SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

Simple Yet Fast Two-Party Signature Based on ECDSA

Binbin Tu[1,2], Yu Chen[1*], Hongrui Cui[3] & Xianfang Wang[2]

[1]*School of Cyber Science and Technology, Shandong University, Qindao, China;*
[2]*Westone Cryptologic Research Center, Westone Information Industry Inc, Beijing, China;*
[3]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China*

Abstract ECDSA is a standardized signature scheme and is widely used in many fields. However, most two-party ECDSA needs a complicated multi-party computation technique to compute the multiplication of many shared secrets and requires expensive zero-knowledge proofs to deal with malicious adversaries.
In this paper, we propose a simple and fast two-party signing protocol based on ECDSA security, whose signing operation of each party is similar to the original ECDSA signing algorithm. Therefore, it can enjoy the same efficiency as ECDSA and reuses the implementation of ECDSA greatly, and upgrades the existing ECDSA application to two-party scenario friendly. For this purpose, we first introduce a variant of ECDSA called combinatorial ECDSA, which is as secure as the standard ECDSA and can be split into two pieces easily. Then, benefiting from the structure of combinatorial ECDSA, we devise a fast two-party signing protocol without complicated multi-party computation and give the security proof based on ECDSA. Compared with the state-of-the-art two-party ECDSA, the signing result of each party in our protocol preserves the structure of the ECDSA signature, which can be verified easily and avoids expensive zero-knowledge proofs. Last, we conduct an experimental evaluation, demonstrating that the performance of our combinatorial ECDSA and two-party signature is similar to ECDSA and the experimental results show that our proposed schemes are practical.

Keywords Two-party signature, ECDSA, Combinatorial ECDSA, Signing key protection

Citation Binbin Tu, Yu Chen, Hongrui Cui, et al. Simple Yet Fast Two-Party Signature Based on ECDSA. Sci China Inf Sci, for review

1 Introduction

The two-party signature [1–10] can split the signing key into two parts and each belongs to one signing party so that a valid signature can be generated by both parties together via a secure protocol. This kind of signature is beneficial to the protection of the signing key because the adversary needs to corrupt both parties to obtain the signing key.

With the benefits of key length and efficiency, ECDSA [11] is widely used in many fields such as TLS [12], DNSSec [13], and many cryptocurrencies, including Bitcoin [14] and Ethereum. As we all know, the security protection of the signing key is the core of signature security, especially in cryptocurrencies, because the signing key loss means concrete financial losses. Therefore, a two-party signature based on ECDSA can provide a high level of signing key protection, which also means better security for your wallet.

However, how to design a two-party signing protocol based on ECDSA is more complicated than many other types of signatures [15, 16], because (i) a shared secret must be inverted, and (ii) a multiplication must be performed on two shared secrets [1]. In order to solve these problems, many heavy multi-party computation (MPC) techniques are applied, such as homomorphic encryption [1–3], and oblivious transfer [5, 8], etc. More precisely, both parties hold part of the shared signing key and a random value, then take in those as privacy input and split the signing algorithm by using MPC protocol to compute a signature. However, computing multiplication between two-party is always complicated. Especially,

[*] Corresponding author (email: cycosmic@gmail.com)

the protocol also requires additional expensive zero-knowledge proofs to ensure that each party works correctly. MacKenzie and Reiter [1] first present a provably secure two-party DSA signature by using homomorphic encryption [17]. Gennaro et al. [2] take advantage of threshold homomorphic encryption, and give the first threshold-optimal ECDSA signature scheme. After that, Lindell [3] points out that [1] and [2] need complicated zero-knowledge proof in the signing phase. So he proposes a new two-party ECDSA and improves the efficiency of the signing operation by modifying the key generation algorithm in which one party encrypts the secret key once and another can use it to sign all the time. However, the protocol is hard to get rid of using homomorphic encryption, causing that key generation operation is heavy. Subsequently, Castagnos et al. [9] generalize Lindell's solution [3] and provide a generic construction for two-party ECDSA from hash proof systems (HPS). But their protocol still requires that the underlying HPS needs homomorphic properties and is also difficult to avoid the heavy key generation operation. Different from before, Doerner et al. [5, 8] point out that using homomorphic encryption leads both to poor performance and to reliance upon little-studied assumptions, and propose a two-party ECDSA based on the ECDSA assumption itself. But their scheme requires additional oblivious transfer to turn the multiplication on private inputs into the addition.

To the best of our knowledge, existing two-party signing protocols based on ECDSA changes the structure of original ECDSA signing algorithm a lot by using heavy MPC technique such as homomorphic encryption, oblivious transfer, etc. Therefore, extending the two-party scenario for existing ECDSA applications, the above schemes need to abandon the existing hardware and software implementation of the ECDSA signing algorithm and re-implement the two-party signing protocol. On the one hand, this method wastes the original signing implementation resources of existing ECDSA applications, on the other hand, the complex two-party signing structure increases the difficulty of implementation and application. Furthermore, in cryptosystems, security depends on its weakest component. Therefore, using more additional tools greatly affects security as well as efficiency. In particular, most two-party signing protocols based on ECDSA have almost completely remolded the original ECDSA signing operation by heavy MPC technique, which makes it difficult to be embedded in the existing ECDSA applications. Motivated by the above discussions, we ask the following challenging questions:

Without complicated MPC techniques, whether there is a simple yet fast two-party signing protocol based on the ECDSA?

1.1 Our Contributions

In this paper, we give a positive answer to the above question. First, we propose a variant of ECDSA called combinatorial ECDSA, and prove its security based on ECDSA itself. The Combinatorial ECDSA can be viewed as a combination of two ECDSA signatures and can easily be split into two parts. Then, benefiting from the structure of combinatorial ECDSA, we design a simple and fast two-party signing protocol without complicated MPC techniques, which can also be proven secure based on the ECDSA security.

Combinatorial ECDSA. From our observation that two ECDSA signatures generated by different signing keys and the same random number can be easily combined into a secure signature, we can construct the combinatorial ECDSA. Intuitively speaking, the signing key of our combinatorial ECDSA contains of two-part ECDSA signing keys x_1, x_2, and the verification key contains of $Q_{\text{add}} = (x_1 + x_2) \cdot G, Q_{\text{mul}} = x_1 \cdot x_2 \cdot G$ which is the combination of two ECDSA verification keys $Q_1 = x_1 \cdot G, Q_2 = x_2 \cdot G$, and the combinatorial ECDSA signature $s = k^{-1}(H(m) + rx_1)(H(m) + rx_2) \bmod q$ can be simply viewed as a multiplication of two ECDSA signature.

We can prove its security based on ECDSA. Informally, in the security proof, the simulator $\mathcal{S}$ can use the ECDSA signing oracle to simulate the signing oracle of the combinatorial ECDSA. $\mathcal{S}$ can choose a part of signing key x_2 and use ECDSA verification key $Q_1 = x_1 \cdot G$ to simulate the combinatorial ECDSA verification key $Q_{\text{add}} = Q + x_2 \cdot G, Q_{\text{mul}} = x_2 \cdot Q$, and simulate the signing oracle of the combinatorial ECDSA by querying the ECDSA signing oracle for message m to get an ECDSA signature $\sigma = (r, s)$ and compute the combinatorial ECDSA signature $\sigma' = (r, s \cdot (H(m) + rx_2) \bmod q)$.

Two-party signature. The Combinatorial ECDSA can be viewed as a combination of two ECDSA signatures. The special structure makes it easy to be split into two pieces and each piece also preserves the structure of ECDSA, at the same time, it can remain the security of ECDSA. Based on the combinatorial

ECDSA, we design a two-party signing protocol without complicated MPC techniques, which can also be proven secure based on the ECDSA.

