Efficient Online-friendly Two-Party ECDSA Signature

Haiyang Xue*
State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences
The University of Hong Kong
haiyangxc@gmail.com

Man Ho Au
The University of Hong Kong
allenau@cs.hku.hk

Xiang Xie
Shanghai Key Laboratory of Privacy-Preserving Computation
xiexiang@matrixelements.com

Tsz Hon Yuen
The University of Hong Kong
thyuen@cs.hku.hk

Handong Cui
The University of Hong Kong
hdcui@cs.hku.hk

ABSTRACT

Two-party ECDSA signatures have received much attention due to their widespread deployment in cryptocurrencies. Depending on whether or not the message is required, we could divide two-party signing into two different phases, namely, offline and online. Ideally, the online phase should be made as lightweight as possible. At the same time, the cost of the offline phase should remain similar to that of a normal signature generation. However, the existing two-party protocols of ECDSA are not optimal: either their online phase requires decryption of a ciphertext, or their offline phase needs at least two executions of multiplicative-to-additive conversion which dominates the overall complexity. This paper proposes an online-friendly two-party ECDSA with a lightweight online phase and a single multiplicative-to-additive function in the offline phase. It is constructed by a novel design of a *re-sharing* of the secret key and a *linear sharing* of the nonce. Our scheme significantly improves previous protocols based on either oblivious transfer or homomorphic encryption. We implement our scheme and show that it outperforms prior online-friendly schemes (i.e., those have lightweight online cost) by a factor of roughly 2 to 9 in both communication and computation. Furthermore, our two-party scheme could be easily extended to the 2-out-of-n threshold ECDSA.

CCS CONCEPTS

• **Security and privacy → Digital signatures**; **Key management**.

KEYWORDS

ECDSA; threshold signature; two-party signature; blockchain; zero-knowledge proof

*This work was done while the author was in The Uiversity of Hong Kong.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484803

ACM Reference Format:
Haiyang Xue, Man Ho Au, Xiang Xie, Tsz Hon Yuen, and Handong Cui. 2021. Efficient Online-friendly Two-Party ECDSA Signature. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3460120.3484803

1 INTRODUCTION

Threshold digital signatures [13, 14] allow distributed signing among n parties such that a given message is signed if and only if at least $t+1$ of the participants agree to sign it. Threshold signature has attracted a lot of attention in the academic community and industry recently, possibly due to its applications in blockchain and cryptocurrencies. A decentralized key management framework of ECDSA [11], the most commonly used signature scheme in blockchain, is urgently required to protect the secret key and hence the cryptocurrency.

Threshold ECDSA. ECDSA [11] is the Elliptical Curve (EC) version of the Digital Signature Algorithm (DSA). The core part of ECDSA involves computing

$$s = k^{-1}(H(m) + xr) \tag{1}$$

where x is the secret key, k is the secret nonce, and r is the public nonce (i.e., the x-coordinate of $k \cdot P$ where P is the EC group generator). The main obstacle of constructing a threshold version of ECDSA is to compute, in a distributed way, s satisfying Equation 1, where k and x are secrets shared among the participants. In detail, given the shares of k and x, computing the shares of k^{-1} and $k^{-1}x$ are the most expensive part of the entire signing procedure. (We remark that, if not specified, the computation in this section is over $\mathbb{Z}_q$ where q is the EC group order.)

A large number of practical protocols aim to address this obstacle in recent years. Lindell [25] proposed a very efficient two-party signing protocol, which utilizes multiplicative shares of k, x, and the additively homomorphic encryption scheme Paillier. The resulting signing procedure only involves homomorphic operations and a decryption operation, and thus is practical. Other follow-up works such as [6, 8, 15, 16, 19, 26, 33] handle the above obstacles with the help of the *multiplicative-to-additive* functionality (denoted as MtA hereafter). Specifically, with a and b as inputs, the MtA functionality securely computes and outputs α and β such that $\alpha + \beta = ab$. Thus it helps two parties to transform shares of k, x into additive share of

k^{-1} and $k^{-1}x$, or to transform shares of k^{-1}, x into additive share of $k^{-1}x$ and computing $k \cdot P$. Then, each participant could easily compute the additive share of s from r and their shares of k^{-1} and $k^{-1}x$. After the reveal of additive share of s, the signing procedure simply sums them up and outputs the signature after verifying its correctness.

However, realizing the MtA functionality always involves complicated zero-knowledge proof systems. This makes MtA the heaviest part of threshold signature schemes. A typical threshold signing invokes the MtA functionality 2 to 4 times. Specifically, given additive shares of $k = k_1 + k_2$ and $x = x_1 + x_2$, two calls of MtA are required to compute shares of k^{-1}, and another two MtA are needed to compute additive shares of $k^{-1}x$. The same holds when given additive shares of k^{-1} and x to jointly compute $k \cdot P$ and additive share of $k^{-1}x$.

Online and Offline Signing. In real-world applications, it is extremely useful to divide an MPC protocol into *offline* and *online* phases. In the offline phase, the message-independent part of the protocol is computed. This is also known as pre-processing or pre-computation and can utilize idle CPU time. This phase could be run several times and in batches. Output of the offline phase is stored for use in the online phase when the message to be processed is known, This separation is commonly discussed in the literature. For example, general-purpose MPC protocols could run the offline phase to generate a batch of random Beaver triples [3] to be used later to handle multiplication gates in the online phase.

As for threshold signatures, several works [6, 15, 16] have already considered splitting the signing process of threshold ECDSA into offline and online phases. In threshold signature, verifying the final signature is inevitable as it allows the detection of malicious behaviors. Thus, we say that the online phase of a threshold signature is *optimal* if it is non-interactive and its cost is the same as a single signature verification. The scheme is said to be *online-friendly* if its online phase is optimal. Existing works of threshold ECDSA either have their online phase [7, 25, 33] requiring a decryption of an additively homomorphic encryption or their offline phase [6, 15, 16, 19, 26, 33] needing 2 to 4 calls of MtA functionality. Table 1 shows a summary of these works.

Starting from [27], several competitive two-party ECDSA protocols [7, 25, 33] utilize multiplicative sharing in combination with additively homomorphic encryption. Concretely, the signing key x and the nonce k are shared multiplicatively as x_1x_2 and k_1k_2. Two parties jointly compute $k^{-1}x$ from multiplicative shares with the additively homomorphic property of the underlying encryption scheme. In spite of Paillier encryption as in [25] or Castagnos and Laguillaumie (CL) encryption as in [7, 33], the final online phase of this method requires a transfer and the decryption of the ciphertext. Although the entire signing procedure is relatively practical, the online phase is still heavy and not optimal.

As shown in Table 1, previous protocols require either a relatively slow online computation, or 2 to 4 calls of MtA in the offline phase when the online phase is optimal. The sub-protocol MtA dominates the overall complexity since it is either computationally expensive (for those based on homomorphic encryption [8, 26]) or needs a very large amount of communication (for those based on oblivious transfer [15]). For example, a single call of Paillier-based MtA needs 14 Paillier encryption/exponentiations and that of OT-based MtA requires communication of approximately 90 KB of data.

Table 1: The offline/online cost of two-party (case of) ECDSA. Enc (resp. Dec) is an execution of Paillier/CL encryption (resp. decryption). "NI" indicates non-interactive online signing. The online phase of "Fast" schemes requires extra dozens of EC point multiplications besides the verification.

Schemes	x	k or k^{-1}	Offline	(NI) Online
[7, 25]	x_1x_2	k_1k_2	Enc	(✓) Dec
[15] 2-of-2	x_1x_2	k_1k_2	2MtA	(✓) optimal
[15] 2-of-n	$x_1 + x_2$	k_1k_2	3MtA	(✓) optimal
[26]	$x_1 + x_2$	$k_1 + k_2$	2MtA	(×) MtA
[8, 19]	$x_1 + x_2$	$k_1 + k_2$	4MtA	(×) Fast
[6, 16]	$x_1 + x_2$	$k_1 + k_2$	4MtA	(✓) optimal
2ECDSA	$x_1 + x_2$	$k_1(r_1 + k_2)$	1MtA	(✓) optimal

In this paper, we focus on the two-party ECDSA. Our goal is giving a construction to achieve (nearly) optimal online performance with minimal number of calls, i.e., a single call, to the MtA functionality.

1.1 Our Contribution

We propose an online-friendly two-party ECDSA, 2ECDSA, such that its online computation is nearly optimal and its offline phase just needs a single call of MtA.

(1) The online phase of our protocol is non-interactive and optimal. It only requires transmitting a single element, and its computation cost is dominated by the verification of the resulting signature. Our offline phase runs in three-pass with a single MtA, thus has significant improvement over [15] and the two-party cases of [6, 8, 16, 19, 26, 33]. Two novel techniques are developed, which may be of independent interest.
   (a) The *linear sharing* of the nonce $k = k_1(r_1 + k_2)$, where k_1, r_1 are chosen by party P_1 and k_2 is chosen by the other party, P_2. It is different from the existing simple additive or multiplicative sharing of the nonce.
   (b) The *re-sharing* of the secret x by using k_2 in the offline phase. New share x_1' (resp. x_2') of the signing key is chosen by P_1 (resp. P_2), such that $x = x_1'k_2 + x_2'$ (which is also a linear function).

   These techniques enable us to construct an online-friendly 2ECDSA with a single MtA. Details are given in Sec. 1.2.

(2) We provide an implementation of our two-party 2ECDSA protocol in Rust, with instantiation of the MtA functionality from Paillier encryption, CL encryption, and oblivious transfer. We give an efficiency comparison with all previous two-party ECDSA and the two-party case of threshold ECDSA. Details are shown in Table 2. On the premise of preserving fast online computation, our scheme reduces the offline cost of Paillier-based protocol to 226 ms and 6.3 KB, and further to 141 ms and 4.1 KB based on Paillier-EC assumption. For CL-based instantiation, the complexity of the

offline phase is 1386 ms and 1.7 KB. Based on oblivious transfer (OT), the offline cost of our scheme is 2.6 ms and 90.9 KB.

(3) Applying 2-out-of-n Shamir secret-sharing [30] to n-party additively shared secret during key generation, our two-party 2ECDSA could be easily extended to the 2-out-of-n ECDSA. The general scheme's signing phase has the same complexity as 2ECDSA.

1.2 Technical Overview

Recall that the main bottleneck of previous schemes is due to the use of complex protocols for the two parties to compute $k^{-1}(H(m)+rx)$ such that $x = x_1 + x_2$ and $k = k_1 + k_2$ (resp. $x = x_1x_2$ and $k = k_1k_2$) in the case of additive sharing (resp. multiplicative sharing). In other words, the bottleneck is inherent in the multiple executions of MtA for these shares.

Starting point. We start from a simple combination of multiplicative sharing of k (i.e. $k = k_1k_2$) and additive sharing of x (i.e. $x = x_1 + x_2$). This has been utilized by Doerner et al. [15] to handle 2-out-of-n threshold scheme, albeit, in a rather inefficient way. To jointly compute $s = H(m)/k_1k_2 + r(x_1+x_2)/k_1k_2$, they adopt three MtA to export the additive shares of $1/k_1 \cdot 1/k_2$, $x_1/k_1 \cdot 1/k_2$, and $1/k_1 \cdot x_2/k_2$ respectively. Thus, their two-party case of 2-out-of-n scheme is worse than their direct two-party scheme where only two MtA are required.

The first attempt: Re-sharing of the secret. We resolve this problem with a *re-sharing* of secret x using a share of nonce k_2. Concretely, in the offline phase the secret $x = x_1 + x_2$ is re-shared to x_1', x_2' using one MtA such that

$$x_1 + x_2 = x_1'k_2 + x_2'.$$

In the online phase, P_2 computes $s_2 = k_2^{-1}(H(m) + rx_2')$, and P_1 could derive the signature component s from s_2, k_1 and x_1' since

$$\begin{aligned} s &= k_1^{-1}(s_2 + rx_1') \\ &= k_1^{-1}k_2^{-1}\left[H(m) + r(x_2' + k_2x_1')\right] \\ &= k^{-1}(H(m) + rx). \end{aligned}$$

The offline phase re-shares the secret $x = x_1 + x_2$ into $x = x_1'k_2 + x_2'$, with a single MtA. Specifically, P_1 chooses a random $x_1' \leftarrow \mathbb{Z}_q$, and then P_1 and P_2 invoke MtA with input x_1' and k_2 respectively, and receive shares t_A and t_B such that $t_A + t_B = x_1'k_2$. Then P_1 masks x_1 with t_A and sends $cc = t_A - x_1$ to P_2. P_2 could extract its new share x_2' from cc, x_2, and t_B, since

$$x_2' = x_1 + x_2 - x_1'k_2 = (t_A - cc) + x_2 - x_1'k_2 = -t_B - cc + x_2.$$

The resulting scheme is online-friendly and requires a single MtA in the offline phase. Unfortunately, this solution is insecure and a malicious adversary may cheat.

Attack on the first attempt. We show that a malicious P_2 can obtain x_1 in the previous scheme. Observe that

$$x_1 = t_A - cc = x_1'k_2 - t_B - cc,$$

where k_2, t_B, cc are known to P_2. The malicious P_2 can set $k_2 = 0$ as the input of MtA and learn P_1's secret $x_1 = -t_B - cc$.

