Promise Σ-protocol: How to Construct Efficient Threshold ECDSA from Encryptions Based on Class Groups

Yi Deng[1,2], Shunli Ma[1,2], Xinxuan Zhang[1,2], Hailong Wang[1,2], Xuyang Song[3], and Xiang Xie[3]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{deng, mashunli, zhangxinxuan, wanghailong9065}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] Shanghai Key Laboratory of Privacy-Preserving Computation, Shanghai, China
{songxuyang, xiexiang}@matrixelements.com

Abstract. Threshold Signatures allow n parties to share the ability of issuing digital signatures so that any coalition of size at least $t + 1$ can sign, whereas groups of t or fewer players cannot. The currently known class-group-based threshold ECDSA constructions are either inefficient (requiring parallel-repetition of the underlying zero knowledge proof with small challenge space) or requiring rather non-standard low order assumption. In this paper, we present efficient threshold ECDSA protocols from encryption schemes based on class groups *with neither assuming the low order assumption nor parallel repeating the underlying zero knowledge proof*, yielding a significant efficiency improvement in the key generation over previous constructions.

Along the way we introduce a new notion of *promise Σ*-protocol that satisfies only a weaker soundness called *promise extractability*. An accepting *promise Σ*-proof for statements related to class-group-based encryptions does not establish the truth of the statement but provides security guarantees (promise extractability) that are sufficient for our applications. We also show how to simulate homomorphic operations on a (possibly invalid) class-group-based encryption whose correctness has been proven via our *promise Σ*-protocol. We believe that these techniques are of independent interest and applicable to other scenarios where efficient zero knowledge proofs for statements related to class-group is required.

1 Introduction

Threshold Digital Signature Schemes [Des88] enable distributed signing amongst a group of individuals such that any subgroup which is larger than a certain predetermined size can jointly sign, whereas any group with fewer players cannot. Specifically, a t-out-of-n threshold signature scheme is a protocol that allows n

parties to jointly generate a common public verification key, along with n shares of the corresponding secret signing key, and allows any subgroup of at least $t+1$ parties to securely sign a given message distributedly. In addition to satisfying the standard unforgeability of signature schemes, threshold variants should provide security that no malicious party can subvert the protocol to extract another party's secret share, and no more than t cannot collude to generate a valid signature.

The Elliptic Curve Digital Signature Algorithm (ECDSA) has been widely used in various applications including TLS, DNSSec, SSH and cryptocurrencies such as Bitcoin and Ethereum. The efficiency and widespread adoptions of ECDSA make its threshold version become an active research topic recently. After the work [GJKR96] and [MR01], many improved protocols have been proposed in recent years both for the specific two-party case [Lin17, DKLs18, CCL+19] and for the more general t-out-of-n case [GGN16, BGG19, GG18, LN18, DKLs19, CCL+20]. Among these schemes, a common used primitive to study threshold ECDSA is additively homomorphic encryption such as Paillier encryption and CL encryption [CL15, CLT18]. The latter is an ElGamal-like encryption scheme based on class groups of unknown order that contain a prime-order subgroup where the discrete logarithm (DL) problem is tractable, and such a distinguished property enables the CL encryption scheme to support much larger message space than the traditional ElGamal scheme.

Protocols based on Paillier encryption. Gennaro et al. [GGN16] extended the technique of [MR01], and introduced a six-round t-out-of-n threshold signature. Boneh et al. [BGG19] optimized their extension in terms of computational efficiency, and reduced the number of rounds to four. Meanwhile Lindell [Lin17] optimized the protocol framework and proposed an efficient protocol in the two-party setting. Subsequently, Gennaro and Goldfeder[GG18, GG20], Lindell and Nof [LN18] presented efficient protocols in the multi-party case that supported efficient distributed key generation. Unfortunately, mainly due to the mismatch between the Paillier modulus and the ECDSA modulus, these schemes all require expensive zero knowledge proofs, such as costly range proofs.

Protocols based on CL encryption. Castagnos et al. [CCL+19, CCL+20] employed the CL encryption and presented bandwidth efficient protocols for the two-party case and multi-party case, respectively. The modulus, which defines the underlying message space of CL encryption, could be set as the same prime modulus as in ECDSA. Thus, these protocols are able to eliminate the expensive range proofs which are required in the Paillier-based protocols, and achieve low communication cost. However, it is challenging to design efficient zero knowledge proofs for CL ciphertexts. As discussed in Section 1.2, a malicious prover holding low-order elements can convince the verifier of an invalid ciphertext with high probability. In order to resist this low-order-element attack, the two-party protocol in [CCL+19] adopts a zero knowledge proof with a single bit challenge to prove the validity of a CL ciphertext, and repeats this subprotocol in parallel to achieve a negligible soundness error. To overcome the inefficiency caused by repetition, [CCL+20] proposed more efficient threshold ECDSA protocols relying on stronger and non-standard low order assumption, which essentially says that no one can find a low order element efficiently in the given class group.

On low order assumption and non-uniform security. We would like to stress that the new low order assumption on the class group of imaginary quadratic fields has been much less studied. Following the Cohen-Lenstra heuristic [Coh00], the probability that any integer d divides the order of the class group is approximately $1/d + 1/d^2$, and it seems inherent that the class group often contains low order elements. Boneh et al. [BBF18] suggested one possible approach to find low order elements in the class group. As stated in [CCL+19], computing square roots or finding elements of order 2 can be efficiently done in class groups knowing the factorization of the discriminant (which is public in the associated schemes). Recently Belabas et al. [BKSW20] show that breaking the low-order assumption is possible if the discriminant belongs to some special class of prime numbers.

We note that the low order assumption on class groups that actually contains low order elements would become false in the presence of non-uniform adversaries, which can be simply hardwired with a low order element in theory. Note that non-uniform security is implicitly required by almost all cryptographic protocols, since we often need to compose them with other protocols.

1.1 Our Contribution

In this paper, we introduce a new notion of *promise* Σ-protocol that satisfies only a weaker soundness called *promise extractability.* The *promise* Σ-protocols for statements involved in class-group-based encryptions relax the requirements of soundness but does provide security guarantees (promise extractability) that are sufficient for our applications. We also show how to simulate homomorphic operations on a (possibly invalid) class-group-based encryption whose correctness has been proven via our *promise* Σ-protocol. We believe that these techniques are of independent interests and applicable to other scenarios where efficient zero knowledge proofs for statements related to class-group is required.

Building on *promise* Σ-protocols, we present efficient two-party and multi-party threshold ECDSA from CL encryptions based on class groups under standard assumptions. Compared to [CCL+19] (resp. [CCL+20]), in the key generation phase our two-party protocol is about $15\times$ (resp. about $2\times$) faster, and about $17\times$ (resp. $2\times$) less in bandwidth. Compared to [CCL+20], *without resorting to the low order assumption and strong root assumption*, our multi-party protocol removes the time-consuming interactive setup phase. It also reduces the number of expensive exponential operation in class groups of each party from $14t - 10$ in [CCL+20] to $10t - 6$ in the signing phase, where t is the threshold. Note that 40-bit soundness error is considered in the above comparison. The improvement will be much better if 128-bit soundness error is required.

1.2 Technical Overview

In this section we mainly give a high-level overview on our new techniques in the two-party ECDSA, which can be naturally extended to the multi-party case.

Before going into a technical discussion, we briefly recall the ECDSA algorithm and its threshold variant. Given a group of points of an elliptic curve $\mathbb{G}$ with a generator G of prime order p, the verification key of a ECDSA algorithm is a point $Q \in \mathbb{G}$ and the signing key is x such that $Q = x \cdot G$. To sign a message m one first hashes it using a (publicly known) hash function H, chooses a random $k \in \mathbb{Z}_p$ and computes $R = kG$, then sets $r = r_x \bmod p$ where r_x is the x-coordinate of the elliptic curve point R. The signature is $(s = (k^{-1}(\mathsf{H}(m) + rx) \bmod p, r)$.

Let us give an example of two-party ECDSA of [Lin17] to illustrate how a distributed ECDSA works. In this case, two parties $\mathcal{P}_1$ and $\mathcal{P}_2$, holding multiplicative shares x_1 and x_2 respectively, execute a coin-tossing-like protocol to generate a verification key $Q = x_1x_2G$, then $\mathcal{P}_1$ computes a ciphertext c_{key} of x_1 using a homomorphic encryption scheme and sends it, together with a zero knowledge proof of its correctness, to $\mathcal{P}_2$. To sign a message m, $\mathcal{P}_1$ and $\mathcal{P}_2$ choose k_1 and k_2 at random respectively, and execute another coin-tossing-like protocol to generate a point $R = k_1k_2G$ and set $r = r_x \bmod p$. Finally, $\mathcal{P}_2$ homomorphically computes an encryption of $s' = k_2^{-1}\mathsf{H}(m) + k_2^{-1}rx_2x_1$ on the ciphertext c_{key} and sends the ciphertext to $\mathcal{P}_1$, who decrypts the ciphertext and obtains s', and outputs a signature $(r, s = k_1^{-1}s')$. As we mentioned before, both of the two popular $\mathsf{Paillier}$ and CL schemes for encrypting x_1 incur large computation/communication overheads.

Promise Σ-protocol for equality of messages. Towards achieving more efficient constructions of two-party ECDSA, our first idea is to use CL encryption scheme to encrypt x_1 and introduce a *promise* Σ-protocol for proving equality of message encoded into $Q_1 = x_1 \cdot G$ and the one encrypted in the CL ciphertext.

To better explain our new notion let us briefly describe a CL encryption scheme and see how a traditional efficient Σ-protocol fails to prove a CL ciphertext. CL encryption scheme works on a tuple of parameters of groups $(\tilde{s}, g, f, g_p, \hat{\mathcal{G}}, \mathcal{G}, \mathcal{F}, \mathcal{G}^p)$ [CL15]. The finite abelian group $\hat{\mathcal{G}}$ is of order $p\hat{s}$ where p is prime, $\hat{s}$ is unknown but with an upper bound $\tilde{s}$ and $\gcd(p, \hat{s}) = 1$. The cyclic group $\mathcal{G} := \langle g \rangle$ of order ps is a subgroup of $\hat{\mathcal{G}}$ in which the Decisional Diffie-Hellman (DDH) assumption holds and $s|\hat{s}$ is also unknown. $\mathcal{G}$ contains a unique cyclic subgroup of order p, $\mathcal{F} := \langle f \rangle$, where the DL problem is *solvable*. Group $\mathcal{G}^p := \langle g_p \rangle$ is the subgroup of p-th powers in $\mathcal{G}$ (of unknown order s). Similar to the $\mathsf{ElGamal}$ scheme, the public and secret keys of a CL scheme are (g_p, f, h) and d (such that $h = g_p^d$) respectively, and a ciphertext of m is of the form $(c_1, c_2) = (g_p^r, h^r f^m)$, where $r \leftarrow [0, S]$ and S chosen in practice is a rough upperbound on the unknown order s of group $\mathcal{G}^p$. Note that since the DL problem over $\mathcal{F}$ is easy, one can decrypt such a CL ciphertext even when $|m| = p$.

When applying a traditional Σ-protocol with large challenge space to prove the plaintext knowledge of a CL ciphertext, one will obtain a transcript $((a_1 = g_p^{s_r}, a_2 = h^{s_r} f^{s_m}), e, (z_r = s_r + er, z_m = s_m + em \bmod p))$ where $s_r \leftarrow [0, U)$ for some sufficiently large U and $s_m \in \mathbb{Z}_p$ are randomness used to mask r and m, respectively. If this transcript is accepted, then it holds that $z_r \in [0, U+(p-1)S)$, $g_p^{z_r} = a_1 c_1^e$ and $h^{z_r} f^{z_m} = a_2 c_2^e$. However, an accepting proof does not imply the correctness of the ciphertext. That is, a traditional Σ-protocol with large challenge space does not enjoy special soundness here. For example, it is easy to see that a malicious prover P^* holding low-order elements (g', h') (g' may be equal to h') could convince the verifier of a false statement $(c_1' = g'c_1, c_2' = h'c_2)$ as long as the challenge e is divided by both the order of g' and h'. This is why the work [CCL+19] adopts a Σ-protocol with a single-bit challenge for proving knowledge of plaintext then parallelizes it to achieve a negligible soundness error.

To overcome this efficiency issue, we have $\mathcal{P}_1$ encode x_1 twice to obtain $Q_1 = x_1 \cdot G$ and a CL ciphertext c_{key} and then use a *promise* Σ-protocol (with large challenge space) to prove equality of the plaintexts. Let $c_{key} = (c_{key,1}, c_{key,2})$ be an encryption of x_1 under the public key $\mathsf{pk} = (g_p, f, h)$ of CL scheme, where $c_{key,1} = g_p^r, c_{key,2} = h^r f^{x_1}$ and r is the randomness used to encrypt x_1. Our *promise* Σ-protocol can be built from the Schnorr protocol and the above "Σ-protocol" (that does not enjoy special soundness) for CL ciphertext (see Section 3.1 for a detailed description).

An interesting observation on the above "Σ-protocol" for CL ciphertext is that, given two accepting transcripts (a_1, a_2, e, z_r, z_m) and $(a_1, a_2, e', z_r', z_m')$ $(e \neq e')$, one could obtain $c_{key,1}^{\tilde{e}} = g_p{}^{\tilde{z}_r}$ and $c_{key,2}^{\tilde{e}} = h^{\tilde{z}_r} f^{\tilde{z}_m}$, where $\tilde{e} = e - e' \in \mathbb{Z}_p$, $\tilde{z}_r = z_r - z_r' \in \mathbb{Z}$ and $\tilde{z}_m = z_m - z_m' \in \mathbb{Z}_p$. Since the Schnorr protocol for $Q_1 = x_1 G$ enjoys special soundness, one can extract x_1 by standard rewinding technique and deduce that, if a prover can make a verifier accept a *promise* Σ-proof for equality of plaintexts of (Q_1, c_{key}), it holds that $c_{key,2}^{\tilde{e}} = h^{\tilde{z}_r} f^{\tilde{z}_m} = h^{\tilde{z}_r} f^{\tilde{e} x_1}$. That is, we can modify $c_{key} = (c_{key,1}, c_{key,2})$ into a valid ciphertext $(c_{key,1}' = c_{key,1}^{\tilde{e}}, c_{key,2}' = c_{key,2}^{\tilde{e}})$ by picking e, e' (with $e \neq e'$) at random and setting $\tilde{e} = e - e'$. This provides us with an additional extraction strategy using the secret key sk: Given the secret key sk, one could efficiently decrypt $(c_{key,1}', c_{key,2}')$ and recover x_1 from the decrypted plaintext $\tilde{e}x_1$, and output x_1 if it satisfies $x_1 \cdot G = Q_1$.

Specifically, our *promise* Σ-protocol for equality of ciphertexts (Q_1, c_{key}) guarantees the following *promise* extractability: for any prover that makes the verifier accept with high probability,

1. With oracle access to this prover, there is an efficient extractor that extracts the message x_1 of Q_1 with probability negligibly close to 1.
2. If the public key pk are honestly generated, then there is an efficient extractor (without access to the prover) that, given the corresponding sk as input, it extracts the plaintext x_1 of Q_1 with probability negligibly close to 1.

Both properties turn out to be very useful in our constructions of two-party and multi-party ECDSA.

Simulating homomorphic operations on an invalid ciphertext. Suppose in the key generation of a two-party ECDSA $\mathcal{P}_1$ sends to $\mathcal{P}_2$ a pair (Q_1, c_{key}) which both encode x_1, along with a *promise* Σ-proof of equality of their plaintexts. Following the framework of [Lin17, CCL+19], in the final step of a two-party signing subprotocol, $\mathcal{P}_1$ and $\mathcal{P}_2$ hold (x_1, k_1, r) and (x_2, k_2, r) respectively, and compute a signature on message m as follows. $\mathcal{P}_2$ computes $b = k_2^{-1}m'$ (where m' is the hash value of m), $a = k_2^{-1}rx_2$, and a ciphertext of $ax_1 + b$ by homomorphic operations on $(c_{key,1}, c_{key,2})$, then sends the ciphertext to $\mathcal{P}_1$, who decrypts the ciphertext and computes a signature $(r, k_1^{-1}(ax_1 + b))$.

Let us abuse notation slightly and denote by $\mathcal{P}_2(r_1)$ the last step of $\mathcal{P}_2$ in which it generates a ciphertext of $ax_1 + b$ using randomness r_1. (It is convenient to think that a and b (along with the public keys) are "hardwired" into $\mathcal{P}_2(r_1)$.) Suppose the ciphertext $(c_{key,1}, c_{key,2})$ is computed under a public key $\mathsf{pk} = (\tilde{s}, p, g_p, f, h)$ of the CL encryption scheme. As shown in [CCL+19], if the ciphertext $(c_{key,1}, c_{key,2})$ is valid, say $c_{key,1} = g_p^r$ and $c_{key,2} = h^r f^{x_1}$, then $\mathcal{P}_2(r_1)$ can compute the ciphertext $(g_p^{r_1} \cdot c_{key,1}^a, h^{r_1} f^b \cdot c_{key,2}^a)$ homomorphically. Furthermore, there is a simulator $\mathcal{S}$ that given only $s' = ax_1 + b$ (without any knowledge of a or b) simulates the honest party $\mathcal{P}_2$ by computing $(g_p^{r_2}, h^{r_2} \cdot f^{s'})$, no matter how the public key pk is generated. However, as mentioned, in order to ensure the correctness of $(c_{key,1}, c_{key,2})$, $\mathcal{P}_1$ needs to run a parallel version of the standard Σ-protocol with single bit challenge for $(c_{key,1}, c_{key,2})$ due to the low-order-element attack, which is the major efficiency bottleneck.

In our case, when using *promise* Σ-protocol for proving equality of plaintexts of (Q_1, c_{key}), as we showed, it guarantees only that the $(c_{key,1}, c_{key,2})$ satisfies $c_{key,1} = g_p^{\tilde{z}/\tilde{e}}$ and $c_{key,2} = h^{\tilde{z}/\tilde{e}} f^{x_1}$ for some $\tilde{e} \in [-p+1, p-1] \setminus \{0\}, \tilde{z} \in [-U - (p-1)S + 1, U + (p-1)S - 1] \setminus \{0\}$. Now if we have the same $\mathcal{P}_2$ and $\mathcal{S}$, we obtain the following two ciphertexts[4]:

$$\begin{aligned}
\mathcal{P}_2(r_1):&\ \ (g_p^{r_1} \cdot c_{key,1}^a, h^{r_1} f^b \cdot c_{key,2}^a) = (g_p^{r_1 + a\tilde{z}/\tilde{e}}, h^{r_1 + a\tilde{z}/\tilde{e}} \cdot f^{ax_1+b});\\
\mathcal{S}(r_2):&\ \ (g_p^{r_2}, h^{r_2} \cdot f^{s'}).
\end{aligned}$$

We observe that it is possible for a malicious $\mathcal{P}_1^*$ to launch a low-order-element attack and tell these two ciphertexts apart. $\mathcal{P}_1^*$ chooses a $y \in \hat{\mathcal{G}}$ of low order (say, order 2) and produces an invalid ciphertext $(c_{key,1} = g_p^r, c_{key,2} = yh^r f^{x_1})$. (one can verify that there always exist $\tilde{e}$ and $\tilde{z}$ such that the above equations hold for this ciphertext.) Note also that $\mathcal{P}_1^*$ can carry out the *promise* Σ-protocol with success probability $\frac{1}{2}$. Once $\mathcal{P}_1^*$ receives the ciphertext from $\mathcal{P}_2$ as above, it can compute $y^a f^{ax_1+b}$ using his secret key and then obtain $a \bmod 2$. (Note that $a = 0 \bmod 2$ if and only if one can solve $y^a f^{ax_1+b}$ and obtain $ax_1 + b$ since the DL problem is tractable in group $\mathcal{F}$.) But if the ciphertext is computed by the simulator $\mathcal{S}$, then $\mathcal{P}_1^*$ always obtains $a = 0 \bmod 2$.