Informally speaking, in our two-party signing protocol, each party takes advantage of a zero-knowledge proof of knowledge of discrete logarithm functionality to ensure that a point $R = k_1 \cdot k_2 \cdot G$ is uniformly distributed, and then based on this point each party gives ECDSA signature $s_1 = k_1^{-1}(H(m)+r\cdot x_1) \bmod q$ and $s_2 = k_2^{-1}(H(m) + r \cdot x_2) \bmod q$ respectively, where $(r_x, r_y) = R, r = r_x \bmod q$, and k_1, k_2 is the random number chosen by each party. The combinatorial ECDSA $s = s_1 \cdot s_2 \bmod q$ can be combined by computing the multiplication of two ECDSA signature s_1, s_2. We can instantiate the zero-knowledge proof functionality efficiently by the Schnorr proof [16] and the Fiat-Shamir transform [18].

Benefiting from the structure of combinatorial ECDSA, a whole signature can be easily combined from two ECDSA signature. Two main difficult problems of constructing distributed ECDSA were avoided, (i) inverted shared secret $k^{-1} = k_1^{-1} \cdot k_2^{-1}$, and (ii) computed the multiplication $k^{-1} \cdot x = k_1^{-1} \cdot k_2^{-1} \cdot x_1 \cdot x_2$ of shared secrets (k_1, x_1) and (k_2, x_2). In previous two-party ECDSA [1,3,5,9], above two problems need to be solved by using heavy MPC techniques and expensive zero-knowledge proofs. Therefore, our two-party signing protocol is more efficient than existing two-party signature based on ECDSA.

Upgrade to two-party scenario easily. In our protocol, the signing operation of each party is similar to the original ECDSA, and the signing results outputted by each party preserve the structure of the ECDSA signature, which can be verified easily by using the ECDSA verification algorithm and avoid expensive zero-knowledge proofs. Therefore, compared with the existing two-party ECDSA [1,3,5,9], our scheme can achieve the same good performance as ECDSA, and reuses the implementation of ECDSA greatly and upgrades the existing ECDSA application to the two-party scenario friendly.

To be specific, extending two-party scenarios from existing ECDSA applications, both parties in our two-party signing protocol can compute random point R together offline. After receiving a message m online, both parties can run the underlying ECDSA signing component to generate the ECDSA signature, respectively. Both signing values can be verified by using the ECDSA verification algorithm and can be combined into the combinatorial ECDSA signature directly. To upgrade the two-party scenario from the existing ECDSA application, our protocol does not modify the underlying ECDSA signing structure and can reuse the implementation of ECDSA greatly.

Performance. We show the theoretical analysis of our combinatorial ECDSA and two-party signature. Our schemes can achieve the same level of efficiency as ECDSA. By avoiding heavy MPC tools, our protocol can use efficient zero-knowledge proof [16,18] rather than expensive zero-knowledge proofs used in previous schemes [1,3,5,9]. In particular, additional zero-knowledge proof can be run in the offline signing phase. What's more, we realize our schemes based on the OpenSSL, and the running times of all algorithms in combinatorial ECDSA and two-party signing protocol are the levels of microseconds. Therefore, our protocol is more efficient than previous schemes [3,9], where the running times of the key generation and signing operation are at least the level of millisecond depicted in [9].

2 Preliminaries

2.1 Notations

For $n \in \mathbb{N}$, 1^n denotes the string of n ones. If S is a set then $s \leftarrow S$ denotes the operation of sampling an element s of S at random. A function is negligible in λ, written $\mathsf{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ.

2.2 Signature

A signature system consists of three algorithms $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$.

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (vk, sk)$. The key generation algorithm takes the security parameter as input and outputs a verification key vk and a secret key sk.
- $\mathsf{Sign}(sk, m) \rightarrow \sigma$. The signing algorithm takes the secret key sk and the message $m \in \mathcal{M}$ (where $\mathcal{M}$ is some fixed message space, possibly depending on n) as input, and outputs a signature σ.
- $\mathsf{Vrfy}(vk, m, \sigma) \rightarrow 0/1$. The verification algorithm takes the verification key vk, the message $m \in \mathcal{M}$, and the signature σ as input, and outputs either 0 or 1.

Correctness. The standard completeness requirement is that for any $(vk,\ sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and any $m \in \mathcal{M}$, we have $\mathsf{Vrfy}(vk, m, \mathsf{Sign}(sk, m)) \rightarrow 1$.

Security. Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary against the existentially unforgeable under chosen message attacks (EUF-CMA). Its advantage function $\mathbf{Adv}^{\text{euf-cma}}_{\Pi,\mathcal{A}}(1^\lambda)$ is defined as

$$\Pr\left[\mathsf{Vrfy}(vk, m^*, \sigma^*) = 1 : \begin{array}{l}(vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda);\\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(1^\lambda, vk);\end{array}\right]$$

Here, $\mathsf{Sign}(sk,\cdot)$ provides access to signing oracle with respect to sk. More precisely, $\mathsf{Sign}(sk,\cdot)$ returns $\mathsf{Sign}(sk, m)$ on input any $m \in \mathcal{M}$. $\mathcal{Q}$ be the set of all m queried by $\mathcal{A}$ to its oracle, $m^* \notin \mathcal{Q}$. The signature is EUF-CMA if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the above security experiment.

2.3 ECDSA

Let $\mathbb{G}$ be an Elliptic curve group of order q with base point (generator) G, and $pp = (\mathbb{G}, q, G)$ is the public parameter. H denotes the hash function, such as $\mathsf{SHA\text{-}256}$. ECDSA algorithm [11] on a message $m \in \{0,1\}^*$ is defined as follows.

- $\mathsf{KeyGen}(pp) \rightarrow (Q, x)$: uniformly choose a secret signing key $x \leftarrow \mathbb{Z}_q$, calculate the verification key $Q = x \cdot G$.
- $\mathsf{Sign}(x, m) \rightarrow (r, s)$:

1. Compute $m' \leftarrow \mathsf{H}(m)$
2. Choose a random $k \leftarrow \mathbb{Z}_q$
3. Compute $R = k \cdot G$. Let $R = (r_x, r_y)$ and compute $r = r_x \bmod q$. If $r = 0$, go back to Step 2
4. Compute $s' = k^{-1} \cdot (m' + r \cdot x) \bmod q$, sets $s = \min\{s', q - s'\}$, and output $\sigma = (r, s)$

- $\mathsf{Vrfy}(Q, m, \sigma) \rightarrow 0/1$:

1. Compute $m' \leftarrow \mathsf{H}(m)$, $(r'_x, r'_y) = s^{-1} \cdot (m' \cdot G + r \cdot Q)$
2. If $r = r'_x \bmod q$, output 1, otherwise, output 0

2.4 Two-Party Signature

The two-party signature [3, 9] contains distributed key generation protocol, distributed signing protocol, and verification algorithm. Two parties run distributed key generation protocol together to generate the shared signing key respectively and the verification key. After getting any message, two parties run the distributed signing protocol together to compute a signature. The signature can be verified by using the verification algorithm.

We review the security of two-party signature following [3, 9]. In the experiment of distributed signature Expt-DSign$_{\mathcal{A},\Pi_2^b}$, $b \in \{1, 2\}$, a PPT adversary $\mathcal{A}$ corrupting the party P_b in protocol Π_2 can interact with an oracle $\Pi_2^b(\cdot,\cdot)$ to sign any messages concurrently, which executes as another party P_{3-b}. $\Pi_2^b(\cdot,\cdot)$ is defined such that the distributed key generation is first run once and then the signing protocols can be executed concurrently. The oracle receives two inputs: the first is a session identifier and the second is either an input or a next message. It works as follows:

In the key generation phase:

- Upon receiving $(0, 0)$ first, the oracle initializes a machine M and acts as the party P_{3-b} to reply.
- Upon receiving $(0, m)$, the oracle hands M the message m as its next incoming message and returns M's reply.
- Upon receiving (sid, m) is received where $sid \neq 0$, the oracle returns $\perp$.

In the signing phase:

- Upon receiving (sid, m), and this is the first oracle query with this identifier sid, the oracle initializes a new machine M_{sid} with the key share and any state stored by M at the end of the key generation phase. The M_{sid} executes as the party P_{3-b} with session identifier sid and input message m to be signed.
- Upon receiving (sid, m), and this is not the first oracle query with this identifier sid, the oracle hands M_{sid} the incoming message m and returns the next message sent by M_{sid}. If M_{sid} concludes, then the output obtained by M_{sid} is returned.

