Multi-signatures for ECDSA and Its Applications in Blockchain

Shimin Pan(✉), Kwan Yin Chan, Handong Cui, and Tsz Hon Yuen

The University of Hong Kong, Hong Kong, China
{smpan,kychan,hdcui,thyuen}@cs.hku.hk

Abstract. Multi-signatures enable a group of t signers to sign a message jointly and obtain a single signature. Multi-signatures help validating blockchain transactions, such as transactions with *multiple inputs* or transactions from *multisig addresses*. However, multi-signatures schemes are always realised naively in most blockchain systems by directly concatenating t ECDSA signatures.

In this paper, we give the *first* multi-signature scheme for ECDSA. Technically, we design a new ephemeral group public key for the set of signers and introduce an interactive signing protocol to output a single ECDSA signature. The signature can be validated by the ephemeral group public key. Then, we instantiate the ECDSA multi-signature scheme with class group, for which we design a secret exchanging mechanism that ensures the hiding content is well-constructed. Moreover, our scheme is able to identify the malicious party in the signing phase and help to minimize unnecessary resource consumption. This ECDSA multi-signatures can be used in blockchain to reduce the transaction cost and provide accountability for signers and backward compatibility with existing ECDSA addresses.

Keywords: Multi-signatures · ECDSA · Signature

1 Introduction

1.1 Motivation

Multi-signatures [16] have been widely used in different scenarios in the blockchain. This cryptographic primitive allows any group S of parties to jointly sign a message and produce a signature, for which verifiers are convinced that each group member S participated in the signing. It can also be used to divide up responsibility for possession of signing keys among multiple players and avoid a single point of failure. There are two major uses of the functionality of multi-signatures. The first use case is formatting a transaction with *multiple inputs* relative to different addresses. The owner of each input can sign on all of the outputs in this transaction[1] and present a signature for this input. In Bitcoin,

[1] This is the default setting in Bitcoin for the signature hash, called SIGHASH_ALL.

K. Nguyen et al. (Eds.): ACISP 2022, LNCS 13494, pp. 265–285, 2022.
https://doi.org/10.1007/978-3-031-22301-3_14

signatures for each input are concatenated. Protocols, such as Taproot, CoinJoin, and PayJoin[2], use multiple inputs and outputs transactions to improve the privacy of Bitcoin transactions. The second use case is the *multisig* address in Bitcoin (and some other blockchain), which contains n public keys. A transaction is valid when there are t valid ECDSA signatures attached relative to public keys among the key list. Each ECDSA signature is verified against one corresponding public key, and these t signers are accountable for generating this multi-signature accordingly.

The efficiency of the naive approach for multi-signatures currently used in Bitcoin is extremely poor. We need k signatures for a transaction with k inputs, or t signatures for a multisig account with a threshold t. Let us consider a transaction with two inputs and two outputs. For the first use case (P2PKH), the transaction size is 374 bytes, and two ECDSA signatures account for 39% (144 bytes) in it. For the second use case (P2SH 2-of-3 multi-signature), the transaction size is 668 bytes, and four ECDSA signatures account for 43% (288 bytes) in it. Therefore, it is important to design a cryptographic solution to reduce the signature size and lower the transaction cost.

1.2 Contribution

We design a new ECDSA multi-signature scheme by introducing the concept of *ephemeral* group public key for a group of signers S. Furthermore, it is integrated with the signing protocol of threshold ECDSA in [15]. The ephemeral group public key is defined *during* the interactive signing and is *different* for each signing instance. Our new scheme is significantly different from existing schemes (e.g., no group public key for [3], or one static group public key for each S [2,4,5,18,19]).

We recall that, in ECDSA, the secret key is x and the public key is $Y = xG$ where G is the group generator. To sign a message m, the signer picks a random k, computes the x-coordinate of $R = k^{-1}G$ as r and calculates $s = k(\mathsf{H}(m) + rx)$ for some hash function H. The signature is (r, s).

A Strawman Protocol. When there are t parties with their keys (x_i, Y_i), a simple multi-signature is setting the group public key as $Y = \sum_i Y_i$. However, this strawman protocol is not secure. For example, an adversary can set $Y_2 = -Y_1 + x_2G$, where Y_1 is the public key of an honest party. Then group key becomes $Y = Y_1 + Y_2 = x_2G$. Hence the adversary can generate a signature using x_2 only. This attack is known as the *rogue public key attack*.

Designing the Group Public Key. In order to deal with the rogue public key attack, the pairing-based multi-signatures [5] and the Schnorr-based multi-signatures [19] defined the group public key as $Y = \sum a_iY_i$ where $a_i = \mathsf{H}_1(\mathrm{S}, Y_i)$[3].

[2] Taproot: https://en.bitcoin.it/wiki/BIP_0341. CoinJoin: https://coinjoin.io. PayJoin: https://en.bitcoin.it/wiki/PayJoin.

[3] The function H_1 is defined in this way for the ease of presentation in the security proof. In practice, we can simply set $a_i = \mathsf{H}_1(i, r, \mathrm{S}, m)$ for all i.

Table 1. Comparison of signatures using multiple secret keys.

	# SK	# PK	Size	Accountability	Keygen
Threshold signature	t	1	$O(1)$	No	Involve n parties
Threshold ring signature	t	n	$O(\log n)$	No	No interaction
Bitcoin native multi-signatures	t	t	$O(t)$	Yes	No interaction
Multi-signatures	t	t	$O(1)$	Yes	No interaction

This static group public key is fixed for all signatures signed by the group of signers S. However, this structure cannot be applied to ECDSA multi-signatures because of the security proof. We instead design a new key structure such that the ephemeral group public key is different for each signature (r, s):

$$Y = \sum a_i Y_i, \quad \text{where } (a_1 || \dots || a_t) = \mathsf{H}(r, \mathrm{S}, m).$$

In the security proof, we show that the unforgeability is reduced to the unforgeability of the standard ECDSA signature with a public key $\hat{Y}$.

1.3 Related Work

Threshold ECDSA and Threshold Ring Signatures. In threshold signatures [11], a signing key is distributed among n parties, and a message can be signed only by a sufficiently large subgroup (Table 1). There are three main differences between threshold signatures and multi-signatures. Firstly, threshold signatures are verified by one public key, while multi-signatures are verified by a set of keys. Secondly, an interactive key generation protocol is needed for threshold signatures, making it hard to cover existing keys and generate new keys. Thirdly, anonymity is a property of threshold signatures while accountability is only offered by multi-signatures. The property of anonymity or accountability may be good for different applications.