We have $\mathcal{P}_2$ randomize a in computing the ciphertext of $ax_1 + b$ to get around this issue. That is, $\mathcal{P}_2$ chooses a random t, raises $c_{key,1}$ and $c_{key,2}$ to the power $a+t$, and then computes a ciphertext $(c_{key,1}^{a+t}, f^b \cdot c_{key,2}^{a+t})$ (Note that, by introducing randomness t, we can drop $g_p^{r_1}$ and h^{r_1} here). For an honest $\mathcal{P}_1$ to decrypt and obtain $ax_1 + b$, $\mathcal{P}_2$ sends back this ciphertext along with $t \bmod p$

[4] In our construction of the two-party protocol, $\mathcal{P}_2$ does homomorphic operations only on $(c_{key,1}, c_{key,2})$ (and not on $(C_{key,1}, C_{key,2})$) and sends back the resulting CL ciphertext.

It appears that revealing information about t would make this randomization useless. However, as we will prove in Section 4, as long as the random string t is sufficiently long and both $p \nmid \mathrm{ord}(g_p)$, $p \nmid \mathrm{ord}(h)$ (we denote by $\mathrm{ord}(y)$ the order of y), the above randomization actually works, i.e., the following two distributions are indistinguishable:

$$\mathcal{P}_2(t_1) : (c_{key,1}^{a+t_1}, f^b \cdot c_{key,2}^{a+t_1}, t_1 \bmod p), \text{ and } \mathcal{S}(t_2) : (c_{key,1}^{t_2}, f^{s'} \cdot c_{key,2}^{t_2}, t_2 \bmod p).$$

To make sure $p \nmid \mathrm{ord}(g_p)$ and $p \nmid \mathrm{ord}(h)$, one can have $\mathcal{P}_1$ generate a CL public key of the form $(g_0, g_p = g_0^p, h_0, h = h_0^p)$. $\mathcal{P}_2$ checks if $g_p = g_0^p$ and $h = h_0^p$ hold, and if so, it takes (g_p, h) as the public key of a standard CL encryption scheme.

Promise Σ-protocol for homomorphic operations. In the multi-party setting, in the signing phase one party needs to prove that it did the same homomorphic operations on given a linear encoding and a CL ciphertext. We also construct a *promise* Σ-protocol for proving such a statement. As before, though a *promise* Σ-proof does not guarantee the statement is true, but the *promise extractability* suffices to prove the security of our protocol.

1.3 Related Work

ECDSA based on oblivious transfer. Instead of using additively homomorphic encryption, Doerner et al. [DKLs18, DKLs19] constructed two-party and multi-party threshold ECDSA based on oblivious transfer. As a consequence, these schemes are fast in computational complexity but at the cost of increasing the bandwidth.

Concurrent work. Very recently, Yuen et al. [YCX21] optimize the underlying zero knowledge proof related to class-group encryption, and construct more efficient two-party and multi-party ECDSA protocols. However, their schemes still rely on the low order assumption and the strong root assumption and the security is proved in the generic group model.

2 Preliminaries

Notation. Let λ be the security parameter. A non-negative function $\mathsf{negl}(\lambda)$ is negligible if for every polynomial $p(\lambda)$, it holds that $\mathsf{negl}(\lambda) \leq 1/p(\lambda)$ for sufficiently large $\lambda \in \mathbb{N}$. Let $\mathsf{poly}(\lambda)$ be a polynomial of λ. PPT stands for probabilistic polynomial time. Denote $\mathrm{ord}(g)$ the order of the element g in a given group.

2.1 CL Encryption from HSM Assumption

Castagnos et al. [CCL+19] gave a specific hard subgroup membership assumption (HSM) [CLT18] which is defined in the context of a group with an easy Dlog subgroup. Their instantiation makes use of class groups of imaginary quadratic fields. We list the definitions and constructions below and refer to [CCL+19] for more details.

Definition 1 ([CCL+19]). *Let* $\mathsf{GenGroup}$ *be a pair of algorithms* $(\mathsf{Gen}, \mathsf{Solve})$. *The* Gen *algorithm taking as inputs the security parameter* λ *and a prime* p *outputs a tuple* $\mathsf{param} = (\tilde{s}, g, f, g_p, \hat{\mathcal{G}}, \mathcal{G}, \mathcal{F}, \mathcal{G}^p)$. *The set* $(\hat{\mathcal{G}}, \cdot)$ *is a finite abelian group of order* $p \cdot \hat{s}$ *where the bitsize of the unknown* $\hat{s}$ *with an upper bound* $\tilde{s}$ *is a function of* λ *and* $\gcd(p, \hat{s}) = 1$. *It is also required that one can efficiently recognise valid encodings of elements in* $\hat{\mathcal{G}}$. *The set* $(\mathcal{F}, \cdot)$ *is the unique cyclic subgroup of* $\hat{\mathcal{G}}$ *of order* p. *The set* $(\mathcal{G}, \cdot)$ *is a cyclic subgroup of* $\hat{\mathcal{G}}$ *of order* $p \cdot s$ *where* s *divides* $\hat{s}$. *By construction* $\mathcal{F} \subset \mathcal{G}$, *and, denoting* $\mathcal{G}^p := \{x^p, x \in \mathcal{G}\}$ *the subgroup of order* s *of* $\mathcal{G}$, *it holds that* $\mathcal{G} = \mathcal{G}^p \times \mathcal{F}$. *The elements* f, g_p *and* $g = f \cdot g_p$ *are respective generators of* $\mathcal{F}$, $\mathcal{G}^p$ *and* $\mathcal{G}$. *Let* $\mathcal{D}$ *(resp.* $\mathcal{D}_p$*) be a distribution over the integers such that the distribution* $\{g^x | x \leftarrow \mathcal{D}\}$ *(resp.* $\{g_p^x | x \leftarrow \mathcal{D}_p\}$*) is at distance less than* $2^{-\lambda}$ *from the uniform distribution in* $\mathcal{G}$ *(resp. in* $\mathcal{G}^p$*). The* Solve *algorithm is a deterministic polynomial time algorithm that solves the discrete logarithm problem in* F. *We suppose moreover that:*

(1) The Dlog problem is easy in $\mathcal{F}$:

$$\Pr\left[\begin{array}{c} \mathsf{param} \leftarrow \mathsf{Gen}(1^\lambda, p); \\ x \leftarrow \mathbb{Z}_p, y = f^x, \\ x^\star \leftarrow \mathsf{Solve}(p, \mathsf{param}, y) \end{array} : x = x^\star \right] = 1.$$

(2) The HSM problem is hard even with access to the Solve *algorithm:*

$$\Pr\left[\begin{array}{c} \mathsf{param} \leftarrow \mathsf{Gen}(1^\lambda, p); \\ x \leftarrow \mathcal{D}, x' \leftarrow \mathcal{D}_p; \\ z_0 = g^x, z_1 = g_p^{x'}; \\ b \leftarrow \{0,1\}; \\ b^\star \leftarrow \mathcal{A}(p, \mathsf{param}, z_b, \mathsf{Solve}(\cdot)) \end{array} : b = b^\star \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

for arbitrary PPT *adversary* $\mathcal{A}$.

In practice, we will use for $\mathcal{D}_p$ the uniform distribution on $\{0, \ldots, S\}$ where $S = 2^{\lambda-2} \cdot \tilde{s}$. Following the notations of [CCL+19], we now describe a standard IND-CPA secure encryption scheme (called CL encryption) under the HSM assumption.

Definition 2. *The additively homomorphic public-key encryption scheme* CL *consists of the following algorithms.*

- $\mathsf{CL.KGen}(1^\lambda, p)$: *Let* $(\tilde{s}, g, f, g_p, \hat{\mathcal{G}}, \mathcal{G}, \mathcal{F}, \mathcal{G}^p) \leftarrow \mathsf{Gen}(1^\lambda, p)$. *Choose* $x \leftarrow \mathcal{D}_p$ *and compute* $h = g_p^x$. *Set* $\mathsf{pk} = (\tilde{s}, p, g_p, f, h)$ *and* $\mathsf{sk} = x$.
- $\mathsf{CL.Enc}_{\mathsf{pk}}(m)$: *Pick* $r \leftarrow \mathcal{D}_p$, *and output* $c = (g_p^r, h^r f^m)$.
- $\mathsf{CL.Dec}_{\mathsf{sk}}(c)$: *Parse* $c = (c_1, c_2)$, *and output* $m \leftarrow \mathsf{Solve}(c_2 / c_1^x)$.

As stated in [CCL+19], we also use the double encoding assumption to ensure the security of the presented two-party ECDSA in the case that the party $\mathcal{P}_2$ is corrupted. The intuition behinds this assumption is that given a one way function evaluated in $x \in \mathbb{Z}_p$ (in our protocol this is the elliptic curve point $Q := xG$) – no polynomial time adversary can produce two invalid CL encryptions of x.

Definition 3 (Double Encoding Assumption [CCL+19]). *The double encoding (DE) problem is δ_{DE}-hard for the one way function $\mathsf{exp}_{\mathbb{G}} : x \mapsto xG$ if for any PPT $\mathcal{A}$, it holds that:*

$$\Pr\left[\begin{array}{c} \mathsf{pp}_{\mathbb{G}} := (\mathbb{G}, G, p) \\ \mathsf{pp}_{\mathcal{G}} := (\tilde{s}, f, g_p, \mathcal{G}, \mathcal{F}, \mathcal{G}^p) \leftarrow \mathsf{Gen}(1^\lambda, p) \\ x \leftarrow \mathbb{Z}_p, Q = xG \\ (\mathsf{pk}, (u_1, u_1^{\mathsf{sk}} f^x), (u_2, u_2^{\mathsf{sk}} f^x) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathbb{G}}, \mathsf{pp}_{\mathcal{G}}, Q)) \end{array} : \begin{array}{c} u_1, u_2 \in \mathcal{G} \backslash \mathcal{G}^p \\ u_2 \cdot u_1^{-1} \in \mathcal{G} \backslash \mathcal{G}^p \\ \textit{and } \mathsf{pk} = g_p^{\mathsf{sk}} \end{array}\right] \leq \delta_{\mathrm{DE}},$$

where $\mathbb{G}$ is a group of points of an elliptic curve with a generator G of prime order p.

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

2.2 Σ-protocol

Denote $\mathcal{L}$ an NP language and $\mathcal{R}$ the associated binary relation. We say an instance x lies in $\mathcal{L}$ if and only if there exists a witness w s.t. $(x, w) \in \mathcal{R}$. Consider two-party protocols with the following pattern: The prover P taking input (x, w) computes a commitment a and hands it to V. The verifier V taking input x samples a random challenge e from a given challenge space and sends it to P. Then P responses z to V. Depending on the transcript (a, e, z), the verifier chooses to accept or reject it.

Definition 4 (Σ-protocol). *A 3-round protocol with the above form is called a Σ-protocol for an NP language $\mathcal{L}$ with an efficiently recognizable relation $\mathcal{R}$ iff. it satisfies the following properties:*

- *Completeness. If P and V behave honestly on input x and private input w to P where $(x, w) \in \mathcal{R}$, then V always accepts.*
- *Special soundness. There exists a PPT algorithm Ext which, given any instance $x \in L$ and two accepting transcripts (a, e, z) and (a, e', z') with $e \neq e'$, computes a witness w s.t. $(x, w) \in \mathcal{R}$.*
- *Special honest verifier zero knowledge (HVZK). There exists a PPT algorithm $\mathcal{S}$ which, taking $x \in \mathcal{L}$ and a challenge e as inputs, outputs (a, z) such that the tuple (a, e, z) is indistinguishable from an accepting transcript generated by a real protocol run between the honest $\mathsf{P}(x, w)$ and $\mathsf{V}(x)$.*

Σ-protocols can be transformed to non-interactive zero knowledge (NIZK) arguments via the Fiat-Shamir heuristic [FS87] and achieve zero knowledge in the random oracle model.

2.3 Threshold ECDSA and Its Security

Let ECDSA run on the elliptic curve group $\mathbb{G}$ of prime-order p with base point G. For a threshold t and a number of parties $n \geq t$, a (t, n)-threshold ECDSA consists of the following two interactive protocols:

$\mathsf{IKeyGen}$: The interactive key generation protocol, which takes the public parameter $(\mathbb{G}, G, p)$ as input. Each party $\mathcal{P}_i$ in the end receives the public key Q and its secret key x_i. The values $x_1, \ldots, x_n$ constitute a (t, n)-threshold secret sharing of the secret signing key x.

ISign: The interactive signing protocol, which take a message m as common input as well as a private input x_i from each party. It outputs an valid signature (r, s) of m or abort the execution.

The verification algorithm Verify is the same as that of the standard ECDSA.

Simulation-based Security and Ideal Functionalities. In this paper, we prove the security of two-party ECDSA according to the standard simulation paradigm with the ideal/real model, in the presence of static adversaries that choose which parties are corrupted before the protocol begins. The ideal/real simulation paradigm is to imagine what properties one would have in an ideal world, then a real world (constructed) protocol is said to be secure if it provides similar properties. Specifically, when proving that a constructed protocol Π achieves the simulation-based security, we always define an ideal functionality $\mathcal{F}$ executed by a trusted party to capture all the properties that need to be met. Then, we construct a simulator $\mathcal{S}$ (essentially plays the role of honest parties) that interacts with the trusted party computing $\mathcal{F}$, invokes the PPT adversary $\mathcal{A}$ internally, and simulates an execution of the real protocol. If $\mathcal{A}$ has negligible advantage to distinguish a real execution with honest parties from the simulation, then Π is considered secure.

We first describe the ECDSA ideal functionality between parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ as follows. Note that when considering two-party ECDSA, we only need to set $n = 2$.

The ECDSA Functionality $\mathcal{F}_{\mathrm{ECDSA}}$

- Upon receiving $\mathsf{KeyGen}(\mathbb{G}, G, p)$ from all parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, where $\mathbb{G}$ is an Elliptic-curve group of order p with generator G, then:
  1. Generate a pair of ECDSA keys (x, Q), where $x \leftarrow \mathbb{Z}_p^*$ is the secret signing key, and $Q = x \cdot G$ is the verification key.
  2. Send Q to all parties.
  3. Ignore future calls to KeyGen.
- Upon receiving $\mathsf{Sign}(sid, m)$ from $\mathcal{P}_1, \ldots, \mathcal{P}_n$, if KeyGen was already called and sid has not been stored, then:
  1. Compute an ECDSA signature (r, s) on m.
  2. Send (r, s) to all parties, and store (sid, m).

As in [Lin17, LN18, CCL+19], we prove the security of our protocol in a hybrid model using the ideal zero knowledge functionality $\mathcal{F}_{\mathrm{zk}}$, and the ideal commit-and-prove functionality $\mathcal{F}_{\text{com-zk}}$.

We now describe the ideal commitment functionality $\mathcal{F}_{\mathrm{com}}$.

The Commitment Functionality $\mathcal{F}_{\text{com}}$

- Upon receiving $(\mathsf{commit}, sid, x)$ from party $\mathcal{P}_i$ for $i \in [n]$, if sid has already been stored then ignore the message. Otherwise, store (sid, i, x) and send $(\mathsf{receipt}, sid, i)$ to all other parties $\mathcal{P}_j$ for all $j \in [n] \backslash \{i\}$.
- Upon receiving $(\mathsf{decommit}, sid, i)$ from party $\mathcal{P}_i$, if (sid, i, x) has been stored, then send $(\mathsf{decommit}, sid, i, x)$ to all other parties $\mathcal{P}_j$ for all $j \in [n] \backslash \{i\}$.

The ideal zero knowledge functionality, denoted $\mathcal{F}_{\text{zk}}$, is defined for a relation $\mathcal{R}$ by $(\emptyset, (x, \mathcal{R}(x, w))) \leftarrow \mathcal{F}_{\text{zk}}((x, w), \emptyset)$, where $\emptyset$ denotes the empty string, and $\mathcal{R}(x, w) = 1$ iff. $(x, w) \in \mathcal{R}$.
On HVZK in practice. We note that, in all previously known works in this line, the zero knowledge functionalities are realized by Σ-protocols or its NIZK version by Fiat-Shamir transformation, which achieve only honest verifier zero knowledge or zero knowledge in the random oracle model.

The Zero Knowledge Functionality $\mathcal{F}_{\text{zk}}^{\mathcal{R}}$ for Relation $\mathcal{R}$

Upon receiving $(\mathsf{prove}, sid, i, x, w)$ from party $\mathcal{P}_i$ for $i \in [n]$, if sid has already been stored then ignore the message. Otherwise, store sid and send $(\mathsf{proof}, sid, i, x, \mathcal{R}(x, w))$ to all other parties $\mathcal{P}_j$ for all $j \in [n] \backslash \{i\}$.

We also use an ideal functionality $\mathcal{F}_{\text{com-zk}}^{\mathcal{R}}$ to commit to NIZK proofs of knowledge for a relation $\mathcal{R}$. This can be achieved by having the prover commit to a NIZK proof of knowledge using the ideal commitment functionality $\mathcal{F}_{\text{com}}$.

The Committed NIZK Functionality $\mathcal{F}_{\text{com-zk}}^{\mathcal{R}}$ for Relation $\mathcal{R}$

- Upon receiving $(\mathsf{com\text{-}prove}, sid, x, w)$ from party $\mathcal{P}_i$ for $i \in [n]$, if sid has already been stored then ignore the message. Otherwise, store (sid, i, x) and send $(\mathsf{proof\text{-}receipt}, sid, i)$ to all other parties $\mathcal{P}_j$ for all $j \in [n] \backslash \{i\}$.
- Upon receiving $(\mathsf{decom\text{-}proof}, sid, i)$ from party $\mathcal{P}_i$, if (sid, i, x) has been stored, then send $(\mathsf{decom\text{-}proof}, sid, i, x, \mathcal{R}(x, w))$ to all other parties $\mathcal{P}_j$ for all $j \in [n] \backslash \{i\}$.

Game-based Security. Following [GJKR96, CCL+20], our construction of multi-party ECDSA is secure under a game-based definition: threshold unforgeability under chosen message attacks described as follows.