The adversary $\mathcal{A}$ controlling P_b with oracle access to $\Pi_2^b(\cdot,\cdot)$ "wins" if it can forge a signature on a message that is not queried. For a detailed explanation we refer the reader to [3]. Let $\Pi_2 = (\mathsf{IKeyGen}, \mathsf{ISign}, \mathsf{Vrfy})$

be a two-party signature, where the $\mathsf{IKeyGen}$ denotes two-party key generation protocol, and the ISign denotes two-party signing protocol. The experiment Expt-DSign$_{\mathcal{A},\Pi_2^b}$, $b \in \{1,2\}$ is defined as follows.

Let $\mathcal{A}$ be a PPT adversary against two-party signature Π_2^b. Its advantage function $\mathbf{Adv}_{\Pi_2^b,\mathcal{A}}(1^\lambda)$ is defined as

$$\Pr\left[\mathsf{Vrfy}(vk, m^*, \sigma^*) = 1 : (m^*, \sigma^*) \leftarrow \mathcal{A}^{\Pi_2^b(\cdot,\cdot)}(1^\lambda)\right]$$

Let $\mathcal{Q}$ be the set of all inputs m such that (sid, m) was queried by $\mathcal{A}$ to its oracle $\Pi_2^b(\cdot,\cdot)$ as the first query with identifier $sid \neq 0$, $m^* \notin \mathcal{Q}$. $\mathsf{Vrfy}(vk, m^*, \sigma^*) = 1$, where vk is the verification key output by P_{3-b} from the key generation phase, and Vrfy is as specified in Π_2. The two-party signing protocol is secure if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the above security experiment.

2.5 Ideal Functionalities

We review three ideal functionalities used in [3,9], including the ideal commitment functionality $\mathcal{F}_{\text{com}}$, the ideal zero-knowledge functionality $\mathcal{F}_{\text{zk}}$, and the committed non-interactive zero-knowledge functionality $\mathcal{F}_{\text{com-zk}}^{\mathsf{R}}$. For a detailed explanation of these ideal functionalities we refer the reader to [3].

The ideal commitment functionality $\mathcal{F}_{\text{com}}$ [19,20] works with parties P_1 and P_2, formally defined as follows.

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

A standard ideal zero-knowledge functionality [21] is defined by $((x, w), \lambda) \leftarrow (\lambda, (x, \mathsf{R}(x, w)))$, where λ denotes the empty string. For a relation R, the ideal zero-knowledge functionality is denoted by $\mathcal{F}_{\text{zk}}^{\mathsf{R}}$, formally defined as follows.

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

For a relation R, the commitments to non-interactive zero-knowledge proofs of knowledge used by the ideal commitment functionality $\mathcal{F}_{\text{com}}$ is denoted by $\mathcal{F}_{\text{com-zk}}^{\mathsf{R}}$, formally defined as follows.

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

3 The Combinatorial ECDSA Based on ECDSA

In this section, we construct the combinatorial ECDSA (cECDSA), which consists of three algorithms $\Pi_1 = (\mathsf{KeyGen},\mathsf{Sign},\mathsf{Vrfy})$ defined as follows. Let $\mathbb{G}$ be an Elliptic curve group of order q with base point (generator) G, and $pp = (\mathbb{G}, q, G)$ is the public parameter. H denotes the hash function, like ECDSA.

- $\mathsf{KeyGen}(pp) \to (Q, x)$: uniformly choose $x_1, x_2 \leftarrow \mathbb{Z}_q$, calculate $Q_1 = x_1 \cdot G, Q_2 = x_2 \cdot G, Q_{\text{add}} = Q_2 + Q_1, Q_{\text{mul}} = x_1 \cdot x_2 \cdot G$. The secret signing key is $x = (x_1, x_2)$ and the verification key is $Q = (Q_{\text{add}}, Q_{\text{mul}})$.
- $\mathsf{Sign}(x, m) \to (r, s)$:

1. Compute $m' \leftarrow \mathsf{H}(m)$
2. Choose a random $k \leftarrow \mathbb{Z}_q$, compute $R = k \cdot G$, and let $R = (r_x, r_y)$
3. Compute $r = r_x \bmod q$. If $r = 0$, go back to Step 2
4. Compute $s' = k^{-1} \cdot (m' + r \cdot x_1) \cdot (m' + r \cdot x_2) \bmod q$, sets $s = \min\{s', q - s'\}$, and output (r, s)

- $\mathsf{Vrfy}(Q, m, \sigma) \to 0/1$:

1. Compute $m' \leftarrow \mathsf{H}(m)$, and $(r'_x, r'_y) = s^{-1} \cdot (m'^2 \cdot G + r \cdot m' \cdot Q_{\text{add}} + r^2 \cdot Q_{\text{mul}})$
2. If $r = r'_x \bmod q$, output 1, otherwise, output 0

Correctness. For any $\mathsf{KeyGen}(pp) \to (Q, x)$ and $m \in \{0,1\}^*$, the signature is $\sigma = (r, s)$, where $r = r_x \bmod q$, $(r_x, r_y) = k \cdot G$, $s = k^{-1} \cdot (m' + r \cdot x_1) \cdot (m' + r \cdot x_2) \bmod q$. We can verify the signature as follows.

$$\begin{aligned} s^{-1}(m'^2 \cdot G + r \cdot m' \cdot Q_{\text{add}} + r^2 \cdot Q_{\text{mul}}) &= s^{-1}(m'^2 + r \cdot m' \cdot (x_1 + x_2) + r^2 \cdot x_1 \cdot x_2) \cdot G \\ &= s^{-1} \cdot (m' + r \cdot x_1)(m' + r \cdot x_2) \cdot G \\ &= k \cdot G \end{aligned}$$

Therefore, $(r'_x, r'_y) = k \cdot G$, $r = r'_x \bmod q$, $\mathsf{Vrfy}(Q, m, \sigma) \to 1$. The combinatorial ECDSA satisfies the correctness.

Lemma 1. Our combinatorial ECDSA defined above is existentially unforgeable under chosen-message attacks if ECDSA is secure.

Proof. Our proof works by showing that, for any PPT adversary $\mathcal{A}$ breaking the EUF-CMA security of the combinatorial ECDSA with non-negligible probability, we can construct a PPT reduction algorithm $\mathcal{S}$ who can break the EUF-CMA security of ECDSA with non-negligible probability.

Assume that $\mathcal{A}$ can output a new combinatorial ECDSA signature (r, s) with non-negligible probability $\varepsilon(\lambda)$ as follows:

$$\mathbf{Adv}^{\text{euf-cma}}_{\text{cECDSA},\mathcal{A}}(1^\lambda) = \Pr[\text{Expt-cECDSA}_{\mathcal{A}}(1^\lambda) = 1] = \varepsilon(\lambda).$$

Then, we use $\mathcal{A}$ to build a reduction algorithm $\mathcal{S}$ to break the security of ECDSA. That is, $\mathcal{S}$ is given an ECDSA verification key $Q_1 = x_1 \cdot G$ as input and tries to give a new ECDSA signature with the help of an ECDSA signing oracle $\mathcal{O}(\cdot)$ that takes in any message and outputs an ECDSA signature. $\mathcal{S}$ works as follows:

1. Choose a random value $x_2 \leftarrow \mathbb{Z}_q$ and compute $Q_2 = x_2 \cdot G, Q_{\text{add}} = Q_1 + Q_2, Q_{\text{mul}} = x_2 \cdot Q_1, Q = (Q_{\text{add}}, Q_{\text{mul}})$.
2. Run $\mathcal{A}$ on input the verification key Q. When $\mathcal{A}$ makes a query of any message m_i to the combinatorial ECDSA signing oracle $\mathcal{O}_1$, $\mathcal{S}$ makes a query of message m_i to the ECDSA signing oracle $\mathcal{O}$ and gets an ECDSA signature $\sigma_i = (r_i, s_i = k_i^{-1}(r_i \cdot x_1 + m'_i))$. Then $\mathcal{S}$ can compute a combinatorial ECDSA signature $\sigma'_i = (r_i, s'_i = k_i^{-1}(r_i \cdot x_1 + m'_i)(r_i \cdot x_2 + m'_i))$ and reply the signature σ'_i to $\mathcal{A}$.
3. When $\mathcal{A}$ outputs a new combinatorial ECDSA signature $\sigma' = (r, s = k^{-1}(r \cdot x_1 + m')(r \cdot x_2 + m'))$ of a message m. Then, $\mathcal{S}$ can outputs a new ECDSA signature $\sigma = (r, s(r \cdot x_2 + m')^{-1} = k^{-1}(r \cdot x_1 + m'))$.