Our solution: Linear sharing of the nonce. A simple solution to rule out the attack of $k_2 = 0$ is to add a zero-knowledge proof of $k_2 \neq 0$ for P_2, but it would be quite expensive. Instead, we apply a re-randomization method to solve this problem. Now, the re-sharing of the secret is changed to $x = x_1'(k_2 + r_1) + x_2'$, where r_1 is chosen by P_1 and could be given to P_2. To achieve that, P_1 now masks x_1 with t_A and a random r_1 by setting $cc = t_A + r_1x_1' - x_1$. Now $x_1 = t_A + r_1x_1' - cc = x_1'(k_2 + r_1) - t_B - cc$. If r_1 is chosen by P_1 after k_2 is chosen by the (malicious) P_2, P_2 learns nothing about x_1 since $k_2 + r_1 = 0$ with probability at most $1/q$.

In order to support this change in the re-sharing of the secret, we need to view P_2's share of the nonce as (k_2+r_1). Interestingly, while we require r_1 to be chosen by P_1 (in order to withstand the above attack), $(k_2 + r_1)^{-1}$ can be computed by P_2 itself (otherwise, we need another round of MtA). Hence, we change the multiplicative sharing of the nonce k into a linear function $k_1(r_1 + k_2)$. In the offline phase, P_1 picks random r_1 and sends it to P_2. Later in the online phase, $(k_2 + r_1)^{-1}$ can be computed by P_2. This is the reason why we use a linear sharing of the nonce.

Additional consistency check is needed when setting up $k \cdot P$ by P_1 and P_2. Details are given in Sec. 3.

1.3 Extension and Instantiations

Extending to 2-out-of-n Access Structures. Our two-party 2ECDSA is presented using 2-out-of-2 additive share of the private key x. It could be easily extended to a 2-out-of-n protocol using Shamir secret-sharing, in a way similar to Gennaro and Goldfeder [19]. A similar approach has been described in [26] and [6]. We also let x be the additive share of each party's contribution x_i, i.e. $x = x_1 + \cdots + x_n$. It is natural to use a 2-out-of-n Shamir secret-sharing [30] to convert it into a 2-out-of-n shares of x in a verifiable manner.

Here, we give a brief overview. For details, please refer to Appendix A. Each P_i chooses a linear function f_i such that $f_i(0) = x_i$ and sends $x_i^{(j)} = f_i(j)$ to P_j for all $j \in [1, n]$. Then, every two parties P_i, P_j could recover x_k (for every $k \in [1, n]$) via interpolating from $x_k^{(i)}$ and $x_k^{(j)}$, i.e., $x_k = \Lambda_i x_k^{(i)} + \Lambda_j x_k^{(j)}$ where Λ_i, Λ_j are Lagrange coefficients. With Shamir's secret sharing, any two parties P_i, P_j could generate $\Lambda_i \sum_{k=1}^n x_k^{(i)}$ and $\Lambda_j \sum_{k=1}^n x_k^{(j)}$ such that $x = \Lambda_i \sum_{k=1}^n x_k^{(i)} + \Lambda_j \sum_{k=1}^n x_k^{(j)}$. They could further invoke the signing of 2ECDSA directly with $\Lambda_i \sum_{k=1}^n x_k^{(i)}$ and $\Lambda_j \sum_{k=1}^n x_k^{(j)}$ as their additive shares of x. The general 2-out-of-n ECDSA has the same signing protocol with 2ECDSA.

On the instantiations of MtA. Existing constructions of MtA from OT, Paillier encryption, and CL-encryption can be directly applied to our 2ECDSA.

Doerner et al. [15] proposed an OT-based MtA from Simplest OT [10] and KOS [23] OT-extension protocols. Their MtA is computationally efficient, while the communication cost is rather high (e.g., 90 KB for 128 bits security). Although improved method was applied, their two-party ECDSA requires a communication cost of at least 168 KB. Integrating the OT-based MtA into our scheme, we obtain the first OT-based two-party signature with communication cost less than 100 KB at 128-bit security.

Paillier-based MtA is first proposed in [27] and improved by [6, 19, 26]. The main issue when using Paillier cryptosystem for MtA is that it operates over $\mathbb{Z}_N$ rather than $\mathbb{Z}_q$, where q is the EC group order. As a result, expensive zero-knowledge range proofs are

Table 2: Comparison of signing with two-party protocols and two-party case of threshold ECDSA from Paillier, OT and CL respectively. These concrete numbers are based on computational security parameter $\lambda = 128$ and statistical security parameter 80. For those Paillier-based schemes, E represents a Paillier exponentiation over $\mathbb{Z}_{N^2}$ (we approximately count 3 operations of $\bmod N$ as one operation of $\bmod N^2$. Some of E are single "short" exponentiation), while for CL-based schemes E means an exponentiation over CL group. M refers to elliptic curve point multiplication which is almost free compared with E. In the communication column, EC points, CL group elements, and ring elements of Paillier are encoded by κ, 7κ (only for $\kappa = 256$), and $2\ell_N$ bits respectively. Estimating in parentheses includes the constant overhead with the standard security recommendation, i.e. $\kappa = 256$ and $\ell_N = 3072$. "Paillier-EC" means that Paillier-EC assumption is applied to eliminate a zero-knowledge proof.

Signing Protocols	Computation		Communication		Passes
	offline	online	offline	online	
LNR18 [26]	28E + 157M (461ms)	14E + 121M (302ms)	$32\ell_N + 67\kappa$ (12KB)	$16\ell_N + 51\kappa$ (6.6KB)	8
GG18 [19]	42E + 40M (1237ms)	17M (3ms)	$40\ell_N + 18\kappa$ (15.5KB)	9κ (288B)	9
CGGMP20 [6]	208E + 44M (2037ms)	2M (0.2ms)	$118\ell_N + 20\kappa$ (44KB)	κ (32B)	4
2ECDSA (Paillier)	14E + 11M (226ms)	2M (0.2ms)	$16\ell_N + 11\kappa$ (6.3KB)	κ (32B)	3
Lin17 [25] (Paillier-EC)	2E + 8M (34ms)	1E + 2M (8ms)	12κ (192B)	$2\ell_N$ (768B)	3
GG18 [19] (Paillier-EC)	18E + 40M (360ms)	17M (3ms)	$16\ell_N + 18\kappa$ (6.6KB)	9κ (288B)	9
2ECDSA (Paillier-EC)	8E + 14M (141ms)	2M (0.2ms)	$10\ell_N + 12\kappa$ (4.1KB)	κ (32B)	3
CCLST19 [7]	4E + 8M (475ms)	1E + 2M (190ms)	6κ (208B)	14κ (505B)	3
CCLST20 [8]	28E + 8M (3316ms)	17M (3ms)	140κ (4.5KB)	9κ (288B)	8
YCX21 [33]	28E + 8M (4550ms)	17M (3ms)	140κ (4.5KB)	9κ (288B)	8
2ECDSA (CL)	11E + 11M (1386ms)	2M (0.2ms)	53κ (1.7KB)	κ (32B)	3
DKLS18 [15]	13M (2.9ms)	2M (0.2ms)	$16\kappa^2$ (169.8KB)	κ (32B)	2
DKLS19 [16]	13M (3.7ms)	2M (0.2ms)	$20\kappa^2$ (180KB)	κ (32B)	7
2ECDSA (OT)	11M (2.6ms)	2M (0.2ms)	$8\kappa^2$ (90.9KB)	κ (32B)	3

required. There are several solutions to simply or even remove these proofs, such as range proof with slack [26], and a non-standard Paillier-EC assumption [19] (refer to Appendix D.3). Using range proof with slack and Paillier-EC assumption, a Paillier-based MtA requires 8 Paillier exponentiations and a transmission of 10 $\log N$ bits. Depending on Paillier-EC assumption is applied or not, we propose two Paillier-based schemes.

Castagnos et al. [7] replace Paillier encryption with Castagnos and Laguillaumie [9] encryption over class group. The key feature of CL-encryption is that it allows instantiations where the message space is exactly $\mathbb{Z}_q$. However, this kind of MtA requires new zero-knowledge proofs performed on unknown order groups, which is the heaviest part of all these constructions. Follow-up works [8, 33] further improve the underlying zero-knowledge proof system.

We note that there are other instantiations from noisy Reed-Solomon encodings (RS) [20] and Ring-LWE [2]. However, these constructions are not very suitable for the parameters related to ECDSA. We leave it as the future work to improve the underlying MtA protocol with these techniques.

1.4 Related Works and Discussion

1.4.1 Related works. Efficient constructions of threshold ECDSA fall into the following three categories. For more details, please refer to [1].

Paillier-based Schemes. Following [27], Lindell [25] proposed a competitive two-party ECDSA utilizing multiplicative sharing of secret and nonce in combination with Paillier encryption. Lindell's scheme has the best overall efficiency while its online phase needs to perform a Paillier decryption. Later, Lindell et al. [26], and Gennaro and Goldfeder [19] proposed a full threshold ECDSA protocol with additive shares. They both require at least 8 communication rounds and their online phase is interactive. Recently, Canetti et al. [6] proposed an online-friendly three-pass threshold ECDSA at the cost of extra overhead.

CL-based Schemes. Castagnos et al. [7] addressed the problem of relying on non-standard assumption in [25] by replacing Paillier encryption with CL-encryption [9] which allows the message space to match that of the signature space ($\mathbb{Z}_q$). Castagnos et al. [8] further extended their work to full threshold by following Gennaro and Goldfeder's blueprint. Very recently, Yuen et al. [33] improves the underlying zero-knowledge proof of Castagnos et al.'s protocol, thus reduces the overall bandwidth and running time. However, the online phase of these schemes either requires the computation of decryption or is interactive.

OT-based Schemes. The OT-based schemes are online-friendly and do not require extra assumptions. Doerner et al. [15] used multiplicative sharing of the signing key and the nonce as [7, 25], and achieved fast online computation with the help of two MtA

from the oblivious transfer. They also generalized their work to 2-of-n cases. Later, they [16] proposed a full threshold scheme. These schemes are very fast in signing time, while the communication cost is the bottleneck.

1.4.2 Discussion. We further discuss issues related to round complexity and which party obtain the final signature in existing approaches.

On the Communication Rounds of Signing. As shown in Table 2, our scheme and [7, 25] require only 3 rounds (i.e., passes). All the other protocols except [15] require more communication rounds. However, as mentioned by [15, Sec. III], it relied on the generic group model (GGM) [31] to achieve this round reduction. Otherwise, 4 rounds are needed.

On the Generation of Final Signature. There are two commonly used syntax regarding the output of the protocol, namely, 1) each party generates a "signature share" and anyone (including a third party) could compute the final signature from the signature shares; 2) only one party obtains the final signature. Notable examples of the former include [6, 8, 16, 19, 26], and the latter include [7, 15, 25] as well as our scheme. The obstacle in adding this property to our scheme and [7, 15, 25] is that the nonce is shared multiplicatively, and thus signature component "s" cannot be reconstructed from a simple linear combination of signature shares.

1.5 Paper Organization.

The rest of paper is organised as follows. We review preliminaries in section 2. Then we propose our protocol and prove its security in section 3. In section 4, we show several instantiations of 2ECDSA. Finally, section 5 presents a comprehensive analysis and comparison with existing schemes.

2 PRELIMINARY

2.1 The ECDSA Signature

Let $\mathbb{G}$ be an elliptic curve group of order q with base point (generator) P. The algorithm makes use of the hash function H. Curve coordinates and scalars are represented in $\kappa = \log q$ bits. The ECDSA scheme works as follows [11].

(1) Keygen(1^κ): on input 1^κ
   - Choose a random $x \leftarrow \mathbb{Z}_q$, set x as the private key.
   - Compute $Q = x \cdot P$, and set Q as the public key.

(2) Sign(x, m): on input sign key x and message m
   - Choose a random $k \leftarrow \mathbb{Z}_q$, compute $R = (r_x, r_y) = k \cdot P$.
   - Compute $r = r_x \mod q$ and $s = k^{-1}(H(m) + rx) \mod q$.
   - Output (r, s) as the signature.

(3) Verify($m; (r, s)$) calculates $(r_x, r_y) = R = s^{-1}H(m) \cdot P + s^{-1}r \cdot Q$ and outputs 1 if and only if $r = r_x \mod q$.

It is well known that for every valid signature (r, s), the pair $(r, -s)$ is also a valid signature. To make (r, s) unique, in this paper, we mandate that the "smaller" of $\{s, -s\}$ is the output.

2.2 Ideal Functionality for Two-Party ECDSA

The ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}$ for two-party ECDSA is shown in Figure 1. It consists of two functions, namely, a key generation function Keygen, called once, and a signing function Sign, called an arbitrary number of times under the generated key.

Consider an elliptic curve group $\mathbb{G}$ of order q with generator P, then:

Keygen: On receiving Keygen($\mathbb{G}, P, q$) from P_1 and P_2
- Generate key pair (Q, x) where $x \leftarrow \mathbb{Z}_q$ and $Q = x \cdot P$.
- Choose a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$.
- Send Q and H to P_1 and P_2.
- Store $(\mathbb{G}, P, q, H, x, Q)$ and ignore further calls.

Sign: On receiving Sign(sid, m) from both P_1 and P_2, where keys have been generated from a call to Keygen and sid has not been used
- Choose $k \leftarrow \mathbb{Z}_q$ and compute $R = (r_x, r_y) = k \cdot P$.
- Compute $r = r_x \mod q$ and $s = k^{-1}(H(m) + rx) \mod q$.
- Send (r, s) to both P_1 and P_2.
- Store (Complete, sid) in the memory.