Definition 5 (Threshold Signature Unforgeability [GJKR96]). *A* (t, n)*-threshold signature scheme* $(\mathsf{IKeyGen}, \mathsf{ISign}, \mathsf{Verify})$ *is said to be unforgeable, if* *for any* PPT *adversary* $\mathcal{A}$ *who corrupts at most* t *parties, given the view of the protocols* $\mathsf{IKeyGen}$ *and* ISign *on input messages* $m_1, \ldots, m_k$ *of its adaptive choice as well as signatures on those messages, the probability that* $\mathcal{A}$ *can produce a signature on any new message* m *(*$m \notin \{m_1, \ldots, m_k\}$*) is negligible.*

3 Promise Σ-protocols

For our purpose, we *ideally* want an additively homomorphic encryption scheme over large message space and an efficient Σ-protocol to prove the validity of certain statements about ciphertexts. Unfortunately, as mentioned before, all currently known constructions are far from satisfactory: The $\mathsf{ElGamal}$ encryption scheme admits an efficient Σ-protocol but only supports a very small message space, while the CL encryption scheme supports large message space but does not admit an efficient Σ-protocol.

We obtain the best of both worlds using the following approach. Consider the following two *keyed* linear-homomorphic encoding schemes (we stress that these secret keys do *not necessarily* enable one to decode a codeword efficiently)[5]

- $(\mathsf{DL.Gen}, \mathsf{DL.Code})$ over elliptic curve group of prime order: $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{DL.Gen}(1^\lambda)$, $cw_0 \leftarrow \mathsf{DL.Code}_{\mathsf{pk}_0}(m)$;
- $(\mathsf{CL.Gen}, \mathsf{CL.Code})$ over class group of unknown order: $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{CL.Gen}(1^\lambda)$, $cw_1 \leftarrow \mathsf{CL.Code}_{\mathsf{pk}_1}(m)$.

We encode a message m twice independently, and then compose in parallel the efficient Σ-protocol for $\mathsf{DL.Code}_{\mathsf{pk}_0}$ with the efficient *insecure* Σ-protocol for $\mathsf{CL.Code}_{\mathsf{pk}_1}$ to prove that "$\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$ and $\mathsf{CL.Code}_{\mathsf{pk}_1}(m)$ encode the same message m", i.e., a statement in the following language:

$$\mathcal{L} = \{(\mathsf{pk}_0, \mathsf{pk}_1, cw_0, cw_1) | \exists m \quad cw_0 = \mathsf{DL.Code}_{\mathsf{pk}_0}(m) \text{ and } cw_1 = \mathsf{CL.Code}_{\mathsf{pk}_1}(m)\}.$$

We observe that, though the composed Σ-protocol does not enjoy the special soundness, it provides some interesting security guarantees that are sufficient for our applications.

We call it *promise* Σ-protocol. Roughly, a *promise* Σ-protocol for the above statement weakens the special soundness property and promises only that one can extract the message m encoded into $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$ by rewinding a successful prover, *or*, extract certain information $\rho(m)$ (for some efficiently computable function $\rho(\cdot)$) about m using both secret keys sk_0 and sk_1 *without access to the prover.*

Definition 6 (Promise Σ-protocol). *Let* λ *be the security parameter and* $\rho(\cdot)$ *be an efficiently computable function. Let encoding schemes* $(\mathsf{DL.Gen}, \mathsf{DL.Code})$*,*

[5] These encoding schemes $\mathsf{DL.Code}$ and $\mathsf{CL.Code}$ may vary with applications, and may be randomized.

$(\mathsf{CL.Gen}, \mathsf{CL.Code})$ *and language* $\mathcal{L}$ *be as above. A promise* Σ*-protocol* (P, V) *for* $\mathcal{L}$ *with respect to* $\rho(\cdot)$ *is a 3-round public coin protocol (with transcript being of the form* (a, e, z)*) that satisfies the following conditions:*

- *Completeness and special honest verifier zero knowledge defined in the same way as* Σ*-protocol.*
- *Promise extractability. For any inverse polynomial* $\epsilon(\lambda)$*, any* PPT P^* *that makes the verifier accept with probability* $\epsilon(\lambda)$*, there is a* PPT *extractor* Ext *such that the following conditions hold.*
  1. *Extraction by rewinding (Special-soundness) for* $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$*. With oracle access to* P^**,* $\mathsf{Ext}^{\mathsf{P}^*}(cw_0, cw_1)$ *extracts* m *of* $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$ *with probability negligibly close to* 1*.*
  2. *Straight-line extraction for* $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$ *using secret keys. If both key pairs* $(\mathsf{pk}_0, \mathsf{sk}_0)$ *and* $(\mathsf{pk}_1, \mathsf{sk}_1)$ *are honestly generated, then given* $(\mathsf{sk}_0, \mathsf{sk}_1)$ *as input the extractor* $\mathsf{Ext}(\mathsf{sk}_0, \mathsf{sk}_1, cw_0, cw_1)$ *(without access to* P^**) extracts* $\rho(m)$ *with probability negligibly close to* 1*, where* m *is message encoded into* $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$*.*

Remark 1. Note that the promise extractability is a weaker notion than the special soundness: An accepting *promise* Σ-proof does not even guarantee the second codeword cw_1 is valid.

However, the second condition of promise extractability implies that if a prover can make the verifier accept the statement with high probability (hence there exist at least two accepting transcripts with the same first message a but different challenges $e \neq e'$), then the one who holds the honestly generated secret keys could extract $\rho(m)$.

Remark 2. As we will see, in our applications the first secret key sk_0 would not allow us to efficiently recover the message encoded into cw_0, but sk_1 would *if* cw_1 *is valid*. As explained above, since cw_1 may be invalid, our straight-line extractor will depart from the normal "decryption" procedure associated with $\mathsf{CL.Code}$. Although we cannot use the first secret key sk_0 to decode cw_0, but it is useful for the straight-line extractor to check if the message extracted out is the right message (see the construction in Section 3.1).

Remark 3. One may use different (rewinding or straight-line) extractor in simulation strategies. Suppose that a malicious party sends out two codewords cw_0 and cw_1, along with a *promise* Σ-proof, to an honest party in a step of a protocol. In case cw_0 and cw_1 are computed under the public keys generated by the malicious party, then the *promise* Σ-proof guarantees that the malicious party "knows" the message encoded into cw_0 , which can be extracted using rewinding by the simulator (playing the role of the honest party) in the security proof. Otherwise, if the corresponding public keys are generated by the honest party, then the *promise* Σ-proof promises that the codeword cw_1 is "decodable", and the simulator can use straight-line extractor (with the corresponding secret keys which are actually generated by itself in a simulation) to extract certain useful information about the message encoded into cw_0.

Theoretically, we can achieve Σ-protocol for such a statement with fully special soundness. Our *promise* Σ-protocol is motivated purely out of efficiency consideration. As we shall see, the weak notion of Σ-protocol is sufficient for our application, and it achieves much better performance than the known constructions of Σ-protocols.

Promise NIZK in the random oracle model. In practice, one can apply the Fiat-Shamir transform to our *promise* Σ-protocol to obtain a non-interactive protocol. One can verify that the resulting protocol also enjoys the *promise* extractability in the random oracle model using the forking technique from [PS96], as well as completeness and zero knowledge property.

Definition 7 (Promise NIZK). *Let λ be the security parameter and $\rho(\cdot)$ be an efficiently computable function. Let $(\mathsf{DL.Gen}, \mathsf{DL.Code})$, $(\mathsf{CL.Gen}, \mathsf{CL.Code})$ and language $\mathcal{L}$ be defined as above. A promise NIZK proof (P, V) for $\mathcal{L}$ in the random oracle model with respect to $\rho(\cdot)$ satisfies the following conditions:*

- *Completeness and zero knowledge defined in the same way as a NIZK proof.*
- *Promise extractability. For any inverse polynomial $\epsilon(\lambda)$, any PPT P^* that generates a proof with an accepted probability $\epsilon(\lambda)$, there is a PPT extractor Ext such that the following conditions hold.*
  1. *Extraction by rewinding for $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$. With oracle access to P^* and the programmability of the random oracle H, $\mathsf{Ext}^{\mathsf{P}^*,\mathsf{H}}(cw_0, cw_1)$ extracts m of $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$ with probability negligibly close to 1.*
  2. *Straight-line extraction for $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$ using secret keys. If both key pairs $(\mathsf{pk}_0, \mathsf{sk}_0)$ and $(\mathsf{pk}_1, \mathsf{sk}_1)$ are honestly generated, then given $(\mathsf{sk}_0, \mathsf{sk}_1)$ as input the extractor $\mathsf{Ext}(\mathsf{sk}_0, \mathsf{sk}_1, cw_0, cw_1)$ (without access to P^*) extracts $\rho(m)$ with probability negligibly close to 1, where m is message encoded into $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$.*

3.1 Promise Σ-protocol for Encryptions

In this section we first consider the following encoding schemes:

- $\mathsf{DL.Code}_G : m \to m \cdot G$, where G is a random generator of an elliptic curve group $\mathbb{G}$ of prime order p, serving the public key pk_0 (and there is no secret key).
- $\mathsf{CL.Code}_{\mathsf{pk}} : m \to \mathsf{CL.Enc}_{\mathsf{pk}}(m; r)$, i.e., $\mathsf{CL.Code}_{\mathsf{pk}}$ is the CL encryption algorithm $\mathsf{CL.Enc}$ (see Section 2), where pk is generated by its corresponding key generation algorithm $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{CL.KGen}(1^\lambda)$.

and the following language:

$$\mathcal{L}_{\mathrm{DLCL}} = \{(G, \mathsf{pk}, Q, c) | \exists m \in \mathbb{Z}_p, r \in [0, S], s.t. \quad Q = m \cdot G \text{ and } c = \mathsf{CL.Enc}_{\mathsf{pk}}(m; r)\}.$$

We construct a *promise* Σ-protocol by composing two "Σ-protocols" for discrete logarithm and CL ciphertext in parallel. Though the latter "Σ-protocol" is insecure as mentioned before, we can still show the composed protocol is a *promise* Σ-protocol. In the following, we fix U such that $(p-1)S/U$ is negligible.

Protocol Σ^1_{prom} for proving the consistency of messages

Common input: $G,\mathsf{pk}=(\tilde{s}, p, g_p, f, h), Q, c=(c_1, c_2)$.
P's Private input: $m \in \mathbb{Z}_p$ and $r \in [0, S]$ s.t. $Q = m \cdot G$, $c_1 = g_p^r$ and $c_2 = h^r f^m$.

1. P chooses $s_m \leftarrow \mathbb{Z}_p$ and $s_r \leftarrow [0, U)$ at random, computes $A = s_m G$, $a_1 = g_p^{s_r}$, $a_2 = h^{s_r} f^{s_m}$. P sends A, a_1, a_2 to verifier V.
2. V chooses and sends a random $e \leftarrow \mathbb{Z}_p$ to P.
3. P computes $z_m = s_m + em \bmod p$ and $z_r = s_r + er$, then sends z_m, z_r to V.
4. V outputs 1 iff. $z_r \in [0, U + (p-1)S)$, $z_m G = A + eQ$, $g_p^{z_r} = a_1 c_1^e$ and $h^{z_r} f^{z_m} = a_2 c_2^e$.

Theorem 1. *If $(p-1)S/U$ is negligible, then protocol Σ^1_{prom} is a promise Σ-protocol with respect to the identity function $\rho : m \to m$.*

The construction and security proof of protocol Σ^1_{prom} are actually subsumed by the following *promise* Σ-protocol in which the first encoding scheme is replaced with the $\mathsf{ElGamal}$ encryption scheme[6].

With replacement of the first encoding scheme in the above with the $\mathsf{ElGamal}$ key generation and encryption algorithm ($\mathsf{EG.KGen}, \mathsf{EG.Enc}$) (Note that here the secret key does not allow one to decrypt ciphertexts since the plaintext is too long), we present a *promise* Σ-protocol for the following language:

$$\begin{aligned}\mathcal{L}_{\text{EGCL}} = \{(\mathsf{pk}_0, \mathsf{pk}_1, C, c) | \exists m \in \mathbb{Z}_p, r_1 \in \mathbb{Z}_p, \text{ and } r_2 \in [0, S], s.t. \\ C = \mathsf{EG.Enc}_{\mathsf{pk}_0}(m; r_1) \text{ and } c = \mathsf{CL.Enc}_{\mathsf{pk}_1}(m; r_2)\}.\end{aligned}$$

Protocol Σ^2_{prom} for proving the equality of plaintexts

Common input: $\mathsf{pk}_0 = (G, P), \mathsf{pk}_1 = (\tilde{s}, p, g_p, f, h), C = (C_1, C_2), c = (c_1, c_2)$.
P's Private input: $m \in \mathbb{Z}_p, r_1 \in \mathbb{Z}_p$ and $r_2 \in [0, S]$ s.t. $C_1 = r_1 G$, $C_2 = r_1 P + mG, c_1 = g_p^{r_2}$ and $c_2 = h^{r_2} f^m$.

[6] It is easy to verify that the straight-line extractor for protocol Σ^1_{prom}, similar to the one for protocol Σ^2_{prom}, does not require the knowledge of sk_0 (which does not exist in protocol Σ^1_{prom}).

1. P chooses $s_1 \leftarrow \mathbb{Z}_p$, $s_2 \leftarrow [0, U)$ and $s_m \leftarrow \mathbb{Z}_p$ at random, and computes $A_1 = s_1 G$, $A_2 = s_1 P + s_m G$, $a_1 = g_p^{s_2}$, $a_2 = h^{s_2} f^{s_m}$. P sends A_1, A_2, a_1, a_2 to verifier V.
2. V chooses and sends a random $e \leftarrow \mathbb{Z}_p$ to P.
3. P computes $z_1 = s_1 + er_1 \bmod p$, $z_2 = s_2 + er_2$ and $z_m = s_m + em \bmod p$, and sends z_1, z_2, z_m to V.
4. V outputs 1 if $z_2 \in [0, U + (p-1)S)$,$z_1 G = A_1 + eC_1$,$z_1 P + z_m G = A_2 + eC_2$,$g_p^{z_2} = a_1 c_1^e$ and $h^{z_2} f^{z_m} = a_2 c_2^e$.

Theorem 2. *If $(p-1)S/U$ is negligible, then protocol Σ^2_{prom} is a promise Σ-protocol with respect to the function $\rho : m \to m$.*

Proof. Completeness is obvious. Special honest verifier zero knowledge property follows from the same arguments as in [CCL+19, GPS06], which we omit here.

We now prove the *promise extractability.* Suppose there is a prover P^* that can make the honest verifier accept with probability $\epsilon(\lambda)$, we can fix a good random tape r_p^* for P^* and define $\mathcal{G}'_{r_p^*} := \{e \in \mathbb{Z}_p : \mathsf{P}^*(r_p^*) \text{ answers } e \text{ correctly}\}$, which is of size greater than $p\epsilon(\lambda)$.

By applying standard rewinding strategy to the prover $\mathsf{P}^*(r_p^*)$, we have an efficient extractor $\mathsf{Ext}^{\mathsf{P}^*(r_p^*)}$ that computes two accepting transcripts $(A_1, A_2, a_1, a_2, e, z_1, z_2, z_m)$ and $(A_1, A_2, a_1, a_2, e', z_1', z_2', z_m')$ with $e \neq e'$. From (A_1, A_2, e, z_1, z_m) and $(A_1, A_2, e', z_1', z_m')$ one can compute the plaintext $m = (z_m - z_m')/(e - e') \bmod p$ of the $\mathsf{ElGamal}$ ciphertext (C_1, C_2) (since the Σ-protocol for an $\mathsf{ElGamal}$ ciphertext satisfies special soundness).

It remains to prove the second condition of *promise extractability* holds. In this case we assume both the public keys pk_0 and pk_1 are honestly generated. Let sk_0 and sk_1 are the corresponding secret keys. From the above two accepting transcripts, we also have that $g_p^{z_2} = a_1 c_1^e, h^{z_2} f^{z_m} = a_2 c_2^e$ and $g_p^{z_2'} = a_1 c_1^{e'}, h^{z_2'} f^{z_m'} = a_2 c_2^{e'}$. Set $\tilde{z}_2 = z_2 - z_2', \tilde{z}_m = z_m - z_m', \tilde{e} = e - e'$. By $\tilde{z}_m = m\tilde{e} \bmod p$ as above, we conclude

$$g_p^{\tilde{z}_2} = c_1^{\tilde{e}}, \text{ and } h^{\tilde{z}_2} f^{m\tilde{e}} = c_2^{\tilde{e}}. \tag{1}$$

This implies that we can efficiently modify the second ciphertext (c_1, c_2) into a valid ciphertext $(c_1' = c_1^{\tilde{e}}, c_2' = c_2^{\tilde{e}})$ of the message $m\tilde{e}$. Furthermore, combining $z_m - z_m' = m\tilde{e} \bmod p$ with the first condition of *promise extractability*, we have

$$\tilde{e} \cdot (C_2 - \mathsf{sk}_0 \cdot C_1) = m\tilde{e} \cdot G \text{ and } (\frac{c_2}{c_1{}^{\mathsf{sk}_1}})^{\tilde{e}} = f^{m\tilde{e}}.$$

This gives rise to the following extractor Ext that can compute the plaintext m from the two secret keys and the two ciphertexts without access to the prover P^* with probability negligibly close to 1.

Extractor $\mathsf{Ext}(\mathsf{sk}_0, \mathsf{sk}_1, (C_1, C_2), (c_1, c_2))$:

1. Run the decryption algorithm $\mathsf{CL.Dec}_{\mathsf{sk}_1}$ on input $((c_1, c_2))$, if it outputs a plaintext m such that $C_2 - \mathsf{sk}_0 \cdot C_1 = mG$, then return m (In this case (c_1, c_2) is a valid ciphertext).
2. Pick two random $e, e' \in \mathbb{Z}_p$, compute $(\frac{c_2}{{c_1}^{\mathsf{sk}_1}})^{e-e'}$ and run $\tilde{z}_m \leftarrow \mathsf{CL.Solve}((\frac{c_2}{{c_1}^{\mathsf{sk}_1}})^{e-e'})$. If $e \neq e'$ and $(e - e') \cdot (C_2 - \mathsf{sk}_0 C_1) = \tilde{z}_m G$ holds for the $\mathsf{ElGamal}$ ciphertext (C_1, C_2), then compute m by solving $(e - e')m = \tilde{z}_m \bmod p$ and return m; otherwise, repeat this step.