As we can see, $\mathcal{S}$ gives a perfect simulation of $\mathcal{A}$'s views in the experment-cECDSA. Since $\mathcal{A}$ outputs a new combinatorial ECDSA signature with non-negligible probability $\varepsilon(\lambda)$, $\mathcal{S}$ can also output a forge of ECDSA with non-negligible probability $\varepsilon(\lambda)$ in the experiment-ECDSA. Its advantage function:

$$\mathbf{Adv}^{\text{euf-cma}}_{\text{ECDSA},\mathcal{S}}(1^\lambda) = \Pr[\text{Expt-ECDSA}_{\mathcal{S}}(1^\lambda) = 1] = \varepsilon(\lambda).$$

We conclude that $\mathcal{S}$ can break the EUF-CMA security of ECDSA, a contradiction. Therefore, combinatorial ECDSA defined above is existentially unforgeable under chosen message attacks.

4 Two-Party Signature

In this section, we present our two-party combinatorial ECDSA (TP-cECDSA) based on ECDSA. Following [3], our protocol is also presented in the $\mathcal{F}_{\text{zk}}$ and $\mathcal{F}_{\text{com-zk}}$ hybrid model. We use zero-knowledge functionality $\mathcal{F}^{\mathsf{R}_{\text{DL}}}_{\text{zk}}$ [3] for the relation $\mathsf{R}_{\text{DL}} = \{(\mathbb{G}, G, q, P, w) | P = w \cdot G\}$ of discrete log values (relative to the given group). We use the standard Schnorr proof [16, 18] for this. For convenience, we omit the group description $(\mathbb{G}, G, q)$ and assume that all values (Elliptic curve points) received are not equal to 0, and each party receiving zero aborts.

4.1 Distributed Key Generation

The parties together generate a tuple of random group element Q. P_1 chooses a random number x_1, computes and sents a random point $Q_1 = x_1 \cdot G$ along with a zero-knowledge proof of knowledge to P_2. P_2 also chooses a random number x_2, computes and sends $Q_2 = x_2 \cdot G$ along with a zero-knowledge proof of knowledge to P_1. The output is the point tuple $Q = (Q_{\text{add}}, Q_{\text{mul}})$, where $Q_{\text{add}} = Q_1 + Q_2, Q_{\text{mul}} = x_1 \cdot x_2 \cdot G$. See Protocol 1 for a full description.

The protocol 1 is simulatable. In particular, assume that P_1 is corrupted. A simulator can define the value sent by P_2 to be $Q_2 = Q_{\mathrm{ECDSA}}$, where Q_{ECDSA} is the verification key in the experiment-ECDSA. If P_2 is corrupted, a simulator can define the value sent by P_1 to be $Q_1 = Q_{\mathrm{ECDSA}}$.

PROTOCOL 1. (Key Generation Protocol $\mathsf{lKeyGen}(pp)$) Given input $pp = (\mathbb{G}, G, q)$ and security parameter 1^λ, work as follows:
1. P_1's first message:
(a) P_1 runs ECDSA.$\mathsf{KeyGen}(pp) \to (x_1, Q_1 = x_1 \cdot G)$.
(b) P_1 sends $(\mathsf{prove}, 1, Q_1, x_1)$ to$\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$.
2. P_2's first message:
(a) P_2 receives $(\mathsf{proof}, 1, Q_1)$ from $\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$. If not, it aborts.
(b) P_2 runs ECDSA.$\mathsf{KeyGen}(pp) \to (x_2, Q_2 = x_2 \cdot G)$.
(c) P_2 sends $(\mathsf{prove}, 2, Q_2, x_2)$ to$\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$.
3. P_1's verification:
(a) P_1 receives $(\mathsf{proof}, 2, Q_2)$ from $\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$. If not, it aborts.
4. Output:
(a) P_1 computes $Q_{\mathrm{add}} = Q_2 + Q_1, Q_{\mathrm{mul}} = x_1 \cdot Q_2, Q = (Q_{\mathrm{add}}, Q_{\mathrm{mul}})$ and stores (x_1, Q).
(b) P_2 computes $Q_{\mathrm{add}} = Q_2 + Q_1, Q_{\mathrm{mul}} = x_2 \cdot Q_1, Q = (Q_{\mathrm{add}}, Q_{\mathrm{mul}})$ and stores (x_2, Q).

4.2 Two-Party Signing Protocol

The main idea of our two-party signing protocol is as follows. First, both parties run a similar "coin tossing protocol" in order to obtain a random point R that will be used in generating the signature. After this, the parties P_1 and P_2 hold the random number k_1 and k_2, respectively, where $R = k_1 \cdot k_2 \cdot G = (r_x, r_y), r = r_x$. Then, the parties receive the message m. P_2 can compute part of signature $(r, k_2^{-1}(r \cdot x_2 + \mathsf{H}(m)))$, and P_1 can compute another part of signature $(r, k_1^{-1}(r \cdot x_1 + \mathsf{H}(m)))$. Both parts of signature can be easily verified and can be combined into a cECDSA signature.

In the signing protocol, P_1 and P_2 first take in the signing key and the message m, and a unique session id sid, and then verify that sid has not been used before (if it has been, the protocol is not executed).

PROTOCOL 2. (Signing Protocol $\mathsf{lSign}(sid, m)$) Given the outputted from Protocol 1 and the message m, and a unique session id sid as input. Protocol 2 is described as follows:
1. P_1's first message:
(a) P_1 chooses a random $k_1 \leftarrow \mathbb{Z}_q$ and computes $R_1 = k_1 \cdot G$.
(b) P_1 sends $(\mathsf{com\text{-}prove}, sid||1, R_1, k_1)$ to $\mathcal{F}_{\text{com-zk}}^{\mathsf{R}_{\mathrm{DL}}}$.
2. P_2's first message:
(a) P_2 receives $(\mathsf{proof\text{-}receipt}, sid||1)$ from $\mathcal{F}_{\text{com-zk}}^{\mathsf{R}_{\mathrm{DL}}}$.
(b) P_2 chooses a random $k_2 \leftarrow \mathbb{Z}_q$ and computes $R_2 = k_2 \cdot G$.
(c) P_2 sends $(\mathsf{prove}, sid||2, R_2, k_2)$ to $\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$.
3. P_1's second message:
(a) P_1 receives $(\mathsf{proof}, sid||2, R_2)$ from $\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$; if not, it aborts.
(b) P_1 sends $(\mathsf{decom\text{-}proof}, sid||1)$ to $\mathcal{F}_{\text{com-zk}}$.
4. P_2's second message:
(a) P_2 receives $(\mathsf{decom\text{-}proof}, sid||1, R_1)$ from $\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$; if not, it aborts.
(b) P_2 computes $R = k_2 \cdot R_1$. Denote $R = (r_x, r_y)$. After receiving m, P_2 computes $r = r_x \bmod q$,$s_2 = k_2^{-1}(r \cdot x_2 + \mathsf{H}(m)) \bmod q$.
(c) P_2 sends s_2 to P_1.
5. P_1 generates output:
(a) P_1 computes $R = k_1 \cdot R_2$. Denote $R = (r_x, r_y)$. Then, P_1 computes $r = r_x \bmod q$.
(b) P_1 computes $s' = k_1^{-1}(r \cdot x_1 + \mathsf{H}(m)) \cdot s_2 \bmod q$, and sets $s = \min\{s', q - s'\}$ (this ensures that the signature is always the smaller of the two possible values).
(c) P_1 verifies that (r, s) is a valid signature with public key Q. If yes it outputs the cECDSA signature (r, s); otherwise, it aborts.