Figure 1: The functionality $\mathcal{F}_{\mathsf{ECDSA}}$ for two-party ECDSA signature.

Setup: On receiving (setup) from P_1 and P_2
- Store and send (setup-complete) to P_1 and P_2.

Multiplication: On receiving (input, sid, $a \in \mathbb{Z}_q$) from P_1, (input, sid, $b \in \mathbb{Z}_q$) from P_2 where sid has not been used, if (setup-complete) exists,
- Sample $\alpha \in \mathbb{Z}_q$ and compute $\beta = ab - \alpha \mod q$.
- Send (output-1, sid, α) to P_1
- Send (output-2, sid, β) to P_2.

Figure 2: The functionality $\mathcal{F}_{\mathsf{MtA}}$ of multiplicative-to-additive protocol.

2.3 The Multiplicative-to-Additive (MtA) Functionality

The $\mathcal{F}_{\mathsf{MtA}}$ functionality, listed in Figure 2, is parameterized by the group order q. It runs with two parties, P_1 and P_2, who may participate in the Setup phase once, and the Multiplication phases as many times as they wish. $\mathcal{F}_{\mathsf{MtA}}$ runs and outputs α, β, from two parties inputs, respectively, a and b, under the restriction that $\alpha + \beta = ab \mod q$.

It could be instantiated from OT [15], Paillier encryption [19, 26], CL encryption [8], etc. Please refer to Sec. 4 for more details.

2.4 Zero-Knowledge Proof

Let $\mathcal{R}$ be a polynomial-time-decidable binary relation. The corresponding language L consists of statement x such that there exists witness w and $(x, w) \in \mathcal{R}$. We specify L as an NP language.

An interactive proof consists of an interactive prover algorithm P and a verifier algorithm V that runs in PPT time. We call (P, V) an interactive proof for relation $\mathcal{R}$ if it has the completeness and soundness properties. Completeness means that for every $x \in L$, $\langle P, V \rangle(x)$ is always 1. Soundness means that for every $x \notin L$ and every prover P^*, $\Pr[\langle P^*, V \rangle(x) = 1]$ is negligible. When the soundness holds for computationally bounded provers, the system is usually

called an "argument". In this paper, both proof and argument are collectively referred to as proof.

Definition 2.1 (zero-knowledge). Let (P, V) be an interactive proof for some language L. (P, V) is zero knowledge if for every PPT verifier V^* there exists a PPT simulator Sim such that the two ensembles $\{View^P_{V^*}(x)\}_{x\in L}$ and $\{\mathsf{Sim}(x)\}_{x\in L}$ are identical.

We could also define statistical (resp. computational) zero-knowledge, if the two ensembles are statistically (resp. computationally) indistinguishable.

Definition 2.2 (proof-of-knowledge). Let $\zeta : \{0,1\}^* \to [0,1]$ be a function. (P, V) is a proof of knowledge for relation $\mathcal{R}$ with knowledge error ζ if the following properties are satisfied:

- *Completeness:* If P and V follow the protocol on input x and private input w to P where $(x, w) \in \mathcal{R}$, then V always accepts.
- *Knowledge Soundness:* there exists a probabilistic oracle machine Ext such that for every prover function P^* and every $x \in L$, Ext satisfies the following: Denote $\epsilon(x)$ the probability that V accepts on input x after interacting with P^*. If $\epsilon(x) > \zeta(x)$, on input x with access to P^*, Ext runs in expected polynomial time and outputs a string w such that $(x, w) \in \mathcal{R}$ with probability at least $\epsilon(x) - \zeta(x)$.

We remark that every zero-knowledge proof of knowledge (ZKPoK) in this paper is transformed to non-interactive [4] using the Fiat-Shamir paradigm [18] in the random oracle model.

ZKPoK of Discrete Logarithm. Define the relation

$$\mathcal{R}_{DL} := \{\big((R, P), x\big) | R, P \in \mathbb{G}, R = x \cdot P\},$$

where the parameters of elliptic curve $(\mathbb{G}; P; q)$ are implicit public parameters. We use the standard Schnorr proof [29] for $\mathcal{R}_{DL}$, apply the Fiat-Shamir [18] transformation to get a non-interactive ZKPoK. Denote by nizkPoK the proof generator and Verifzk the verify algorithm for $\mathcal{R}_{DL}$.

3 TWO-PARTY SIGNATURES FOR ECDSA

In this section, we present a two-party protocol 2ECDSA. We first describe the distributed key generation phase that is executed once, followed by the signing phase which may run multiple times. The process is also illustrated in Fig. 3.

Let MtA be the multiplicative-to-additive functionality, nizkPoK be the NIZK proof for discrete logarithm relation $\mathcal{R}_{DL}$. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ be a hash function.

Although we make a logical presentation of the signing procedure in four phases, in the actual protocol they are intertwined. We could reorder the messages in the instantiations such that messages of MtA from P_2 to P_1 come first, followed by all messages from P_1 to P_2, which results in a 3-pass signing protocol.

Distributed Key Generation Phase Keygen$(\mathbb{G}, P, q)$**.**
Given the joint input $(\mathbb{G}, P, q)$ and security parameter λ:

(1) P_1**'s first message**:
- P_1 chooses a random $x_1 \leftarrow \mathbb{Z}_q$, and computes $Q_1 = x_1 \cdot P$ and nizk1 = nizkPoK(Q_1, x_1)
- P_1 sends f1 = H$(Q_1,$ nizk1$)$ to P_2

(2) P_2**'s first message**:
- P_2 chooses a random $x_2 \leftarrow \mathbb{Z}_q$, and computes $Q_2 = x_2 \cdot P$ and nizk2 = nizkPoK(Q_2, x_2)
- P_2 sends Q_2, nizk2 to P_1

(3) P_1**'s second message**:
- On receiving Q_2, nizk2 from P_2, P_1 verifies nizk2. If Verifzk(nizk2)=0, abort
- Else, P_1 sends Q_1, nizk1 to P_2

(4) P_2**'s verification**:
- On receiving Q_1, nizk1 from P_1, P_2 verifies nizk1.
- If f1 $\neq$ H$(Q_1,$ nizk1$)$ or Verifzk(nizk1)=0, abort.

(5) **Compute output:**
- P_1 computes $Q = Q_1 + Q_2$, stores (Q, x_1, Q_1, Q_2).
- P_2 computes $Q = Q_2 + Q_1$, stores (Q, x_2, Q_1, Q_2).

Distributed Sign Phase Sign(sid, m). They begin with the session id sid, m the message to be signed, and additive share of secret key. The protocol is divided into 4 logical steps, and only the last one is online.

- P_1 has (Q, x_1, Q_1, Q_2) as the output of Keygen, message m and session id sid.
- P_2 has (Q, x_2, Q_1, Q_2) as the output of Keygen, message m and session id sid.

(1) **Commitment of** P_2**'s nonce**:
- P_2 chooses a random $k_2 \leftarrow \mathbb{Z}_q$, and computes $R_2 = k_2 \cdot P$ with nizk3 = nizkPoK(R_2, k_2)
- P_2 computes and sends f2 = H$(R_2,$ nizk3$)$ to P_1.

(2) **MtA and Consistency Check**:
- P_1 chooses a random $x_1' \leftarrow \mathbb{Z}_q$, and computes $Q_1' = x_1' \cdot P$.
- P_1 and P_2 invoke the MtA functionality with input x_1' and k_2 respectively and receives t_A, t_B such that

$$t_A + t_B = x_1' k_2 \mod q.$$

- P_1 chooses a random $r_1 \leftarrow \mathbb{Z}_q$, and computes $cc := t_A + x_1' r_1 - x_1 \mod q$ and sends (r_1, cc) to P_2.
- P_2 checks the consistency by checking

$$(t_B + cc) \cdot P \stackrel{?}{=} (r_1 + k_2) \cdot Q_1' - Q_1.$$

- if the consistency check passes, P_2 computes

$$x_2' = x_2 - (t_B + cc) \mod q.$$

(3) **Nonce Key Exchange**:
- P_1 chooses a random $k_1 \leftarrow \mathbb{Z}_q$, computes and sends $R_1 = k_1 \cdot P$ with nizk4 = nizkPoK(R_1, k_1) to P_2.
- P_2 aborts if Verifzk(nizk4)=0, otherwise sends $(R_2,$ nizk3$)$ to P_1 and computes $R = (r_x, r_y) = (k_2 + r_1) \cdot R_1$, and $r = r_x \mod q$.
- P_1 aborts if f2 $\neq$ H$(R_2,$ nizk3$)$ or Verifzk(nizk3)=0, otherwise computes $R = (r_x, r_y) = k_1 \cdot R_2 + k_1 r_1 \cdot P$ and $r = r_x \mod q$.

(4) **Online Signature**:
- Given m, P_2 computes $h = H(m)$ and sends

$$s_2 = (k_2 + r_1)^{-1}(h + r x_2') \mod q$$

to P_1.
- On receiving s_2, P_1 computes

$$s = k_1^{-1}(s_2 + r x_1') \mod q.$$

- P_1 aborts if $\mathsf{Verify}(m;(r,s))=0$, else returns (r,s) as the final signature.

Correctness. By the definition of x_2', we have

$$\begin{aligned} x_2' &= x_2 - (t_B + cc) \\ &= x_2 - (t_B + t_A + x_1' r_1 - x_1) \\ &= x_2 - (x_1' k_2 + x_1' r_1) + x_1 \end{aligned}$$

over $\mathbb{Z}_q$, thus $x_1'(r_1+k_2)+x_2' = x_1 + x_2 \mod q$.

Let $k := k_1(r_1+k_2) \mod q$, then $R = k \cdot P$. We have

$$\begin{aligned} s &= k_1^{-1}(s_2 + r x_1') \\ &= k_1^{-1}\left[(r_1+k_2)^{-1}(H(m)+rx_2') + rx_1'\right] \\ &= k_1^{-1}(r_1+k_2)^{-1}\left[H(m) + r(x_2' + x_1' r_1 + x_1' k_2)\right] \\ &= k_1^{-1}(r_1+k_2)^{-1}\left[H(m) + r(x_1+x_2)\right] \\ &= k_1^{-1}(r_1+k_2)^{-1}(H(m)+rx) \end{aligned}$$

over $\mathbb{Z}_q$. Thus, (r,s) is a valid signature of m.

3.1 Security of 2ECDSA

Theorem 3.1. *The two-party 2ECDSA protocol in Figure 3 securely computes $\mathcal{F}_{ECDSA}$ in the random oracle model in the presence of a malicious static adversary under the real/ideal definition. Concretely, there exists a simulator for the scheme such that any probabilistic polynomial time adversary, who corrupted P_1 or P_2, can distinguish a real execution of the protocol from a simulated one with only negligible probability.*

We present a sketch of the proof here. Please refer to Appendix B for the full proof.

Simulator $\mathcal{S}$ could only access an ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}$ for computing ECDSA signatures. All $\mathcal{S}$ learns in the ideal world is the public key Q generated in the key generation phase and several signatures (r,s) for messages m of its choice in the signing phase. In the real world, the adversary, having either corrupted P_1 or P_2 will also see all the interactions with the non-corrupted party. Thus $\mathcal{S}$ must be able to simulate the adversary's view of these interactions, while only knowing the expected output.

The proof proceeds in two cases: the adversary corrupts P_1, and the adversary corrupts P_2.

$\mathcal{S}$ simulates P_2- Corrupted P_1. In the key generation phase, after receiving public key Q from $\mathcal{F}_{\mathsf{ECDSA}}$ and receiving f1 from $\mathcal{A}$, $\mathcal{S}$ could extract Q_1, nizk1 such that $\mathsf{f1} = \mathsf{H}(Q_1, \mathsf{nizk1})$ in the random oracle model with overwhelming probability. Furthermore, if nizk1 is accepted, there exists knowledge extractor to successfully output x_1. Then, $\mathcal{S}$ computes $Q_2 = Q - Q_1$ and generates nizk2 by querying the zero knowledge simulator Sim. The indistinguishability between the simulated and real key generation is obvious.

During the signing phase, $\mathcal{S}$ could receive a signature (r,s) of message m from $\mathcal{F}_{\mathsf{ECDSA}}$, and recover R via verification algorithm.

(1) $\mathcal{S}$ invokes $\mathcal{A}$ with a totally random f2.
(2) $\mathcal{S}$ interacts with $\mathcal{A}$ on behave of $\mathcal{F}_{\mathsf{MtA}}$ and in doing so receives its input x_1' and output shares t_A. $\mathcal{S}$ checks the consistency of Q_1', r_1, cc received from $\mathcal{A}$ by verifying $Q_1' \stackrel{?}{=} x_1' \cdot P$ and $cc \stackrel{?}{=} t_A + x_1' r_1 - x_1 \mod q$.
(3) On receiving R_1, nizk4, $\mathcal{S}$ sets $R_2 = k_1^{-1} \cdot R - r_1 \cdot P$ (where k_1 is extracted from accepting proof nizk4). $\mathcal{S}$ generates nizk3 to $\mathcal{A}$ by querying zero knowledge simulator Sim. Then, it puts $(R_2 || \mathsf{nizk3}, \mathsf{f2})$ into the hash list of H to indicate that $\mathsf{f2} = \mathsf{H}(R_2, \mathsf{nizk3})$.
(4) After extracting k_1 from nizk4, $s_2 = k_1 s - x_1' r \mod q$ could be easily computed from s, x_1' and k_1.