Note that if $e \neq e'$ and $(e - e') \cdot (C_2 - \mathsf{sk}_0 \cdot C_1) = \tilde{z}_m \cdot G$, then the $\mathsf{ElGamal}$ ciphertext is valid and we can compute the unique plaintext m from $(e - e')m = \tilde{z}_m \bmod p$. Since the size of $\mathcal{G'}_{r_p^*}$ is greater than $p\epsilon(\lambda)$, a single step 2 of Ext will output the plaintext of (C_1, C_2) with probability at least $\epsilon^2(\lambda) - \frac{1}{2^\lambda}$, and hence it will succeed in expected time at most $\mathcal{O}\Big(\frac{1}{\epsilon(\lambda)^2}T\Big)$, where T is the running time of a single repetition of step 2. $\square$

3.2 Promise Σ-protocol for Homomorphic Operations

Suppose we have a tuple $(\mathsf{pk}_0, \mathsf{pk}_1, C = (C_1, C_2), c = (c_1, c_2))$ that has already been proven to be in $\mathcal{L}_{\text{EGCL}}$ via the *promise* Σ-protocol Σ^2_{prom} described in the previous section. We call such a pair (C, c) *semi-equal*.

Given such a $(\mathsf{pk}_0, \mathsf{pk}_1, C = (C_1, C_2), c = (c_1, c_2))$, we consider the following encoding schemes both derived from homomorphic operations:

- $\mathsf{DL.Code}_{\mathsf{pk}'_0 = (\mathsf{pk}_0, C_1, C_2)} : (a, b) \to (aC_1 + rG, aC_2 + bG + rP)$, where $\mathsf{pk}_0 = (G, P)$ is the public key of the $\mathsf{ElGamal}$ encryption, r is selected randomly from $\mathbb{Z}_p$. We let the secret key corresponding to pk'_0 be the secret key of the $\mathsf{ElGamal}$ encryption.
- $\mathsf{CL.Code}_{\mathsf{pk}'_0 = (\mathsf{pk}_1, c_1, c_2)} : (a, b) \to (c_1^a, c_2^a f^b)$, where pk_1 is the public key of the CL encryption. We let the secret key corresponding to pk'_1 be the secret key of the CL encryption.

We now present a *promise* Σ-protocol for the following language with respect the above two schemes:

$$\mathcal{L}_{\text{affine}} = \{((\mathsf{pk}_0 = (G,P), C_1, C_2), (\mathsf{pk}_1 = (\tilde{s}, p, g_p, f, h), c_1, c_2), (C'_1, C'_2), (c'_1, c'_2))| \\ \exists a \in [0, pS), b, r \in \mathbb{Z}_p, \text{s.t. } C'_1 = aC_1 + rG \wedge C'_2 = aC_2 + bG + rP \wedge c'_1 = c_1^a \wedge c'_2 = c_2^a f^b\}$$

Such a statement essentially says that the tuple $(C' = (C'_1, C'_2), c' = (c'_1, c'_2))$ is generated from $(C = (C_1, C_2), c = (c_1, c_2))$ by doing the *same* affine homomorphic operations.

The protocol proceeds as follows.

Protocol Σ^3_{prom} for correctness of homomorphic operations

Common input: $(((G,P),C_1,C_2),((\tilde{s},p,g_p,f,h),c_1,c_2),(C'_1,C'_2),(c'_1,c'_2))$.
P**'s Private input:** $a \in [0,pS), b, r \in \mathbb{Z}_p$.

1. P randomly chooses $s_a \in [0,pU), s_b, s_r \in \mathbb{Z}_p$, and computes $A_1 = s_aC_1 + s_rG, A_2 = s_aC_2 + s_bG + s_rP, a_1 = c_1^{s_a}, a_2 = c_2^{s_a} f^{s_b}$, then sends (A_1, A_2, a_1, a_2) to V.
2. V chooses randomly $e \in \mathbb{Z}_p$ and sends it to P.
3. P computes $z_a = s_a + ea$ in $\mathbb{Z}$, $z_b = s_b + eb \bmod p$ and $z_r = s_r + er \bmod p$, then sends (z_a, z_b) to V.
4. V first checks whether $z_a \in [0, p(U + (p-1)S))$, and accepts iff. the following conditions hold: $z_aC_1 + z_rG = A_1 + eC'_1, z_aC_2 + z_bG + z_rP = A_2 + eC'_2, c_1^{z_a} = a_1 {c'_1}^e, c_2^{z_a} f^{z_b} = a_2 {c'_2}^e$.

Theorem 3. *If $(p-1)S/U$ is negligible, and $((C_1,C_2),(c_1,c_2))$ is a* semi-equal *pair under the encoding schemes $\mathsf{EG.Enc}_{\mathsf{pk}_0}$ and $\mathsf{CL.Enc}_{\mathsf{pk}_1}$ respectively, then protocol Σ^3_{prom} is a promise Σ-protocol with respect to the function $\rho_m : (a,b) \to (am + b \bmod p)$, where m is such that $(C_1,C_2) = \mathsf{EG.Enc}_{\mathsf{pk}_0}(m)$.*

Proof. Again, here we omit the proofs of completeness and the HVZK property, and just prove the *promise extractability.*

Suppose an adversarial prover P^* convinces V with a non-negligible probability, we could obtain two accepting transcripts $(A_1, A_2, a_1, a_2, e, z_a, z_b, z_r)$ and $(A_1, A_2, a_1, a_2, e', z'_a, z'_b, z'_r)$ with $e \neq e' \bmod p$ using the similar proof strategy as the previous section. Subsequently following the special soundness of the Σ-protocol for $\mathsf{ElGamal}$ ciphertexts, one can compute the affine factors $(a \bmod p) = (z_a - z'_a)/(e - e') \bmod p$ and $b = (z_b - z'_b)/(e - e') \bmod p$, as well as the randomness $r = (z_r - z'_r)/(e - e') \bmod p$ such that $C'_1 = aC_1 + rG$ and $C'_2 = aC_2 + bG + rP$.

We now turn to the second property of *promise extractability.* Note that from the above two accepting transcripts it yields

$$c_1^{\Delta z_a} = {c'_1}^{\Delta e}, c_2^{\Delta z_a} f^{\Delta z_b} = {c'_2}^{\Delta e}, \tag{2}$$

where $\Delta z_a = z_a - z'_a, \Delta z_b = z_b - z'_b$ and $\Delta e = e - e'$.

From the fact that $(C = (C_1,C_2), c = (c_1,c_2))$ are semi-equal, it follows from the equality (1) that $c_1^{\tilde{e}} = g_p^{\tilde{z}}, c_2^{\tilde{e}} = h^{\tilde{z}} f^{m\tilde{e}}$ for some $\tilde{e}$. Combining these two equalities with the equality (2), we have

$$g_p^{\tilde{z}\Delta z_a} = {c'_1}^{\tilde{e}\Delta e}, h^{\tilde{z}\Delta z_b} f^{(am+b)\tilde{e}\Delta e} = {c'_2}^{\tilde{e}\Delta e}. \tag{3}$$

This essentially says that one can efficiently modify the second codeword (c'_1, c'_2) into a valid codeword $(c''_1 = {c_1}^{\tilde{e}\Delta e}, c''_2 = {c'_2}^{\tilde{e}\Delta e})$ of the message $(am + b)\tilde{e}\Delta e$.

We set $\hat{e} = \tilde{e}\Delta e$ and $\hat{z} = \tilde{z}\Delta z_a$. Let $(\mathsf{sk}_0, \mathsf{sk}_1)$ be the (honest generated) secret keys of the underlying $\mathsf{ElGamal}$ and CL encryption scheme. Thus, combining the first condition of promise extractability and the equality (3), we have

$$(am+b)G = C_2' - \mathsf{sk}_0 \cdot C_1', {c_1'}^{\hat{e}} = g_p^{\hat{z}}, {c_2'}^{\hat{e}} = h^{\hat{z}} f^{(am+b)\hat{e}},$$

which allow us to construct a straight-line extractor $\mathsf{Ext}(\mathsf{sk}_0, \mathsf{sk}_1, \cdot)$ to extract $\rho(m) = am + b$ in the same way as in Section 3.1. $\square$

We prove in Appendix A that the *promise* Σ-protocols described above are indeed *promise* NIZKs in the random oracle model after applying Fiat-Shamir transformation.

4 Simulating Homomorphic Operations on an Invalid Ciphertext

Recall that in the final stage of a two-party signing subprotocol of [Lin17, CCL+19], $\mathcal{P}_2$ holds (x_2, k_2, r) and computes $b = k_2^{-1}m'$ (where m' is the hash value of m), $a = k_2^{-1}rx_2$, and a ciphertext of $ax_1 + b$ by homomorphic operations on the ciphertext $(c_{key,1}, c_{key,2})$ of x_1, which it received in the key generation phase. $\mathcal{P}_2$ sends the resulting ciphertext of $ax_1 + b$ to $\mathcal{P}_1$, who decrypts the ciphertext and computes a signature $(r, k_1^{-1}(ax_1 + b))$.

In our settings, in the key generation phase $\mathcal{P}_1$ computes a pair $(Q_1, c_{key} = (c_{key,1}, c_{key,2}))$ which both encode x_1, and then runs an efficient *promise* Σ-protocol (with challenge of polynomial length) to prove the knowledge of x_1. This *promise* Σ-proof guarantees only that the $(c_{key,1}, c_{key,2})$ satisfies $c_{key,1}^{\tilde{e}} = g_p^{\tilde{z}}$ and $c_{key,2}^{\tilde{e}} = h^{\tilde{z}} f^{\tilde{e}x_1}$ for some $\tilde{e} \in [-p+1, p-1] \setminus \{0\}, \tilde{z} \in [-U-(p-1)S+1, U+(p-1)S-1] \setminus \{0\}$. As discussed in the introduction, if we have the same $\mathcal{P}_2$ as in [CCL+19], then the same simulator $\mathcal{S}$ would fail.

Instead, we have $\mathcal{P}_2$ choose a random t, raise $c_{key,1}$ and $c_{key,2}$ to the power $a+t$ (randomizing a), and then compute a ciphertext $(c_{key,1}^{a+t}, f^b \cdot c_{key,2}^{a+t})$. For an honest $\mathcal{P}_1$ to decrypt and obtain $ax_1 + b$, $\mathcal{P}_2$ sends back this ciphertext along with $t \bmod p$. Specifically, we consider the following $\mathcal{P}_2$ and $\mathcal{S}$ (think that a,b and $s' = ax_1 + b$ (along with the public keys) are "hardwired" into $\mathcal{P}_2$ and $\mathcal{S}$, respectively):

$$\mathcal{P}_2(t_1) : (c_{key,1}^{a+t_1}, f^b c_{key,2}^{a+t_1}, t_1 \bmod p), \text{ and } \mathcal{S}(t_2) : (c_{key,1}^{t_2}, f^{s'} c_{key,2}^{t_2}, t_2 \bmod p). \quad (4)$$

We now give a formal proof that, if the random string t is sufficiently long and $p \nmid \mathrm{ord}(g_p)$, $p \nmid \mathrm{ord}(h)$, then these two distributions above are statistically close. As mentioned, the last two conditions can be achieved by having $\mathcal{P}_1$ generate a public key of the CL encryption scheme of the form $(g_0, g_p = g_0^p, h_0, h = h_0^p)$. Recall the following notations and their properties:

- $\mathsf{param} := (\tilde{s}, g, f, g_p, \hat{\mathcal{G}}, \mathcal{G}, \mathcal{F}, \mathcal{G}^p)$ and S are the parameters of groups we work on, satisfying that 1) p is a prime, $|\hat{\mathcal{G}}| = p\hat{s}$ for some unknown $\hat{s} < \tilde{s}$ and $S = 2^{\lambda-2}\tilde{s}$; 2) $\gcd(p, \hat{s}) = 1$.

- $\mathsf{pk} := (\tilde{s}, p, g_p, f, h)$ is the public key such that $p \nmid \mathrm{ord}(g_p)$ and $p \nmid \mathrm{ord}(h)$. As discussed above, this property can be easily achieved even if the public key is maliciously generated.
- (x_1, a, b, s') satisfies $s' = ax_1 + b \bmod p$.
- $(c_{key,1}, c_{key,2})$ and h'. $(c_{key,1}, c_{key,2})$ is the ciphertext as above, and satisfies $c_{key,1}^{\tilde{e}} = g_p^{\tilde{z}}$ and $c_{key,2}^{\tilde{e}} = h^{\tilde{z}} f^{\tilde{e}x_1}$ for some $\tilde{e} \in [-p+1, p-1] \setminus \{0\}$, $\tilde{z} \in [-U-(p-1)S, U+(p-1)S] \setminus \{0\}$. h' is an arbitrary $\tilde{e}$-th root of $h^{\tilde{z}}$, which satisfies that $c_{key,2} = h' f^{x_1}$.

We define $\mathfrak{p} := (\mathsf{param}, \mathsf{pk}, x_1, a, b, s', c_{key,1}, c_{key,2}, h')$, and prove the following lemma.

Lemma 1. *Let $\mathfrak{p}$ be defined as above. Then the statistical distance between the two distributions $\{t_1 \leftarrow [0, pS) : \mathcal{P}_2(t_1)\}$ and $\{t_2 \leftarrow [0, pS) : \mathcal{S}(t_2)\}$ in (4) is exponentially small.*

Proof. From the facts that $p \nmid \mathrm{ord}(g_p)$, $p \nmid \tilde{e}$ and $c_{key,1}^{\tilde{e}} = g_p^{\tilde{z}}$, it follows $p \nmid \mathrm{ord}(g_p^{\tilde{z}})$ and $\gcd(p, \mathrm{ord}(g_p^{\tilde{z}}) = \mathrm{ord}(c_{key,1}^{\tilde{e}})) = 1$. Since

$$\mathrm{ord}(c_{key,1}^{\tilde{e}}) = \mathrm{ord}(c_{key,1}) / \gcd(\mathrm{ord}(c_{key,1}), \tilde{e}) \text{ and } p \nmid \tilde{e},$$

we have $\gcd(p, \mathrm{ord}(c_{key,1})) \le \gcd(p, \mathrm{ord}(c_{key,1}^{\tilde{e}})) \cdot \gcd(p, \gcd(\mathrm{ord}(c_{key,1}), \tilde{e})) = 1$, i.e., $p \nmid \mathrm{ord}(c_{key,1})$.

Similarly, one can deduce $p \nmid \mathrm{ord}(h')$ from the facts that $p \nmid \mathrm{ord}(h)$, $p \nmid \tilde{e}$ and $h'^{\tilde{e}} = h^{\tilde{z}}$. Observe also that $\mathrm{ord}(c_{key,1}) | p\hat{s}$ and $\mathrm{ord}(h') | p\hat{s}$, we have

$$\mathrm{ord}(c_{key,1}) | \hat{s} \text{ and } \mathrm{ord}(h') | \hat{s}. \tag{5}$$

We define the following deterministic algorithm $\mathfrak{f}$ with $\mathfrak{p}$ hardwired:

$$\mathfrak{f}_{\mathfrak{p}}(t') = (c_{key,1}^{a+t'}, h'^{a+t'} f^{ax_1 + t'x_1 + b}, t' \bmod p),$$

and observe that,

$$\begin{aligned} \mathcal{P}_2(t_1) &= (c_{key,1}^{a+t_1}, f^b \cdot c_{key,2}^{a+t_1}, t_1 \bmod p); \\ &= (c_{key,1}^{a+t_1 \bmod \hat{s}}, h'^{a+t_1 \bmod \hat{s}} \cdot f^{ax_1 + t_1 x_1 + b \bmod p}, t_1 \bmod p) = \mathfrak{f}_{\mathfrak{p}}(t_1) \end{aligned} \tag{6}$$

By defining $t_2^* := t_2 - p^{-1}pa \bmod p\hat{s}$, where p^{-1} satisfies that $p^{-1}p \equiv 1 \bmod \hat{s}$ (recall $\gcd(p, \hat{s}) = 1$), we have:

$$\begin{aligned} \mathcal{S}(t_2) &= (c_{key,1}^{t_2}, f^{s'} \cdot c_{key,2}^{t_2}, t_2 \bmod p) \\ &= (c_{key,1}^{t_2^* + p^{-1}pa \bmod \hat{s}}, h'^{t_2^* + p^{-1}pa \bmod \hat{s}} \cdot f^{ax_1 + t_2^* x_1 + b \bmod p}, t_2^* + p^{-1}pa \bmod p) \\ &= (c_{key,1}^{t_2^* + a \bmod \hat{s}}, h'^{t_2^* + a \bmod \hat{s}} \cdot f^{ax_1 + t_2^* x_1 + b \bmod p}, t_2^* \bmod p) = \mathfrak{f}_{\mathfrak{p}}(t_2^*) \end{aligned} \tag{7}$$

It is easy to verify that $\{t_1 \leftarrow [0, p\hat{s}) : \mathfrak{f}_{\mathfrak{p}}(t_1)\}$ is identical to $\{t_2 \leftarrow [0, p\hat{s}) : \mathfrak{f}_{\mathfrak{p}}(t_2^* = t_2 - p^{-1}pa \bmod p\hat{s})\}$, which implies that

$$\{t_1 \leftarrow [0, p\hat{s}) : \mathcal{P}_2(t_1)\} \equiv \{t_2 \leftarrow [0, p\hat{s}) : \mathcal{S}(t_2)\}. \tag{8}$$

Denote by D_1 the distribution $\{t_1 \leftarrow [0, pS) : t_1 \bmod p\hat{s}\}$ and by D_2 the distribution $\{t_1 \leftarrow [0, p\hat{s}) : t_1\}$. The statistical distance $\mathsf{SD}(D_1, D_2)$ between the two distributions D_1 and D_2 is

$$\mathsf{SD}(D_1, D_2) = \frac{1}{2} \sum_{t \in [0, p\hat{s})} |\Pr[t_1 \leftarrow D_1 : t_1 = t] - \Pr[t_1 \leftarrow D_2 : t_1 = t]|.$$

Suppose that $\tilde{s} = k\hat{s}$ for some $k > 1$. We have $pS = 2^{\lambda-2}p\tilde{s} = k2^{\lambda-2}p\hat{s}$, and therefore

$$\Pr_{t \in [0,p\hat{s})}[t_1 \leftarrow D_1 : t_1 = t] = \frac{\lfloor k2^{\lambda-2} \rfloor}{k2^{\lambda-2}p\hat{s}} \text{ or } \frac{\lfloor k2^{\lambda-2} \rfloor + 1}{k2^{\lambda-2}p\hat{s}} \in [\frac{1}{p\hat{s}} - \frac{1}{k2^{\lambda-2}p\hat{s}}, \frac{1}{p\hat{s}} + \frac{1}{k2^{\lambda-2}p\hat{s}}].$$

Thus, we conclude

$$\mathsf{SD}(D_1, D_2) = \frac{1}{2} \sum_{t \in [0, p\hat{s})} |\Pr[t_1 \leftarrow D_1 : t_1 = t] - \Pr[t_1 \leftarrow D_2 : t_1 = t]| < \frac{1}{k2^{\lambda-1}}.$$

Then $\mathsf{SD}(\mathfrak{f}_\mathfrak{p}(D_1), \mathfrak{f}_\mathfrak{p}(D_2)) \leq 1/k2^{\lambda-1}$ since the *deterministic* $\mathfrak{f}_\mathfrak{p}$ doesn't amplify the statistical distance. And we have $\mathsf{SD}(\mathcal{P}_2(D_1), \mathcal{P}_2(D_2)) \leq 1/k2^{\lambda-1}$ from (6) . Similarly, the statistical distance between $\{t_2 \leftarrow [0, pS) : \mathcal{S}(t_2)\}$ and $\{t_2 \leftarrow [0, p\hat{s}) : \mathcal{S}(t_2)\}$ is also less than $1/k2^{\lambda-1}$. Combining (8), it follows

$$\mathsf{SD}(\{t_1 \leftarrow [0, pS) : \mathcal{P}_2(t_1)\}, \{t_2 \leftarrow [0, pS) : \mathcal{S}(t_2)\}) < \frac{1}{k2^{\lambda-2}}.$$

□

5 Two-party ECDSA

We now present an efficient construction for two-party ECDSA protocol and prove its security under a simulation-based definition. We follow the framework of [Lin17, CCL+19], but apply the *promise* Σ-protocol to avoid doing parallel repetition which is the main efficiency bottleneck in [CCL+19]. Our protocol, as depicted in Fig. 1, differs from [CCL+19] as follows (labeled with colored boxes in Fig. 1):

1. $\mathcal{P}_1$ is required to generate a CL public key of the form $(g_p = \hat{g}_p^p, h = \hat{h}^p)$ to make it sure $p \nmid \mathrm{ord}(g_p)$ and $p \nmid \mathrm{ord}(h)$.
2. $\mathcal{P}_1$ proves via the *promise* Σ-protocol Σ^1_{prom} described in Section 3.1 that Q_1 and the CL ciphertext c_{key} encode the same message x_1, i.e., $(Q_1, c_{key}) \in \mathcal{L}_{\mathrm{DLCL}}$.
3. In the signing phase, $\mathcal{P}_2$ generates a ciphertext by homomorphic operations together with $t_p = t \bmod p$ in the same way described in Section 4.