Offline/Online. As we can see, the first three steps in the above signing protocol run by two parties are used to agree on the random point $R = k_1 \cdot k_2 \cdot G$, and the message to be signed is only used in the last step. Thus, it is possible to run the first three steps to pre-store many random points in the offline phase. In the online signing phase, after receiving the message m, each party can take advantage of stored random points to compute signatures together. As is shown in our protocol 2, the signing operation of P_1 and P_2 is similar to the original signing operation of ECDSA [1].

Output to both parties, or a combiner. Observe that since the validity of the signature can be checked by any other parties, it is possible for P_1 to send P_2 part of a signature, or even both P_1 and P_2 to send combiner [2] part of a signature without verifying the part of signature sent by another party. Since every part of the signature can be seen as an ECDSA signature, this will not affect security at all. In particular, the combiner can easily combine the whole cECDSA signature and verifies its validation by running the verification algorithm of cECDSA.

4.3 Proof of Security

In this section, we prove that Π_2 comprised of Protocols 1 and 2 for key generation and signing, respectively, constitutes a secure two-party signing protocol. Our proof of security based on the hybrid model with ideal functionalities $\mathcal{F}_{\text{com}}$, $\mathcal{F}_{\text{zk}}$, $\mathcal{F}^{\mathsf{R}}_{\text{com-zk}}$ [3, 23, 24].

Lemma 2. Assume that ECDSA is existentially unforgeable under chosen-message attack and that the zero-knowledge proofs and commitments are as described. Then, Protocols 1 and 2 constitute a secure two-party signing protocol.

Proof. We separately prove security for the case of a corrupted $P_b, b \in \{1, 2\}$ and define two hybrid experiments Game_{1_b}, Game_{2_b} as follows.

Game_{1_b}: The PPT adversary $\mathcal{A}$ corrupts $P_b, b \in \{1, 2\}$. Another party P_{3-b} interacts with $\mathcal{A}$ in real execution of protocol 1 and 2.

Game_{2_b}: The PPT adversary $\mathcal{A}$ corrupts $P_b, b \in \{1, 2\}$. We construct a PPT simulator $\mathcal{S}$ to interact with $\mathcal{A}$ like P_{3-b} in Game_{1_b}, except that $\mathcal{S}$ sets the verification key Q^* in experiment-ECDSA as Q_{3-b} in the experiment-TP-cECDSA and interacts with $\mathcal{A}$ based on the ECDSA signing oracle.

In Game_{2_1}, there exist a PPT adversary $\mathcal{A}$ that corrupts P_1. We construct a PPT simulator $\mathcal{S}$ to simulates the execution for $\mathcal{A}$. Formally:

1. $\mathcal{S}$ takes $(1^\lambda, Q^*)$ as input and has the access to the signing oracle of the experiment ECDSA, where Q^* is the public verification key in experiment ECDSA.

2. $\mathcal{S}$ runs $\mathcal{A}$ on input 1^λ and simulates oracle Π_2^1 for $\mathcal{A}$, answering as described in the following steps:

   a) $\mathcal{S}$ replies $\perp$ to all queries from $\mathcal{A}$ before it queries $(0, 0)$.

   b) After $\mathcal{A}$ sends $(0, 0)$ to the oracle Π_2^1, $\mathcal{S}$ receives $(0, m_1)$ which is P_1's first message in the key generation subprotocol (any other query is ignored). $\mathcal{S}$ computes the oracle reply as follows:

      i. $\mathcal{S}$ parses m_1 into the form $(\mathsf{prove}, 1, Q_1, x_1)$ that P_1 sends to $\mathcal{F}_{\text{zk}}^{\mathsf{R}_{\text{DL}}}$ in the hybrid model.

      ii. $\mathcal{S}$ verifies that $Q_1 = x_1 \cdot G$. If yes, then it sets $Q_2 = Q^*$. Then $\mathcal{S}$ sets the oracle reply of Π_2^1 to be $(\mathsf{proof}, 2, Q_2)$ and internally hands this to $\mathcal{A}$ (as if sent by $\mathcal{F}_{\text{zk}}^{\mathsf{R}_{\text{DL}}}$).

   c) Finally, $\mathcal{A}$ can compute $Q_{\text{add}} = Q_2 + Q_1 = Q^* + Q_1, Q_{\text{mul}} = x_1 \cdot Q_2, Q = (Q_{\text{add}}, Q_{\text{mul}})$ and stores (x_1, Q). $\mathcal{S}$ can compute $Q_{\text{add}} = Q_2 + Q_1, Q_{\text{mul}} = x_1 \cdot Q_2, Q = (Q_{\text{add}}, Q_{\text{mul}})$ and stores (x_1, Q). The distributed key generation phase is completed.

   d) Upon receiving a query of the form (sid, m) where sid is a new session identifier, $\mathcal{S}$ queries its signing oracle in experiment-ECDSA with m and receives back an ECDSA signature (r, s). Using the ECDSA verification procedure, $\mathcal{S}$ computes the Elliptic curve point R. Then, queries received by $\mathcal{S}$ from $\mathcal{A}$ with identifier sid are processed as follows:

[1] In offline/online mode of ECDSA, the signer can pre-store many random numbers and compute the random points in the offline, and then he uses the stored random numbers and points to sign any messages in the online.

[2] In the threshold signature, there always exists a combiner [22] that can combine every part of signature sent from each party into a whole signature.

    i. The first message (sid, m_1) is processed by first parsing the message m_1 as $(\mathsf{com\text{-}prove}, sid||1,R_1,k_1)$. If $R_1 = k_1 \cdot G$ then $\mathcal{S}$ sets $R_2 = k_1^{-1} \cdot R$; else it chooses R_2 at random. $\mathcal{S}$ sets the oracle reply to $\mathcal{A}$ to be the message $(\mathsf{proof}, sid||2,R_2)$ that $\mathcal{A}$ expects to receive. (Note that the value R_2 is computed using R from the ECDSA signature and k_1 as sent by $\mathcal{A}$.)

    ii. The second message (sid, m_2) is processed by parsing the message m_2 as $(\mathsf{decom\text{-}proof}, sid||1)$ from $\mathcal{A}$. If $R_1 \neq k_1 \cdot G$ then $\mathcal{S}$ simulates P_2 aborting and the experiment concludes (since the honest P_2 no longer participates in any executions of the protocol and so all calls to Π_2^1 are ignored). Otherwise, $\mathcal{S}$ computes $s_2 = k_1 \cdot s$ where s is the value from the signature received from experiment-ECDSA, and sets the oracle reply to $\mathcal{A}$ to be s_2.

3. Whenever $\mathcal{A}$ halts and outputs a pair $(m^*, \sigma = (r^*, s^*))$, $\mathcal{S}$ outputs halts and $(m^*, \sigma^* = (r^*, s^*/(\mathsf{H}(m^*) + r^* \cdot x_1) \bmod q))$.

As we can see, if $\mathcal{A}$ outputs a valid forgery $(m^*, \sigma = (r^*, s^*))$ of our two-party signature, where m^* is not queried and $s^* = k^*(\mathsf{H}(m^*) + r^* \cdot x_1)(\mathsf{H}(m^*) + r^* \cdot x) \bmod q$, $R^* = k^* \cdot G = (r_x^*, r_y^*)$, $r^* = r_x^* \bmod q$, $\mathsf{H}(\cdot)$ denotes a hash function. Therefore, $\mathcal{S}$ can computes a valid forgery $(m^*, \sigma^* = (r^*, s^*/(\mathsf{H}(m^*) + r^* \cdot x_1) \bmod q) = (m^*, \sigma^* = (r^*, k^*(\mathsf{H}(m^*) + r^* \cdot x) \bmod q)$ of ECDSA, where m^* is not queried, $Q^* = x \cdot G$ is the verification key of ECDSA, and x is the signing key of ECDSA.