Note that $\mathcal{S}$ would abort if any of the checks does not pass.

The difference between a real execution and the simulation is how R_2 and s_2 are computed, and the consistency of Q_1' and x_1' is check. In the simulation, R_2 is $k_1^{-1} \cdot R - r_1 \cdot P$ whereas in the real execution $R_2 = k_2 \cdot P$ where $k_2 \leftarrow \mathbb{Z}_q$. Since $\mathcal{F}_{\mathsf{ECDSA}}$ samples R uniformly at random from $\mathbb{G}$, the distribution in both cases is identical. Ext extracts k_1 with knowledge error $1/q$. Conditional on the correctness of k_1, in the simulation

$$s_2 = k_1 s - x_1' r = (r_1+k_2)^{-1}(H(m) + rx_2') \mod q$$

which is identical to that in the real execution. The conditions $Q_1' = x_1' \cdot P$ and $cc = t_A + x_1' r_1 - x_1 \mod q$ in the simulation are equivalent to $(t_B + cc) \cdot P = (r_1 + k_2) \cdot Q_1' - Q_1$ in the real game.

This implies that the view of a corrupted P_1 in the real execution is indistinguishable from that of the simulation, i.e., the advantage of any PPT adversary who corrupts P_1 to distinguish the real execution and simulated execution given by $\mathcal{S}$ is negligible.

$\mathcal{S}$ simulates P_1- Corrupted P_2. After receiving public key Q from $\mathcal{F}_{\mathsf{ECDSA}}$, $\mathcal{S}$ invokes $\mathcal{A}$ with a totally random f1, and receives $(Q_2, \mathsf{nizk2})$. After extracting x_2 from the accepting proof nizk2, $\mathcal{S}$ computes $Q_1 = Q - Q_2$ and nizk1 by querying zero knowledge simulator Sim. Then $\mathcal{S}$ fixes the computation of f1 with random oracle, and sends Q_1, nizk1 to $\mathcal{A}$. The indistinguishability between the simulated and real key generation is obvious.

During the signing phase, $\mathcal{S}$ could receive a signature (r,s) of message m from $\mathcal{F}_{\mathsf{ECDSA}}$, and recover R via verification algorithm.

(1) $\mathcal{S}$ invokes $\mathcal{A}$ with $\mathsf{Sign}(\mathsf{sid}, m)$ and receives f2 from $\mathcal{A}$. Then, $\mathcal{S}$ could extract R_2, nizk3 and further k_2 such that $\mathsf{f2} = \mathsf{H}(R_2, \mathsf{nizk3})$ and $R_2 = k_2 \cdot P$ with overwhelming probability with the help of random oracle and knowledge extractor (if the proof is accepted).
(2) $\mathcal{S}$ interacts with $\mathcal{A}$ on behave of $\mathcal{F}_{\mathsf{MtA}}$ and in doing so receives its input k_2' and output shares t_B. $\mathcal{S}$ samples a random $r_1 \leftarrow \mathbb{Z}_q$ and a random $cc \leftarrow \mathbb{Z}_q$, computes
$$Q_1' = (k_2' + r_1)^{-1}[(t_B + cc) \cdot P + Q_1],$$
and sends Q_1', r_1, cc to $\mathcal{A}$.
(3) $\mathcal{S}$ computes $R_1 = (r_1 + k_2)^{-1} \cdot R$, generates nizk4 by querying zero knowledge simulator Sim, and sends them to $\mathcal{A}$.
(4) On receiving $(R_2, \mathsf{nizk3})$ from $\mathcal{A}$, $\mathcal{S}$ checks the consistency of k_2' with R_2
(5) $\mathcal{S}$ checks the correctness of s_2 by verifying
$$s_2 \cdot (r_1 \cdot P + R_2) \stackrel{?}{=} h \cdot P + r \cdot (x_2 - t_B - cc) \cdot P,$$
where $h = H(m)$. If all the check conditions pass, $\mathcal{S}$ would output (r,s) as output.

Note that $\mathcal{S}$ would abort if any of the checks does not pass.

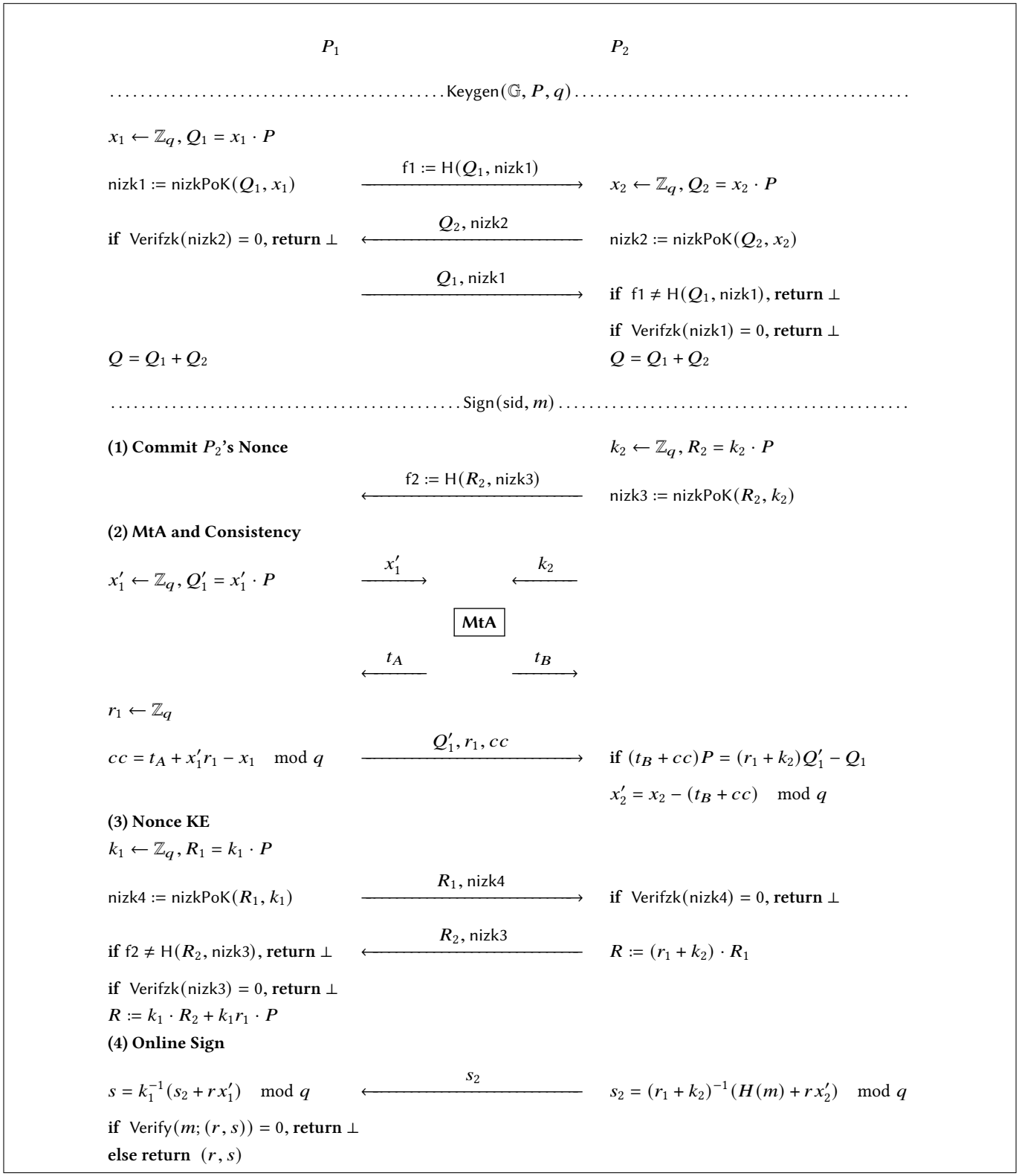

Figure 3: Two-party protocol 2ECDSA from a single MtA.

The difference between a real execution and the simulation is how R_1 and Q_1' are computed, and the condition that (r, s) is output or not.

In the simulation, R_1 is $R_1 = (r_1 + k_2)^{-1} \cdot R$ whereas in the real execution $R_1 = k_1 \cdot P$ where $k_1 \leftarrow \mathbb{Z}_q$. Since $\mathcal{F}_{\mathsf{ECDSA}}$ samples R uniformly at random from $\mathbb{G}$, the distribution in both cases is identical. In the simulation, Q_1' is $(k_2' + r_1)^{-1}[(t_B + cc) \cdot P + Q_1]$ whereas in the real execution $Q_1' = x_1' \cdot P$ for $x_1' \leftarrow \mathbb{Z}_q$. Since r_1 and cc are sampled randomly and the consistency check always passes, the distribution in both cases is identical.

In the real execution, the Verify algorithm checks that $s \cdot R \stackrel{?}{=} h \cdot P + r \cdot Q$, which holds if and only if the following holds,

$$s_2(r_1 + k_2) \stackrel{?}{=} h + r(x_2 - t_B - cc) \mod q. \tag{2}$$

Since

$$s_2 \cdot (r_1 \cdot P + R_2) = h \cdot P + r(x_2 - t_B - cc) \cdot P \tag{3}$$

holds if and only if Equation 2 is correct, the condition to output (r, s) in both the real and simulated case is identical.

This implies that the advantage of any PPT adversary who corrupts P_2 to distinguish the real execution and simulated execution given by $\mathcal{S}$ is negligible.

4 INSTANTIATIONS OF MTA AND THEIR APPLICATIONS TO 2ECDSA

There are several constructions of MtA which could be directly applied to our 2ECDSA.

4.1 MtA from Oblivious Transfer

Following Gilboa's semi-honest oblivious transfer [21], and motivated by two-party ECDSA, Doerner et al. [15] proposed a MtA against malicious adversary. Their construction is based upon Simplest OT [10] and KOS [23] OT-extension protocols. Please refer to Appendix C for a detailed description. Their MtA is computationally very efficient, while the communication is rather large. To compute a MtA, at least $8\kappa^2$ bits must be transferred. Concretely, when $\kappa = 256$, the communication of a single execution of MtA is $\approx$ 90 KB.

Their OT-based MtA could be applied to 2ECDSA. The online phase of both [15] and our protocols are extremely lightweight: sending a single field element and computing two elliptic curve point multiplications. In our protocol, a single MtA is required while the two-party ECDSA of [15] needs two. Thus, our solution requires roughly half of the communication and computation of [15]. A concrete comparison is given in Sec. 5.

4.2 MtA from Paillier

Gennaro and Goldfeder [19] presented a Multiplicative-to-Additive conversion protocol from Paillier encryption, which has appeared many times before [12, 25, 27]. Lindell et al. [26] also proposed a similar protocol and call it private multiplication. We recall it and explain its application to our 2ECDSA.

Let $pk = N$ and $sk = \phi(N)$ be the public and secret keys of Paillier encryption. Denote the encryption of message m by $\mathsf{Enc}(pk, m)$, the decryption of ciphertext c by $\mathsf{Dec}(sk, c)$. Denote by $c_1 \oplus c_2 = c_1 c_2 \mod N^2$ the addition of the plaintext in ciphertexts c_1 and c_2, and by $a \odot c = c^a \mod N^2$ the multiplication of the plaintext in c by a scalar a.

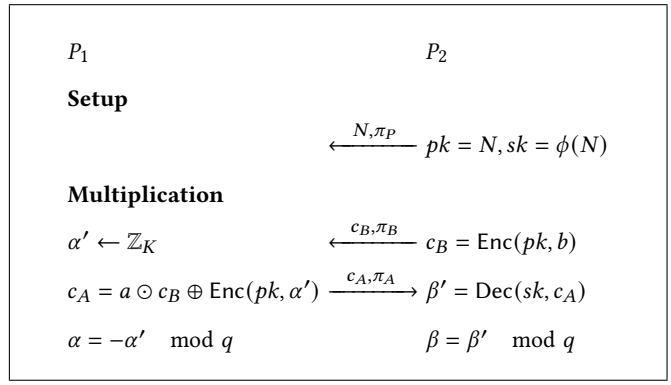

Figure 4: The Paillier-based MtA from [19]. K is the parameter to be determined.

The Paillier-based MtA between P_1 with input a and P_2 with input b works in Figure 4 where

- π_P is a ZKPoK for $\{(N; \phi(N)) | \gcd(N, \phi(N)) = 1\}$;
- π_B is a ZKPoK for $\{(c_B; b) | c_B = \mathsf{Enc}(pk, b) \wedge b \in \mathbb{Z}_q\}$;
- π_A is a ZKPoK for $\{(c_A, c_B; a, \alpha') | c_A = a \odot c_B \oplus \mathsf{Enc}(pk, \alpha') \wedge a \in \mathbb{Z}_q \wedge \alpha' \in \mathbb{Z}_K\}$.

Please refer to Appendix D for the full description of Paillier-based MtA and these ZKPoK protocols (w/o slack regarding soundness).