We use the ideal zero knowledge functionality $\mathcal{F}_{\mathrm{zk}}$ for the following NP relation (where the parameters of the elliptic curve $(\mathbb{G}, G, p)$ are implicit public

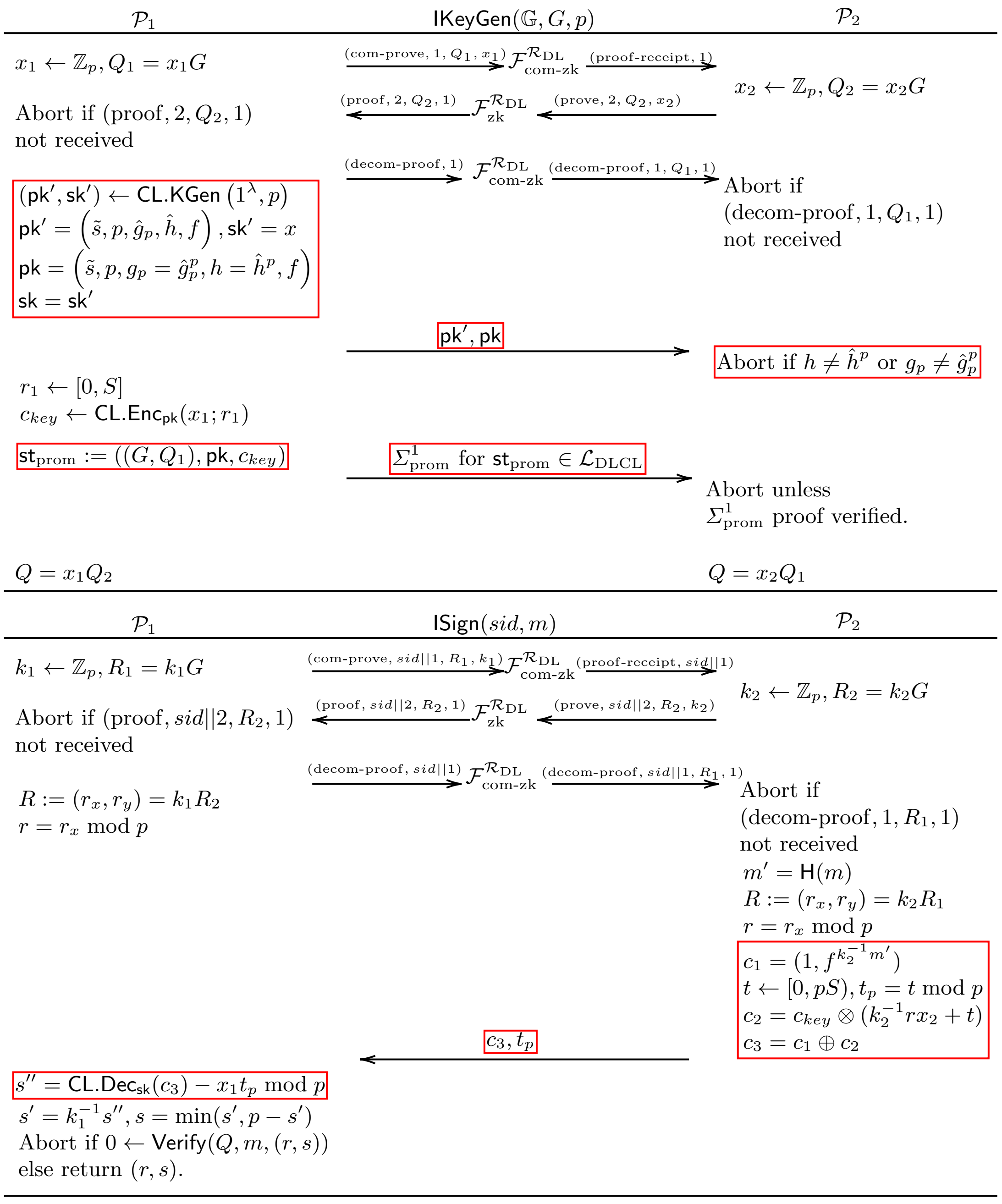
$\mathcal{P}_1$
$\mathsf{lKeyGen}(\mathbb{G}, G, p)$
$\mathcal{P}_2$
$x_1 \leftarrow \mathbb{Z}_p, Q_1 = x_1 G$
$(\text{com-prove}, 1, Q_1, x_1)$
$\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{com-zk}}$
$(\text{proof-receipt}, 1)$
$x_2 \leftarrow \mathbb{Z}_p, Q_2 = x_2 G$
Abort if $(\text{proof}, 2, Q_2, 1)$ not received
$(\text{proof}, 2, Q_2, 1)$
$\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{zk}}$
$(\text{prove}, 2, Q_2, x_2)$
$(\text{decom-proof}, 1)$
$\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{com-zk}}$
$(\text{decom-proof}, 1, Q_1, 1)$
Abort if $(\text{decom-proof}, 1, Q_1, 1)$ not received
$(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{CL.KGen}\left(1^\lambda, p\right)$
$\mathsf{pk}' = \left(\tilde{s}, p, \hat{g}_p, \hat{h}, f\right), \mathsf{sk}' = x$
$\mathsf{pk} = \left(\tilde{s}, p, g_p = \hat{g}_p^p, h = \hat{h}^p, f\right)$
$\mathsf{sk} = \mathsf{sk}'$
$\mathsf{pk}', \mathsf{pk}$
Abort if $h \neq \hat{h}^p$ or $g_p \neq \hat{g}_p^p$
$r_1 \leftarrow [0, S]$
$c_{key} \leftarrow \mathsf{CL.Enc}_{\mathsf{pk}}(x_1; r_1)$
$\mathsf{st}_{\text{prom}} := ((G, Q_1), \mathsf{pk}, c_{key})$
Σ^1_{prom} for $\mathsf{st}_{\text{prom}} \in \mathcal{L}_{\text{DLCL}}$
Abort unless Σ^1_{prom} proof verified.
$Q = x_1 Q_2$
$Q = x_2 Q_1$
$\mathcal{P}_1$
$\mathsf{lSign}(sid, m)$
$\mathcal{P}_2$
$k_1 \leftarrow \mathbb{Z}_p, R_1 = k_1 G$
$(\text{com-prove}, sid\|1, R_1, k_1)$
$\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{com-zk}}$
$(\text{proof-receipt}, sid\|1)$
$k_2 \leftarrow \mathbb{Z}_p, R_2 = k_2 G$
Abort if $(\text{proof}, sid\|2, R_2, 1)$ not received
$(\text{proof}, sid\|2, R_2, 1)$
$\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{zk}}$
$(\text{prove}, sid\|2, R_2, k_2)$
$(\text{decom-proof}, sid\|1)$
$\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{com-zk}}$
$(\text{decom-proof}, sid\|1, R_1, 1)$
$R := (r_x, r_y) = k_1 R_2$
$r = r_x \bmod p$
Abort if $(\text{decom-proof}, 1, R_1, 1)$ not received
$m' = \mathsf{H}(m)$
$R := (r_x, r_y) = k_2 R_1$
$r = r_x \bmod p$
$c_1 = (1, f^{k_2^{-1} m'})$
$t \leftarrow [0, pS), t_p = t \bmod p$
$c_2 = c_{key} \otimes (k_2^{-1} r x_2 + t)$
$c_3 = c_1 \oplus c_2$
c_3, t_p
$s'' = \mathsf{CL.Dec}_{\mathsf{sk}}(c_3) - x_1 t_p \bmod p$
$s' = k_1^{-1} s'', s = \min(s', p - s')$
Abort if $0 \leftarrow \mathsf{Verify}(Q, m, (r, s))$
else return (r, s).

Fig. 1: Two-party ECDSA Key Generation and Signing Protocols

inputs): $\mathcal{R}_{\mathrm{DL}} = \{(Q; w) | Q = wG\}$. Functionality $\mathcal{F}_{\mathrm{zk}}$ can be efficiently instantiated by Schnorr protocol. Note that instead of using the $\mathcal{F}_{\mathrm{zk}}$-hybrid model, we use the *promise* Σ-protocol directly in our construction.

In Fig 1 we denote by $\otimes$ and $\oplus$ the homomorphic operations, defined as $c_{key} \otimes k = (c_{key,1}^{k}, c_{key,2}^{k})$ and $c_1 \oplus c_2 = (c_{1,1} \cdot c_{2,1}, c_{1,2} \cdot c_{2,2})$, where c_{key}, c_1, c_2 are CL ciphertexts and k is an integer.

Theorem 4. *Under the DDH assumption, the HSM assumption and the Double Encoding assumption, the protocol described in Fig. 1 securely computes $\mathcal{F}_{\mathrm{ECDSA}}$ for a two-party case in the $(\mathcal{F}_{\mathrm{zk}}, \mathcal{F}_{\text{com-zk}})$-hybrid model in the presence of a malicious static adversary under the simulation-based definition.*

Our construction is to some extent derived from the one in [CCL+19] except that the *promise* Σ-protocol only enjoys a *weaker* special soundness. On one hand, if the adversary $\mathcal{A}$ corrupts party $\mathcal{P}_2$ which only verifies a *promise* Σ-proof, we can simulate $\mathcal{P}_1$ in the same manner as in [CCL+19]; On the other hand, if $\mathcal{P}_1$ is corrupted by $\mathcal{A}$ who plays the role of a prover in a *promise* Σ-protocol, we could construct a simulator to generate an indistinguishable view from the adversarial perspective by leveraging the extraction by rewinding (the first property of *promise extraction*) and the technique to simulate homomorphic operations (described in Section 4). The detailed proof of Theorem 4 is presented in Appendix B.

6 Multi-party (Threshold) ECDSA

In this section, we show how to use *promise* Σ-protocols to remove the low order assumption and strong root assumption for the multi-party (threshold) ECDSA of [CCL+20]. The resulting protocol is more efficient than the one of [CCL+20] in terms of both bandwidth and computational efficiency. Our techniques also apply the multi-party protocol of [LN18] to improve bandwidth efficiency at the cost of relatively high computational complexity as in the case [CCL+20].

6.1 Improvment on [CCL+20] with *promise* Σ-protocols

In the threshold ECDSA of [CCL+20], their zero knowledge proof for proving the well-formedness of a CL ciphertext requires a random group generator g_p due to the need of strong root assumption. This leads to a costly interactive setup phase to generate such g_p. Without relying the assumption, we could remove this phase.

We modify the threshold ECDSA protocol in [CCL+20] with *promise* Σ-protocols in the following way (labeled with colored boxes in Fig. 2 and Fig. 3):

1. After generating the CL public/secret key pair $(\hat{\mathsf{pk}}_i = (\tilde{s}, p, \hat{g}_p, \hat{h}, f), \hat{\mathsf{sk}}_i)$, we have each party refresh the public key to obtain a new $\mathsf{pk}_i = (\tilde{s}, p, g_p = \hat{g}_p^p, h = \hat{h}^p, f)$ as in the two-party case, and additionally generate a public/secret key pair of $\mathsf{ElGamal}$ encryption $(\mathsf{pk}'_i, \mathsf{sk}'_i) \leftarrow \mathsf{EG.KGen}(1^\lambda)$ in the key generation and broadcast $(\mathsf{pk}'_i, \hat{\mathsf{pk}}_i, \mathsf{pk}_i)$.

2. In **Phase** 1 of the signing phase of [CCL+20], we have each party $\mathcal{P}_i$ encrypt k_i using CL encryption scheme, as well as encoding it using $\mathsf{ElGamal}$ encryption scheme, then use the *promise* Σ-protocol Σ^2_{prom} decsribed in Section 3.1 to prove the plaintexts equality.
3. In **Phase** 2 of the signing phase of [CCL+20], instead of generating a CL ciphertext $c_{k_j\gamma_i}$ of $k_j\gamma_i - \beta_{j,i} \bmod p$, $\mathcal{P}_i$, like $\mathcal{P}_2$ in its final step of the two-party signing protocol described in the previous section, homomorphically computes a CL ciphertext of $k_j\gamma_i + k_j\hat{t}_{j,i} - \beta_{j,i} \bmod p$, where $\hat{t}_{j,i}$ is selected uniformly from a sufficient large space $[0, pS)$. $\mathcal{P}_i$ generates a ciphertext $c_{k_jw_i}$ of $k_jw_i + k_jt_{j,i} - v_{j,i} \bmod p$ in the same way. And $\mathcal{P}_i$ sends $c_{k_j\gamma_i}, c_{k_jw_i}$, along with $\hat{t}_{p,ji} = \hat{t}_{j,i} \bmod p$ and $t_{p,ji} = t_{j,i} \bmod p$ (for $\mathcal{P}_j$ to derandomized the plaintexts) to $\mathcal{P}_j$.

$\mathcal{P}_i$	$\mathsf{IKeyGen}\ (\mathbb{G}, G, p)$	All players $\{\mathcal{P}_j\}_{j\neq i}$
$u_i \leftarrow \mathbb{Z}_p$ $(\mathsf{kgc}_i, \mathsf{kgd}_i) \leftarrow \mathsf{Com}(u_iG)$		
$\boxed{\begin{array}{l}(\hat{\mathsf{sk}}_i = d_i, \hat{\mathsf{pk}}_i) \leftarrow \mathsf{CL.KGen}\left(1^\lambda, p\right)\\ \text{where } \hat{\mathsf{pk}}_i = \left(\tilde{s}, p, \hat{g}_p, \hat{h}_i, f\right)\\ \mathsf{pk}_i = \left(\tilde{s}, p, g_p = \hat{g}_p^p, h_i = \hat{h}_i^p, f\right), \mathsf{sk}_i = \hat{\mathsf{sk}}_i\\ (\mathsf{pk}'_i,\ \mathsf{sk}'_i) \leftarrow \mathsf{EG.KGen}\left(1^\lambda\right)\end{array}}$		
	$\xrightarrow{\boxed{\hat{\mathsf{pk}}_i, \mathsf{pk}_i, \mathsf{pk}'_i \text{ and } \mathsf{kgc}_i}}$ $\xrightarrow{\mathsf{kgd}_i}$	$\boxed{\begin{array}{l}\text{Abort if } \exists i \text{ s.t.}\\ g_p \neq \hat{g}_p^p \text{ or } h_i \neq \hat{h}_i^p\end{array}}$
		$Q_i \leftarrow \mathsf{Open}(\mathsf{kgc}_i, \mathsf{kgd}_i)$ s.t. $Q_i = u_iG$ $Q = \sum_{i=1}^n Q_i$
Perform (t,n)-VSS share of u_i: $p_i(X) = u_i + \sum_{k=1}^t a_{i,k}X^k \bmod p$ Set $\{\sigma_{i,j} := p_i(j)\}_{j\in[n]}$ and $\{V_{i,k} := a_{i,k}G\}_{k\in[t]}$	$\xrightarrow{\text{Send } \sigma_{i,j} \text{to } \mathcal{P}_j}$ $\xrightarrow{\{V_{i,k}\}_{k\in[t]}}$	
$\{\sigma_{j,i}\}_j$ are additive shares of $x_i := \sum_{j\in[n]} p_j(i)$ where $\{x_i\}_{i\in[n]}$ are (t,n)-Shamir shares of x. $\pi_{\mathsf{kg},i} \leftarrow \mathsf{ZKPoK}_{X_i}\{(x_i) : X_i = x_iG\}$	$\xrightarrow{\pi_{\mathsf{kg},i}}$	Abort if $\exists i\ \pi_{\mathsf{kg},i}$ is rejected

Fig. 2: Multi-Party Key Generation Protocol

Following [GG18, CCL+20], we also use cryptographic primitives such as Feldman's verifiable secret sharing (VSS) scheme and a non-malleable equivocable commitment. We refer to [GG18, CCL+20] for more details of the two schemes.