Threshold ring signature [6] differs from threshold signature as the group G can be dynamically formed, and there is no interactive setup phase. The drawback of threshold ring signatures is that the verification involves all n public keys in G, and the state-of-the-art signature size is $O(\log n)$.

Multi-signatures. There are two approaches to construct multi-signatures. One is naively implemented by concatenating |S| signatures signed by S signing keys. Alternatively, researchers designed cryptographic algorithms to compress these |S| signatures into a single one, such as Schnorr-based multi-signatures [3,19,20] and pairing-based multi-signatures [4,5,18]. Multi-signatures with a predefined key range G such that $S \subset G$ is also named Accountable-Subgroup Multi-signatures (ASM) [20]. The accountability means that the subgroup S of actual signers is known to the verifiers.

Recent researches on Schnorr follow the paper [19] known as MuSig. MuSig has been proved to be insecure in [13], which states that there is no OMDL reduction to the MuSig. Later in Crypto 2021, other multi-signatures were proposed [1,21]. The recent attractive Schnorr multi-signatures results could not be adapted to ECDSA setting directly due to the complexity of the inversion computation.

An ECDSA-based multi-signature scheme is proposed in [17]. Their scheme relies on a trusted group manager to generate the ECDSA signature from $t-1$ parties. Moreover, the secret keys of the $t-1$ parties are all derived by the group manager. Apparently, it is not secure in the security model given by [19]. It is also not secure against the rogue public key attack. Konstantinos *et al.* tried to do signature compression in 2021 [10] but their scheme only compresses t signatures into $(t+1)/2$ and reached a relatively large signature size.

As an ECDSA multi-signature, our scheme requires no trusted party and the signature requires only the same size as the standard ECDSA. Consequently, this scheme shows superiority in functionality and the optimal signature size. Compared to Schnorr multi-signatures, it has better compatibility with current-used ECDSA key pairs in most blockchain systems.

1.4 Paper Organization

This paper is organized as the following. Section 2 shows notations and the multi-signature primitive. Section 3 introduces a modified multiplicative-to-additive scheme. Section 4 presents the generic multi-signature scheme and the security proof. Section 5 shows a scheme instance, which utilizes the Castagnos-Laguillaumie encryption. Section 6 shows the implementation of the previous instance with Rust and the bandwidth analysis. Section 7 shows details of how our scheme interacts with the Bitcoin system. Section 8 draws some conclusions.

2 Preliminaries

We define notations and the multi-signature primitive in this section. Other building components are listed in Appendix A.

Notation $x \leftarrow_{\$} S$ is uniformly sampling an element x from the set S and $[n]$ denotes the set $\{1, \ldots, n\}$. PPT stands for probabilistic polynomial time and $\mathsf{negl}(n)$ is a negligible function on n. $\mathcal{G}_{\text{ECC}} = (\mathbb{G}, G, q)$ is the ECC group generated by G with order q.

For the definition of multi-signature, we consider that given in [19], where multi-signature is a tuple of four PPT algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$:

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{params}$: it generates system parameters from the security parameter.
- $\mathsf{KeyGen}(\mathsf{params}) \rightarrow (\mathsf{sk}, \mathsf{pk})$: it is the key generation protocol which, on input parameters, outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$ where pk is the public key and sk is the secret one.

– $\mathsf{Sign}(\mathsf{params}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_t\}, \mathrm{S} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}, m) \rightarrow \sigma/\perp$: it is an interactive protocol. Parties keep their sk_i secret and work with others in S to sign a message m. The protocol outputs either a signature or $\perp$.
– $\mathsf{Verify}(\mathsf{params}, S = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}, m, \sigma) \rightarrow \{0, 1\}$: it checks whether the signature σ is valid or not.

Correctness. For all messages m, if $\sigma \leftarrow \mathsf{Sign}(\mathsf{params}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_t\}, \mathrm{S} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}, m)$ where sk_i is the secret key corresponding to the public key pk_i for $i \in [t]$, then $1 \leftarrow \mathsf{Verify}(\mathsf{params}, \mathrm{S}, m, \sigma)$.

Security model. We use the game-based security definition for multi-signatures [19]. The security game involves one honest party, and all other parties are corrupted by an adversary $\mathcal{A}$. After calling the signing oracle on inputs of the form (m_i, S_i) and getting back valid signatures σ_i, the adversary $\mathcal{A}$ wins the game by outputting a valid signature σ involving the public key of the honest party. A formal definition is given below.

1. The system setups based on the security parameters $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. The honest party generates a key pair $(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{params})$ and the adversary $\mathcal{A}$ receives pk^* as input.
3. For any adversary-specified message m and public-key set $\mathrm{S} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}$ containing pk^*, the honest party runs $\mathsf{Sign}(\mathsf{params}, \mathsf{sk}^*, \mathrm{S}, m)$ interactively with $\mathcal{A}$ and works as the signing oracle for $\mathcal{A}$. It could be abort when wrong messages discovered.
4. Finally, $\mathcal{A}$ returns a message m^*, a public key set S^* and a signature σ^* such that the tuple (m^*, S^*) has not been queried previously. $\mathcal{A}$ wins the game if $\mathsf{pk}^* \in \mathrm{S}^*$ and the signature is valid, i.e. $\mathsf{Verify}(\mathsf{params}, \mathrm{S}^*, m^*, \sigma^*) = 1$.

A multi-signature scheme is said to be unforgeable if no PPT adversary wins the game with non-negligible probability.

3 Multiplicative-to-Additive Share Conversion Protocol

Multiplicative-to-additive (MtA) protocol [14] was introduced as a building block for threshold ECDSA. The MtA protocol involves two parties $\{P_1, P_2\}$ having messages $a \in \mathbb{Z}_p$ and $b \in \mathbb{Z}_p$ as their private input respectively. The protocol turns a multiplicative result $ab \mod q$ to an additive result $\alpha + \beta \mod q$, where P_1 and P_2 outputs α and β respectively.

3.1 Definition

Generic MtA Protocol. The original MtA scheme [14] is constructed with the Paillier encryption, and it requires a range proof. We give a 3 round generic MtA protocol, abstracted from the construction in [7]. This generic protocol relies on any additive homomorphic encryption $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EvalSum}, \mathsf{EvalScal})$ with a message space equal to $\mathbb{Z}_q$[4].

[4] If the message space of the additive homomorphic encryption is larger than q (e.g., Paillier encryption), then an extra zero-knowledge range proof is needed for all ciphertexts, to ensure that $\alpha = ab - \beta$ in Step 2 is still within the message space.

Setup Phase. For preset system parameters $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda)$, P_1 generates keys by running $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{params})$.