As noted by previous works [6, 19, 25, 26], π_A and π_B are very expensive. Several works have been done to reduce or even remove these proofs. Lindell [25], Gennaro and Goldfeder [19] suggested to eliminate π_A by relying on a non-standard Paillier-EC assumption (Appendix D.3). Lindell et al. [26], Gennaro and Goldfeder [19], and Canetti et al. [6] found that it is enough to use proofs with a lot of slack regarding soundness. The slack version of π_A and π_B roughly cost computing 6 Paillier exponentiations and sending $6 \log N$ bits. Thus, the overall cost of Paillier-based MtA is computing approximately 14 Paillier exponentiations and sending $16 \log N$ bits. Under the Paillier-EC assumption, it could be further reduced to 8 Paillier exponentiations and $10 \log N$ bits respectively.

4.2.1 Applying Paillier-based MtA to 2ECDSA. We propose two versions of Paillier-based 2ECDSA depending on whether or not MtA relies on Paillier-EC assumption. When applying Paillier-EC assumption, a proof-of-knowledge for the discrete logarithm of Q_1' should be added against corrupted P_1. (In order to extract x_1' to querying the oracle in Paillier-EC assumption.) Actually, Gennaro and Goldfeder [19] also proposed another non-standard assumption to further eliminate π_B, and their technique could also be applied to our protocol. Nevertheless, to simplify the analysis, we do not include it in this paper and just analyze Gennaro and Goldfeder's scheme based on Paillier-EC.

Lindell's two-party protocol [25] has the best overall performance. Nevertheless, their online phase is comparatively expensive, i.e., sending a Paillier ciphertext and computing Paillier decryption. The online phase of the two-party case of Lindell's threshold scheme [26] is worst, since 14 Paillier exponentiations are required. The online phase of [19] is fast, while their offline phase requires at

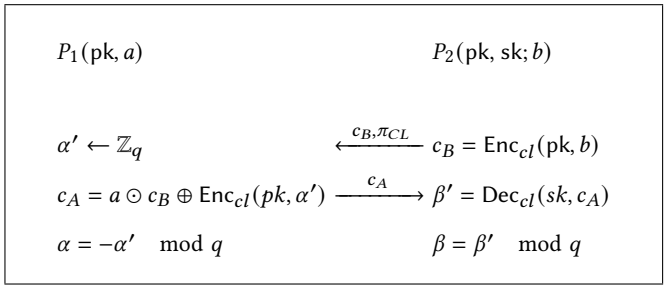

Figure 5: The CL-based MtA extracted from [7, 8].

least computing 18 Paillier exponentiations and sending 16 ℓ_N bits depending on Paillier-EC is required or not. The online phase of the two-party case of Canetti et al. [6] and our scheme are extremely lightweight. However, the offline phase of [6] needs 204 Paillier exponentiations. Our scheme relies on 8 or 14 Paillier exponentiations and sending 10 ℓ_N or 16 ℓ_N bits depending on whether Paillier-EC is required. A concrete comparison is given in Sec. 5.

4.3 MtA with CL-encryption Achieving Game-based Security

To address the problem of non-standard assumption in Paillier-based MtA, Castagnos et al. [7, 8] replaces Paillier with CL encryption [9] which allows instantiations where the message space is exactly $\mathbb{Z}_q$.

Let $(\tilde{s}, f, \hat{g}_q, g_q, \hat{G}, F, q)$ be public parameters of CL encryption (as defined in [8]), where $F = \langle f \rangle$ is the subgroup of $\hat{G}$ with order q, g_q is a random element in $\langle \hat{g}_q \rangle$. Denote by $\mathsf{sk}, \mathsf{pk} = g_q^{\mathsf{sk}}$ secret-public key pair. CL encryption $\mathsf{Enc}_{cl}(\mathsf{pk}, m, r)$ computes $(c_1, c_2) = (g_q^r, \mathsf{pk}^r f^m)$ as the encryption of $m \in \mathbb{Z}_q$ with randomness $r \leftarrow [0, S]$ (for some S), and $\mathsf{Dec}_{cl}(\mathsf{sk}, c_1, c_2)$ computes $\log_f(c_2/c_1^{\mathsf{sk}})$ to decrypt. We also denote by $\oplus$ the addition of the plaintext in two ciphertexts, and by $\odot$ the scalar multiplication on ciphertext.

We abstract CL-based MtA from Castagnos et al.'s protocol [8]. P_2 generates public-secret keypair and sends pk to P_1 in the setup phase. The CL-based MtA protocol is presented in Fig. 5, where π_{CL} is a ZKPoK for relation

$$\{(\mathsf{pk}, c_1, c_2; m, r) | \mathsf{pk} \in \hat{G}, r \in [0, S], c_1 = g_q^r \wedge c_2 = \mathsf{pk}^r f^m\}.$$

Please refer to Appendix E for the proof π_{CL}.

The CL-encryption requires 2 exponentiations on the class group while the decryption requires 1. The prover of π_{CL} needs 2 exponentiations while the verifier computes 4 exponentiations. The size of π_{CL} is about a CL ciphertext. Thus, the overall cost of CL-based MtA is sending 3 CL ciphertexts and computing 9 exponentiations over the class group.

Note that the scheme of Figure 5 dose not provide full simulation-based security when P_1 is corrupted. We remark that simulation based security could be achieved by adding an expensive ZKPoK for the affine operation of c_A on c_B. In our specific usage, we choose the same strategy as Castagnos et al. [8, Sec. 2.2] to achieve a weaker game-based threshold unforgeability given in Appendix E.2, thus do not add this ZKPoK.

4.3.1 Applying CL-based MtA to 2ECDSA. Instantiating MtA by CL will result in a CL-based 2ECDSA[1]. In both [7, 8] and our protocol, CL encryption/decryption and/or π_{CL} dominate the overall complexity.

Although the overall cost of [7] is best, a CL ciphertext should be transferred and a decryption computation is required making the online phase computationally expensive. In particular, it is more than 50 times slower compared with [8, 33], and nearly 1000 times slower than ours in the online phase. The online phase of two-party case of [8, 33] is fast (although it is interactive), while it costs 28 exponentiations and 140κ bits (when $\kappa = 256$). Our online phase is non-interactive and extremely fast, and our offline cost is just 11 exponentiations and 53κ bits. A concrete comparison is given in Sec. 5.

5 IMPLEMENTATION AND COMPARISON

In this section, we give a comprehensive implementation and comparison of two-party ECDSA from Paillier, OT and CL-encryption. Although [6, 8, 16, 19, 26] support threshold larger than two, we only consider their performance in two-party case.

We benchmark our implementation using Rust on a MacBook Pro 13-inch 2019 with Intel Core i5 @ 1.4 GHz CPU and 16 GB 2133 MHz LPDDR3 RAM running macOS Mojave v10.14.5. For simplicity, we evaluate the protocols in a single laptop and consider only the computation time. We remark that the comparison will further favor our scheme if latency is taken into account. The reason is, as shown in Sec. 1.4, our scheme has fewer communication rounds than other schemes. The results are the median time of running 100 times. All benchmarks were taken over curve secp256k1 which is recommended by NIST [24] and is the curve used in Bitcoin, among many other blockchains and cryptocurrencies. We use SHA-256 to instantiate the hash functions, random oracles and/or the PRG.

Overall comparison is shown in Figure 6 and Table 2. The detailed comparisons are given in the following subsections.

5.1 Paillier-based Schemes

We set security parameters $\kappa = 256$ (achieving 128-bits computational security), 80 bits statistical security, and 80 bits soundness error for our Paillier-based 2ECDSA. The underlying MtA is that in Appendix D. Lin17 [25] uses 40-bits statistical security and soundness error. LNR18 [26] and CGGMP20 [6] set the same statistical and soundness parameters with us. GG18 [19] sets 128-bits computational security, 256-bit statistical security and 128-bits soundness error. We note that these works recommend 2048-bit module N. To make a fair comparison with CL and OT-based schemes, we set a 3072-bit module in the implementation and comparison.

We implement two versions of 2ECDSA and GG18 (one with and one without Paillier-EC), LNR18, and CGGMP20 based on the elementary code of ZenGo [32] and run ZenGo's code on Lin17. The results of computation and communication on signing are presented in Table 3. The key generation of our scheme has a similar cost with GG18, LNR18, and CGGMP20.

Without Paillier-EC assumption, our scheme improves GG18's offline complexity by a factor of 5 for computation and a factor of 2

[1] π_{CL} requires a setup step to guarantee that g_q is a random element of the subgroup $\langle \hat{g}_q \rangle$. It is executed in the key generation phase.

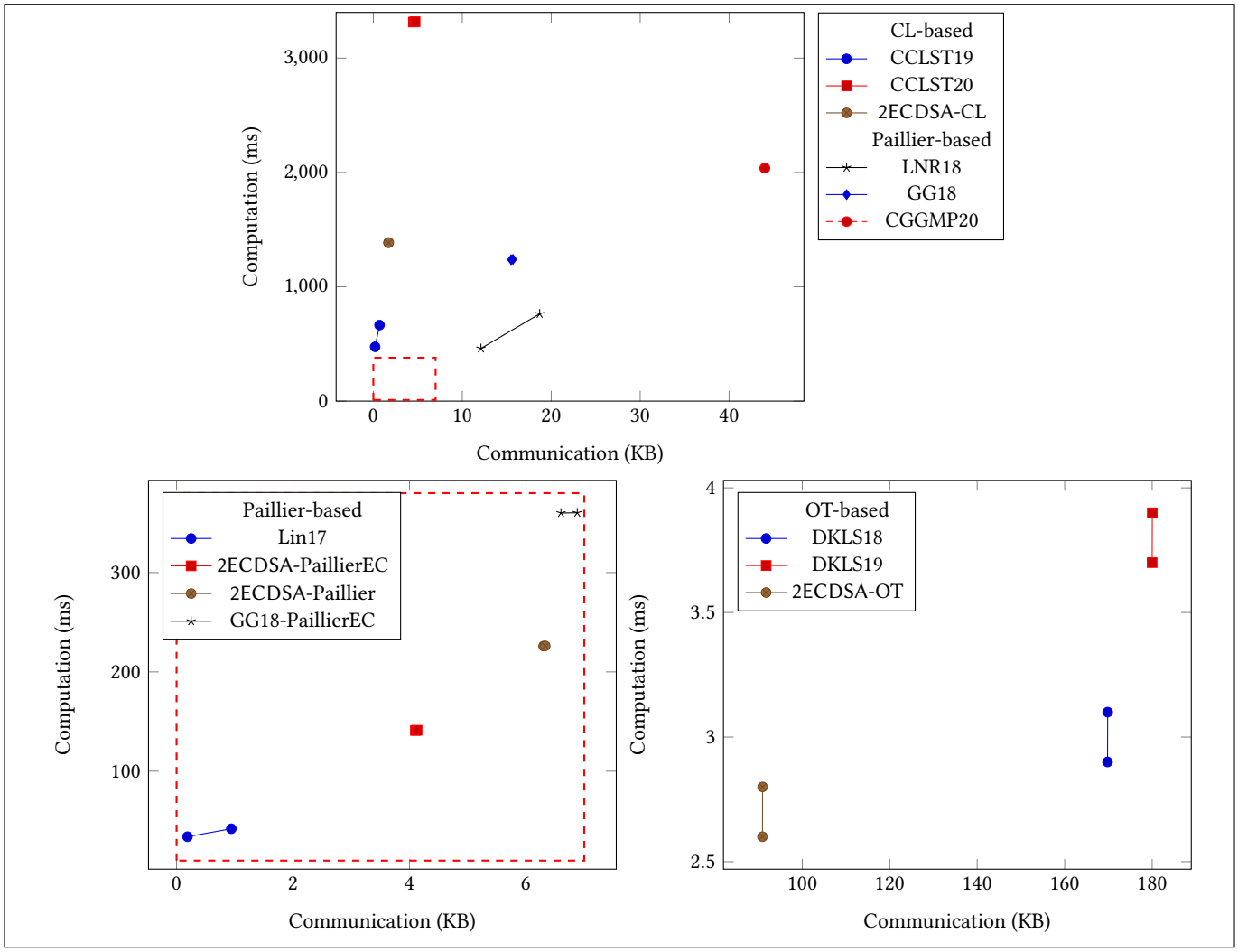

Figure 6: Cost comparison of all two-party schemes. For every scheme, the south-west point is the offline cost, while north-east point is the overall cost. Thus, the distance between these two points indicates the online complexity. The dashed red rectangles share the same area.

for communication. Our implementation outperforms CGGMP20's work by a factor of 9 for computation and a factor of 8 for communication. Without Paillier-EC assumption, we improve GG18's computation by a factor of 2 and its communication by a factor of 1.6.

5.2 OT-based Schemes

We set security parameters $\kappa = 256$ (achieving 128-bits computational security), 80-bits statistical security and soundness error as DKLS18-19 [15, 16] did. We implement OT-based 2ECDSA using MtA of [28] and run their code on DKLS18 and DKLS19 protocols. The results of computation and communication on signing are reported in Table 4. The key generation of our scheme has the same complexity as DKLS19.

The offline communication of our scheme is 90.9 KB, which outperforms DKLS19-20 by a factor of roughly 2.

Table 3: Cost comparison of Paillier-based schemes.

