsf{hk}} \in \mathcal{L}$; 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 $(b_2 \mathsf{hk} + x \bmod q)$ in the HSM case; and $(b_1 \bmod N)$, $(b_2 \bmod N)$, $(b_1 \mathsf{hk} + x \bmod N)$ and $(b_2 \mathsf{hk} + x \bmod N)$ in the DCR case. Since $u_2 u_1^{-1} \in \mathcal{X} \backslash \mathcal{L}$, $b_1 \neq b_2 \bmod q$ for HSM, while for DCR $b_1 \neq b_2 \bmod N$, 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 Section 4.2, 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 here define the notion of an ECDSA-friendly HPS, essentially it is a HPS which meets all of the aforementioned properties, and which suffices to ensure simulation based security in the protocol of Subsection 3.6.

Definition 7 ($(\varUpsilon, \mathcal{F}, \delta_s, \delta_{\mathcal{L}}, \delta_{\mathsf{DE}})$-ECDSA-friendly HPS). *Let $\mathcal{X}, \varPi$ and $\mathcal{F}$ be groups such that $\mathcal{F}$ is a cyclic subgroup of $\varPi$ of prime order q, generated by f, and such that there exists an efficient isomorphism from $(\mathbf{Z}/q\mathbf{Z}, +)$ to $(\mathcal{F}, \cdot)$, 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 $(\varUpsilon, \mathcal{F}, \delta_s, \delta_{\mathcal{L}}, \delta_{\mathsf{DE}})$-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 $\mathsf{PHF} := (\{\mathsf{hash}_{\mathsf{hk}}\}_{\mathsf{hk} \in K_{\mathsf{hk}}}, K_{\mathsf{hk}}, \mathcal{X}, \mathcal{L}, \varPi, K_{\mathsf{hp}}, K'_{\mathsf{hp}}, \mathsf{projkg})$ which is $(\varUpsilon, \mathcal{F})$-decomposable, δ_s-smooth over $\mathcal{X}$ on $\mathcal{F}$, and such that the DE problem is δ_{DE}-hard for $(\mathsf{PHF}, \mathsf{exp}_{\mathbb{G}})$.*

3.5 Zero-Knowledge Proofs

Proofs of knowledge. We use the $\mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{com\text{-}zk}}$ hybrid model. Ideal ZK functionalities are used for the following relations, were the parameters of the elliptic curve $(\mathbb{G}, P, q)$ are implicit public inputs:

1. $R_{DL} := \{(Q, w) | Q = wP\}$, proves the knowledge of the discrete log of an elliptic curve point.
2. $R_{\mathsf{HPS-DL}} := \{(\mathsf{hp}, (c_1, c_2), Q_1); (x_1, w) | (c_1, c_2) = \mathsf{Enc}((u, w); (\mathsf{hp}, x_1)) \wedge (c_1, w) \in \mathsf{R} \wedge Q_1 = x_1 P\}$, 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 $\mathcal{F}_{\mathsf{zk}}^{R_{DL}}$, $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R_{DL}}$ can be instantiated using Schnorr proofs [Sch91]. For the $R_{\mathsf{HPS-DL}}$ 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, 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 $\mathcal{X} \times \varPi$ is not a valid ciphertext.

For our instantiations, we will introduce specific and efficient proofs. Note that in any case, we needn't 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.

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

We here provide our generic construction for two-party ECDSA signing from hash proof systems (Figure 5), along with a proof that the protocol is secure in the Ideal/Real paradigm (Theorem 1). To this end, we must argue the indistinguishability of an adversary's view – corrupting either party P_1 or P_2 – in real and simulated executions. In Cramer-Shoup like encryption schemes (resulting from HPSs as described in Subsection 3.3), the chosen plaintext attack indistinguishability of ciphertexts allows for the challenger in the security game to sample the secret hashing key hk, and compute the resulting projection key hp. Thus hk is *known* to the challenger. Indeed here, in order to prove indistinguishability, the challenger 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 challenger *must* know the secret hashing key. The hardness of the subset membership problem ensures this goes unnoticed to any polynomial time adversary. Then the smoothness of the projective hash family allows one to replace the plaintext value by some random element from the plaintext space, thus guaranteeing the indistinguishability of the resulting encryption scheme. 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 challenger 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.

Moreover, though this technique suffices for a game-based definition, it does not for simulation-based security definitions. Thus, in order to be able to prove their protocol secure using simulation, they use a rather non-standard and interactive assumption, called Paillier-EC assumption and recalled in Appendix IV. Thanks to the framework we have chosen to adopt, we are able to avoid such an interactive 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}.