Conversion Phase

1. P_1 encrypts a and generates a zero-knowledge (ZK) proof for it.
   - P_1 computes the encryption $c_A = \mathsf{Enc}_{\mathsf{ek}}(a; \rho)$ using a randomness ρ.
   - P_1 creates a zero-knowledge (ZK) proof π_A, relative to the relation R_{Enc}, that c_A is well-formed, where $R_{\mathsf{Enc}} = \{(c_A, \mathsf{ek}) : (a, \rho) | c_A = \mathsf{Enc}_{\mathsf{ek}}(a; \rho)\}$.
   - P_1 sends c_A and π_A to P_2.
2. P_2 manipulates c_A to the ciphertext of $\alpha = ab - \beta \bmod q$, where β is the randomness.
   - P_2 picks a random β in $\mathbb{Z}_q$.
   - P_2 computes $c_B = \mathsf{EvalSum}_{\mathsf{ek}}(\mathsf{EvalScal}_{\mathsf{ek}}(c_A, b), \mathsf{Enc}_{\mathsf{ek}}(-\beta; \rho'))$.
   - P_2 gives the ZK proof π_B, relative to relation R_B, that c_B is calculated from (b, β) and <u>is consistent with $H = bG$ where G is the ECC generator.</u>

$$R_B = \left\{ (\underline{H}, \underline{G}, c_A, c_B, \mathsf{ek}) : (b, \beta, \rho') | \begin{array}{l} \underline{H = bG} \wedge c_B = \\ \mathsf{EvalSum}_{\mathsf{ek}}(\mathsf{EvalScal}_{\mathsf{ek}}(c_A, b), \mathsf{Enc}_{\mathsf{ek}}(-\beta, \rho')) \end{array} \right\}$$

   - P_2 sends c_B and π_B to P_1.
3. P_1 checks π_B and then computes $\alpha = \mathsf{Dec}_{\mathsf{dk}}(c_B)$.

MtAwc Protocol. The standard MtA protocol does not include the underlined steps. If we further want to check b in c_B is consistent with value H, these steps are retained and the protocol is named as MtAwc (Multiplicative-to-additive with check). The MtA(wc) ptotocol could be proved secure even without any ZK proof as shown in [14]. Both MtA and MtAwc are used in our multi-signature.

4 Multi-signatures for ECDSA

In this section, we give a new ECDSA multi-signature scheme. In the naive approach of concatenating t ECDSA signatures, all parties can determine who is not signing correctly. Hence, we choose to build our ECDSA multi-signatures upon the interactive signing protocol with identifiable abort in [15]. Moreover, the new proposed ZK proof technique is detailed in Appendix C.

4.1 Construction

We denote a non-malleable equivocable commitment scheme a tuple of 5 algorithms $(\mathsf{KeyGen}_e, \mathsf{Com}_e, \mathsf{Decom}_e, \mathsf{KeyGen}'_e, \mathsf{Equiv}_e)$ and a trapdoor commitment scheme with efficient ZK proof by $(\mathsf{KeyGen}_z, \mathsf{Com}_z, \mathsf{Decom}_z, \mathsf{KeyGen}'_z, \mathsf{TCom}_z, \mathsf{TDecom}_z)$.

Our protocol contains 4 algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{params}$: On security parameter λ, this algorithm generates an ECC group $\mathcal{G}_{\mathrm{ECC}} = (\mathbb{G}, G, q)$. It chooses hash functions $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q$, and $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^*$. It runs $\mathsf{pk}_e \leftarrow \mathsf{KeyGen}_e(1^\lambda)$ and $\mathsf{pk}_z \leftarrow \mathsf{KeyGen}_z(1^\lambda)$. It outputs $\mathsf{params} = (\mathbb{G}, G, q, \mathsf{H}, \mathsf{H}_1, \mathsf{pk}_e, \mathsf{pk}_z)$.
- $\mathsf{KeyGen}(\mathsf{params}) \to (\mathsf{sk}, \mathsf{pk})$: Each party picks a random secret key $x_i \leftarrow_\$ \mathbb{Z}_q$ and generates its own public key as $Y = xG$. Each party additionally runs the setup phase of the MtA protocol. This algorithm will finally output the key pair for the current party (x, Y). The key generation is identical to the standard ECDSA.
- $\mathsf{Sign}(\mathsf{params}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_t\}, \mathrm{S} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}, m) \to \sigma/\perp$: On input a group of public keys S of size t and a message m, player P_i with secret key x_i generate and share its MtA public key, then runs the following steps interactively.
  - Phase 1. Each player P_i picks $k_i, \gamma_i \leftarrow_\$ \mathbb{Z}_q$. All players broadcast their commitment C_i to $\gamma_i G$, where $(C_i, D_i) \leftarrow \mathsf{Com}_e(\mathsf{pk}_e, \gamma_i G)$.
  - Phase 2. For convenience, we define the quantities $k = \sum_{i\in[t]} k_i$, $\gamma = \sum_{i\in[t]} \gamma_i$. As a result $k\gamma = \sum_{i,j\in[t]} k_i\gamma_j \mod q$. Each pair of players P_i and P_j runs MtA together for k_i and γ_j and respectively receives back the result α_{ij} with β_{ij}, such that $k_i\gamma_j = \alpha_{ij} + \beta_{ij} \mod q$. Upon receiving α_{ij} and β_{ji}, P_i constructs $\delta_i = k_i\gamma_i + \sum_{i\neq j} \alpha_{ij} + \sum_{i\neq j} \beta_{ji} \mod q$.
  - Phase 3. All parties broadcast their own δ_i and reconstruct $\delta = \sum_{i\in[t]} \delta_i = \sum_{i,j\in[t]} k_i\gamma_j \mod q$.
  - Phase 4. Each party P_i broadcasts the decommitment D_i. P_i obtains $\gamma_j G = \mathsf{Decom}_e(\mathsf{pk}_e, C_j, D_j)$ for all $j \neq i$ and constructs $R = \delta^{-1}(\sum_{i\in[t]} \gamma_i G) = (k\gamma)^{-1}(\sum_{i\in[t]} \gamma_i G) = k^{-1}G$ and gets r as the x-coordinate of R.
  - Phase 5. Each party broadcasts $\bar{R}_i = k_i R$ and gives a consistency proof π_{k_i} between $\bar{R}_i$ and $\mathsf{Enc}(k_i)$ which is the first message sent in MtA protocol in Phase 2. The protocol aborts if the following check fails