Schemes	Computation		Communication	
	Offline	Online	Offline	Online
LNR18 [26]	461ms	302ms	12.1KB	6.6KB
GG18 [19]	1237ms	3ms	15.5KB	288B
CGGMP20 [6]	2037ms	0.2ms	44KB	32B
2ECDSA (Paillier)	226ms	0.2ms	6.3KB	32B
Lin17 [25] (Paillier-EC)	34ms	8ms	192B	768B
GG18 [19] (Paillier-EC)	360ms	3ms	6.6KB	288B
2ECDSA (Paillier-EC)	141ms	0.2ms	4.1KB	32B

Table 4: Cost comparison of OT-based schemes.

Schemes	Computation		Communication	
	Offline	Online	Offline	Online
DKLS18 [15]	2.9ms	0.2ms	169.8KB	32B
DKLS19 [16]	3.7ms	0.2ms	180KB	32B
2ECDSA (OT)	2.6ms	0.2ms	90.9KB	32B

Table 5: Cost comparison of CL-based schemes.

Schemes	Computation		Communication	
	Offline	Online	Offline	Online
CCLST19 [7]	475ms	190ms	505B	208B
CCLST20 [8]	3316ms	3ms	4.5KB	288B
YCX21 [33]	4550ms	3ms	4.5KB	288B
2ECDSA (CL)	1386ms	0.2ms	1.7KB	32B

5.3 CL-based Schemes

For CL-based instantiations, we set 128-bits computational security, take 80-bits statistical distance and soundness error as YCX21 [33] did. CCLST19 [7] used 40-bits statistical and soundness security, and CCLST20 [8] utilized 128-bits for soundness error and 80-bits for statistical distance. We implement CL-based 2ECDSA and two-party schemes of CCLST20 and YCX21 and run ZenGo's code on CCLST19. The cost comparison is reported in Table 5. The key generation of our scheme has the same complexity with [8].

The offline complexity of our scheme is 1.7KB and 1386ms. CCLST20's offline phase needs at least 2× cost than our work. The offline phase of YCX21 requires 2× communication and 3× computation than our scheme.

6 CONCLUSION

We propose an online-friendly two-party 2ECDSA such that its online computation is extremely fast and its offline phase requires only a single execution of MtA. Our scheme could be efficiently instantiated with constructions of MtA from Paillier, CL encryptions and oblivious transfer. Furthermore, our scheme can be easily extended to the more general case of 2-out-of-n.

Our work focuses on the two-party ECDSA. We believe the idea of this work will lead to improvements of the full threshold ECDSA, and we leave this for further work.

ACKNOWLEDGMENTS

We would like to thank Xuyang Song and Xueli Wang for their help in the experiments. Haiyang Xue is supported by the National Natural Science Foundation of China (No. 62172412), the National Key Research and Development Program of China (No. 2020YFB1807502). Man Ho Au is supported by the National Natural Science Foundation of China (No. 61972332), the Research Grant Council of Hong Kong (GRF Project 15211120).

REFERENCES

[1] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. 2020. A Survey of ECDSA Threshold Signing. (2020). https://eprint.iacr.org/2020/1390.pdf.
[2] Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. 2020. Efficient Protocols for Oblivious Linear Function Evaluation from Ring-LWE. In *SCN*. Springer, 130–149.
[3] Donald Beaver. 1991. Efficient multiparty protocols using circuit randomization. In *CRYPTO*. Springer, 420–432.
[4] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-interactive zero-knowledge and its applications. In *STOC*. 103–112.
[5] Fabrice Boudot. 2000. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*. Springer, 431–444.
[6] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. 2020. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. In *ACM CCS*. 1769–1787.
[7] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. 2019. Two-party ECDSA from hash proof systems and efficient instantiations. In *CRYPTO*. Springer, 191–221.
[8] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. 2020. Bandwidth-efficient threshold EC-DSA. In *PKC*. Springer, 266–296.
[9] Guilhem Castagnos and Fabien Laguillaumie. 2015. Linearly homomorphic encryption from DDH. In *CT-RSA*. Springer, 487–505.
[10] Tung Chou and Claudio Orlandi. 2015. The simplest protocol for oblivious transfer. In *LATINCRYPT*. Springer, 40–58.
[11] William M Daley and Raymond G Kammer. 2000. *Digital signature standard (DSS)*. Technical Report. BOOZ-ALLEN AND HAMILTON INC MCLEAN VA.
[12] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P Smart. 2012. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In *SCN*. Springer, 241–263.
[13] Yvo Desmedt. 1987. Society and group oriented cryptography: A new concept. In *CRYPTO*. Springer, 120–127.
[14] Yvo Desmedt and Yair Frankel. 1989. Threshold cryptosystems. In *CRYPTO*. Springer, 307–315.
[15] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. 2018. Secure two-party threshold ECDSA from ECDSA assumptions. In *IEEE Symposium on Security and Privacy*. IEEE, 980–997.
[16] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. 2019. Threshold ECDSA from ECDSA assumptions: the multiparty case. In *IEEE Symposium on Security and Privacy*. IEEE, 1051–1066.
[17] Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE, 427–438.
[18] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*. Springer, 186–194.
[19] Rosario Gennaro and Steven Goldfeder. 2018. Fast multiparty threshold ECDSA with fast trustless setup. In *ACM CCS*. 1179–1194.
[20] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. 2017. Maliciously secure oblivious linear function evaluation with constant overhead. In *ASIACRYPT*. Springer, 629–659.
[21] Niv Gilboa. 1999. Two party RSA key generation. In *CRYPTO*. Springer, 116–129.
[22] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. 2019. Efficient RSA key generation and threshold paillier in the two-party setting. *Journal of Cryptology* 32, 2 (2019), 265–323.
[23] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2015. Actively secure OT extension with optimal overhead. In *CRYPTO*. Springer, 724–741.
[24] C Kerry and P Gallagher. 2013. FIPS PUB 186-4: Digital Signature Standard (DSS). *Federal Information Processing Standards Publication. National Institute of Standards und Technology* (2013).
[25] Yehuda Lindell. 2017. Fast secure two-party ECDSA signing. In *CRYPTO*. Springer, 613–644.
[26] Yehuda Lindell and Ariel Nof. 2018. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM CCS*. 1837–1854. Refer https://eprint.iacr.org/2018/987.pdf for the full version..
[27] Philip MacKenzie and Michael K Reiter. 2001. Two-party generation of DSA signatures. In *CRYPTO*. Springer, 137–154.
[28] NEUCRYPO. 2021. mp-ecdsa. https://gitlab.com/neucrypt/mpecdsa.
[29] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of cryptology* 4, 3 (1991), 161–174.
[30] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[31] Victor Shoup. 1997. Lower bounds for discrete logarithms and related problems. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 256–266.
[32] ZenGo X. 2021. multi-party-ecdsa. https://github.com/ZenGo-X/multi-party-ecdsa.
[33] Tsz Hon Yuen, Handong Cui, and Xiang Xie. 2021. Compact Zero-Knowledge Proofs for Threshold ECDSA with Trustless Setup. In *PKC*. Springer, 481–511.

A KEY GENERATION PROTOCOL FOR 2-OUT-OF-n ACCESS STRUCTURES

Distributed Key Generation Phase Keygen$(\mathbb{G}, P, q)$**.** Given the joint input $(\mathbb{G}, P, q)$:

- Phase 1: Each player P_i selects $x_i \leftarrow \mathbb{Z}_q$ and computes $X_i = x_i \cdot P$ and a ZKPoK of discrete logarithm using x_i. Then, P_i broadcasts a commitment of X_i and corresponding ZKPoK.
- Phase 2: Upon receiving commitments from other parties, P_i decommits X_i and corresponding ZKPoK. Then, P_i checks all the received commitments and the ZKPoKs.
- Phase 3: Each party P_i runs $(2, n)$ Feldman-VSS [17] using his secret x_i. Specifically, let $\{x_1^{(i)}, \cdots, x_n^{(i)}\}$ be the secret sharing of x_i, P_i adds the private shares received from Feldman-VSS, i.e., $v_i = \sum_{j=1}^{n} x_i^{(j)} \mod q$. Note that $V_i = v_i \cdot P$ is publicly computable.
- Phase 4: Each player P_i proves in zero-knowledge that it knows v_i, the discrete logarithm of V_i. Each player sets $Q = \sum_{i=1}^{n} X_i$ to be the public key, otherwise aborts. Note that $x = \sum_{i=1}^{n} u_i \mod q$ be the secret key.

Assuming that P_i and P_j are the two parties to sign m, they could compute $\Lambda_i v_i$, $\Lambda_j v_j$ respectively (such that $x = \Lambda_i v_i + \Lambda_j v_j$), where Λ_i and Λ_j are Lagrange coefficients.

B PROOF OF THE MAIN THEOREM

Here is the proof of Theorem 3.1.

The simulator $\mathcal{S}$ could only access to an ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}$ for computing ECDSA signatures. All $\mathcal{S}$ learns in the ideal world is the public key Q generated in the key generation phase and several signatures for messages of its choice in the signature phase. In the real world, the adversary, having either corrupted P_1 or P_2 will also see all the interactions with the non-corrupted party. Thus $\mathcal{S}$ must be able to simulate the adversary's view of these interactions, while only knowing the expected output. The proof proceeds in two cases: the adversary corrupts P_1, and the adversary corrupts P_2.

$\mathcal{S}$ simulates P_2- Corrupted P_1. We first show that if adversary $\mathcal{A}$ corrupts P_1, there exists simulator $\mathcal{S}$ such that the output distribution of $\mathcal{S}$ is indistinguishable from $\mathcal{A}$'s view in the real execution of the protocol.

Simulator $\mathcal{S}$ maintains a hash list L_h for H. On any query y to H, if $\exists (y, h_y) \in L_h$, return h_y, else return $h_y \leftarrow \{0,1\}^\kappa$ and add (y, h_y) to L_f.

Key Generation Phase.

(1) Given input Keygen$\mathbb{G}, P, q$), $\mathcal{S}$ sends Keygen$\mathbb{G}, P, q$) to $\mathcal{F}_{\mathsf{ECDSA}}$ and receives back the public key Q.
(2) $\mathcal{S}$ invokes $\mathcal{A}$ upon input Keygen$\mathbb{G}, P, q$) and receives f1.
   - if there exists $(Q_1||\mathsf{nizk1}, \mathsf{f1}) \in L_h$, check Verifzk(nizk1). If Verifzk(nizk1) = 1, extract x_1 with Ext. If $Q_1 = x_1 \cdot P$, compute $Q_2 = Q - Q_1$.
   - otherwise, choose random Q_2.

   With the help of zero knowledge simulator Sim, $\mathcal{S}$ computes nizk2, the non-interactive proof of knowledge for the discrete log of Q_2.
(3) $\mathcal{S}$ sends Q_2, nizk2 to $\mathcal{A}$.
(4) $\mathcal{S}$ receives Q_1, nizk1 from $\mathcal{A}$. If $(Q_1||\mathsf{nizk1}, \mathsf{f1}) \notin L_h$, abort. $\mathcal{S}$ simulates P_2 aborting if Verifzk(nizk1) = 0 or $Q_1 \neq x_1 \cdot P$.
(5) $\mathcal{S}$ sends continue to $\mathcal{F}_{\mathsf{ECDSA}}$ for P_2 to receive output and stores (Q, x_1, Q_1, Q_2).

To pass the check of f1 = H(Q_1, nizk1), $\mathcal{A}$ must have queried $(Q_1, \mathsf{nizk1})$ to the random oracle. Thus, the difference between the real execution and the ideal execution simulated by $\mathcal{S}$ is the generation of Q_2 and nizk2. In the real execution, $Q_2 = x_2 \cdot P$ where $x_2 \leftarrow \mathbb{Z}_q$, and nizk2 = nizkPoK($Q_2, x_2$), while in the later $Q_2 = Q - Q_1$ and nizk2 $\leftarrow$ Sim(Q_2, P) where Q is returned by $\mathcal{F}_{\mathsf{ECDSA}}$. Ext extracts x_1 with knowledge error $1/q$. Since $\mathcal{F}_{\mathsf{ECDSA}}$ samples Q uniformly at random from $\mathbb{G}$, conditional on the extraction of x_1, the distribution of Q_2 in both cases is identical. Since Sim perfectly simulate the proof, the distribution of nizk2 is also identical.

Signing Phase.

- Given input Sign(sid, m), $\mathcal{S}$ sends Sign(sid, m) to $\mathcal{F}_{\mathsf{ECDSA}}$ and receives signature (r, s).
- Using the verification procedure, $\mathcal{S}$ recovers R from (r, s).

(1) Commitment: $\mathcal{S}$ invokes $\mathcal{A}$ with input Sign(sid, m) and sends a random string f2 $\leftarrow \{0,1\}^\kappa$ to $\mathcal{A}$.
(2) MtA and consistency check:
   - $\mathcal{S}$ interacts with $\mathcal{A}$ on behave of $\mathcal{F}_{\mathsf{MtA}}$ and in doing so receives $\mathcal{A}$'s input x_1' and output shares t_A.
   - On receiving Q_1', r_1, cc from $\mathcal{A}$, $\mathcal{S}$ checks the consistency by verifying $Q_1' \stackrel{?}{=} x_1' \cdot P$ and $cc \stackrel{?}{=} t_A + x_1' r_1 - x_1 \mod q$, and simulates P_2 aborting if these equations do not hold.
(3) Nonce key exchange:
   - Upon receiving $(R_1, \mathsf{nizk4})$ from $\mathcal{A}$, $\mathcal{S}$ simulates P_2 aborting if Verifzk(nizk4) = 0. Else $\mathcal{S}$ extracts k_1 utilizing Ext algorithm.
   - $\mathcal{S}$ computes $R_2 = k_1^{-1} \cdot R - r_1 \cdot P$, generates nizk3 by querying zero knowledge simulator Sim, and adds $(R_2||\mathsf{nizk3}, \mathsf{f2})$ to the hash list L_h.
(4) Online Signature: $\mathcal{S}$ computes $s_2 = k_1 s - x_1' r \mod q$ where x_1' and k_1 are extracted from nizk1 and nizk4 respectively, and sends R_2, nizk3, s_2 to $\mathcal{A}$.