In Game_{2_2}, there exist a PPT adversary $\mathcal{A}$ that corrupts P_2. We construct a PPT simulator $\mathcal{S}$ to simulates the execution for $\mathcal{A}$. Formally:

1. $\mathcal{S}$ takes $(1^\lambda, Q^*)$ as input and has the access to the signing oracle of the experiment ECDSA, where Q^* is the public verification key in experiment ECDSA.

2. $\mathcal{S}$ runs $\mathcal{A}$ on input 1^λ and simulates oracle Π_2^2 for $\mathcal{A}$, answering as described in the following steps:

   a) $\mathcal{S}$ replies $\perp$ to all queries from $\mathcal{A}$ before it queries $(0, 0)$.

   b) After $\mathcal{A}$ sends $(0, 0)$ to the oracle Π_2^2, $\mathcal{S}$ sets $Q_1 = Q^*$. Then $\mathcal{S}$ sets the oracle reply of Π_2^2 to be $(\mathrm{proof}, 1, Q_1)$ and internally hands this to $\mathcal{A}$ (as if sent by $\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$). $\mathcal{S}$ receives $(0, m_1)$ which is P_2's first message in the key generation subprotocol. $\mathcal{S}$ parses m_1 into the form $(\mathsf{prove}, 2, Q_2, x_2)$ that P_2 sends to $\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$ in the hybrid model. $\mathcal{S}$ verifies that $Q_2 = x_2 \cdot G$. If no, $\mathcal{S}$ simulates P_1 aborting.

   c) Finally, $\mathcal{A}$ can compute $Q_{\mathrm{add}} = Q_2 + Q_1$, $Q_{\mathrm{mul}} = x_2 \cdot Q_1$, $Q = (Q_{\mathrm{add}}, Q_{\mathrm{mul}})$ and stores (x_2, Q). $\mathcal{S}$ can compute $Q_{\mathrm{add}} = Q_2 + Q_1, Q_{\mathrm{mul}} = x_2 \cdot Q_1, Q = (Q_{\mathrm{add}}, Q_{\mathrm{mul}})$ and stores (x_2, Q). The distributed key generation phase is completed.

   d) Upon receiving a query of the form (sid, m) where sid is a new session identifier, $\mathcal{S}$ computes the oracle reply to be $(\mathsf{proof\text{-}receipt}, sid||1)$ as $\mathcal{A}$ expects to receive, and hands it to $\mathcal{A}$. Next, $\mathcal{S}$ queries its signing oracle in experiment-ECDSA with m and receives back a signature (r, s). Using the ECDSA verification procedure, $\mathcal{S}$ computes the Elliptic curve point R. Then, queries received by $\mathcal{S}$ from $\mathcal{A}$ with identifier sid are processed as follows:

      i. The first message (sid, m_1) is processed by first parsing the message m_1 as $(\mathsf{prove}, sid||2,R_2,k_2)$ that $\mathcal{A}$ sends to $\mathcal{F}_{\mathrm{zk}}^{\mathsf{R}_{\mathrm{DL}}}$. $\mathcal{S}$ verifies that $R_2 = k_2 \cdot G$ and that R_2 is a non-zero point on the curve; otherwise, it simulates P_1 aborting. $\mathcal{S}$ computes $R_1 = k_2^{-1} \cdot R$ and sets the oracle reply to be $(\mathsf{decom\text{-}proof}, sid||1, R_1)$ as if coming from $\mathcal{F}_{\mathrm{com\text{-}zk}}^{\mathsf{R}_{\mathrm{DL}}}$.

      ii. The second message (sid, m_2) is processed by parsing m_2 as s_2. $\mathcal{S}$ computes $s' = k_2 \cdot s_2 \cdot s$ and check whether it is a combinatorial ECDSA or not. If yes, $\mathcal{S}$ generates signature $\sigma' = (r, s')$. Otherwise, $\mathcal{S}$ simulates P_1 aborting.

3. Whenever $\mathcal{A}$ halts and outputs a pair $(m^*, \sigma = (r^*, s^*))$, $\mathcal{S}$ outputs halts and $(m^*, \sigma^* = (r^*, s^*/(\mathsf{H}(m^*) + r^* \cdot x_2) \bmod q)$.

As we can see, if $\mathcal{A}$ outputs a valid forgery $(m^*, \sigma = (r^*, s^*))$ of our two-party signature, where m^* is not queried and $s^* = k^*(\mathsf{H}(m^*) + r^* \cdot x_2)(\mathsf{H}(m^*) + r^* \cdot x) \bmod q$, $R^* = k^* \cdot G = (r_x^*, r_y^*)$, $r^* = r_x^* \bmod q$, $\mathsf{H}(\cdot)$ denotes the hash function. Therefore, $\mathcal{S}$ can computes a valid forgery $(m^*, \sigma^* = (r^*, s^*/(\mathsf{H}(m^*) + r^* \cdot x_2) \bmod q) = (\mathsf{H}(m^*), \sigma^* = (r^*, k^*(\mathsf{H}(m^*) + r^* \cdot x) \bmod q)$ of ECDSA, where m^* is not queried and $Q^* = x \cdot G$, x is the signing key of ECDSA.

We set the probability of a PPT adversary $\mathcal{A}$ outputs a valid forgery of our two-party signature in Game$_{i_b}$ is $\Pr[\text{Expt-DSign}^{i}_{\mathcal{A},\Pi_2^b}(1^\lambda)=1]=\mathbf{Adv}_{\Pi_2^b,i,\mathcal{A}}(1^\lambda)$, where $i\in\{1,2\}$,$b\in\{1,2\}$.

We prove the security of the protocol in the $\mathcal{F}_{\text{zk}}^{\mathsf{R}_{\text{DL}}}$ and $\mathcal{F}_{\text{com-zk}}^{\mathsf{R}_{\text{DL}}}$ hybrid model for relations R_{DL}. Note that if the commitment and zero-knowledge protocols are UC-secure [19], then this means that the output in the hybrid and real protocols is computationally indistinguishable.

In Game$_{2_b}$, the simulator $\mathcal{S}$ interacts with $\mathcal{A}$ based on the signing oracle in experiment-ECDSA. As we can see, $\mathcal{S}$ can interact with $\mathcal{A}$ as the oracle Π_2^b. Therefore, $\mathcal{S}$ can simulate the view of $\mathcal{A}$ perfectly in either Game$_{1_b}$ or Game$_{2_b}$. That is, $\Pr[\text{Expt-DSign}^{1}_{\mathcal{A},\Pi_2^b}(1^\lambda)=1]=\Pr[\text{Expt-DSign}^{2}_{\mathcal{A},\Pi_2^b}(1^\lambda)=1]$.

Our proof works by showing that, in the Game$_{2_b}$, for any PPT adversary $\mathcal{A}$ attacking our two-party signature in the experiment-TP-cECDSA with the probability $\Pr[\text{Expt-DSign}^{2}_{\mathcal{A},\Pi_2^b}(1^\lambda)=1]=\mathbf{Adv}_{\Pi_2^b,2,\mathcal{A}}(1^\lambda)$, we can construct a simulator $\mathcal{S}$ who forges an ECDSA signature in Π the experiment-ECDSA. Because ECDSA is EUF-CMA secure, there exists a negligible function negl such that,

$$\Pr[\text{Expt-Sign}_{\mathcal{A},\Pi}(1^\lambda)=1]=\mathbf{Adv}^{\text{euf-cma}}_{\Pi,\mathcal{A}}(1^\lambda)\leqslant\mathsf{negl}(\lambda).$$

Therefore, we conclude that

$$\mathbf{Adv}_{\Pi_2^b,1,\mathcal{A}}(1^\lambda)=\Pr[\text{Expt-DSign}^{1}_{\mathcal{A},\Pi_2^b}(1^\lambda)=1]\leqslant\mathsf{negl}(\lambda),$$

thus Π_2^b is secure, separately for $b=1$ and $b=2$.