$$G = \sum_{i\in[t]} \bar{R}_i. \tag{1}$$

  - Phase 6. All players compute $(a_1||\ldots||a_t) = \mathsf{H}_1(r, \mathrm{S}, m)$, in which a_i stands for the masks of all parties' public keys. The group public key is denoted as $Y = \sum_{Y_i\in \mathrm{S}} a_i Y_i$. Consequently, the corresponding secret key is $x = \sum_{i\in[t]} a_i x_i$, and it could not be controlled by any single party. As a result, $k\sum_{i\in[t]} a_i x_i = \sum_{i,j\in[t]} k_i(a_j x_j) \mod q$.
    Each pair of players P_i and P_j runs MtAwc together for k_i and $a_j x_j$, with the public value $B = a_j Y_j$. The return values are respectively marked as μ_{ij} for P_i and ν_{ij} for P_j. Hence $k_i(a_j x_j) = \mu_{ij} + \nu_{ij} \mod q$.
    Upon receiving μ_{ij} and ν_{ji}, P_i constructs $\sigma_i = k_i a_i x_i + \sum_{i\neq j} \mu_{ij} + \sum_{i\neq j} \nu_{ji}$ mod q
  - Phase 7. <u>All parties broadcast T_i, where $(T_i, \cdot) \leftarrow \mathsf{Com}_z(\mathsf{pk}_z, \sigma_i)$, with a zero-knowledge proof π_{T_i} of σ_i.</u>

Table 2. Identify abortion

Phase	Failure	Detecting adversary
2	MtA	Detect directly
4	Decommitment	Detect directly
5	$\bar{R}_i$ consistency	Detect directly
5	Equation (1)	a. P_i publishes k_i, γ_i, α_{ij} and β_{ij} b. All compute δ_i' and check $\delta_i = \delta_i'$
6	MtAwc	Detect directly
7	T_i consistency	Detect directly
8	S_i consistency	Detect directly
8	Equation (2)	a. P_i publishes k_i and μ_{ij} b. P_j computes $\sigma_i G = k_i a_i x_i G + \sum_{i \neq j} \mu_{ij} G + \sum_{i \neq j} \nu_{ji} G$ c. P_i prove $\sigma_i G$ and S_i consistent
9	σ invalid	Detect by checking $s_i R = \mathsf{H}(m)\bar{R}_i + rS_i$

- Phase 8. <u>Each party gives the ZK proof π_{σ_i} on the consistency between T_i in Phase 7 and the newly generated value $S_i = \sigma_i R$. Upon receiving all S_i, parties aborts when</u>

$$Y \neq \sum_{i \in [t]} S_i. \tag{2}$$

- Phase 9. All parties broadcast $s_i = k_i \mathsf{H}(m) + \sigma_i r$ and reconstruct s as $s = \sum_{i \in [t]} s_i$. The protocol aborts if (r, s) is not a valid ECDSA signature for the message m and the public key y.

– Verify$(\mathsf{params}, S = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}, m, \sigma) \to \{0, 1\}$: The algorithm takes as inputs the public keys of signers as $\mathrm{S} = \{Y_i\}$, the message m and the signature (r, s). The verification is done in two steps.
  - Generate ephemeral group public key. Compute $(a_1 || \ldots || a_t) = \mathsf{H}_1(r, \mathrm{S}, m)$ and $Y = \sum_{i \in [t]} a_i Y_i$.
  - Verify ECDSA signature. Verify $\sigma = (r, s)$ using Y, by computing $R' = \mathsf{H}(m) \cdot s^{-1} G + r s^{-1} Y$ and checking if the x-coordinate of R' mod q is r.

Note: Steps with underlining are optional. With these steps, one is able to determine which party did not collaborated properly by referring to Table 2, which uses the technique given by [15]. Otherwise, the protocol will give *anonymous abort*. We could prevent intentionally anonymous aborting by identifying the malicious party.

4.2 Security Proof

Theorem 1. *Our ECDSA multi-signature is unforgeable in the random oracle model if the standard ECDSA is unforgeable.*

Proof. In the bird's eyes, we prove the standard ECDSA is forgeable with non-negligible probability if our multi-signature is threaten by an adversary $\mathcal{A}$ with non-negligible advantage ϵ. The forger $\mathcal{F}$ internally invokes adversary $\mathcal{A}$ for and tries to break the standard ECDSA scheme with the power it.

Without loss of generality, the proof assumes only 1 honest party, named P_1 corresponding to public key pk_1, and other parties $\{P_i\}_{i>1}$ are all corrupted. We assume the adversary to be a *rushing adversary*, which means corrupted parties always send their messages after the honest party in each round.

Simulation of Setup. The simulator $\mathcal{S}$ picks $\mathcal{G}_{\mathrm{ECC}}$ and runs key generation $(\mathsf{pk}_e, \mathsf{tk}_e) \leftarrow \mathsf{KeyGen}'_e(1^\lambda)$ and $(\mathsf{pk}_z, \mathsf{tk}_z) \leftarrow \mathsf{KeyGen}'_z(1^\lambda)$ honestly.

Simulation of KeyGen. The key generation procedure needs to embed the standard ECDSA public key $\hat{\mathsf{pk}} = \hat{Y}$ into the multi-signature scheme. The simulator sets the public key for P_1, i.e. the simulated party, to $Y_1 = \hat{Y}$.

Simulation of H *and* H_1. $\mathcal{S}$ forwards whatever the standard ECDSA hash function returns for H and simulates H_1 as a normal random oracle query.

Simulation of Sign. For signing on message m, $\mathcal{S}$ firstly queries the ECDSA instance with a random message $\hat{m} \leftarrow_\$ \mathbb{Z}_q$ and gets back the signature $(\hat{r}, \hat{s})$.

They are expected to fulfill the equation $\hat{R} = \mathsf{H}(\hat{m})\hat{s}^{-1}G + \hat{r}\hat{s}^{-1}\hat{Y}$ where $\hat{r}$ is the x-coordinate of $\hat{R}$. Denote $\Delta = \mathsf{H}(\hat{m}) - \mathsf{H}(m)$. $\mathcal{S}$ picks random numbers $d_1, d_2 \in \mathbb{Z}_q$ such that:

$$(\hat{s}/d_2)(d_2\hat{R} + d_1d_2/\hat{s}G) = \mathsf{H}(\hat{m})G + \hat{r}\hat{Y} + d_1G = \mathsf{H}(m)G + \hat{r}Y_1 + (d_1 + \Delta)G.$$

Now suppose $R' = d_2\hat{R} + d_1d_2/\hat{s}G$ and its x-coordinate as r', and denote $s' = \hat{s}/d_2$. Then we have:

$$s'(R') = \mathsf{H}(m)G + r'(\hat{r}/r'Y_1 + (d_1 + \Delta)/r'G).$$

(r', s') is a valid ECDSA signature on a message m and the corresponding group public key is $\hat{r}/r'Y_1 + (d_1 + \Delta)/r'G$. To form such a group public key, we set $a_1 = \hat{r}/r'$ with $\sum_{j>1} a_jx_j = (d_1 + \Delta)/r'$ in Phase 6 by the random oracle model.