The difference between a real execution and the simulation is how R_2 and s_2 are computed, and the consistency of Q_1' and x_1' is check. In the simulation, R_2 is $k_1^{-1} \cdot R - r_1 \cdot P$ whereas in the real execution $R_2 = k_2 \cdot P$ where $k_2 \leftarrow \mathbb{Z}_q$. Since $\mathcal{F}_{\mathsf{ECDSA}}$ samples R uniformly at random from $\mathbb{G}$, the distribution in both cases is identical. Ext extracts k_1 with knowledge error $1/q$. Conditional on the correctness of k_1, in the simulation

$$s_2 = k_1 s - x_1' r = (r_1 + k_2)^{-1}(H(m) + r x_2') \mod q$$

which is identical to that in the real execution. Conditions $Q_1' = x_1' \cdot P$ and $cc = t_A + x_1' r_1 - x_1 \mod q$ in the simulation are equivalent to $(t_B + cc) \cdot P = (r_1 + k_2) \cdot Q_1' - Q_1$ in the real game.

This implies that the view of a corrupted P_1 in the real execution is indistinguishable from that of the simulation, i.e., the advantage of any PPT adversary who corrupts P_1 to distinguish the real execution and simulated execution given by $\mathcal{S}$ is negligible.

$\mathcal{S}$ simulates P_1- Corrupted P_2. We show that if an adversary $\mathcal{A}$ corrupts P_2, there exists a simulator $\mathcal{S}$ such that the output distribution of $\mathcal{S}$ is indistinguishable with $\mathcal{A}$'s view in the real execution of the protocol. Simulator $\mathcal{S}$ maintains a hash list L_h for

H. On any query y to H, if $\exists(y, h_y) \in L_h$, return h_y, else return $h_y \leftarrow \{0,1\}^\kappa$ and add (y, h_y) to L_h.

Key Generation Phase.

(1) Given input KeygenG, P, q), $\mathcal{S}$ sends KeygenG, P, q) to $\mathcal{F}_{\text{ECDSA}}$ and receives back the public key Q.
(2) $\mathcal{S}$ invokes $\mathcal{A}$ with input KeygenG, P, q) and sends a random string f1 $\leftarrow \{0,1\}^\kappa$ to $\mathcal{A}$.
(3) Upon receiving $(Q_2, \text{nizk2})$ from $\mathcal{A}$, $\mathcal{S}$ computes $Q_1 = Q - Q_2$ and generates nizk1 with the help of zero knowledge simulator Sim.
(4) $\mathcal{S}$ adds $(Q_1||\text{nizk1}, \text{f1})$ to the hash list L_h.
(5) $\mathcal{S}$ sends Q_1, nizk1 to $\mathcal{A}$ and stores (x_2, Q, Q_1, Q_2).

The difference between real execution and ideal execution simulated by $\mathcal{S}$ is the generation of Q_1 and nizk1. In the real execution, $Q_1 = x_1 \cdot P$ where $x_1 \leftarrow \mathbb{Z}_q$, and nizk1 = nizkPoK$(Q_1, x_1)$, while in the later $Q_1 = Q - Q_2$ and nizk1 $\leftarrow$ Sim(Q_1, P) where Q is returned by $\mathcal{F}_{\text{ECDSA}}$. Ext extracts x_2 with knowledge error $1/q$. Since $\mathcal{F}_{\text{ECDSA}}$ samples Q uniformly at random from $\mathbb{G}$, conditional on the extraction of x_2, the distribution of Q_1 in both cases is identical. Since Sim perfectly simulate the proof, the distribution of nizk1 is also identical.

Signing Phase.

- Given input Sign(sid, m), $\mathcal{S}$ sends Sign(sid, m) to $\mathcal{F}_{\text{ECDSA}}$ and receives signature (r, s).
- Using the verification procedure, $\mathcal{S}$ recovers R from (r, s).

(1) Commitment: $\mathcal{S}$ invokes $\mathcal{A}$ with input Sign(sid, m) and receives f2 from $\mathcal{A}$.
   - if $\exists(R_2||\text{nizk3}, \text{f2}) \in L_h$, $\mathcal{S}$ extracts k_2 such that $R_2 = k_2 \cdot P$ with the help of knowledge extractor (if the proof is accepted). Then $\mathcal{S}$ samples a random $r_1 \leftarrow \mathbb{Z}_q$ and computes $R_1 = (r_1 + k_2)^{-1} \cdot R$.
   - otherwise, samples a random $r_1 \leftarrow \mathbb{Z}_q$ and a random point R_1 and generates nizk4 with the help of zero knowledge simulator Sim
(2) MtA and consistency check:
   - $\mathcal{S}$ interacts with $\mathcal{A}$ on behave of $\mathcal{F}_{\text{MtA}}$ and in doing so receives its input k_2' and output shares t_B.
   - $\mathcal{S}$ samples a random $cc \leftarrow \mathbb{Z}_q$, computes
   $$Q_1' = (k_2' + r_1)^{-1}[(t_B + cc) \cdot P + Q_1],$$
   and sends (Q_1', r_1, cc) to $\mathcal{A}$.
(3) Nonce key exchange: $\mathcal{S}$ computes $R_1 = (r_1 + k_2)^{-1} \cdot R$, generates nizk4 by querying zero knowledge simulator Sim, and sends them to $\mathcal{A}$.
(4) Online signature: Upon receiving R_2, nizk3, s_2 from $\mathcal{A}$, $\mathcal{S}$ checks the proof of NIZK3, and whether f2 = H$(R_2, \text{nizk3})$ and
   $$s_2 \cdot (r_1 \cdot P + R_2) \stackrel{?}{=} h \cdot P + r \cdot (x_2 - t_B - cc) \cdot P.$$
   If the checks pass, $\mathcal{S}$ returns (r, s) as the final signature, else aborts.

The difference between a real execution and the simulation is how R_1 and Q_1' are computed, and the condition that (r, s) is output or not.

In the simulation, R_1 is $R_1 = (r_1 + k_2)^{-1} \cdot R$ whereas in the real execution $R_1 = k_1 \cdot P$ where $k_1 \leftarrow \mathbb{Z}_q$. Since $\mathcal{F}_{\text{ECDSA}}$ samples R uniformly at random from $\mathbb{G}$, the distribution in both cases is identical. In the simulation, Q_1' is $(k_2' + r_1)^{-1}[(t_B + cc) \cdot P + Q_1]$ whereas in the real execution $Q_1' = x_1' \cdot P$ for $x_1' \leftarrow \mathbb{Z}_q$. Since r_1 and cc are sampled randomly and the consistency check always passes, the distribution in both cases is identical.

In the real execution, the Verify algorithm checks that $s \cdot R \stackrel{?}{=} h \cdot P + r \cdot Q$, i.e., implicitly checks

$$s_2(r_1 + k_2) \stackrel{?}{=} h + r(x_2 - t_B - cc) \mod q. \tag{4}$$

Since

$$s_2 \cdot (r_1 \cdot P + R_2) = h \cdot P + r(x_2 - t_B - cc) \cdot P \tag{5}$$

holds if and only if Equation 4 is correct, the condition to output (r, s) in both the real and simulated case is identical.

This implies that the view of a corrupted P_2 in the real execution is indistinguishable with that of the simulation, i.e., the advantage of any PPT adversary who corrupts P_2 to distinguish the real execution and simulated execution given by $\mathcal{S}$ is negligible.

C MTA FROM OBLIVIOUS TRANSFER

We recall the MtA from OT proposed in [15].

Let $\mathcal{F}_{\text{OTe}}^\ell$ be the Correlated OT-extension functionality that allows arbitrarily many Correlated OT instances to be executed in batches of size ℓ. The input of the receiver is a vector of choice bits while the sender's input is a vector of correlated elements. The functionality samples ℓ random pads and sends them to the sender. To the receiver, it sends the pads if the sender's corresponding choice bits were 0, otherwise the sum of the pads and their corresponding correlations. Please refer to [15, Sec. IV and Appendix A] for the concrete definition and instantiation.

The OT-based MtA is constructed in the $\mathcal{F}_{\text{OTe}}^\ell$ hybrid model. It is parameterized by the statistical security parameter s, the curve order q, and $\kappa = |q|$. Let $\mathbf{g} = \mathbf{g}^{\text{G}}||\mathbf{g}^{\text{R}}$ be a coefficient vector where $\mathbf{g}^{\text{G}}$ satisfies $\mathbf{g}_i^{\text{G}} = 2^{i-1}$, and $\mathbf{g}^{\text{R}}$ is a public random vector. Assume the input of Alice and Bob is $a, b \in \mathbb{Z}_q$ respectively, they execute the following protocol to export α, β such that $\alpha + \beta = ab$.

Encoding:

- Bob samples $\gamma \leftarrow \{0,1\}^{\kappa+2s}$, and encodes its input as $\mathbf{b} = \text{Bits}(b - \langle \mathbf{g}^{\text{R}}, \gamma \rangle)||\gamma$.
- Alice samples $\hat{a} \leftarrow \mathbb{Z}_q$ and sets $\mathbf{a} = \{a||\hat{a}\}_{j \in [1, 2\kappa+2s]}$.

Multiplication:

- Alice who plays as sender and Bob as receiver, invokes functionality $\mathcal{F}_{\text{OTe}}^\ell$ with their encoded input where $\ell = 2\kappa + 2s$. They receive as outputs, respectively, $\{\text{t}_{Aj}||\hat{\text{t}}_{Aj}\}_{j \in [1,2\kappa+2s]}$ and $\{\text{t}_{Bj}||\hat{\text{t}}_{Bj}\}_{j \in [1,2\kappa+2s]}$
- Alice and Bob generate two shared random values by calling the random oracle, i.e., $(\chi, \hat{\chi}) \leftarrow \text{RO}(\text{transcript})$.
- Alice computes and sends $\text{r} = \{\chi \text{t}_{Aj} + \hat{\chi}\hat{\text{t}}_{Aj}\}_{j \in [1,2\kappa+2s]}$, $u = \chi a + \hat{\chi}\hat{a}$ to Bob.
- Bob aborts if $\chi \text{t}_{Bj} + \hat{\chi}\hat{\text{t}}_{Bj} \neq \mathbf{b}_j u - \text{r}_j$ for any $j \in [1, 2\kappa + 2s]$.
- Alice and Bob compute their output shares respectively, i.e., $\alpha = \sum_{j \in [1,2\kappa+2s]} \mathbf{g}_j \text{t}_{Aj}$, and $\beta = \sum_{j \in [1,2\kappa+2s]} \mathbf{g}_j \text{t}_{Bj}$.

D MTA FROM PAILLIER

We recall the MtA protocol from Paillier encryption, which has appeared many times before [6, 12, 19, 25–27].

Paillier is a well-known additive homomorphic encryption scheme. Let $pk = N$ and $sk = \phi(N)$ be the public and secret keys of Paillier encryption respectively. Denote the encryption of message m with randomness r by $\mathsf{Enc}(pk, m, r) = r^N(1+N)^m \mod N^2$, the decryption of ciphertext c by $\mathsf{Dec}(sk, c)$. For simplicity, we may eliminate the randomness r and denote encryption as $\mathsf{Enc}(pk, m)$. We denote by $c_1 \oplus c_2 = c_1 c_2 \mod N^2$ the addition of the plaintext in ciphertexts c_1 and c_2, and by $a \odot c = c^a \mod N^2$ the multiplication of the plaintext in c by a scalar a.

Zero-knowledge proofs for the following relation are needed for the Paillier-based MtA.

Proof correct Paillier keypair generation Define the relation

$$\mathcal{R}_{\mathrm{P}} = \{(N, \phi(N)) | \gcd(N, \phi(N)) = 1\}$$

of valid Paillier public keys. We remark that standard Paillier is defined for $N = pq$ with p, q prime. As did in [25], we only require $\gcd(N, \phi(N)) = 1$ since all we need is the additive homomorphic property. The proof $\mathsf{ZKPoK}_{\mathcal{R}_{\mathrm{P}}}$ for $\mathcal{R}_{\mathrm{P}}$ was proposed in [22, Sec.3.3] and [25, Appendix A] which (for a soundness error 2^{-t}) requires $3t$ Paillier encryption and $4t$ GCD computation. Please refer to [25, Appendix A] for the description of $\mathsf{ZKPoK}_{\mathcal{R}_{\mathrm{P}}}$.