5 Efficiency and Experimental Results

5.1 Theoretical Analysis

As is shown in Table 1 and Figure 1, we review ECDSA and show our combinatorial ECDSA and two-party signature. In Table 2, we give the theoretical analysis of communication cost and efficiency about ECDSA, combinatorial ECDSA, Lindell's two-party ECDSA [3] and our two-party signature, respectively. For convenience, we omit some simple operations of modular multiplication and addition. Intuitively speaking, Lindell's two-party ECDSA [3] uses Paillier homomorphic encryption to realize the multiplication of shared secret key and a random value, and bases on a special zero-knowledge proof to prove that a value encrypted in a given ciphertext is the discrete log of a given Elliptic curve point. The above two operations are the main factors affecting the communication cost and efficiency of their two-party signature. Our combinatorial ECDSA signature can be seen as a combination of two ECDSA signatures in the structure, only increases twice point multiplication and once point addition operation in the key generation and verification phase. Benefit from these minimal costs, we greatly simplify the inversion and multiplication of shared secrets in designing a two-party signature.

Table 1 Comparisons of ECDSA and combinatorial ECDSA

algorithms	ECDSA	cECDSA
key generation	$x\leftarrow\mathbb{Z}_q$, $vk=Q=x\cdot G$	$x_1,x_2\leftarrow\mathbb{Z}_q,Q_1=x_1\cdot G,Q_2=x_2\cdot G$ $Q_{\text{add}}=Q_1+Q_2,Q_{\text{mul}}=x_1\cdot x_2\cdot G$ $vk=Q=(Q_{\text{add}},Q_{\text{mul}})$
signing	$k\leftarrow\mathbb{Z}_q,R=kG=(r_x,r_y)$, $r=r_x \bmod q$, $s=k^{-1}(m+r\cdot x) \bmod q$	$k\leftarrow\mathbb{Z}_q,R=kG=(r_x,r_y)$ $r=r_x \bmod q$, $s=k^{-1}(m+r\cdot x_1)(m+r\cdot x_2) \bmod q$
verification	$s^{-1}\cdot m\cdot G+s^{-1}\cdot r\cdot Q=(r'_x,r'_y)$, $r=^? r'_x$	$s^{-1}\cdot m^2\cdot G+s^{-1}\cdot r\cdot m\cdot Q_{\text{add}}+s^{-1}\cdot r^2\cdot Q_{\text{mul}}=(r'_x,r'_y)$, $r=^? r'_x$

More precisely, with regard to Lindell's two-party ECDSA [3], in the key generation phase, each party needs four times interactions, and P$_1$ computes key generation and encryption of Paillier scheme, runs discrete log proof, proves Paillier public-key validity and runs the special zero-knowledge proof PDL-ZK. P$_2$ needs to run discrete log proof and verify the public-key validity and the proof of PDL-ZK.

P_1	the key generation algorithm	P_2
$x_1 \leftarrow \mathbb{Z}_q$		$x_2 \leftarrow \mathbb{Z}_q$
$Q_1 = x_1 \cdot G$	$\xrightarrow{(\textsf{prove},1,Q_1,x_1)} \mathcal{F}_{\text{zk}}^{\mathsf{R}_{\text{DL}}} \xrightarrow{(\textsf{proof},1,Q_1)}$	$Q_2 = x_2 \cdot G$
	$\xleftarrow{(\textsf{proof},2,Q_2)} \mathcal{F}_{\text{zk}}^{\mathsf{R}_{\text{DL}}} \xleftarrow{(\textsf{prove},2,Q_2,x_2)}$	
$Q_{\text{add}} = Q_1 + Q_2$		$Q_{\text{add}} = Q_1 + Q_2$
$Q_{\text{mul}} = x_1 \cdot Q_2$		$Q_{\text{mul}} = x_2 \cdot Q_1$
$Q = (Q_{\text{add}}, Q_{\text{mul}})$		$Q = (Q_{\text{add}}, Q_{\text{mul}})$
P_1	the signing algorithm	P_2
$k_1 \leftarrow \mathbb{Z}_q$		
$R_1 = k_1 \cdot G$	$\xrightarrow{(\textsf{com-prove},sid\|\|1,R_1,k_1)} \mathcal{F}_{\text{com-zk}}^{\mathsf{R}_{\text{DL}}} \xrightarrow{(\textsf{proof-receipt},sid\|\|1)}$	$k_2 \leftarrow \mathbb{Z}_q$
		$R_2 = k_2 \cdot G$
	$\xleftarrow{(proof,sid\|\|2,R_2)} \mathcal{F}_{\text{zk}}^{\mathsf{R}_{\text{DL}}} \xleftarrow{(\textsf{prove},sid\|\|2,R_2,k_2)}$	
P_1 aborts if $(\textsf{proof}, sid\|\|2, R_2)$ not received		
	$\xrightarrow{(\textsf{decom-proof},sid\|\|1)} \mathcal{F}_{\text{com-zk}}^{\mathsf{R}_{\text{DL}}} \xrightarrow{(\textsf{decom-proof},sid\|\|1,R_1)}$	P_2 aborts if $(\textsf{decom-proof},$ $sid\|\|1, R_1)$ not received
$R = k_1 \cdot R_2 = (r_x, r_y)$		
$r = r_x \bmod q$		$m' = H(m)$
		$R = k_2 \cdot R_1 = (r_x, r_y)$
$s = k_1^{-1}(m' + r \cdot x_1) \cdot s_2 \bmod q$		$r = r_x \bmod q$
If $\mathsf{Vrfy}(Q, m, (r, s)) \to 0$,	$\xleftarrow{s_2}$	$s_2 = k_2^{-1}(m' + r \cdot x_2) \bmod q$
P_1 aborts, else returns (r, s)		

Figure 1 TP-cECDSA key generation and signing protocols

Table 2 Communication cost and efficiency comparison

schemes	pk	sk	key generation	signing(online)	verification
ECDSA	$x \cdot G$	x	1PM	1Hash+1MM	2PM+1PA
[3]-P_1	pk_{enc}, $x_1 \cdot x_2 \cdot G$	x_1, sk_{dec}	1PM+1KeyGen+1Enc 1DL-ZK+1PDL-ZK(p)+1P-ZK(p)	1Dec 1Vrfy(ECDSA)	—
[3]-P_2	pk_{enc}, $x_1 \cdot x_2 \cdot G$	x_2	2PM+1PDL-ZK(v) 1DL-ZK+1P-ZK(v)	1Enc 1SM+1HA	—
cECDSA	$x_1 \cdot x_2 \cdot G$, $(x_1 + x_2)G$	x_1, x_2	2PM	1Hash+3MM	3PM+2PA
Ours-P_1	$x_1 \cdot x_2 \cdot G$, $(x_1 + x_2)G$	x_1	2PM+1PA 1DL-ZK	1Hash+1MM	—
Ours-P_2	$x_1 \cdot x_2 \cdot G$, $(x_1 + x_2)G$	x_2	2PM+1PA 1DL-ZK	1Hash+1MM	—

‡ x, x_1 and x_2 denote the signing secret key. G denotes a base point. pk_{enc} and sk_{dec} indicate the public key and secret key of Paillier encryption. Enc and Dec denote the encryption and decryption operation of Paillier encryption. Hash denotes the hash operation of SHA-256. MM, PM, PA and SM denote modular multiplication, point multiplication, point addition and scalar multiplication operation. PDL-ZK(v/p),P-ZK(v/p) and DL-ZK denote PDL-ZK(verification/prove operation) [3], P-ZK(verification/prove operation) [3] and DL-ZK(both of verification and prove operation). Vrfy(ECDSA) denotes the verification operation of ECDSA. The symbol — denotes that each party needs not this operation.

In the online signing phase, P_2 needs to compute encryption and homomorphic scalar multiplication and homomorphic addition of the Paillier scheme. P_1 computes Paillier decryption and ECDSA verification.

Since our two-party signature avoids the complicated homomorphic encryption, it also avoids expensive zero-knowledge proof, such as P-ZK and PDL-ZK. Therefore, in the key generation phase, each party needs twice interactions, twice point multiplication operation, and once point addition operation and discrete log proof (each running as prover once and as verifier once). In the online signing phase, P_1 and P_2 take advantage of a pre-stored random value and compute part of the signature, and the operation is similar to the original ECDSA signing. Therefore, both the key generation and signing operations of each party in our two-party signature are more simple and efficient.