Theorem 1. *Assume* HPS *is a* $(\Upsilon, \mathcal{F}, \delta_s, \delta_{\mathcal{L}}, \delta_{\mathsf{DE}})$*-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 5 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

P_1	$\mathsf{IKeyGen}(\mathbb{G}, P, q)$	P_2
$x_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$Q_1 \leftarrow x_1 P$	$\xrightarrow{(\mathsf{com\text{-}prove}, 1, Q_1, x_1)} \mathcal{F}^{R_{DL}}_{\mathsf{com\text{-}zk}} \xrightarrow{(\mathsf{proof\text{-}receipt}, 1)}$	
		$x_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$
P_1 aborts if $(\mathsf{proof}, 2, Q_2)$ not received.	$\xleftarrow{(\mathsf{proof}, 2, Q_2)} \mathcal{F}^{R_{DL}}_{\mathsf{zk}} \xleftarrow{(\mathsf{prove}, 2, Q_2, x_2)}$	$Q_2 \leftarrow x_2 P$
	$\xrightarrow{(\mathsf{decom\text{-}proof}, 1)} \mathcal{F}^{R_{DL}}_{\mathsf{com\text{-}zk}} \xrightarrow{(\mathsf{decom\text{-}proof}, 1, Q_1)}$	
$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$ $\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$ Sample $(u, w) \in \mathsf{R}$		
$c_{\mathsf{key}} \leftarrow \mathsf{Enc}((u, w); (\mathsf{hp}, x_1))$	$\xrightarrow{(\mathsf{prove}, 3, (\mathsf{hp}, c_{\mathsf{key}}, Q_1), (x_1, w))} \mathcal{F}^{R_{\mathsf{HPS-DL}}}_{\mathsf{zk}} \xrightarrow{(\mathsf{proof}, 3, (\mathsf{hp}, c_{\mathsf{key}}, Q_1))}$	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	$\mathsf{ISign}(m, sid)$	P_2
$k_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$		
$R_1 \leftarrow k_1 P$	$\xrightarrow{(\mathsf{com\text{-}prove}, sid\Vert 1, R_1, k_1)} \mathcal{F}^{R_{DL}}_{\mathsf{com\text{-}zk}} \xrightarrow{(\mathsf{proof\text{-}receipt}, sid\Vert 1)}$	$k_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $R_2 \leftarrow k_2 P$
P_1 aborts if $(\mathsf{proof}, sid\Vert 2, R_2)$ not received.	$\xleftarrow{(\mathsf{proof}, sid\Vert 2, R_2)} \mathcal{F}^{R_{DL}}_{\mathsf{zk}} \xleftarrow{(\mathsf{prove}, sid\Vert 2, R_2, k_2)}$	
	$\xrightarrow{(\mathsf{decom\text{-}proof}, sid\Vert 1)} \mathcal{F}^{R_{DL}}_{\mathsf{com\text{-}zk}} \xrightarrow{(\mathsf{decom\text{-}proof}, sid\Vert 1, R_1)}$	P_2 aborts if $(\mathsf{decom\text{-}proof}, sid\Vert 1, R_1)$ 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)	$\xleftarrow{c_3}$	$c_3 \leftarrow \mathsf{EvalSum}(\mathsf{hp}, c_1, c_2)$

Fig. 5: Two-Party ECDSA Key Generation and Signing Protocols from HPSs

$\mathcal{A}$ 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 $R_{\mathsf{HPS-DL}}$, 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 $\mathcal{F}^{R_{DL}}_{\mathsf{com\text{-}zk}}$, 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 $\mathcal{F}^{R_{DL}}_{\mathsf{zk}}$ thereby $\mathcal{S}$ simulating a proof of knowledge of x_2 s.t. $Q_2 = x_2 P$.
5. $\mathcal{S}$ receives $(\mathsf{decom} - \mathsf{proof}, 1)$ as $\mathcal{A}_1$ intends to send to $\mathcal{F}^{R_{DL}}_{\mathsf{com\text{-}zk}}$ 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 $\mathcal{F}^{R_{\mathsf{HPS-DL}}}_{\mathsf{zk}}$.
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 $\mathcal{F}_{\mathsf{zk}}^{R_{\mathsf{HPS-DL}}}$ – 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) – 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 $\mathsf{lKeyGen}(\mathbb{G}, P, q)$ and sends $(\mathsf{proof\text{-}receipt}, 1)$ as $\mathcal{A}_2$ expects to receive from $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R_{DL}}$.
3. $\mathcal{S}$ receives $(\mathsf{prove}, 2, Q_2, x_2)$ as $\mathcal{A}_2$ intends to send to $\mathcal{F}_{\mathsf{zk}}^{R_{DL}}$.
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 $\mathcal{F}_{\mathsf{com\text{-}zk}}^{R_{DL}}$. 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 $\mathcal{F}_{\mathsf{zk}}^{R_{\mathsf{HPS-DL}}}$.
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 $\mathcal{F}^{R_{DL}}_{\mathsf{com\text{-}zk}}$.
4. $\mathcal{S}$ receives $(\mathsf{prove}, sid||2, R_2, k_2)$ as $\mathcal{A}_2$ intends to send to $\mathcal{F}^{R_{DL}}_{\mathsf{zk}}$.
5. Using the extracted value k_2, $\mathcal{S}$ verifies that R_2 is a non zero point and that $R_2 = k_2P$. 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 $\mathcal{F}^{R_{DL}}_{\mathsf{com\text{-}zk}}$. 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 $\alpha = k_2^{-1} \cdot (m' + r \cdot x_2 \cdot \tilde{x}_1) \mod q$ 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 $\mathcal{A}_2$.