The interaction messages will be given on how to simulate the real protocol with the previous $\hat{\mathsf{pk}}$ instance.

- Phase 1. P_1 runs the protocol and broadcasts C_1 as required. All other players also broadcast the commitment C_i for γ_iG.
- Phase 2. $\mathcal{S}$ interactively runs MtA with other parties using the MtA encryption keys as the following.
  - Initiator for MtA with k_1 and γ_j. $\mathcal{S}$ runs correctly for P_1 using k_1. $\mathcal{S}$ extracts P_j's value γ_j and β_{1j} and computes $\alpha_{1j} = k_1\gamma_j - \beta_{1j} \bmod q$.
  - Respondent for MtA with k_j and γ_1. $\mathcal{S}$ runs correctly for P_1 using γ_1. $\mathcal{S}$ extracts P_j's value k_j and computes $\alpha_{j1} = k_j\gamma_1 - \beta_{j1} \bmod q$ using its own share β_{j1}.
- Phase 3. $\mathcal{S}$ broadcasts δ_1 according to the scheme and receives back δ_i for $i > 1$. $\mathcal{S}$ reconstructs $\delta = \sum_{i\in[t]} \delta_i$.

– Phase 4a. Party P_i reveals D_i to decommit $\gamma_i G$. $\mathcal{S}$ computes $R = \delta^{-1}(\sum_{i\in[t]} \gamma_i G)$.

$\mathcal{S}$ checks whether the published values are consistent. Using the value k_i extracted in MtA, $\mathcal{S}$ can also validate whether $\sum_{i\in[t]} k_i R = G$. We say that an execution is fail-1 if this checking does not passed. If it is fail-1, $\mathcal{S}$ runs Phase 5 of the protocol as required using k_1 and one of the adversary's ZK proofs will fail and the protocol aborts. If it is not fail-1, then:

– Phase 4b. $\mathcal{S}$ rewinds $\mathcal{A}$ to the decommitment step and computes $\Gamma_1 = \delta R' - \sum_{j>1} \gamma_j G$ using γ_j extracted from Phase 2. Then $\mathcal{S}$ runs $D_1' \leftarrow \mathsf{Equiv}_e(\mathsf{pk}_e, \mathsf{tk}_e, C_1, \Gamma_1)$. Then $\mathcal{S}$ reveals D_1' as the decommitment instead.
  All parties can compute $R' = \delta^{-1}(\Gamma_1 + \sum_{j>1} \gamma_j G)$ and get r' as the x-coordinate of R'.
– Phase 5. $\mathcal{S}$ computes $\bar{R}_1 = G - \sum_{j>1} k_j(R')$ using the extracted k_j. $\mathcal{S}$ simulates the consistency proof and outputs $\bar{R}_1$.
– Phase 6. All players compute $(a_1||\ldots||a_t) = \mathsf{H}_1(r', \mathrm{S}, m)$.
  $\mathcal{S}$ interactively runs MtAwc with other parties using the MtAwc encryption keys as the following.
  • Initiator for MtAwc with k_1 and $a_j x_j$. $\mathcal{S}$ runs correctly for P_1 using k_1. $\mathcal{S}$ extracts x_j and ν_{1j} from π_B and computes $\mu_{1j} = k_1(a_j x_j) - \nu_{1j} \bmod q$.
  • Respondent for MtAwc with k_j and $a_1 x_1$. $\mathcal{S}$ does not have $\mathsf{sk}_1 = x_1$ of P_1. $\mathcal{S}$ just randomly picks $\tilde{x_1} \leftarrow_\$ \mathbb{Z}_q$ and interacts with P_i as if it is x_1.

Now $\mathcal{S}$ has already obtained the values $x_2, \ldots, x_t$. $\mathcal{S}$ rewinds $\mathsf{H}_1(r', S, m)$ and sets $a_1 = \hat{r}/r'$ and a_2 such that $\sum_{j>1} a_j x_j = (d_1 + \Delta)/r'$. $\mathcal{S}$ sets new $(a_1||a_2||\ldots)$ as the output of $\mathsf{H}_1(r', \mathrm{S}, m)$. We first consider the distribution of a_2. Since $a_3, \ldots, a_n$ are randomly chosen from $\mathbb{Z}_p$, a_2 itself is uniformly distributed from $\mathbb{Z}_p$. The values of all a_i satisfy the relation $\hat{r}/a_1 = (d_1 + \Delta)/\sum_{j>1} a_j x_j$. The relation is hidden by $\mathcal{S}$'s random choice of d_1 and Δ.

The value a_1 is calculated from $\hat{r}$ (the x-coordinate of $\hat{R}$ generated for a random message $\hat{m}$) and r' (the x-coordinate of R', calculated from the random number d_1, d_2). Assume that the division of the two x-coordinates is uniformly distributed in $\mathbb{Z}_p$, then a_1 is also uniformly distributed from $\mathbb{Z}_p$. Hence rewinding will succeed with non-negligible probability.

We remark that $\mathcal{S}$ cannot get x_1 so it will never get the complete σ_1 by itself. $\mathcal{S}$ can only compute another value: $\sigma_A = \sum_{i,j>1} k_i a_j x_j + \sum_{i>1} \mu_{i1} + \sum_{i>1} \nu_{1i} \bmod q$ using the values extracted from MtAwc.

– Phase 7. $\mathcal{S}$ computes $(T_1, \mathsf{aux}_{T_1}) \leftarrow \mathsf{TCom}_z(\mathsf{pk}_z, \mathsf{tk}_z)$ and uses a simulator of the ZK proof to generate π_{T_1}. $\mathcal{S}$ broadcasts T_1 and π_{T_1}.

$\mathcal{S}$ can detect if the values published by the adversary are consistent. Using the extractor of π_{T_i}, $\mathcal{S}$ can extract σ_i and check if $\sigma_A = \sum_{i>1} \sigma_i$. We say that an execution is fail-2 if this checking is incorrect.