ZKPoK for Paillier Encryption with Range Proof. Define the relation

$$\mathcal{R}_{\mathrm{PwR}} := \{(N, q, c; m, r) | c = r^N(1+N)^x \mod N^2 \wedge x \in \mathbb{Z}_q\}$$

of Paillier encryption with range proof. Lindell [25, Appendix A] proposed a rather expensive ZKPoK for $\mathcal{R}_{\mathrm{PwR}}$ based on a proof from [5, Sec.1.2.2]. Later, Lindell et al. [26], Gennaro and Goldfeder [19], and Canetti et al. [6] gave an efficient proof with slack under Strong-RSA assumption and proved that the proof with slack is enough for MtA. Please refer to D.1 for a ZKPoK for $\mathcal{R}_{\mathrm{PwR}}$ with slack.

ZKPoK for Range-Bounded Affine Operation. Define the relation $\mathcal{R}_{\mathrm{AffRan}}$ as

$$\{(pk, q, c_A, c_B; a, \alpha) | c_A = a \odot c_B \oplus \mathsf{Enc}(pk, \alpha) \wedge a \in \mathbb{Z}_q \wedge \alpha \in \mathbb{Z}_K\}$$

of Paillier encryption with range proof. We may also define the relation $\mathcal{R}'_{\mathrm{AffRan}}$ as

$$\{(N, q, c_A, c_B; a, \alpha) | c_A = (c_B)^a(1+N)^\alpha \mod N^2 \wedge a \in \mathbb{Z}_q \wedge \alpha \in \mathbb{Z}_K\}.$$

Let $\mathsf{ZKPoK}_{\mathcal{R}'_{\mathrm{AffRan}}}$ (resp. $\mathsf{ZKPoK}_{\mathcal{R}_{\mathrm{AffRan}}}$) be the ZKPok proof for $\mathcal{R}'_{\mathrm{AffRan}}$ (resp. $\mathcal{R}_{\mathrm{AffRan}}$). Please refer to D.2 for this proof with slack.

With those ZKPoKs, the Paillier-based MtA works as in the following. We remark that all the ZKPoKs are transferred to non-interactive using the Fiat-Shamir paradigm [18]. Its security relies on the semantic security of Paillier encryption. As noted by [19, 25], π_A could be eliminated under the non-standard Paillier-EC assumption which is recalled in D.3.

In the setup phase,

- P_2 generates a Paillier key-pair $pk = N$, $sk = \phi(N)$.
- P_2 computes a $\mathsf{ZKPoK}_{\mathcal{R}_{\mathrm{P}}}$ proof for the correctness of Paillier public key.
- P_2 sends pk, $\mathsf{ZKPoK}_{\mathcal{R}_{\mathrm{P}}}$ to P_1 who will verify the proof.

In the multiplication phase, let $a, b \in \mathbb{Z}_q$ be the input of P_1 and P_2 respectively.

- P_2 initiates the protocol
  - Compute $c_B = \mathsf{Enc}(pk, b)$ and a $\mathsf{ZKPoK}_{\mathcal{R}_{\mathrm{PwR}}}$ proof π_B.
  - Send (c_B, π_B) to P_1.
- On receiving c_B, π_B, P_1 do the following
  - Verify π_B, and abort if it fails.
  - Choose $\alpha' \leftarrow \mathbb{Z}_K$. Set output $\alpha = -\alpha' \mod q$.
  - Compute the ciphertext $c_A = a \odot c_B \oplus \mathsf{Enc}(pk, \alpha')$, and a $\mathsf{ZKPoK}_{\mathcal{R}_{\mathrm{AffRan}}}$ proof π_A with witness a, α'. (or Compute $c_A = a \odot c_B(1+N)^{\alpha'} \mod N^2$), and a $\mathsf{ZKPoK}_{\mathcal{R}'_{\mathrm{AffRan}}}$ proof π_A.)
  - Send (c_A, π_A) to P_2.
- Upon receiving (c_A, π_A), P_2 do the following
  - Verify π_A, and abort if it fails.
  - Compute $\beta' = \mathsf{Dec}(sk, c_A)$ and output $\beta = \beta' \mod q$.

D.1 Range Proof for Paillier with Slack under Strong-RSA Assumption

We recall the range proof for Paillier with slack, i.e. the following relation $\mathcal{R}_{\mathrm{PwR}}$) proposed in [26, Sec.6.2.6].

$$\mathcal{R}_{\mathrm{PwR}} := \{(N, q, c; m, r) | c = r^N(1+N)^x \mod N^2 \wedge x \in \mathbb{Z}_q\}.$$

The soundness only guarantees that $x \in [-2^{t+l}q, -2^{t+l}q]$. As analyzed in [26, Sec. 6.2], the following range proof with slack is enough for Paillier-based MtA by adding $2^{t+l}q$ to a and setting $K = 2^{t+l+s}q^2$.

Let t, ℓ, s be security parameters. In the implementation, we set $t = s = 128$ and $\ell = 80$ for 80 bits statistical security. Let N_0 be the modulus for Pedersen, and $g, h \in QR_{N_0}$ be random. The Pedersen commitment of x with randomness ρ is $g^x h^\rho \mod N_0$. It is assumed that the Paillier modulus N was generated by the Prover and the Pedersen parameters (N_0, g, h) were generated by the Verifier. We instruct the parties, in the setup phase, to prove in zero-knowledge that all the parameters were generated correctly.

In the setup phase, the verifier V sends the prover P parameters N_0, g, h and zero-knowledge proof that all the parameters are generated correctly.

- Prover's first message: P chooses $\rho, \gamma, \tau \leftarrow \mathbb{Z}_{N_0}, \alpha, a \in \mathbb{Z}_{2^{t+\ell}q}$, $\beta \leftarrow \mathbb{Z}_N$ and computes $\tilde{C} = g^x h^\rho \mod N_0, A = \beta^N(1+N)^\alpha \mod N^2, B = g^\alpha h^\gamma \mod N_0$, and $D = g^a h^\tau \mod N_0$. Then Prover sends $(\tilde{C}, A, B, D)$ to V.
- Challenge: V chooses $e \in \{0,1\}^{2t}$ and $e_1 \in \mathbb{Z}_{2^t}$ and sends them to P.
- Prover's second message: P computes $z_1 = \alpha + ex$ (over the integer), $z_2 = \beta r^e \mod N$, $z_3 = \gamma + \rho e$, $z_4 = a + xe_1$, $z_5 = \tau + e_1\rho$. P sends $(z_1, z_2, z_3, z_4, z_5)$ to V.
- Verification: Accept if and only if
  - $z_2^N(1+N)^{z_1} = A(C)^e \mod N^2$
  - $g^{z_1}h^{z_3} = B(\tilde{C})^e \mod N_0$
  - $z_4 \in [2^t q, 2^{t+\ell}q)$
  - $g^{z_4}h^{z_5} = D(\tilde{C})^{e_1} \mod N_0$

Table 6: The CL encryption scheme.

$\mathsf{Enc}_{cl}(\mathsf{pk}, m)$	$\mathsf{Dec}_{cl}(\mathsf{sk}, c_1, c_2)$
Pick $r \leftarrow [0, S]$ Return $(g_q^r, \mathsf{pk}^r f^m)$	Compute $M = c_2/c_1^{\mathsf{sk}}$ Return $\log_f(M)$

D.2 Proof of Paillier-Pedersen Range-Bounded Affine Operation

We recall the proof for affine operation over Paillier ciphertext in given range, i.e. for relation $\mathcal{R}'_{\text{AffRan}}$) proposed in [26, Sec.6.2.7]. $\mathcal{R}'_{\text{AffRan}}$ is defined as

$$\{(N, q, c_A, c_B; a, \alpha) | c_A = (c_B)^a (1+N)^\alpha \mod N^2 \\ \wedge a \in \mathbb{Z}_q \wedge \alpha \in \mathbb{Z}_{q^2 2^{t+\ell+s}}\}$$

Let t, ℓ, s be security parameters. In the implementation, we set $t = s = 128$ and $\ell = 80$ for 80 bits statistical security. The proof works as follows. In the setup phase, the verifier V sends the prover P parameters N_0, g, h and zero-knowledge proof that all the parameters are generated correctly. (In our specific usage to MtA, $N_0 = N$ since the verifier generates N.)

- Prover's first message: P chooses $b \in \mathbb{Z}_{2^{t+\ell}q}$, $\beta \leftarrow \mathbb{Z}_{2^{2t+2\ell+s}q^2}$, $\rho_1, \rho_2, \rho_3, \rho_4 \leftarrow \mathbb{Z}_N$. P computes $A = c_B^b (1+N)^\beta \mod N^2$, $B_1 = g^b h^{\rho_1} \mod N$, $B_2 = g^\beta h^{\rho_2} \mod N$, $B_3 = g^a h^{\rho_3} \mod N$, $B_4 = g^\alpha h^{\rho_4} \mod N$. P sends (A, B_1, B_2, B_3, B_4) to V.
- Challenge: V sends a random $e \in \mathbb{Z}_{2^t}$ to P.
- Prover's second message: P computes and sends $z_1 = b + ea$, $z_2 = \beta + e\alpha$, $z_3 = \rho_1 + e\rho_3$, $z_4 = \rho_2 + e\rho_4$ to V.
- Verification: Accept if and only if
  - $z_1 \in [2^t q, 2^{t+\ell} q)$
  - $z_2 \in [2^{2t+\ell+s} q^2, 2^{2t+2\ell+s} q^2)$
  - $c_B^{z_1} (1+N)^{z_2} = A(c_A)^e \mod N^2$
  - $g^{z_1} h^{z_3} = B_1 (B_3)^e \mod N$
  - $g^{z_2} h^{z_4} = B_2 (B_4)^e \mod N$

D.3 Paillier-EC Assumption

The Paillier-EC assumption states the security of encryption even the adversary is given a restricted oracle. Concretely, it says that any PPT adversary $\mathcal{A}$ can only win the following experiment (i.e. the experiment outputs 1) with 1/2 plus a negligible probability.

- Generate a Paillier key pair (pk, sk)
- Choose $w_0, w_1 \leftarrow \mathbb{Z}_q$ and compute $Q_0 = w_0 \cdot P$
- Choose a random $b \leftarrow \{0, 1\}$ and compute $c = \mathsf{Enc}(pk, w_b)$
- Let $b' = \mathcal{A}^{O(\cdot,\cdot,\cdot)}(pk, c, Q_0)$, where $O(c, a, b) = 1$ iff $\mathsf{Dec}(c) = a + bw_b \mod q$ and the oracle O holds once it returns 0.
- The experiment returns 1 if and only if $b' = b$.

E THE CL-ENCRYPTION, ZKPOK FOR CL-ENCRYPTION AND GAME-BASED THRESHOLD UNFORGEABILITY.

We recall the CL encryption [9] and ZKPoK for CL encryption from [8].

Let $(\tilde{s}, f, \hat{g}_q, g_q, \hat{G}, F, q)$ be public parameters of CL encryption (as defined in [8]), where $F = \langle f \rangle$ is the subgroup of $\hat{G}$ with order q, g_q is a random element in $\langle \hat{g}_q \rangle$. Denote by $\mathsf{sk}, \mathsf{pk} = g_q^{\mathsf{sk}}$ secret-public key pair. CL encryption and decryption works as Table 6, where S is an integer to be determined.

We denote by $c \oplus \hat{c}$ the addition of the plaintext in ciphertexts c and $\hat{c}$, and by $a \odot c$ the multiplication of the plaintext in c by a scalar a.

E.1 ZKPoK for correctness of CL encryption

We recall the zero-knowledge argument of knowledge for correctness of ciphertext from [8]. Define relation

$$R_{\text{CL}} := \{(\mathsf{pk}, c_1, c_2; m, r) | \mathsf{pk} \in \hat{G}, r \in [0, S], \\ c_1 = g_q^r \wedge c_2 = \mathsf{pk}^r f^m\}.$$

Let C be the challenge space and $C = |C|$. In the setup, the prover should send public parameters to verifier and proves g_q is a random element of $\langle \hat{g}_q \rangle$ (using the proof given in [8, Sec. 3.2]).

- Prover's first message: P chooses $r_1 \leftarrow [0, 2^{80} \tilde{s} C]$, $r_1 \leftarrow \mathbb{Z}_q$, computes $t_1 = g_q^{r_1}, t_2 = \mathsf{pk}^{r_1} f^{r_1}$. P sends (t_1, t_2) to V.
- Challenge: V sends a random $e \leftarrow C$ to P.
- Prover's second message: P computes and sends $z_1 = r_1 + er$, $z_2 = r_1 + em$ to V.
- Verification: Accept if and only if
  - $z_1 \in [0, 2^{40} \tilde{s} C (2^{40} + 1)]$
  - $z_2 \in \mathbb{Z}_q$
  - $g_q^{z_1} = t_1 c_1^e$
  - $\mathsf{pk}^{z_2} f^{z_2} = t_2 (c_2)^e$

E.2 Game-based Threshold Unforgeability

We recall the definition of game-based threshold unforgeaility given in [8, Sec. 2.2], and specify it to two-party case.

Definition E.1 (Two-party signature unforgeability). Consider a two-party signature (Keygen, Sign, Verify), and a PPT algorithm $\mathcal{A}$, having corrupted one of two parties, and which is given the view of two-party protocols Keygen and Sign on input messages of its choice (chosen adaptively) as well as signatures on these messages. Let M be the set of aforementioned messages. The two-party signature is said to be unforgeable under chosen message attack if for any such $\mathcal{A}$ the probability that $\mathcal{A}$ can produce a signature on a message $m \notin M$ is negligible.