Game_0	Game_1
$Q \leftarrow x_1x_2P$	$Q \leftarrow x_1x_2P$
$\vdots$	$\vdots$
$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$	$\mathsf{hk} \hookleftarrow \mathcal{D}_{\mathsf{hk}}$
$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$	$\mathsf{hp} \leftarrow \mathsf{projkg}(\mathsf{hk})$
	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 $\mathcal{A}_2$
$\vdots$	$\vdots$
$R \leftarrow k_1k_2P$, $r \leftarrow r_x \bmod q$	$R \leftarrow k_1k_2P$, $r \leftarrow r_x \bmod q$
$\vdots$	$\vdots$
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)

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 $Q = x_1x_2P$ 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

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 $(\mathsf{prove}, sid\vert\vert 2, R_2, k_2)$
$\vdots$	$\vdots$
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 $(\mathsf{prove}, sid\vert\vert 2, R_2, k_2)$	Extr. k_2 from $(\mathsf{prove}, sid\vert\vert 2, R_2, k_2)$	Extr. k_2 from $(\mathsf{prove}, sid\vert\vert 2, R_2, k_2)$
$\vdots$	$\vdots$	$\vdots$
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		

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 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 $y \in \langle \varUpsilon \rangle$. Recall that $\varPsi$ is a generator for K_{hk} such that that $\mathsf{hash}_{\varPsi}(\varUpsilon) = f$ (*cf.* Remark 2). We denote $b \in \mathbf{Z}/q\mathbf{Z}$ the unique value such that $\mathsf{hash}_{\varPsi}(y) = f^b$. 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 $\mathcal{F}$ 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 $\mathsf{hk} = \gamma \cdot \varPsi$ and $\mathsf{hk}' = \gamma' \cdot \varPsi$, such that $\mathsf{hash}_{\mathsf{hk}}(\varUpsilon) = f^{\gamma}$ and $\mathsf{hash}_{\mathsf{hk}'}(\varUpsilon) = f^{\gamma'}$. Now since $\mathsf{hash}_{\varPsi}(y) = f^b$, it holds that

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

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 $\mathsf{Game}_3{}'$, 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 $\mathsf{Game}_3{}'$ are redundant, we only keep check (2).

Observation. The view of $\mathcal{A}_2$ in Game_2 and in $\mathsf{Game}_3{}'$ 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 $\mathsf{Game}_3{}'$ 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 $c_3 = (u_3, e_3) \in \mathcal{X} \times \varPi$ sent by $\mathcal{A}_2$. By the decomposability of PHF we know there exist unique $z_3 \in \mathcal{L}$, $y_3 \in \langle \varUpsilon \rangle$ such that $u_3 = z_3 y_3$. Moreover there exists a unique $b_3 \in \mathbf{Z}/q\mathbf{Z}$ such that $\mathsf{hash}_{\varPsi}(y_3) = f^{b_3}$. By the homomorphic properties of PHF the decryption algorithm applied to c_3 with decryption key hk (resp. hk') returns $\perp$ if $e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1} = e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(z_3)^{-1} \cdot \mathsf{hash}_{\mathsf{hk}}(y_3)^{-1} \notin \mathcal{F}$ (resp. $e_3 \cdot \mathsf{hash}_{\mathsf{hk}'}(u_3)^{-1} = e_3 \cdot \mathsf{hash}_{\mathsf{hk}'}(z_3)^{-1} \cdot \mathsf{hash}_{\mathsf{hk}'}(y_3)^{-1} \notin \mathcal{F}$). 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 $\mathcal{F}$. 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: $e_3 = \mathsf{hash}_{\mathsf{hk}}(z_3)f^{h_3} = \mathsf{hash}_{\mathsf{hk}'}(z_3)f^{h_3}$ with $h_3 \in \mathbf{Z}/q\mathbf{Z}$. We thus have:

$$\begin{aligned} \alpha &:= \log_f(e_3 \cdot \mathsf{hash}_{\mathsf{hk}}(u_3)^{-1}), \\ &= h_3 - b_3 \cdot \gamma \bmod q \end{aligned} \qquad \text{and} \qquad \begin{aligned} \alpha' &:= \log_f(e_3 \cdot \mathsf{hash}_{\mathsf{hk}'}(u_3)^{-1}), \\ &= h_3 - b_3 \cdot \gamma' \bmod q \end{aligned}$$

such that, injecting Equation (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 ($\alpha - \alpha' = k_2^{-1}rx_2(\tilde{x}_1 - x_1) \bmod q$). This occurs if $b_3 = k_2^{-1}rx_2 b \bmod q$, 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:

$$\alpha = k_2^{-1}(m' + r\tilde{x_1}x_2) \Leftrightarrow \alpha' + k_2^{-1}rx_2(\tilde{x}_1 - x_1) = k_2^{-1}(m' + r\tilde{x_1}x_2) \Leftrightarrow \alpha' = k_2^{-1}(m' + rx_2x_1).$$

3. ($\alpha \neq \alpha' \bmod q$) and ($\alpha - \alpha' \neq k_2^{-1}rx_2(\tilde{x}_1 - x_1) \bmod q$). We here consider three sub-cases:
   (a) Either both tests fail for α and test (2) fails for α'; i.e. $\alpha \neq k_2^{-1}(m' + r\tilde{x}_1x_2) \bmod q$; and $\alpha, \alpha' \neq k_2^{-1}(m' + rx_1x_2) \bmod q$. This results in identical views from $\mathcal{A}_2$'s perspective.
   (b) Either the check on α' passes. This means that:

$$\alpha' = k_2^{-1}(m' + rx_1x_2) \bmod q.$$

   Since $\alpha - \alpha' \neq k_2^{-1}rx_2(\tilde{x}_1 - x_1) \bmod q$ 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:

$$\begin{aligned} & \tilde{x}_1 = (k_2\pi_1 - m' - k_2b_3b^{-1}(\pi_0 - \tilde{x}_1))(x_2r)^{-1} \bmod q \\ \Leftrightarrow \quad & \tilde{x}_1(1 - k_2b_3(bx_2r)^{-1}) = (k_2\pi_1 - m' - k_2b_3b^{-1}\pi_0)(x_2r)^{-1} \bmod q \end{aligned}$$

As we dealt with $b_3 = k_2^{-1}rx_2b \bmod q$ in case 2, here $b_3 \neq k_2^{-1}rx_2b \bmod q$, and $1 - k_2b_3(bx_2r)^{-1}$ is invertible mod q so we can write:

$$\tilde{x}_1 = (k_2\pi_1 - m' - k_2b_3b^{-1}\pi_0)(x_2r)^{-1}(1 - k_2b_3(bx_2r)^{-1})^{-1} \bmod q, \qquad (2)$$

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 (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 (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 $\alpha = k_2^{-1}(m' + rx_1x_2) \bmod q$. 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. □

4 Instantiation in Class Groups of an Imaginary Quadratic Field

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. For that we will use a linearly homomorphic encryption scheme modulo a prime number, denoted CL in the following, introduced in [CL15] using a group with an easy Dlog subgroup, with a concrete instantiation using class groups of quadratic fields. In order to define a HPS, we use the recent results of [CLT18] that enhance the CL framework by introducing a hard subgroup membership assumption (HSM). We first give the definition of this assumption in the context of a group with an easy Dlog subgroup, then the instantiation with class groups, and then define a HPS from HSM and prove that it has the required properties to instantiate the generic construction in Section 3.

4.1 A Hard Subgroup Membership Assumption

To start with, we explicitly define the generator $\mathsf{GenGroup}$ used in the framework of a group with an easy Dlog subgroup introduced in [CL15] and enhanced in [CLT18], with small modifications as discussed below.

Definition 8. *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 Dlog 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*:*

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

Remark 3. In this definition, there are a few modifications compared to the definition of [CLT18]. Namely we take as input the prime q instead of having Gen generating it, and we output the group $\widehat{G}$ from which the group G with an easy Dlog subgroup F is produced. In practice, with the concrete instantiation with class groups, this is a just a matter of rewriting: the prime q was generated independently of the rest of the output in [CL15,CLT18] so it can be an input of the algorithm, and the group $\widehat{G}$ would be the class group which was implicitly defined by its discriminant. We note 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.