To enable a threshold signing protocol where a subset $S \subseteq [n]$ of parties collaborate to sign a message m, given the (t,n) shares $\{x_i\}_{i\in[n]}$ of x obtained

$\mathcal{P}_i$	**Phase 1**	All players $\{\mathcal{P}_j\}_{j \neq i}$
$k_i, \gamma_i \leftarrow \mathbb{Z}_p, r_i \leftarrow [0,\ S]$ $c_{k\,i} \leftarrow \mathsf{CL.Enc}_{\mathsf{pk}_i}(k_i; r_i)$ $(\mathsf{c}_i, \mathsf{d}_i) \leftarrow \mathsf{Com}(\gamma_i G)$		
$\boxed{\begin{array}{l} r'_i \leftarrow \mathbb{Z}_p \\ C_{k\,i} \leftarrow \mathsf{EG.Enc}_{\mathsf{pk}'_i}(k_i; r'_i) \end{array}}$	$\xrightarrow{\boxed{\mathsf{c}_i, c_{k\,i}, C_{k_i}}}$	
$\boxed{\mathsf{st}_i := (\mathsf{pk}_i, \mathsf{pk}'_i, C_{k_i}, c_{k_i})}$	$\xrightarrow{\boxed{\Sigma^2_{\text{prom}}\text{-proof } \pi_i \text{ for } \mathsf{st}_i \in \mathcal{L}_{\text{EGCL}}}}$	Abort if a proof fails
	Phase 2	
$\beta_{j,i}, v_{j,i} \leftarrow \mathbb{Z}_p, B_{j,i} = v_{j,i} G$		
$\boxed{\begin{array}{l} t_{j,i}, \hat{t}_{j,i} \leftarrow [0, pS) \\ t_{p,ji} = t_{j,i} \bmod p, \hat{t}_{p,ji} = \hat{t}_{j,i} \bmod p \\ c_{k_j \gamma_i} \leftarrow c_{k_j} \otimes (\gamma_i + \hat{t}_{j,i}) \oplus \left(1, f^{-\beta_{j,i}}\right) \\ c_{k_j w_i} \leftarrow c_{k_j} \otimes (w_i + t_{j,i}) \oplus \left(1, f^{-v_{j,i}}\right) \end{array}}$	$\xrightarrow{\boxed{c_{k_j \gamma_i}, c_{k_j w_i}, B_{j,i}, t_{p,ji}, \hat{t}_{p,ji}}}$	$\boxed{\begin{array}{l} \alpha_{j,i} \leftarrow \mathsf{CL.Dec}_{\mathsf{sk}_j}(c_{k_j \gamma_i}) - k_j \hat{t}_{p,ji} \\ \mu_{j,i} \leftarrow \mathsf{CL.Dec}_{\mathsf{sk}_j}(c_{k_j w_i}) - k_j t_{p,ji} \\ \text{Abort if } \mu_{j,i} G + B_{j,i} \neq k_j W_i \end{array}}$
$\delta_i = k_i \gamma_i + \sum_{j \neq i} (\alpha_{i,j} + \beta_{j,i})$ $\sigma_i = k_i w_i + \sum_{j \neq i} (\mu_{i,j} + v_{j,i})$		
	Phase 3	
	$\xrightarrow{\delta_i}$	$\delta = \sum_{i \in S} \delta_i = k\gamma$
	Phase 4	
$\pi_{\gamma_i} \leftarrow \mathsf{ZKPoK}_{\Gamma_i}\{(\gamma_i) : \Gamma_i = \gamma_i G\}$	$\xrightarrow{\mathsf{d}_i}$ $\xrightarrow{\pi_{\gamma_i}}$	$\Gamma_i \leftarrow \mathsf{Open}(\mathsf{c}_i, \mathsf{d}_i) = \gamma_i G$ Abort if a proof fails. $(R := (r_x, r_y)) = \delta^{-1} \left(\sum_{i \in S} \Gamma_i\right)$ and $r = r_x \bmod p$
	Phase 5	
$m' = \mathsf{H}(m), s_i = m' k_i + r \sigma_i$ $\ell_i, \rho_i \leftarrow \mathbb{Z}_p$ $V_i = s_i R + \ell_i G, A_i = \rho_i G$ $(\hat{\mathsf{c}}_i, \hat{\mathsf{d}}_i) \leftarrow \mathsf{Com}(V_i, A_i)$ $\hat{\pi}_i \leftarrow \mathsf{ZKPoK}_{(V_i, A_i)}\{(s_i, \ell_i, \rho_i) :$ $V_i = s_i R + \ell_i G \wedge A_i = \rho_i G\}$	$\xrightarrow{\hat{\mathsf{c}}_i}$ $\xrightarrow{\hat{\mathsf{d}}_i}$ $\xrightarrow{\hat{\pi}_i}$	
$U_i = \rho_i V, T_i = \ell_i A$ $\left(\tilde{\mathsf{c}}_i, \tilde{\mathsf{d}}_i\right) \leftarrow \mathsf{Com}(U_i, T_i)$	$\xrightarrow{\tilde{\mathsf{c}}_i}$ $\xrightarrow{\tilde{\mathsf{d}}_i}$ $\xrightarrow{s_i}$	Abort if a proof fails $V = -m' G - rQ + \sum_{i \in S} V_i$ $A = \sum_{i \in S} A_i$ Abort if $\sum_{i \in S} T_i \neq \sum_{i \in S} U_i$ $s = \sum_{i \in S} s_i$ Abort if (r, s) is not valid, else return (r, s).

Fig. 3: Multi-Party Threshold Signing Protocol

in the key generation phase, each party can compute the additive shares $\{w_i\}_{i \in S}$ of x using the appropriate Lagrangian coefficients, as well as $\{W_i = w_i G\}_{i \in S}$. We also use the symbols $\otimes, \oplus$ defined in Section 5 to represent homomorphic operations.

Note that the only subprotocol of the construction of [CCL+20] that requires the low order assumption and the strong root assumption is the Σ-protocol for the correctness of CL ciphertexts. When replacing such a subprotocol with our *promise* Σ-protocol (as in the above second modification), we remove these two stronger assumptions since our *promise* Σ-protocol per se does not rely on any assumptions. Thus we have the following theorem, and leave its proof in Appendix C.

Theorem 5. *Under the assumption that the standard ECDSA is existentially unforgeable, the DDH assumption, the HSM assumption, and the assumption that* Com *is equivocable and non-malleable, then the protocol of Fig. 2 and 3 is an existentially unforgeable threshold signature scheme.*

6.2 Improving the bandwidth efficiency of [LN18]

Lindell and Nof [LN18] propose an efficient multi-party ECDSA but with higher bandwidth due to the usage of $\mathsf{Paillier}$ encryption and expensive zero knowledge range proofs in a subprotocol, called $\pi^{\mathsf{priv}}_{\mathsf{mult}}$. In the first round of the protocol each party $\mathcal{P}_i$ sends a $\mathsf{Paillier}$ encryption c_i of x_i (under its own public key), and receives back c_j. In the second round $\mathcal{P}_i$ selects a random $r_{i \to j}$ and homomorphically generates $c_{i \to j}$ which is an encryption of $x_j \cdot y_i + r_{i \to j}$, then sends it to $\mathcal{P}_j$.$\mathcal{P}_i$ decrypts $c_{j \to i}$ to obtain $z_{j \to i}$, and computes $z_i = \sum_{j \in [n] \setminus \{i\}} z_{j \to i} + x_i y_i - \sum_{j \in [n] \setminus \{i\}} r_{i \to j}$.To ensure the parties follow the protocol, each party provides two zero knowledge proofs for every other one at the end of each round: One zero knowledge proof for correctness of the ciphertext, and the other for proving the correctness of the homomorphic operations in generating $c_{i \to j}$.

Similarly, within the subprotocol $\pi^{\mathsf{priv}}_{\mathsf{mult}}$, we can replace the above two zero knowledge proofs with our *promise* Σ-protocol Σ^2_{prom} and Σ^3_{prom} described in Section 3 via encoding the secret message into an $\mathsf{ElGamal}$ ciphertext and an CL ciphertext instead of a $\mathsf{Paillier}$ ciphertext, which achieves better bandwidth efficiency. We stress that, due to the relatively heavy computation over class groups, this replacement will increase the computational complexity as the case in [CCL+20].

7 Comparisons

In this section, we compare implementations of our protocols with the state-of-the-art ones. For fair comparison, we implement four two-party protocols with Rust, including our protocol, the protocol in [CCL+19], its variant in [CCL+20] and the protocol in [Lin17], and two multi-party ECDSA including our protocol and the one in [CCL+20]. The elliptic curve is secp256k1 and the bit length of the discriminant of the class group is chosen as 1827, which ensures that our protocols have 128-bit security. We use Pari C library to handle arithmetic operations in class groups and Paillier encryption. The running times are measured on a single core of an Intel(R) Core(TM) i7-9700K @ 3.6GHz.

Two-Party ECDSA Protocol. In the theoretical aspect, we compare our two-party ECDSA protocol with [CCL+19] and its improved variant in [CCL+20] which reduces the repetition rounds of the zero knowledge proof to $\kappa/10$ times with soundness error $2^{-\kappa}$.

The theoretical comparisions are given in Table 1 and Table 2. Since the exponential operation in class groups is much costly than in elliptic curve and dominates the computation cost, we only list the number of exponentiations in class groups, and denote it as #CL-Exp. $|\mathcal{G}|$ and $|\mathbb{G}|$ are size of group elements in $\mathcal{G}$ and $\mathbb{G}$, respectively. L, L_p are the length of the integers sampled from $\mathcal{D}_p$ and $\mathbb{Z}_p$. In our implementation, $|\mathbb{G}| = 33$ Bytes, $|\mathcal{G}| = 345$ Bytes, $L = 115$ Bytes and $L_p = 32$ Bytes.

	Keygen (#CL-Exp)	Signing (#CL-Exp)	Assumptions (related to class group)
Ours	11	3	HSM + Double Encoding
[CCL+19]	$4\kappa + 2$	5	HSM + Double Encoding
[CCL+20]	$(6\kappa)/10 + 2$	5	HSM + Double Encoding

Table 1: Theoretical Comparisons in Computation of Two-Party Protocols

	Keygen (Bytes)	Signing (Bytes)
Ours	$5\lvert\mathbb{G}\rvert + 4\lvert\mathcal{G}\rvert + 8L_p + L$	$4\lvert\mathbb{G}\rvert + 2\lvert\mathcal{G}\rvert + 7L_p$
[CCL+19]	$(4+\kappa)\lvert\mathbb{G}\rvert + (2\kappa+2)\lvert\mathcal{G}\rvert + (6+\kappa)L_p + \kappa L$	$4\lvert\mathbb{G}\rvert + 2\lvert\mathcal{G}\rvert + 6L_p$
[CCL+20]	$(4+\kappa/10)\lvert\mathbb{G}\rvert + (\kappa/5+2)\lvert\mathcal{G}\rvert + (6+\kappa/10)L_p + (\kappa/10)L$	$4\lvert\mathbb{G}\rvert + 2\lvert\mathcal{G}\rvert + 6L_p$

Table 2: Theoretical Comparisons in Communication of Two-Party Protocols

	Keygen (ms)	Signing (ms)	Keygen (Bytes)	Signing (Bytes)
Ours	967	391	1916	1046
[CCL+19] ($\kappa = 40$)	14107	442	35814	1014
[CCL+20] ($\kappa = 40$)	2275	442	4494	1014
[Lin17] ($\kappa = 40$)	6120	41	96805	1092
[CCL+19] ($\kappa = 128$)	44740	442	112374	1014
[CCL+20] ($\kappa = 128$)	6471	442	11454	1014
[Lin17] ($\kappa = 128$)	19032	41	305189	1092

Table 3: Concrete Performance of Two-Party Protocols

As shown in Table 1, in the key generation phase our two-party ECDSA protocol is about $15\times$ (resp. about $2\times$) faster than the protocol in [CCL+19] (resp. in [CCL+20]) when $\kappa = 40$. The improvement is about $44\times$ (resp. about

7×) when $\kappa = 128$. Our protocol is slightly better than the ones in [CCL+19] and [CCL+20] in the signing phase.

Our protocol also reduces the communication cost significantly. As in Table 2, the improvement of communication is about 17× (resp. 2×) in the key generation phase compared to the protocol in [CCL+19] (resp. in [CCL+20]) when $\kappa = 40$. The improvement is about 54× (resp. about 6×) when $\kappa = 128$. The communication cost in the signing phase is almost the same.

In the concrete apsect, we compare all the four protocols. The running time and consumed bandwidth of our protocol and the protocols in [CCL+19] and [CCL+20] shown in Table 3 meet the theoretical analysis above. Further, we compare our protocol with the one in [Lin17], where $\mathsf{Paillier}$ modulus is chosen as 3072 to get 128-bit security. In the key generation phase, our protocol improves the computation performance by a factor about 6× (resp. 20×) when $\kappa = 40$ (resp. $\kappa = 128$). Our protocol also reduces the bandwidth by a factor about 47× (resp. 149×) when $\kappa = 40$ (resp. $\kappa = 128$).

	Keygen (#$\mathsf{CL\text{-}Exp}$)	Signing (#$\mathsf{CL\text{-}Exp}$)	Assumptions (related to class group)
Ours	$2n+1$	$10t-6$	HSM
[CCL+20]	$((2n-1)\kappa)/10+2$	$14t-10$	HSM + Low Order + Strong Root

Table 4: Theoretical Comparisons in Computation of Multi-Party Protocols

Multi-Party ECDSA Protocol. The improvement of our multi-party ECDSA protocol on [CCL+20], which is essentially based on [GG18], is very obvious. Using the same notations as above, we show the theoretical comparisons in Table 4 and Table 5, and the concrete comparison in Table 6.

	Keygen (Bytes)	Signing (Bytes)
Ours	$((4+t)\lvert\mathbb{G}\rvert + \lvert\mathcal{G}\rvert + 5L_p)(n-1)$	$(17\lvert\mathbb{G}\rvert + 8\lvert\mathcal{G}\rvert + 19L_p + L)(t-1)$
[CCL+20]	$((3+t)\lvert\mathbb{G}\rvert + (\kappa/10+3)\lvert\mathcal{G}\rvert + 10L_p + (\kappa/10)L)(n-1)$	$(9\lvert\mathbb{G}\rvert + 8\lvert\mathcal{G}\rvert + 16L_p + L)(t-1)$

Table 5: Theoretical Comparisons in Communication of Multi-Party Protocols

	Keygen (ms)	Signing (ms)	Keygen (Bytes)	Signing (Bytes)
Ours	$186n+95$	$1137t-539$	$33tn+637n-33t-637$	$4044t-4044$
[CCL+20] ($\kappa = 40$)	$739n-163$	$1258t-834$	$33tn+3792n-33t-3792$	$3684t-3684$
[CCL+20] ($\kappa = 128$)	$2287n-934$	$1252t-842$	$33tn+7434n-33t-7434$	$3684t-3684$

Table 6: Concrete Performance of Multi-Party Protocols

In terms of computational complexity, our multi-party ECDSA protocol is about 4× (resp. 12×) faster than the protocol in [CCL+20] in the key generation phase when $\kappa = 40$ (resp. $\kappa = 128$), which can be seen both in the theoretical and concrete aspects. The signing phase of our construction is slightly better than that in [CCL+20], and it is about 10% faster in the concrete aspect.

In terms of communications, since we eliminate the need of costly interactive setup phase, our protocol outperforms the one in [CCL+20] in the key generation phase for both $\kappa = 40$ and $\kappa = 128$, factors vary according to the number of parties n and the threshold t. In the signing phase the communication overhead is slightly larger while our solution remains of the same order of magnitude.

Finally, it is worth noting that all our constructions are based on HSM assumption (along with other assumptions in elliptic curve group) just as in [CCL+19], instead of using stronger and non-standard assumptions: the low order assumption and the strong root assumption as in [CCL+20].

Acknowledgments

We would like to thank the anonymous reviewers for their valuable suggestions. We are supported by the National Natural Science Foundation of China (Grant No. 61932019, No. 61772521 and No. 61772522) and the Key Research Program of Frontier Sciences, CAS (Grant No. QYZDB-SSW-SYS035).

Bibliography

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712.

[BGG19] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security. In Tanja Lange and Orr Dunkelman, editors, *Progress in Cryptology – LATINCRYPT 2017*, pages 352–377, Cham, 2019. Springer International Publishing.

[BKSW20] Karim Belabas, Thorsten Kleinjung, Antonio Sanso, and Benjamin Wesolowski. A note on the low order assumption in class group of an imaginary quadratic number fields. Cryptology ePrint Archive, Report 2020/1310, 2020. https://eprint.iacr.org/2020/1310.

[CCL+19] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ecdsa from hash proof systems and efficient instantiations. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 191–221, Cham, 2019. Springer International Publishing.

[CCL+20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold ec-dsa. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 266–296, Cham, 2020. Springer International Publishing.

[CL15] Guihem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from ddh. In *Topics in Cryptology — CT-RSA 2015*, pages 487–505. Springer International Publishing, 2015.

[CLT18] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In *Advances in Cryptology – ASIACRYPT 2018*, pages 733–764. Springer International Publishing, 2018.

[Coh00] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 2000.

[Des88] Yvo Desmedt. Society and group oriented cryptography: a new concept. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 120–127, Berlin, Heidelberg, 1988. Springer.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ecdsa from ecdsa assumptions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 980–997, 2018.

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1051–1066, 2019.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO'86*, LNCS 263, pages 186–194. Springer, 1987.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1179–1194, New York, NY, USA, 2018. Association for Computing Machinery.

[GG20] Rosario Gennaro and Steven Goldfeder. One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. https://eprint.iacr.org/2020/540.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *Applied Cryptography and Network Security*, pages 156–174, Cham, 2016. Springer International Publishing.

[GJKR96] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 354–371, Berlin, Heidelberg, 1996. Springer.

[GPS06] Marc Girault, Guillaume Poupard, and Jacques Stern. On the fly authentication and signature schemes based on groups of unknown order. *Journal of Cryptology*, 19:463–487, 2006.

[Lin17] Yehuda Lindell. Fast secure two-party ecdsa signing. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 613–644, Cham, 2017. Springer International Publishing.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1837–1854, New York, NY, USA, 2018. Association for Computing Machinery.

[MR01] Philip MacKenzie and Michael K. Reiter. Two-party generation of dsa signatures. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 137–154, Berlin, Heidelberg, 2001. Springer.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Advances in Cryptology - EUROCRYPT'96*, LNCS 1070, pages 387–398. Springer, 1996.

[YCX21] Tsz Hon Yuen, Handong Cui, and Xiang Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In *Public-Key Cryptography - PKC 2021*, volume 12710, pages 481–511. Springer, 2021.

A Promise NIZK in the Random Oracle Model

Let $(\mathsf{DL.Gen}, \mathsf{DL.Code})$, $(\mathsf{CL.Gen}, \mathsf{CL.Code})$ and language $\mathcal{L}$ be defined as in Section 3. We here present a concept of *promise* NIZK in the random oracle model.

Definition 8 (Promise NIZK). *Let λ be the security parameter and $\rho(\cdot)$ be an efficiently computable function. A promise NIZK protocol (P, V) for $\mathcal{L}$ with respect to $\rho(\cdot)$ satisfies the following conditions:*

- *Completeness and zero knowledge defined in the same way as a NIZK protocol.*
- *Promise extractability. For any inverse polynomial $\epsilon(\lambda)$, any PPT P^* that generates a proof with an accepted probability $\epsilon(\lambda)$, there is a PPT extractor Ext such that the following conditions hold.*
    1. *Extraction by rewinding for $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$. With oracle access to P^* and the programmability of the random oracle, $\mathsf{Ext}^{\mathsf{P}^*,\mathsf{H}}(cw_0, cw_1)$ extracts m of $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$ with probability negligibly close to 1.*
    2. *Straight-line extraction for $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$ using secret keys. If both key pairs $(\mathsf{pk}_0, \mathsf{sk}_0)$ and $(\mathsf{pk}_1, \mathsf{sk}_1)$ are honestly generated, then given $(\mathsf{sk}_0, \mathsf{sk}_1)$ as input the extractor $\mathsf{Ext}(\mathsf{sk}_0, \mathsf{sk}_1, cw_0, cw_1)$ (without access to P^*) extracts $\rho(m)$ with probability negligibly close to 1, where m is message encoded into $\mathsf{DL.Code}_{\mathsf{pk}_0}(m)$.*

Applying the Fiat-Shamir transform to our *promise* Σ-protocol Σ^2_{prom} for equality of plaintexts, we could obtain the following *promise* NIZK Π_{prom} by using a hash function H in the random oracle model.