If it is fail-2, then in Phase 8, $\mathcal{S}$ sets $S_1 = (k_1 a_1 \tilde{x_1} + \sum_{j>1} \mu_{1j} + \sum_{j>1} \nu_{j1})R'$, simulates a consistency proof using the simulator of the ZK proof, and outputs S_1. At least one of the adversary's ZK proofs will fail and the protocol will abort.

If it is not fail-2, then:

- Phase 8. $\mathcal{S}$ computes $S_1 = Y - \sum_{j>1} \sigma_i R'$. $\mathcal{S}$ simulates a consistency proof using the simulator of the ZK proof and outputs S_1.
- Phase 9. As the simulator $\mathcal{S}$ already knew k_i, $a_i x_i$ for all $i > 1$, it could compute $s_A = \sum_{i>1} s_i = \mathsf{H}(m) \sum_{i>1} k_i + \sigma_A r$, and outputs $s_1 = s' - s_A$.

Attacking Standard ECDSA. In the final step of the security game, $\mathcal{A}$ is required to present a valid signature (r^*, s^*) on a message m^* such that the honest party's public key Y_1 is inside the public key set $\mathrm{S}^* = (y_1^*, \ldots, y_{t^*}^*)$. WLOG, suppose $Y_1^* = Y_1$. Since the signature is valid, we have $(a_1^* || \ldots || a_{t^*}^*) = \mathsf{H}_1(r^*, \mathrm{S}^*, m^*)$, $Y^* = \sum_{i \in [t^*]} a_i^* Y_i^*$,

$$s^*(R^*) = \mathsf{H}(m^*)G + r^* Y^*, \tag{3}$$

and the x-coordinate of R^* mod q is r^*.

$\mathcal{S}$ rewinds $\mathcal{A}$ to the query of $\mathsf{H}_1(r^*, \mathrm{S}^*, m^*)$ and returns another fresh random $(\tilde{a_1}^* || \tilde{a_2}^* || \ldots || \tilde{a_{t^*}}^*)$ instead. Now $\mathcal{A}$ returns the signature $(r^*, \tilde{s}^*)$, and

$$\tilde{s}^*(R^*) = \mathsf{H}(m^*)G + r^* \tilde{Y}^*. \tag{4}$$

By dividing Eq. (3) and (4), we have:

$$(s^* - \tilde{s}^*)kG = (s^* - \tilde{s}^*)(R^*) = r^*(a_i^* - \tilde{a_i}^*) \sum Y_i = r^*((a_1^* - \tilde{a_1}^*)Y_1 + \sum_{i>1} (a_i^* - \tilde{a_i}^*) x_i G)$$

Hence $\mathcal{S}$ can extract the discrete logarithm of Y_1 i.e. x_1 from the final equation, which helps itself to generate a valid signature for the underlying standard ECDSA. By the random choice of $\hat{m}$ in the signing oracle query, m^* is different from all existing $\hat{m}$ with an overwhelming probability.

Analysis. The differences between the real and the simulated views can be listed as the following. In Phase 2, the MtA protocol the values $c_i = \mathsf{Enc}_{\mathsf{ek}_i}(k_i)$ are published. In the real protocol $R = \sum_i k_i G$ and in the simulated protocol we have R^* instead. The views are indistinguishable as the encryption scheme secure is IND-CPA secure. In Phase 4b of the simulated protocol, the decommitment D_1' is returned. By the non-malleability property of the equivocable commitment, it is indistinguishable from the real decommitment D_1. By the zero-knowledge property of the ZK proofs, the simulation of Phase 5 and 8 are correct. In Phase 6, a_2 is set to $\sum_{j>1} a_j x_j = (d_1 + \Delta)/r'$. It is uniformly distributed in $\mathbb{Z}_q$ by the random choice of $d_1 \in \mathbb{Z}_q$. Also, a_1 is set to $\hat{r}/r'$. Note that $\hat{r}$ is related to $\hat{s} = s'd_2$, which is uniformly distributed in $\mathbb{Z}_p$ by the random choice of $d_2 \in \mathbb{Z}_q$.

5 Instantiating with Class Group

We use the additive homomorphic encryption introduced by Castagnos and Laguillaumie [9] defined on a group with hard subgroup membership (HSM).

5.1 Hard Subgroup Membership Group

HSM Group. It is an abstract group introduced in [5] and named as HSM for the hard subgroup membership assumption [22], which constructs a subgroup where the discrete logarithm (DL) is easy. The generation algorithm takes security parameter 1^λ as input and it outputs the group as $\mathcal{G}_{\mathrm{HSM}} = (\mathbb{G}, \mathbb{G}^q, \mathbb{F}, g, g_q, f, \tilde{s}, q)$. Specifically, the primary group is $(\mathbb{G}, \cdot)$ generated by g, in which the real order $q \cdot \hat{s}$ is unknown but we can determine the prime factor q with $\tilde{s}$ as the upper bound of $\hat{s}$. The subgroup $(\mathbb{F}, \cdot)$ with generator f and order q could be determined. And the subgroup $\mathbb{G}^q$ of order $\hat{s}$ could be generated by g^q. Apparently, we have $\mathbb{G} = \mathbb{G}^q \times \mathbb{F}$. The DL problem in the subgroup $\mathbb{F}$ is easy to solve by a PPT algorithm Solve without any trapdoor. Given the group description $\mathcal{G}_{\mathrm{HSM}} = (\mathbb{G}, \mathbb{G}^q, \mathbb{F}, g, g_q, f, \tilde{s}, q)$ and input $y = f^x$, the algorithm computes discrete logarithm $x \leftarrow \mathsf{Solve}_{\mathcal{G}_{\mathrm{HSM}}}(y)$ in polynomial time.

HSM Group from Class Group. The HSM group could by instantiated by class groups of imaginary quadratic order. The $\mathsf{GGen}_{\mathrm{HSM}}$ first picks a random prime $\tilde{q}$ such that $q\tilde{q} \equiv 1 \pmod 4$ and $(q/\tilde{q}) = -1$. For fundamental discriminant $\Delta_K = -q\tilde{q}$ and non-maximal order of discriminant $\Delta_q = q^2\Delta_K$, class group $\tilde{\mathbb{G}} = Cl(\Delta_q)$ orders $h(\Delta_q) = q \cdot h(\Delta_K)$ where $h(\Delta_K)$ is the order of $Cl(\Delta_K)$. Let I be the ideal lying above small prime r and ϕ_q^{-1} be the Algorithm 1 in [8]. The generators f and g_q for the subgroup $\mathbb{F} = \langle f \rangle$ and $\mathbb{G}^q = \langle g_q \rangle$ can be computed by $g_q = [\phi_q^{-1}(I^2)]^q$ and $f = [(q^2, q)]$. Accordingly, $g = f \cdot g_q$ generates $\mathbb{G} = \langle g \rangle$. The algorithm outputs $\mathcal{G}_{\mathrm{HSM}} = (\mathbb{G}, \mathbb{G}^q, \mathbb{F}, g, g_q, f, \tilde{s}, q)$.