Table 3 Computation time of ECDSA, cECDSA, and TP-cECDSA on the standard NIST curves NID.X9.62.prime256v1

schemes	key generation (μs)	offline signing (μs)	online signing (μs)	verification (μs)
ECDSA	11.07	37.03	2.05	87.56
cECDSA	22.91	34.90	3.91	140.62
Ours-P_1	61.42	100.2	2.06	—
Ours-P_2	65.41	84.29	1.98	—

‡ The symbol — denotes that each party needs not this operation.

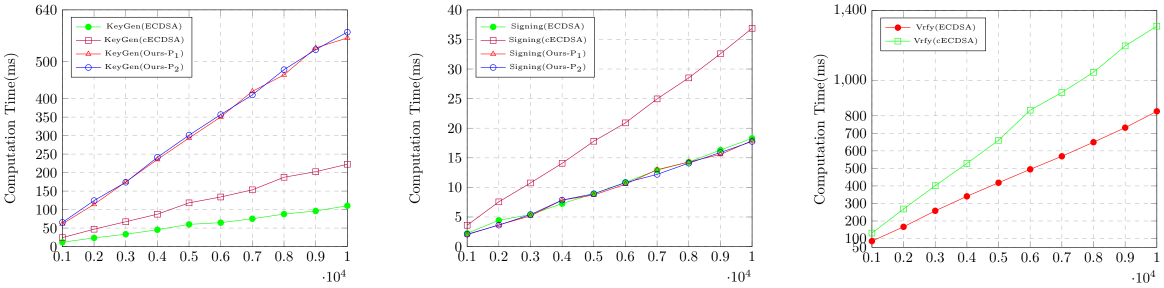

Figure 2 Efficiency comparison of (a) keyGen, (b) signing and (c) verification operation

5.2 Experimental Analysis

In order to evaluate the practical performance of our schemes, we implement the ECDSA, combinatorial ECDSA, and our two-party signature based on the OpenSSL library. The implementation is given in https://github.com/tbb-tobebetter/TP-cECDSA. The program is executed on an Intel Core i5 CPU 2.3 GHz and 8GB RAM running macOS High Sierra 10.13.3 system.

Experiment Setting and Computation Time. Following Lindell's two-party ECDSA [3], the first three signing steps of protocol 2 can be operated in the offline phase, so we omit this part and only realize the online signing phase of each party. In online signing phase, the operation of P_1 and P_2 is similar to calculating an ECDSA signature. As depicted in Table 3, Figure 2, we run our implementations on the standard NIST curves NID.X9.62.prime256v1. In Table 3, we test the average running times of all algorithms in ECDSA, cECDSA, and our two-party signature. In Figure 2, we run ECDSA, cECDSA, and our two-party signature many times respectively, and show the efficiency of the key generation, online signing, and verification operation. Based on the experimental data and analysis results, the average running times of all algorithms in combinatorial ECDSA and our two-party signature are the levels of microseconds. Especially, the running times of the online signing algorithm in TP-cECDSA-P_1 and TP-cECDSA-P_2 is similar to the original ECDSA signing algorithm. As depicted in [9], the average running times of the key generation in [3, 9] is the level of seconds, and the average running times of the signing operation in [3, 9] are the level of a millisecond. Therefore, our protocol can compare with the state-of-the-art two-party signature based on ECDSA [3, 9].

6 Conclusion

In this work, we first construct a combination ECDSA based on ECDSA itself, which can be seen as a combination of two ECDSA in the structure and can be split into two parts easily. Benefiting from the structure, we develop a simple and fast two-party signing protocol based on the ECDSA security, which can avoid the traditional problems of constructing two-party ECDSA showed in [1]. Although these problems can be solved by using additional complicated MPC techniques, such as homomorphic encryption [1–3], and oblivious transfer [5, 8], etc, there are some defects in security and efficiency. Our protocol ensures that each part of the signature outputted from each party can be verified and can be combined into a combination ECDSA signature easily, so heavy MPC and expensive zero-knowledge proofs are avoided. In particular, after agreeing on random points in the offline phase, the signing operation of each party is similar to the original ECDSA in the online phase. Therefore, it can reuse the implementation of existing ECDSA and upgrade existing ECDSA applications to two-party scenarios easily.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant No. 61772522).

References

1 MacKenzie, P.D., Reiter, M.K.: Two-party generation of DSA signatures. In: Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. pp. 137–154 (2001), https://doi.org/10.1007/3-540-44647-8_8

2 Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings. pp. 156–174 (2016), https://doi.org/10.1007/978-3-319-39555-5_9

3 Lindell, Y.: Fast secure two-party ECDSA signing. In: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II. pp. 613–644 (2017), https://doi.org/10.1007/978-3-319-63715-0_21

4 Boneh, D., Gennaro, R., Goldfeder, S.: Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In: Progress in Cryptology - LATINCRYPT 2017 - 5th International Conference on Cryptology and Information Security in Latin America, Havana, Cuba, September 20-22, 2017, Revised Selected Papers. pp. 352–377 (2017), https://doi.org/10.1007/978-3-030-25283-0_19

5 Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 980–997 (2018), https://doi.org/10.1109/SP.2018.00036

6 Lindell, Y., Nof, A.: 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 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 1837–1854 (2018), https://doi.org/10.1145/3243734.3243788

7 Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 1179–1194 (2018), https://doi.org/10.1145/3243734.3243859

8 Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ECDSA from ECDSA assumptions: The multiparty case. In: 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019. pp. 1051–1066 (2019), https://doi.org/10.1109/SP.2019.00024

9 Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ECDSA from hash proof systems and efficient instantiations. In: Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III. pp. 191–221 (2019), https://doi.org/10.1007/978-3-030-26954-8_7

10 Zhang, Y., He, D., Zhang, M., Choo, K.R.: A provable-secure and practical two-party distributed signing protocol for SM2 signature algorithm. Frontiers Comput. Sci. 14(3), 143803 (2020), https://doi.org/10.1007/s11704-018-8106-9

11 American National Standards Institute: X9.62: Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ecdsa) (2005)

12 Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., Möller, B.: Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS). RFC 4492, 1–35 (2006), https://doi.org/10.17487/RFC4492

13 Hoffman, P.E., Wijngaards, W.C.A.: Elliptic curve digital signature algorithm (DSA) for DNSSEC. RFC 6605, 1–8 (2012), https://doi.org/10.17487/RFC6605

14 Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), https://nakamotoinstitute.org/bitcoin/

15 Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978), http://doi.acm.org/10.1145/359340.359342

16 Schnorr, C.: Efficient identification and signatures for smart cards. In: Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. pp. 239–252 (1989), https://doi.org/10.1007/0-387-34805-0_22

17 Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T.: Efficient RSA key generation and threshold paillier in the two-party setting. In: Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings. pp. 313–331 (2012), https://doi.org/10.1007/978-3-642-27954-6_20

18 Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. pp. 186–194 (1986), https://doi.org/10.1007/3-540-47721-7_12

19 Lindell, Y.: Highly-efficient universally-composable commitments based on the DDH assumption. In: Paterson, K.G. (ed.) Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 446–466. Springer (2011), https://doi.org/10.1007/978-3-642-20465-4_25

20 Fujisaki, E.: Improving practical uc-secure commitments based on the DDH assumption. In: Zikas, V., Prisco, R.D. (eds.) Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9841, pp. 257–272. Springer (2016), https://doi.org/10.1007/978-3-319-44618-9_14

21 Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols - Techniques and Constructions. Information Security and Cryptography, Springer (2010), https://doi.org/10.1007/978-3-642-14303-8

22 Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 565–596. Springer (2018), https://doi.org/10.1007/978-3-319-96884-1_19

23 Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. 13(1), 143–202 (2000), https://doi.org/10.1007/s001459910006

24 Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA. pp. 136–145. IEEE Computer Society (2001), https://doi.org/10.1109/SFCS.2001.959888