Protocol Π_{prom} for proving the equality of plaintexts

Common input: $\mathsf{st}_{\text{prom}} = ((G, P), (\tilde{s}, p, g_p, f, h), (C_1, C_2, c_1, c_2))$.
P's Private input: $m \in \mathbb{Z}_p$, $r_1 \in \mathbb{Z}_p$ and $r_2 \in [0, S]$ s.t. $C_1 = r_1 G$, $C_2 = r_1 P + mG$, $c_1 = g_p^{r_2}$ and $c_2 = h^{r_2} f^m$.

Proof generation by P.

1. Choose $s_1 \leftarrow \mathbb{Z}_p$ and $s_2 \leftarrow [0, U)$ and $s_m \leftarrow \mathbb{Z}_p$ at random.
2. Compute $A_1 = s_1 G$, $A_2 = s_1 P + s_m G$, $a_1 = g_p^{s_2}$, $a_2 = h^{s_2} f^{s_m}$.
3. Compute $e = \mathsf{H}(\mathsf{st}_{\text{prom}}, \{A_i\}_{i=1}^2, \{a_i\}_{i=1}^2)$.
4. Compute $z_1 = s_1 + er_1 \mod p$, $z_2 = s_2 + er_2$ and $z_m = s_m + em \mod p$.
5. Send $\pi_{\text{prom}} = (\{A_i\}_{i=1}^2, \{a_i\}_{i=1}^2, z_1, z_2, z_m)$ to V.

Proof verification by V.

1. Parse $\pi_{\text{prom}} = (\{A_i\}_{i=1}^2, \{a_i\}_{i=1}^2, z_1, z_2, z_m)$.
2. Compute $e = \mathsf{H}(\mathsf{st}_{\text{prom}}, \{A_i\}_{i=1}^2, \{a_i\}_{i=1}^2)$.
3. Output 1 iff $z_2 \in [0, U + (p-1)S)$, $z_1 G = A_1 + eC_1$, $z_1 P + z_m G = A_2 + eC_2$, $g_p^{z_2} = a_1 c_1^e$ and $h^{z_2} f^{z_m} = a_2 c_2^e$.

Lemma 2. *If $(p-1)S/U$ is negligible, then protocol Π_{prom} is a promise NIZK proof in the random oracle with respect to the identity function $\rho : m \to m$.*

Proof. Note that it follows from standard arguments that the protocol Π_{prom} satisfies completeness, zero knowledge in the random oracle model.

Here we just give a proof sketch for the *promise extractability.* Observe that if the probability that P^* makes the verifier accept Π_{prom} is greater than $\epsilon(\lambda)$ in the random oracle model, then we can define the set $\mathcal{G}'_{r_p^*}$ with a fixed good random tap r_p^* as in the previous section, and show that the size of $\mathcal{G}'_{r_p^*}$ is greater than $p\epsilon(\lambda)$ using the forking lemma in [PS96], which yields an extractor Ext with failing probability 0 as showed previously, and thus the first condition of *promise extractability* holds.

With the same way as in proving Theorem 2, we could extract the plaintext m using the both secret keys without rewinding the prover, thus conclude the second condition of *promise extractability*: *straight-line extraction.*

Similarly, we can obtain *promise* NIZK proofs for the other two *promise* Σ-protocols Σ^1_{prom} and Σ^3_{prom} described in Section 3.1 and 3.2 respectively.

B Proof of Theorem 4

Proof. We prove the security by constructing a PPT simulator $\mathcal{S}$ such that any polynomial-time adversary $\mathcal{A}$ which either corrupts $\mathcal{P}_1$ or $\mathcal{P}_2$ cannot distinguish a real execution of the protocol from a simulated one with non-negligible probability. Specifically, the simulator $\mathcal{S}$ only has access to a trusted party computing the ideal functionality $\mathcal{F}_{\text{ECDSA}}$, it learns in the ideal world the verification key Q and the signature (r, s) for message m as outputs of KeyGen and Sign, respectively.

First, we show how to simulate the key generation and signing when $\mathcal{A}$ corrupts $\mathcal{P}_1$.

Simulating Key Generation Phase When $\mathcal{A}$ Corrupts $\mathcal{P}_1$:

1. Given input $\mathsf{KeyGen}(\mathbb{G}, G, p)$, the simulator $\mathcal{S}$ sends it to $\mathcal{F}_{\text{ECDSA}}$ and receives a verification key Q.
2. $\mathcal{S}$ invokes $\mathcal{A}$ on input $\mathsf{KeyGen}(\mathbb{G}, \mathsf{G}, \mathsf{p})$ and receives $(\mathsf{com\text{-}prove}, 1, Q_1, x_1)$ as $\mathcal{A}$ intends to send to $\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{com-zk}}$.
3. $\mathcal{S}$ verifies whether $Q_1 = x_1 \cdot G$ ($\mathcal{S}$ uses the extractor to compute x_1). If so, then computes $Q_2 = x_1^{-1} \cdot Q$; otherwise just chooses a random Q_2.
4. $\mathcal{S}$ sends $(\mathsf{proof}, 2, Q_2, 1)$ to $\mathcal{A}$ as if sent by $\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{zk}}$.
5. $\mathcal{S}$ receives $(\mathsf{decom\text{-}proof}, 1)$ as $\mathcal{A}$ intends to send to $\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{com-zk}}$. If $Q_1 \neq x_1 G$, then $\mathcal{S}$ sends abort to the trusted party computing $\mathcal{F}_{\text{ECDSA}}$.
6. $\mathcal{S}$ verifies $h = \hat{h}^p, g_p = \hat{g}_p^p$ and the *promise* Σ-protocol for $\mathcal{L}_{\text{DLCL}}$. If all pass, $\mathcal{S}$ continues; otherwise aborts the simulation.

7. $\mathcal{S}$ sends continue to $\mathcal{F}_{\text{ECDSA}}$ for $\mathcal{P}_2$ to receive the output, and stores (x_1, Q, c_{key}).

Simulating Signing Phase When $\mathcal{A}$ Corrupts $\mathcal{P}_1$:

1. Upon receiving input $\mathsf{Sign}(sid, m)$, the simulator $\mathcal{S}$ sends it to $\mathcal{F}_{\text{ECDSA}}$ and receives a signature (r, s).
2. $\mathcal{S}$ computes $R = (r, r_y)$ with the ECDSA verification algorithm.
3. $\mathcal{S}$ invokes $\mathcal{A}$ with input $\mathsf{Sign}(sid, m)$, and simulates the first three rounds, which is the same as in the key generation stage to compute Q. We briefly describe the procedure as follows:
   (a) $\mathcal{S}$ receives $(\mathsf{com\text{-}prove}, sid||1, R_1, k_1)$ as $\mathcal{A}$ intends to send to $\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{com-zk}}$.
   (b) $\mathcal{S}$ verifies $R_1 = k_1 \cdot G$. If so, it computes $R_2 = k_1^{-1} \cdot R$; otherwise it samples a random R_2. $\mathcal{S}$ sends $(\mathsf{proof}, sid||2, R_2, 1)$ to $\mathcal{A}$.
   (c) $\mathcal{S}$ receives $(\mathsf{decom\text{-}proof}, sid||1)$ as $\mathcal{A}$ intends to send to $\mathcal{F}^{\mathcal{R}_{\text{DL}}}_{\text{com-zk}}$. If $R_1 \neq k_1 \cdot G$, then $\mathcal{S}$ sends abort to the trusted party computing $\mathcal{F}_{\text{ECDSA}}$.
4. $\mathcal{S}$ computes $s' = k_1 \cdot s \bmod p$ and $f^{s'}$. Then $\mathcal{S}$ chooses $t \leftarrow [0, pS)$, computes $c_{3,1} = c^t_{key,1}, c_{3,2} = c^t_{key,2} \cdot f^{s'}$, and sends $(c_{3,1}, c_{3,2})$ and $t_p = t \bmod p$ to $\mathcal{A}$.

In this case, the differences between the real protocol and the simulated one is the way $\mathcal{S}$ generates Q_2, R_2 and (c_3, t_p).

Since Q received from $\mathcal{F}_{\text{ECDSA}}$ is uniformly random, the distribution of $Q_2 = x_1^{-1} \cdot Q$ is uniform as well, which is identical to the real protocol. The analysis of R_2 is similar to Q_2.

We now show that the distributions of (c_3, t_p) in the two protocols are negligibly close. $\mathcal{S}$ does not abort in the key generation phase in this case, which means the *promise* Σ-protocol passes the verification and $h = \hat{h}^p, g_p = \hat{g_p}^p$, which implies $p \nmid \text{ord}(h), p \nmid \text{ord}(g_p)$.

Let $a = k_2^{-1} r x_2$, $b = k_2^{-1} m'$. The distribution of (c_3, t_p) in the real protocol is $\{t \leftarrow [0, pS) : (c^{a+t}_{key,1}, f^b \cdot c^{a+t}_{key,2}, t \bmod p)\}$, while in the simulated protocol, the distribution is $\{t \leftarrow [0, pS) : (c^t_{key,1}, f^{ax_1+b} \cdot c^t_{key,2}, t \bmod p)\}$. Since $p \nmid \text{ord}(h), p \nmid \text{ord}(g_p)$ and the promise Σ-protocol passes the verificiation, all the preconditions of Lemma 1 hold. Therefore, by Lemma 1 we have the above two distributions are negligibly close according to Lemma 1. This completes the proof of the case where $\mathcal{A}$ corrupts $\mathcal{P}_1$.

We now shows how to simulate the key generation and signing when $\mathcal{A}$ corrupts $\mathcal{P}_2$.

Simulating Key Generation Phase When $\mathcal{A}$ Corrupts $\mathcal{P}_2$:

1. Given input $\mathsf{KeyGen}(\mathbb{G}, G, p)$, the simulator $\mathcal{S}$ sends it to the trusted party computing $\mathcal{F}_{\text{ECDSA}}$ and receives a verification key Q.

2. $\mathcal{S}$ invokes $\mathcal{A}$ on input $\mathsf{KeyGen}(\mathbb{G}, G, p)$ and sends $(\mathsf{proof\text{-}receipt}, 1)$ to $\mathcal{A}$ as if sent by $\mathcal{F}_{\text{com-zk}}^{\mathcal{R}_{\text{DL}}}$.
3. $\mathcal{S}$ receives $(\mathsf{prove}, 2, Q_2, x_2)$ as $\mathcal{A}$ intends to send to $\mathcal{F}_{\text{zk}}^{\mathcal{R}_{\text{DL}}}$. $\mathcal{S}$ checkes if $Q_2 = x_2 \cdot G$. If so, computes $Q_1 = x_2^{-1} \cdot Q$, and sends $(\mathsf{decom\text{-}proof}, 1, Q_1, 1)$ to $\mathcal{A}$ as if sent by $\mathcal{F}_{\text{com-zk}}^{\mathcal{R}_{\text{DL}}}$; otherwise $\mathcal{S}$ simulates $\mathcal{P}_1$ aborting.
4. $\mathcal{S}$ runs $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{CL.KGen}(1^\lambda)$, where $\mathsf{pk}' = (\tilde{s}, p, \hat{g}_p, \hat{h}, f), \mathsf{sk}' = x$. $\mathcal{S}$ computes $h = \hat{h}^p, g_p = \hat{g}_p^p$, sets $\mathsf{pk} = (\tilde{s}, p, g_p, h, f), \mathsf{sk} = \mathsf{sk}'$, and sends $(\mathsf{pk}', \mathsf{pk})$ to $\mathcal{A}$.
5. $\mathcal{S}$ chooses $\tilde{x}_1 \leftarrow \mathbb{Z}_p, \tilde{r}_1 \leftarrow [0, S]$, computes $c_{key} \leftarrow \mathsf{CL.Enc}_{\mathsf{pk}}(\tilde{x}_1; \tilde{r}_1)$. Then $\mathcal{S}$ runs the simulator of Σ_{prom}^1 on the (invalid) instance $((G, Q_1), \mathsf{pk}, c_{key})$ to generate a proof of euqality of messages.
6. $\mathcal{S}$ sends $\mathsf{continue}$ to $\mathcal{F}_{\text{ECDSA}}$ for $\mathcal{P}_1$ to receive ouput, and stores (x_2, Q).

Simulating Signing Phase When $\mathcal{A}$ Corrupts $\mathcal{P}_2$:

1. On input $\mathsf{Sign}(sid, m)$, the simulator $\mathcal{S}$ sends it to $\mathcal{F}_{\text{ECDSA}}$ and receives a signature (r, s) on the message m.
2. $\mathcal{S}$ computes $R = (r, r_y)$ with the ECDSA verification algorithm.
3. $\mathcal{S}$ invokes $\mathcal{A}$ with input $\mathsf{Sign}(sid, m)$, and sends $(\mathsf{proof\text{-}receipt}, sid||1)$ to $\mathcal{A}$ as if sent by $\mathcal{F}_{\text{com-zk}}^{\mathcal{R}_{\text{DL}}}$.
4. $\mathcal{S}$ receives a query $(\mathsf{prove}, sid||2, R_2, k_2)$ to $\mathcal{F}_{\text{zk}}^{\mathcal{R}_{\text{DL}}}$ by $\mathcal{A}$. $\mathcal{S}$ verifies that $R_2 = k_2 \cdot G$. If so, it computes $R_1 = k_2^{-1} \cdot R$, and sends $(\mathsf{decom\text{-}proof}, sid||1, R_1, 1)$ to $\mathcal{A}$ as it expects to receive from $\mathcal{F}_{\text{com-zk}}^{\mathcal{R}_{\text{DL}}}$; otherwise, $\mathcal{S}$ simulates $\mathcal{P}_1$ aborting.
5. $\mathcal{S}$ receives (c_3, t_p) from $\mathcal{A}$, and decrypts c_3 to get $s' \in \mathbb{Z}_p$ using sk_2. $\mathcal{S}$ checks whether $s' - \tilde{x}_1 t_p = k_2^{-1}(m' + r x_2 \tilde{x}_1) \bmod p$. If so $\mathcal{S}$ sends $\mathsf{continue}$ to the trusted party computing $\mathcal{F}_{\text{ECDSA}}$ such that $\mathcal{P}_1$ receives the output; otherwise it sends abort to $\mathcal{F}_{\text{ECDSA}}$.

We prove the security in this case by constructing a sequence of hybrids, where the first hybrid is the real execution and the last one is the simulation. We show the views of $\mathcal{A}$ in adjacent hybrids are indistinguishable, which implies the real execution is indistinguishable from the simulated one.

Hybrid$_0$: This is the real execution of the protocol.

Hybrid$_1$: This hybrid is the same as **Hybrid$_0$**, except that in the key generation phase, the secret key $\mathsf{sk} = x$ is used to encrypt x_1 instead of using the public key h. More specifically, c_{key} is computed as $(u, u^x \cdot f^{x_1})$, where $u = g_p^{r_1}, r_1 \leftarrow [0, S]$.

It is easy to know that the views in **Hybrid$_0$** and **Hybrid$_1$** are identical.

Hybrid$_2$: This hybrid is the same as **Hybrid$_1$**, except that $\mathcal{P}_1$ runs the simulator for Σ_{prom}^1 protocol to generate the proof of euqality of messages. Due to (honest-verifier) zero-knowledge property of *promise* Σ-protocol[7], we have that $\mathbf{Hybrid}_2$ is indistinguishable from $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_3$**:** This hybrid is the same as $\mathbf{Hybrid}_2$, except that in the key generation phase, c_{key} is generated as $(u, u^x \cdot f^{x_1})$, where $u = g^{r_1}, r_1 \leftarrow \mathcal{D}$.

There is a minor change of the public key of CL encryption in the protocol. h and g_p are computed as $h = \hat{h}^p$ and $g_p = \hat{g}_p^p$, respectively. Where $\hat{g}_p^p$ and $\hat{h}$ are generated from $\mathsf{Gen}(1^\lambda, p)$. Note that since $\gcd(p, \mathrm{ord}(\mathcal{G}^p)) = 1$, then g_p is a generator of $\mathcal{G}^p$. Due to the HSM assumption, we have that $\mathbf{Hybrid}_3$ is indistinguishable from $\mathbf{Hybrid}_2$.

$\mathbf{Hybrid}_4$**:** This hybrid is the same as $\mathbf{Hybrid}_3$, except that it switches to the ideal world as follows: 1) Q and R are received from $\mathcal{F}_{\mathrm{ECDSA}}$; 2) x_2 and k_2 are extracted from $(\mathsf{prove}, 2, Q_2, x_2)$ and $(\mathsf{prove}, sid||2, R_2, k_2)$, respectively, 3) $\mathcal{S}$ computes c_{key} by encrypting a fresh random message $\tilde{x}_1 \leftarrow \mathbb{Z}_p$ using CL encryption scheme; 4) In the last step of signing, $\mathcal{S}$ checks both a) $s'' = k_2^{-1}(m' + r\tilde{x}_1 x_2)$ and b) $(s'' k_2) \cdot P = m' \cdot P + r \cdot Q$. If both fail, $\mathcal{S}$ aborts. Otherwise, returns (r, s).

Note that the differences of 1) and 2) do not change the distribution of $\mathbf{Hybrid}_3$. Observing that the CL encryption scheme used in our protocol is exactly the same as the one based on Hash Proof System described in [CCL+19] [8], in other words, our CL encryption scheme can be viewed as being derived from the projective hash family in [CCL+19]. Thus, one can verify the indistinguishability of the two ciphertexts c_{key} in $\mathbf{Hybrid}_4$ and $\mathbf{Hybrid}_3$ using the same reasoning as in the proof of Game_2 to Game_3 in Theorem 1 of [CCL+19]. As stated in [CCL+19], the extra check b) ensures that if a PPT adversary tells apart these two hybrids by causing one to abort while the other does not, then it can be used to break the double encoding assumption. Due to the space limit, we do not repeat the proof and refer to [CCL+19] for more details. We conclude the indistinguishability of $\mathbf{Hybrid}_4$ and $\mathbf{Hybrid}_3$.

$\mathbf{Hybrid}_5$**:** This hybrid is the same as $\mathbf{Hybrid}_4$, except that $\mathcal{S}$ does not check b) in the last step of signing.

These two hybrids differ if and only if check a) fails in both of them, while check b) passes. If this happens $\mathcal{S}$ has decrypted c_3 to the value $s'' = k_2^{-1}(m' + r x_1 x_2)$. Since $\mathcal{S}$ has extracted k_2, x_2, receives r from $\mathcal{F}_{\mathrm{ECDSA}}$ and knows m', $\mathcal{S}$ can compute x_1 from s'', thereby computing the discrete logarithm of point Q. Thus distinguishing these two hybrids reduces to the hardness of breaking the DL problem in $\mathbb{G}$. We have that $\mathbf{Hybrid}_5$ is indistinguishable from $\mathbf{Hybrid}_4$.