5.2 CL Encryption for HSM Group

We review the additive homomorphic encryption raised by Castagnos and Laguillaumie [9], in which message space is a cyclic group with prime order q.

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{params}$: it calls group generation algorithm $\mathsf{GGen}_{\mathrm{HSM}}$ described previously, then outputs system parameters as $\mathsf{params} = (\mathbb{G}, \mathbb{G}^q, \mathbb{F}, g, g_q, f, \tilde{s}, q)$. Moreover, we define the statistical distance ϵ_d with constant $S = \tilde{s} \cdot 2^{\epsilon_d}$.
- $\mathsf{KeyGen}(\mathsf{params}) \rightarrow (\mathsf{ek}, \mathsf{dk})$: it picks $\mathsf{dk} \leftarrow_{\$} [0, S]$ and sets public key $\mathsf{ek} = g_q^{\mathsf{dk}}$.
- $\mathsf{Enc}_{\mathsf{ek}}(m) \rightarrow C$: it picks random number in $\rho \leftarrow_{\$} [0, S]$. It composes the ciphertext $C = (C_1, C_2)$ where $C_1 = f^m \mathsf{ek}^\rho$ and $C_2 = g_q^\rho$.
- $\mathsf{Dec}_{\mathsf{dk}}(C) \rightarrow m$: it computes $M = C_1 / C_2^{\mathsf{dk}}$ and calls Solve for $m \leftarrow \mathsf{Solve}_{\mathcal{G}_{\mathrm{HSM}}}(M)$.
- $\mathsf{EvalSum}_{\mathsf{ek}}(C, C') \rightarrow \hat{C}$: it computes the addition by $\hat{C} = (C_1 C_1', C_2 C_2')$ for $C = (C_1, C_2)$ and $C' = (C_1', C_2')$.
- $\mathsf{EvalScal}_{\mathsf{ek}}(C, s) \rightarrow C'$: it scales the message with the scalar s by computing $C' = (C_1^s, C_2^s)$ for inputted ciphertext $C = (C_1, C_2)$.

5.3 ZK Proof with CL Encryption

Instantiating the MtA protocol with CL encryption, the relation R_{Enc} turns to be $\{(m, \rho) | \mathsf{pk} \in \mathbb{G}^q, \rho \in [0, S] : C_1 = f^m \mathsf{pk}^\rho \wedge C_2 = g_q^\rho\}$. And the relation R_B turns to be $\{(\underline{H}, \underline{G}, c_A, c_B, \mathsf{ek}) : (b, \beta, \rho) : \underline{H = bG} \wedge C_1 = \hat{C_1}^{b} \mathsf{pk}^{\rho} f^{-\beta} \wedge C_2 = \hat{C_2}^{b} g_q^{\rho}\}$ where G is the ECC generator. Consequently, the ZK proof for R_{Enc} follows immediately the Algorithm 5 in [22]. And we give the ZK protocol for R_B and its security analysis in Appendix C where the relation is formally named $\mathcal{R}_{\mathsf{Aff}}\underline{wc}$.

Table 3. Bandwidth (bytes) and running time (ms) of each party for a t-party signing

Phase	Sent size	Receive size	Running time
1	32	32 (t − 1)	0.00t + 0.20
2a	4899	4899 (t − 1)	2397.24t + 1832.19
2b	5292 (t − 1)	5292 (t − 1)	588.15t − 2.81
3	32	32 (t − 1)	0.01t + 0.00
4	64	64 (t − 1)	0.03t + 0.37
5	4049	4049 (t − 1)	2749.42t + 1548.67
6	5356 (t − 1)	5356 (t − 1)	584.92t + 5.08
7	128	128 (t − 1)	0.54t + 0.80
8	160	160 (t − 1)	0.92t + 0.70
9	32	32 (t − 1)	0.25t + 0.49
Total	10648t + 9396	20044t − 20044	6321.48t + 3385.70

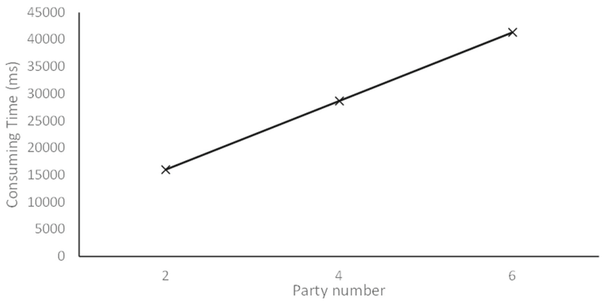

Fig. 1. Total running time of each party for a t-party signing

6 Implementation

We implement the multi-signature with Rust language[5] relying on a modified class group and the related curve library[6] Our implementation targets at 128-bit security and picks the SHA-256 hash function, the Secp256k1 ECC curve and

[5] https://github.com/multisig-ecdsa/multisig-ecdsa.
[6] https://github.com/ZenGo-X/class and https://github.com/ZenGo-X/curv.

a class group with $||\Delta_K|| = 3392$ [12] accordingly. The message size and bandwidth requirement analysis are given theoretically, and all broadcast messages are consider as sending it once (Fig. 1 and Table 3). Benchmark is performed on an AMD Ryzen 7 5800H @3.20 GHz computer with 8 GB RAM.

7 Applications in Blockchain

Nowadays, blockchain plays an increasingly essential role among decentralized cryptocurrencies and many of them rely on the ECDSA signatures. In Bitcoin, a flexible way to check the ownership is adopted, which is known as the Bitcoin script. But the Bitcoin script is not Turing-complete and prevents our scheme to work fully native. We discusses how to adapt our scheme to the Bitcoin here.

Advantages for Using ECDSA Multi-signatures. (1) The signature size is minimized in our scheme and could be extremely bandwidth efficient. (2) Compared to the Schnoor-based or pairing-based multi-signatures, our ECDSA multi-signatures could better fit into the current blockchains. (3) Compared to the threshold ECDSA, our scheme does not require interactive key generation.

Construction of t-of-n Multi-signature. In the *multisig* address in Bitcoin, an address can be associated with a set of n public keys G and a threshold value $t \leq n$. Any set of t signers S can authorize a transaction on behalf of G.