We recall here the definition of a hard subgroup membership (HSM) problem within a group with an easy Dlog subgroup as defined in [CLT18]. HSM is closely related to Paillier's DCR assumption. Such hard subgroup membership problems are based on a long line of assumptions on the hardness of distinguishing powers in groups. In short, DCR and HSM are essentially the same assumption but in different groups, hence there is no direct reduction between them. 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 [CLT18] within class groups, this being said, cryptography based on class groups is now well established, and is seeing renewed interest as it allows versatile and efficient solutions such as encryption switching protocols [CIL17], inner product functional encryption [CLT18] or verifiable delay functions [BBBF18,Wes19].

In Def. 8, one has $G = F \times G^q$. The assumption is that it is hard to distinguish the elements of G^q in G.

Definition 9 **(**HSM **assumption).** *We say that* $\mathsf{GenGroup}$ *is the generator of a* HSM *group with easy Dlog 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.*

Class groups Our instantiation makes use of class groups of orders of imaginary quadratic fields. We refer the interested reader to [BH01] for background on this algebraic object and its early use in cryptography. We here briefly sketch an instantiation of algorithm GenGroup in Definition 8, following [CL15, Fig. 2]. The formal description is given in Fig. 6 below and concrete details can be found in [CL15]. 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 $h(\Delta_q) = q \cdot h(\Delta_K)$ where $h(\Delta_K)$ is the class number, *i.e.*, the order of $Cl(\Delta_K)$, the class group of fundamental discriminant Δ_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. Informally, 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]).

Following [CL15, Fig. 2], 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], 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 $\langle f \rangle$.

Note that following [CL15] the bit size of q must have at least λ bits, where λ is the security parameter, which is the case for ECDSA: q will be the order of the elliptic curve. 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

```
$\underline{\mathsf{Gen}(1^\lambda, q)}$

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 $C(\Delta_q)$ and $G^q \leftarrow \langle g_q \rangle$
7. Set $g \leftarrow g_p \cdot f$ and $G \leftarrow \langle g \rangle$
8. Return $(\tilde{s}, g, f, g_q, \widehat{G}, G, F, G^q)$
```

Fig. 6: Group generator Gen

method to solve the Dlog 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 $\mathcal{O}(L_{|\Delta_K|}[1/2, o(1)])$, which allows to use asymptotically shorter keys compared to protocols using classical problems that are solved in subexponential complexity $\mathcal{O}(L[1/3, o(1)])$ (see Section 5 for concrete sizes for η).

4.2 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.

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 $x \in G$, $\mathsf{hash}_{\mathsf{hk}}(x) := x^{\mathsf{hk}}$.
- 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).

Lemma 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 1 in Appendix II) 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. □

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 Subsection 3.3 using the above HPS results in the encryption scheme called $\mathsf{HSM\text{-}CL}$ in [CLT18], which is linearly homomorphic modulo q and $\mathsf{ind} - \mathsf{cpa}$ under the HSM assumption. We describe this scheme in Fig. 7 for completeness. 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 $\{0, \ldots, S\}$.

```
Algorithm KeyGen($1^\lambda, q$)

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 $pk \leftarrow (\tilde{s}, g_q, f, p, h)$
4. Set $sk \leftarrow x$
5. Return $(pk, sk)$
```

```
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 Solve($M$)
```

Fig. 7: Description of the $\mathsf{HSM\text{-}CL}$ encryption scheme

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* HSM-*CL encryption scheme.* As explained in Lemma 1, 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).

4.3 A zero-knowledge proof for $R_{\mathsf{CL-DL}}$

We describe here the ZKPoK for $R_{\mathsf{HPS-DL}}$ used for our instantiation with the encryption scheme of Fig. 7 and denote it $R_{\mathsf{CL-DL}}$. It relies on the Schnorr-like GPS (statistically) 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 $c_{key} := (c_1, c_2)$. If c_{key} is a valid encryption of x_1 under public key pk it holds that $c_{key} = (g_q^r, f^{x_1}\mathsf{pk}^r)$ for some $r \in \{0, \dots, S\}$. The protocol $R_{\mathsf{CL-DL}}$ provides a ZKPoK for the following relation:

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

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` $u_1 \in [0, A + S[$ $t_1 \cdot c_2^k = \mathsf{pk}^{u_1} \cdot f^{u_2}$ $t_2 + [k]Q = [u_2]P$ $t_3 \cdot c_1^k = g_q^{u_1}$

Fig. 8: The zero-knowledge proof of knowledge $R_{\mathsf{CL-DL}}$

The following theorem, whose proof is given in Appendix III, states the security of the zero-knowledge proof of knowledge $R_{\mathsf{CL-DL}}$.