$\mathbf{Hybrid}_6$**:** This game is the same as $\mathbf{Hybrid}_5$, except that in the key generation phase, c_{key} is computed as $(u, u^x \cdot f^{\tilde{x}_1})$, where $u = g_p^{r_1}, r_1 \leftarrow [0, S]$.

[7] As mentioned before, one can obtain a *promise* NIZK via the Fiat-Shamir transformation and achieve zero knowledge in the random oracle model.

[8] When the public key of a CL encryption scheme generated by an honest $\mathcal{P}_1$, the distribution of CL public key pk_2 in our protocol is identical to that of CL public keys in [CCL+19] since $p \nmid \mathrm{ord}(\mathcal{G}^p)$.

The analysis is the same as the proof of the indistinguishability of **Hybrid**$_2$ and **Hybrid**$_3$, and we have that **Hybrid**$_6$ is indistinguishable from **Hybrid**$_5$.

Hybrid$_7$**:** This game is the same as **Hybrid**$_6$, except that in the key generation phase c_{key} is generated in an honest way using the public key. The same proof of the indistinguishability of **Hybrid**$_0$ and **Hybrid**$_1$ applies here, and we have that **Hybrid**$_7$ is indistinguishable from **Hybrid**$_6$.

Putting the above together, we conclude Theorem 4.

C Proof of Theorem 5

Proof. Observe that simulating the key generation phase can be done in a similar way to the work of [CCL+20], since the changes we made have no impact on the security of the protocol. Here we just give a simulator $\mathcal{S}$ for the signing phase.

Simulating $\mathcal{P}_1$ in Signing

1. As in a real execution, $\mathcal{S}$ samples $k_1, \lambda_1 \leftarrow \mathbb{Z}_p$. It computes $C_{k_1} \leftarrow \mathsf{EG.Enc}_{\mathsf{pk}'_1}(k_1), c_{k_1} \leftarrow \mathsf{CL.Enc}_{\mathsf{pk}_1}(k_1)$, and $(\mathsf{c}_1, \mathsf{d}_1) \leftarrow \mathsf{Com}(\gamma_1 G)$, then generates a *promise* Σ-proof π_1 for the equality of plaintexts of C_{k_1}, c_{k_1}. It broadcasts $(\mathsf{c}_1, C_{k_1}, c_{k_1}, \pi_1)$ before receiving $\{\mathsf{c}_j, C_{k_j}, c_{k_j}, \pi_j\}_{j\in S, j\neq 1}$ from $\mathcal{A}$. $\mathcal{S}$ checks the proofs are valid and extracts the encrypted values $\{k_j\}_{j\in S, j\neq 1}$ and computes $\tilde{k} = \Sigma_{i\in S} k_i$ using $\mathsf{Ext}^{\mathcal{A}}(C_{k_j}, c_{k_j})$.
2. (a) For $j \in S, j \neq 1$, $\mathcal{S}$ computes $\beta_{j,1}, \hat{t}'_{j,1}, c_{k_j\gamma_1}$ as in a real execution of the protocol, it samples a random $\mu_{j,1} \leftarrow \mathbb{Z}_p$, $t'_{j,1} \leftarrow [0, pS)$ and sets $c_{k_j w_1} = c_{k_i} \otimes t'_{j,1} \oplus (1, f^{\mu_{j,1}})$, and $B_{j,1} = k_j \cdot W_1 - \mu_{j,1} \cdot G$. $\mathcal{S}$ sends $(c_{k_j\gamma_1}, \hat{t}'_{p,j1}, c_{k_j w_1}, t'_{p,j1}, B_{j,1})$ to $\mathcal{P}_j$, where $\hat{t}'_{p,j1} = \hat{t}'_{j,1} \bmod p, t'_{p,j1} = t'_{j,1} \bmod p$.
   (b) When it receives $(c_{k_1\gamma_j}, \hat{t}'_{p,j1}, c_{k_1 w_j}, t'_{p,j1}, B_{1,j})$ from $\mathcal{P}_j$, it decrypts as in a real execution of the protocol to obtain $\alpha_{1,j}$ and $\mu_{1,j}$.
   (c) $\mathcal{S}$ verifies that $\mu_{1,j} G + B_{1,j} = k_1 W_j$. If so, since $\mathcal{S}$ also knows k_1 and w_j, it computes $\nu_{1,j} = k_1 w_j - \mu_{1,j} \bmod p$.

   $\mathcal{S}$ computes $\delta_1 = k_1\gamma_1 + \Sigma_{k\neq 1}\alpha_{1,k} + \Sigma_{k\neq 1}\beta_{k,1}$. It can compute

$$\begin{aligned}\Sigma_{i>1}\sigma_i &= \Sigma_{i>1}(k_i w_i + \Sigma_{j\neq i}(\mu_{i,j} + \nu_{j,i}))\\ &= \Sigma_{i>1}\Sigma_{j\neq i}(\mu_{i,j} + \nu_{j,i}) + \Sigma_{i>1} k_i w_i\\ &= \Sigma_{i>1}(\mu_{i,1} + \nu_{1,i}) + \Sigma_{i>1, j>1} k_i w_j,\end{aligned}$$

   and chooses the random values $\mu_{i,1}$ and it compute all of the shares $\nu_{1,j} = k_1 w_j - \mu_{1,j} \bmod p$.
3. $\mathcal{S}$ broadcasts δ_1 and receives all the $\{\delta_j\}_{j\in S, j\neq 1}$ from $\mathcal{A}$. Let $\delta = \Sigma_{i\in S}\delta_i$.
4. (a) $\mathcal{S}$ broadcasts d_1 which decommits to Γ_1, and $\mathcal{A}$ reveals $\{\mathsf{d}_j\}_{j\in S, j\neq 1}$ which decommit to $\{\Gamma_j\}_{j\in S, j\neq 1}$.

(b) $\mathcal{S}$ proves knowledge of γ_1 s.t. $\Gamma_1 = \gamma_1 G$, and for $j \in S, j \neq 1$, receives the PoK of γ_j s.t. $\Gamma_j = \gamma_j G$. $\mathcal{S}$ extracts $\{\gamma_j\}_{j \in S, j \neq 1}$ and computes $\gamma = \Sigma_{i \in S} \gamma_i \bmod p$ and $k = \delta \cdot \gamma^{-1} \bmod p$.

If $k = \tilde{k} \bmod p$, $\mathcal{S}$ proceeds as follows:

(c) requests a signature (r, s) for m from its ECDSA signing oracle.

(d) computes $R = (r_x, r_y) = s^{-1}(\mathsf{H}(m) \cdot G + r \cdot Q) \in \mathbb{G}$ (note that $r = r_x \bmod p$).

(e) rewinds $\mathcal{A}$ to the decommitment step at 4.(a) and equivocates $\mathcal{P}_1$'s commitment to open to $\hat{\Gamma}_1 := \delta \cdot R - \Sigma_{i>1} \Gamma_i$. It also simulates the proof of knowledge of $\hat{\gamma}_1$ s.t. $\hat{\Gamma}_1 = \hat{\gamma}_1 G$. Note that $\delta^{-1}(\hat{\Gamma}_1 + \Sigma_{i>1} \Gamma_i) = R$.

Else if $k \neq \tilde{k} \bmod p$, then $\mathcal{S}$ proceeds as follows:

(c) computes $R = (r_x, r_y) = \delta^{-1}(\Sigma_{i \in S} \Gamma_i) = k \cdot G$ and $r = r_x \bmod p$

5. If $\tilde{k} = k$, now $\mathcal{S}$ knows $\Sigma_{j \in S, j \neq 1} s_j$ held by $\mathcal{A}$ since $s_j = k_j m + \sigma_j r$, and proceeds as follows:

(a) computes s_1 held by $\mathcal{P}_1$ as $s_1 = s - \Sigma_{j \in S, j \neq 1} s_j$

(b) continues the steps of phase 5 as in a real execution.

If $\tilde{k} = k$, $\mathcal{S}$ does the following

(a) sample a ramdom $\tilde{s}_1 \leftarrow \mathbb{Z}_p$

(b) sample $\ell_1, \rho_1 \leftarrow \mathbb{Z}_p$, compute $V_1 = s_1 R + \ell_1 G; A_1 = \rho_1 G; (\hat{\mathsf{c}}_1, \hat{\mathsf{d}}_1) \leftarrow \mathsf{Com}(V_1, A_1)$ and send $\hat{\mathsf{c}}_1$ to $\mathcal{A}$

(c) receive $\{\hat{\mathsf{c}}_j\}_{j \neq 1}$ and decommit by broadcasting $\hat{\mathsf{d}}_1$. Prove knowledge of (s_1, ℓ_1, ρ_1) s.t. $(V_1 = s_1 R + \ell_1 G) \wedge (A_1 = \rho_1 G)$.

(d) for $j \in S, j \neq 1$, $\mathcal{S}$ receive $\hat{\mathsf{d}}_j$ and the ZKPoK of (s_j, ℓ_j, ρ_j) s.t. $(V_j = s_j R + \ell_j G) \wedge (A_j = \rho_j G)$.

(e) compute $V = -\mathsf{H}(m)G - rQ + \Sigma_{i \in S} V_i, A = \Sigma_{i \in S} A_i, T_1 := \ell_1 A$ and sample a random $U_1 \leftarrow \mathbb{G}$.

(f) compute $(\tilde{\mathsf{c}}_1, \tilde{\mathsf{d}}_1) \leftarrow \mathsf{Com}(U_1, T_1)$ and send $\tilde{\mathsf{c}}_1$ to $\mathcal{A}$. Upon receiving $\{\tilde{\mathsf{c}}_j\}_{j \neq 1}$ from $\mathcal{A}$, broadcast $\tilde{\mathsf{d}}_1$ and receive the $\{\tilde{\mathsf{d}}_j\}_{j \neq 1}$.

(g) now since $\Sigma_{i \in S} T_i \neq \Sigma_{i \in S} U_i$ both $\mathcal{A}$ and $\mathcal{S}$ abort.

To argue our protocol remains secure, we first construct $\tilde{\mathcal{P}}_1$ which is statistically indistinguishable from real $\mathcal{P}_1$, and then prove that $\mathcal{S}$ is indistinguishable from $\tilde{\mathcal{P}}_1$. We construct $\tilde{\mathcal{P}}_1$ as follows.

$\tilde{\mathcal{P}}_1$ acts the same way as $\mathcal{P}_1$ except that in **Phase** 2, for every j, instead of choosing a random $\nu_{j,1}$ and generating $c_{k_j w_1}, B_{j,1}$, $\tilde{\mathcal{P}}_1$ does the following:

1. Compute $k_j \leftarrow \mathsf{Ext}^{\mathcal{A}}(C_{k_j}, c_{k_j})$.
2. Sample a random $\mu_{j,1} \leftarrow \mathbb{Z}_p$.
3. Set $c_{k_j w_1} = c_{k_j} \otimes t_{j,1} \oplus (1, f^{\mu_{j,1}})$ where $t_{j,1} \leftarrow [0, pS)$, $B_{j,1} = k_j \cdot W_1 - \mu_{j,1} \cdot G$ and $\nu_{j,1} = k_j w_1 - \mu_{j,1}$.

In $\mathcal{P}_1$, we have $c_{k_j w_1} = c_{k_j} \otimes (w_1 + t_{j,1}) \oplus (1, f^{-\nu_{j,1}}), B_{j,1} = \nu_{j,1} \cdot G$, and $\{\nu_{j,1}\}$ is a uniform distribution over $\mathbb{Z}_p$, while in $\tilde{\mathcal{P}}_1$, we have $c_{k_j w_1} = c_{k_j} \otimes$

$t_{j,1} \oplus (1, f^{w_1 k_j - v_{j,1}}), B_{j,1} = v_{j,1} \cdot G$. Note that $\{\nu_{j,1} = k_j w_1 - \mu_{j,1}\}$ is a uniform distribution over $\mathbb{Z}_p$, since $\mu_{j,1}$ is sampled from $\mathbb{Z}_p$ uniformly. It follows from Lemma 1 that the distribution $\{c_{k_j w_1}, B_{j,1}, t_{j,1} \bmod p, \nu_{j,1}\}$ of $\mathcal{P}_1$ is statistically close to the one of $\tilde{\mathcal{P}}_1$. Thus, $\tilde{\mathcal{P}}_1$ is statistically indistinguishable from real $\mathcal{P}_1$ for any malicious $\mathcal{A}$.

We distinguish three cases accoring to the result of the check $\tilde{k} = k$ in step 4 of the simulator:

Case 0: The adversary $\mathcal{A}$ aborts befor the check;
Case 1: k computed by $\mathcal{S}$ equals $\tilde{k}$;
Case 2: k computed by $\mathcal{S}$ does not equal $\tilde{k}$.

We now prove $\mathcal{S}$ is indistinguishable from $\tilde{\mathcal{P}}_1$ in each case.

Case 0. Note that $\tilde{\mathcal{P}}_1$ and $\mathcal{S}$ proceed exactly in the same way before the check, and thus $\tilde{\mathcal{P}}_1$ is indistinguishable from $\mathcal{S}$.

Case 1. We construct a series of hybrids to complete the proof in this case.

Hybrid$_0$: $\mathcal{S}_0$ acts the same way as $\tilde{\mathcal{P}}_1$.

Hybrid$_1$: $\mathcal{S}_1$ acts the same way as $\mathcal{S}_0$ except that it simulates the *promise* Σ-proof in **Phase** 1 and proofs of ZKPoK in **Phase** 4 and **Phase** 5 using corresponding simulators and equivocates the commitment c_1 to open to $\gamma_1 G$.

It follows from the hiding of the equivocable commitment and (HV)ZK of *promise* Σ-protocol and ZKPoK that **Hybrid$_1$** is indistinguishable from **Hybrid$_0$**.

Hybrid$_2$: $\mathcal{S}_2$ acts the same way as $\mathcal{S}_1$ except that it uses the secret key sk_1 instead of the public key pk_1 and r_1 to compute $c_{k_1} \leftarrow (u_1, u_1^{\mathsf{sk}_1} f^{k_1})$ where $u_1 = g_p^{r_1}$.

It is easy to know that **Hybrid$_2$** is identical to **Hybrid$_1$**.

Hybrid$_3$: $\mathcal{S}_3$ acts the same way as $\mathcal{S}_2$ except that it replaces the first element u_1 in c_{k_1} with $\bar{u}_1 = g_p^{r_1} f^{b_1} \in \mathcal{G} \backslash \mathcal{G}^p$ by sampling $r_1 \in \mathbb{Z}_s$ and $b_1 \in \mathbb{Z}_p$ at random.

From HSM assumption, we have that **Hybrid$_3$** is indistinguishable from **Hybrid$_2$**.

Hybrid$_4$: $\mathcal{S}_4$ acts the same way as $\mathcal{S}_3$ except that it generates $C_{k_1} \leftarrow \mathsf{EG.Enc}_{\mathsf{pk}_1'}(0^n)$, and $(\mathsf{c}_1, \mathsf{d}_1) \leftarrow \mathsf{Com}(G)$.

Observe that the secret key of $\mathsf{ElGamal}$ encryption is not used during the entire execution. It is easy to verify that the indistinguishability of **Hybrid$_4$** and **Hybrid$_3$** follows from the IND-CPA security of $\mathsf{ElGamal}$ encryption and the hiding property of the commitment.

Hybrid$_5$: $\mathcal{S}_5$ acts the same way as $\mathcal{S}_4$ except that when $\tilde{k} = k$, it gets a signature (r, s) from the signing oracle and follows the simulation stretagy as $\mathcal{S}$.

One can apply the same argument of indistinguishability of Game_2 and Game_3 in Lemma 4 of [CCL+20] and prove that, conditioning on $\tilde{k} = k$, $\mathbf{Hybrid}_5$ is statistically indistinguishable from $\mathbf{Hybrid}_4$.

$\mathbf{Hybrid}_6$: $\mathcal{S}_6$ acts the same way as $\mathcal{S}_5$ except that it replaces the first element $\bar{u}_1$ of c_{k_1} with $u_1 \in \mathcal{G}^p$, where $u_1 = g_p^{r_1}$, $r_1 \leftarrow [0, S]$.

From the HSM assumption, we have that $\mathbf{Hybrid}_6$ and $\mathbf{Hybrid}_5$ are indistinguishable.

$\mathbf{Hybrid}_7$: $\mathcal{S}_7$ acts the same way as $\mathcal{S}_6$ except that it uses the public key pk_1 instead of sk_1 to compute $c_{k_1} \leftarrow (g_p^{r_1}, h^{r_1} f^{k_1})$, where $r_1 \leftarrow [0, S]$.

It's easy to verify that $\mathbf{Hybrid}_7$ is identical to $\mathbf{Hybrid}_6$.

$\mathbf{Hybrid}_8$**:** $\mathcal{S}_8$ computes $C_{k_1} \leftarrow \mathsf{EG.Enc}_{\mathsf{pk}'_1}(k_1)$ and $(\mathsf{c}_1, \mathsf{d}_1) \leftarrow \mathsf{Com}(\gamma_1 G)$, which is the only difference with $\mathcal{S}_7$.

From the IND-CPA security of $\mathsf{ElGamal}$ encryption and the hiding of the commitment, we have that $\mathbf{Hybrid}_8$ is indistinguishable from $\mathbf{Hybrid}_7$.

$\mathbf{Hybrid}_9$**:** $\mathcal{S}_9$ is identical to $\mathcal{S}$. Note that the differences between $\mathbf{Hybrid}_9$ and $\mathbf{Hybrid}_8$ are: 1). In $\mathbf{Hybrid}_9$ $\mathcal{S}_9$ acts as an honest prover to generate *promise* Σ-proof in **Phase** 1 and zero knowledge proofs in step 4(b), while $\mathcal{S}_8$ invokes the corresponding simulators, and 2). $\mathcal{S}_9$ open c_1 in an honest way, while $\mathcal{S}_8$ equivocates it to open in step 4(a).

From the same reasoning for indistinguishability of $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$, it follows $\mathbf{Hybrid}_9$ and $\mathbf{Hybrid}_8$ are indistinguishable.

It remains to prove the following lemma to complete the proof for **Case 1**.

Case 2. We construct three new hybrids between the simulator and the adversary here. In $\mathbf{Hybrid}_0$, $\mathcal{S}_0$ acts the same way as $\tilde{\mathcal{P}}_1$; In $\mathbf{Hybrid}_1$, $\mathcal{S}_1$ acts the same way as $\mathcal{S}_0$ except that it chooses U_1 as a random group element; In $\mathbf{Hybrid}_2$, $\mathcal{S}_2$ is identical to $\mathcal{S}$. Following the reasoning similar to that in [CCL+20], one can prove these three hybrids are computationally indistinguishable under the DDH assumption.

In sum, if the adversary $\mathcal{A}$ could output a new signature to break the existential unforgeability of our threshold protocol, then we have an efficient algorithm $\mathcal{A}^{\mathcal{S}}$ that breaks the existential unforgeability of the standard ECDSA. This concludes Theorem 5.