Theorem 2. *The protocol described in Figure 8 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.*

4.4 Two-Party Distributed ECDSA Protocol from HSM

The protocol results from a direct application of Subsection 3.6 using the HPS defined in Subsection 4.2, an the $R_{\mathsf{CL-DL}}$ proof of the previous subsection. Therefore we defer the detailed protocol to Appendix V, and simply state the following theorem.

Theorem 3. *Assuming* GenGroup *is the generator of a* HSM *group with easy Dlog subgroup F, then the protocol of Appendix V 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).*

5 Implementation and Efficiency Comparisons

In this section we compare an implementation of our protocol with Lindell's protocol of [Lin17]. For fair comparison, we implement both protocols 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 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.

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 Fig. 9a. 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.

Curve	Sec. Param.	Modulus	Expo. (ms)	Keygen (ms)	Signing (ms)	Keygen (b)	Signing (b)
P-256	112	2048	**7**	**2 133**	**20**	881 901	5 636
P-256	128	3072	**22**	**6 340**	**49**	1 317 101	7 684
P-384	192	7680	214	65 986	**437**	3 280 429	17 668
P-521	256	15360	1196	429 965	2 415	6 549 402	33 832

(a) Lindell's Protocol with Paillier

Curve	Sec. Param.	Discriminant	Expo. (ms)	Keygen (ms)	Signing (ms)	Keygen (b)	Signing (b)
P-256	112	1348	32	5 521	101	**178 668**	**4 748**
P-256	128	1827	55	9 350	170	**227 526**	**5 706**
P-384	192	3598	**212**	**35 491**	649	**427 112**	**10 272**
P-521	256	5971	**623**	**103 095**	**1 888**	**688 498**	**16 078**

(b) Our Protocol with HSM-CL

Fig. 9: Experimental results (timings in ms, sizes in bits)

5.2 Our Protocol with HSM-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 $R_{\mathsf{CL-DL}}$ zero-knowledge proof of knowledge of Fig. 8. Note that exponentiations in $\langle f \rangle$ are almost free as seen in Subsection 4.1. 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 of our comparisons for any of the considered security levels.

The results are summarized in Fig. 9b. 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].

5.3 Comparison

Figure 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 level 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 shown in the introduction, 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. Appendix V). 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.

6 Instantiation of our Generic Construction Using DCR

As stated at the end of the introduction, we can instantiate the generic construction of Section 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 $R_{\mathsf{HPS-DL}}$ 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 $R_{\mathsf{HPS-DL}}$ 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.

7 Conclusion

Inspired by Lindell's scheme, we have provided the first generic construction for two-party ECDSA signing from hash proof systems which are homomorphic modulo a prime number. Theoretically, our construction allows for a simulation-based proof of security that is both tight and requires no interactive assumptions, due to the structure of the underlying semantically secure homomorphic encryption schemes. Practically, we provide a detailed instantiation, and C implementation, from class groups of imaginary quadratic fields using the CL framework. This yields a better performance than Lindell's Paillier-based scheme for high levels of security, and same order of magnitude for standard levels. Our solution becomes faster than Lindell's from 192-bits of security upwards. Improvements could come from advances in ideal arithmetic in imaginary quadratic fields (see [IJS10] for instance). Recent proposals of verifiable delay functions based on class groups should also motivate research in this area (for example the Chia Network [Chi] has opened a competition for this).

Moreover, the bottleneck of our instantiation is the use of binary challenges in a zero knowledge proof of knowledge, used during key generation, in order to cope with the fact we are working in a cyclic subgroup of a group of unknown order and that we can not check that elements belong to the subgroup. There have been many proposals to deal with generalized Schnorr proofs in groups of unknown order (see for instance the framework of [CKY09] using safeguard groups, or [TW12]). For the case of subgroups of $(\mathbf{Z}/n\mathbf{Z})^\times$, efficient solutions for this type of proofs enlarge the challenge space, and rely on variants of the strong RSA assumption. For class groups, there have been informal proposals (see [DF02] for instance). However, 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). Moreover, as suggested in [BBF18], there may be other approaches to find low order elements in class groups. Advances in our understanding of class groups would lead to substantial efficiency improvements in several areas of cryptography.

Last but not least, our work focuses on the two party case. We believe that the ideas of our generic construction will lead to improvements in the general case of threshold ECDSA signatures. We leave this for future work.

Acknowledgements: The authors would like to thank Benoit Libert for fruitful discussions. We also thank Rosario Gennaro and Steven Goldfeder for questions that eventually led us to find a problem in the original proof. This work was supported by the Universita' degli Studi di Catania,"Piano della Ricerca 2016/2018 Linea di intervento 2", and the French ANR ALAMBIC project (ANR-16-CE39-0006).

References

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.

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.

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.

BH03. M. L. Bauer and S. Hamdy. On class group computations using the number field sieve. In *ASIACRYPT 2003*, *LNCS* 2894, pages 311–325. Springer, Heidelberg, November / December 2003.

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.

Boy86. C. Boyd. Digital multisignature. *Cryptography and Coding*, pages 241–246, 1986.

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.

CH89. R. A. Croft and S. P. Harris. Public-key cryptography and reusable shared secret. *Cryptography and Coding*, pages 189–201, 1989.

Chi. Chia. https://www.chia.net/.

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.

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.

CL15. G. Castagnos and F. Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA 2015*, *LNCS* 9048, pages 487–505. Springer, Heidelberg, April 2015.

CLT18. 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.

Coh00. H. Cohen. *A course in computational algebraic number theory.* Springer-Verlag, 2000.

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.

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.

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.

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.

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.

GJKR96. 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.

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.

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.

HL10. C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer-Verlag, 1st edition, 2010.

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.

IJS10. L. Imbert, M. J. Jacobson Jr., and A. Schmidt. Fast ideal cubing in imaginary quadratic number and function fields. *Advances in Mathematics of Communications*, 4(2):237–260, 2010.

Jac00. M. J. Jacobson Jr. Computing discrete logarithms in quadratic orders. *Journal of Cryptology*, 13(4):473–492, September 2000.

Lin16. Y. Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. http://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.

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.

MR04. P. D. MacKenzie and M. K. Reiter. Two-party generation of DSA signatures. *Int. J. Inf. Sec.*, 2(3-4):218–239, 2004.

PAR18. PARI Group, Univ. Bordeaux. *PARI/GP version* `2.11.1`, 2018. available from http://pari.math.u-bordeaux.fr/.

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.

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).

Wes19. B. Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology – EUROCRYPT 2019*, pages 379–407, Cham, 2019. Springer International Publishing.

Auxiliary Supporting Material

I A brief definition of interactive zero-knowledge proofs

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 — who detains a witness for a given statement — must convince V that the statement is true without revealing anything other than the truth of this statement to V.

Specifically, if $\mathcal{L}$ is a language, $x \in \mathcal{L}$ is a *true* statement while $x \notin \mathcal{L}$ is a *false* statement; 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, the following properties must hold:

- *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.

II A technical Lemma on Distributions

Lemma 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 $\mathcal{U}_{\mathcal{A}} := \{a, b, x_b, x_a | (a, b, x) \hookleftarrow \mathcal{V};\ x_b \leftarrow x \mod b;\ x_a \overset{\$}{\leftarrow} \mathbf{Z}/a\mathbf{Z}\}$ 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 $\beta^* \overset{\$}{\leftarrow} \{0, 1\}$
3. If $\beta^* = 0$, then $x_a^* \overset{\$}{\leftarrow} \mathbf{Z}/a\mathbf{Z}$
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. □

III Proof of Theorem 2

In this section, we prove the following theorem.

Theorem 1. *The protocol described in Figure 8 is a statistical zero-knowledge proof of knowledge with soundness $2^{-\ell}$, as long as ℓ is polynomial and $\ell S/A$ is negligible.*

Proof. We prove completeness, soundness and zero-knowledge. Completeness follows easily by observing that when $((\mathsf{pk}, (c_1, c_2), Q_1); (x_1, r)) \in R_{\mathsf{CL-DL}}$, 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 $R_{\mathsf{CL-DL}}$) without knowing the witnesses (x_1, r) for $(\mathsf{pk}, (c_1, c_2), Q_1)$ in the relation $R_{\mathsf{CL-DL}}$.

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[1] to that in a simulated execution.

[1] 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

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.

□

IV Lindell's new interactive assumption

In order to prove the security of his 2-party ECDSA, Lindell introduced in [Lin17] the following ad hoc interactive assumption, called Paillier-EC assumption. It is defined via the following random experiment.

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

where $1 \leftarrow \mathcal{O}_{c^\star}(c', \alpha, \beta)$ if and only if $\mathsf{Paillier.Dec}(1^\lambda, sk, c') = \alpha + \beta\omega_{b^\star} \mod q$ and $\mathcal{O}$ stops after the first time it returns 0.

The Paillier-EC assumption is hard if for every probabilistic polynomial-time adversary $\mathcal{A}$ there exists a negligible function ν such that $\Pr[\mathbf{Exp}_{\mathcal{A}}(1^\lambda) = 1] \le \frac{1}{2} + \nu(n)$.

V Two-Party ECDSA from **HSM**

P_1	$KeyGen(\mathbb{G}, P, q, \widehat{G}, g_q)$	P_2
$x_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $Q_1 \leftarrow x_1P$		
	$\xrightarrow{(\mathsf{com\text{-}prove},1,Q_1,x_1)} \mathcal{F}^{R_{DL}}_{\mathsf{zk-com}} \xrightarrow{(\mathsf{proof\text{-}receipt},1)}$	
P_1 aborts if $(\mathsf{proof}, 2, Q_2)$ not received.		$x_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $Q_2 \leftarrow x_1P$
	$\xleftarrow{(\mathsf{proof},2,Q_2)} \mathcal{F}^{R_{DL}}_{\mathsf{zk}} \xleftarrow{(\mathsf{prove},2,Q_2,x_2)}$	
	$\xrightarrow{(\mathsf{decom\text{-}proof},1)} \mathcal{F}^{R_{DL}}_{\mathsf{zk-com}} \xrightarrow{(\mathsf{decom\text{-}proof},1,Q_1)}$	
$x, \rho \hookleftarrow \mathcal{D}_q$ $h \leftarrow g_q^x$		
$c_{key} = (c_{key,1}, c_{key,2})$ $= (g_q^\rho, h^\rho f^{x_1})$	$\xrightarrow{(\mathsf{prove},3,(h,c_{key},Q_1),(x_1,\rho))} L_{\mathsf{CL-DL}} \xrightarrow{(\mathsf{proof},3,(h,c_{key},Q_1))}$	P_2 aborts unless $(\mathsf{decom\text{-}proof}, 1, Q_1)$, $(\mathsf{proof}, 3, (h, c_{key}, Q_1))$ received and $h \in \widehat{G}$.
$Q \leftarrow x_1Q_2$		$Q \leftarrow x_2Q_1$

P_1	$Sign(m, \mathsf{sid})$	P_2
$k_1 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $R_1 \leftarrow k_1P$		
	$\xrightarrow{(\mathsf{com\text{-}prove},\mathsf{sid}\Vert 1,R_1,k_1)} \mathcal{F}^{R_{DL}}_{\mathsf{zk-com}} \xrightarrow{(\mathsf{proof\text{-}receipt},\mathsf{sid}\Vert 1)}$	
		$k_2 \xleftarrow{\$} \mathbf{Z}/q\mathbf{Z}$ $R_2 \leftarrow k_2P$
P_1 aborts if $(\mathsf{proof}, sid\Vert 2, R_2)$ not received.	$\xleftarrow{(\mathsf{proof},\mathsf{sid}\Vert 2,R_2)} \mathcal{F}^{R_{DL}}_{\mathsf{zk}} \xleftarrow{(\mathsf{prove},\mathsf{sid}\Vert 2,R_2,k_2)}$	
	$\xrightarrow{(\mathsf{decom\text{-}proof},\mathsf{sid}\Vert 1)} \mathcal{F}^{R_{DL}}_{\mathsf{zk-com}} \xrightarrow{(\mathsf{decom\text{-}proof},\mathsf{sid}\Vert 1,R_1)}$	P_2 aborts if $(\mathsf{decom\text{-}proof}, sid\Vert 1, R_1)$ not received. $m' \leftarrow H(m)$
$R = (r_x, r_y) \leftarrow k_1R_2$ $r \leftarrow r_x \mod q$		$R = (r_x, r_y) \leftarrow k_2R_1$ $r \leftarrow r_x \mod q$ $\tau \hookleftarrow \mathcal{D}_q$ $c_1 = (c_{1,1}, c_{1,2}) \leftarrow (g_q^\tau, h^\tau f^{k_2^{-1}m'})$ $c_2 = (c_{2,1}, c_{2,2}) \leftarrow (c_{key,1}^{k_2^{-1}rx_2}, c_{key,2}^{k_2^{-1}rx_2})$ $c_3 = (c_{3,1}, c_{3,2}) \leftarrow (c_{1,1}c_{2,1}, c_{1,2}c_{2,2})$
	$\xleftarrow{\qquad\qquad c_3 \qquad\qquad}$	
$\alpha \leftarrow \mathsf{Solve}(c_{3,2}/c_{3,1}^x)$ $\hat{s} \leftarrow \alpha \cdot k_1^{-1}$ $s \leftarrow \min(\hat{s}, q - \hat{s})$ If not $\mathsf{Verif}(Q, m, (r, s))$ P_1 aborts Else Return (r, s)		