From the current method of forming group public key in our ECDSA multi-signature, we could also construct a m-of-n multi-signature. The idea is to replace the key aggregation protocol in Phase 6 to $(a_1||\ldots||a_t) = \mathsf{H}_1(r, \mathrm{S}, G, m)$.

Combining with Mixing Service. Currently, cryptocurrencies utilizes mixing services to make transaction anonymous but these services always takes a high transaction fee. With our ECDSA multi-signature scheme, users could collaborate by themselves and form a mixing transaction with a single ECDSA signature. Moreover, users don't need to generate auxiliary information when signing the message, because we require nothing other than the original keys.

8 Conclusion

In this paper, we give the first multi-signatures for ECDSA by designing a novel *ephemeral group public key* for the set of signers and using a generic MtA protocol for signing. This scheme can identify the malicious party and is adaptable to the class group, which minimizes the communication cost maximally. As it only produces a single signature, this scheme can be used in blockchain to save transaction cost with the accountability for signers and backward-compatibility with existing addresses.

A Definition for Building Blocks

A.1 ECDSA

ECDSA is a variant of DSA scheme over elliptic curve. It contains a tuple of 4 algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{params}$ generates parameters and calls $\mathsf{GGen}_{\mathrm{ECC}} = (\mathbb{G}, G, q)$ and picks a hash function $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$. It returns $\mathsf{params} = (\mathbb{G}, G, q, \mathsf{H})$. $\mathsf{KeyGen}(\mathsf{params}) \rightarrow (\mathsf{sk}, \mathsf{pk})$ takes security parameter params as input and returns a secret key $\mathsf{sk} = x \leftarrow_{\$} \mathbb{Z}_q$ with a public key $\mathsf{pk} = xG$. $\mathsf{Sign}(\mathsf{sk}, m) \rightarrow \sigma$ computes $R = k^{-1}G$ and takes the x coordinate of R mod q as r. It computes $s = k(\mathsf{H}(m) + xr)$ mod q and returns signature $\sigma = (r, s)$. $\mathsf{Verify}(\mathsf{pk}, \sigma) \rightarrow b$ outputs the verification result $b \in \{0,1\}$ according to whether $R' = \mathsf{H}(m) \cdot s^{-1}G + rs^{-1}\mathsf{pk}$ and the x coordinate of R' mod q is r.

A.2 Additive Homomorphic Encryption

An additive homomorphic encryption allows users to compute the sum of two message in ciphertext. It contains $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EvalSum}, \mathsf{EvalScal})$. $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{params}$ takes security parameters and outputs the system parameter params. $\mathsf{KeyGen}(\mathsf{params}) \rightarrow (\mathsf{ek}, \mathsf{dk})$ computes an encryption key and a decryption key from the system parameters. $\mathsf{Enc}_{\mathsf{ek}}(m) \rightarrow C$ gets the encryption of a message m under the encryption key ek as the ciphertext C. $\mathsf{Dec}_{\mathsf{dk}}(C) \rightarrow m$ recovers the plaintext m from the decryption key dk. $\mathsf{EvalSum}_{\mathsf{ek}}(C, C') \rightarrow \hat{C}$ evaluates the ciphertext $\hat{C} = \mathsf{Enc}_{\mathsf{ek}}(a + b)$ for $C = \mathsf{Enc}_{\mathsf{ek}}(a)$ and $C' = \mathsf{Enc}_{\mathsf{ek}}(b)$. $\mathsf{EvalScal}_{\mathsf{ek}}(C, s) \rightarrow C'$ scales $C = \mathsf{Enc}_{\mathsf{ek}}(a)$ to $C' = \mathsf{Enc}_{\mathsf{ek}}(s \cdot a)$.

The security of the additive homomorphic encryption follows the standard definition of indistinguishability against chosen plaintext attack (IND-CPA).

A.3 Trapdoor Commitment

A commitment scheme contains a algorithms tuple as $(\mathsf{KeyGen}, \mathsf{Com}, \mathsf{Decom})$. $\mathsf{KeyGen}(1^\lambda) \rightarrow \mathsf{pk}$ generates a public key pk. $\mathsf{Com}(\mathsf{pk}, M) \rightarrow (C, D)$ takes the public key pk with a message M then outputs the commitment string C and decommitment string D. $\mathsf{Decom}(\mathsf{pk}, C, D) \rightarrow \{M, \perp\}$ takes the public key pk, the commitment string C, the decommitment string D as input and outputs M if it succeeds and $\perp$ otherwise.

A commitment scheme is considered secure if it fulfills the $\mathsf{correctness}$, hiding and $\mathsf{binding}$ properties. For correctness, it requires that for all messages M and $\mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, then $M \leftarrow \mathsf{Decom}(\mathsf{pk}, \mathsf{Com}(\mathsf{pk}, M))$. Hiding means that every message M_1 and M_2 and $\mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\mathsf{Com}(\mathsf{pk}, M_1)$ and $\mathsf{Com}(\mathsf{pk}, M_2)$ is statistically indistinguishable. The binding property holds if adversary $\mathcal{A}$ wins the game with probability $\Pr[\mathcal{A}$ wins binding game$] \leq \mathsf{negl}(\lambda)$.

Trapdoor Commitment with Efficient ZK Proof. A commitment scheme has the additional algorithms $(\mathsf{KeyGen}', \mathsf{TCom}, \mathsf{TDecom})$ fulfilling the following. $\mathsf{KeyGen}'(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{tk})$ generates a public key pk and a trapdoor tk. $\mathsf{TCom}(\mathsf{pk}, \mathsf{tk}) \rightarrow (C, \mathrm{aux})$ gives commitment C and auxiliary information aux such that TDecom could open it with any message specified. $\mathsf{TDecom}(C, \mathrm{aux}, M) \rightarrow D$ give out the decommitment D by using aux.

The additional algorithm is required to be $\mathsf{trapdoorness}$. We say a commitment scheme fulfilling the trapdoorness property if for all messages M, the following distributions: $\{(\mathsf{pk}, M, C, D) : \mathsf{pk} \leftarrow \mathsf{KeyGen}(1^\lambda), (C, D) \leftarrow \mathsf{Com}(\mathsf{pk}, M)\}$ and $\{(\mathsf{pk}, M, C, D) : (\mathsf{pk}, \mathsf{tk}) \leftarrow \mathsf{KeyGen}'(1^\lambda), (C, \mathrm{aux}) \leftarrow \mathsf{TCom}(\mathsf{pk}, \mathsf{tk}); D \leftarrow \mathsf{TDecom}(C, \mathrm{aux}, M)\}$ are computationally indistinguishable.

Non-malleable Equivocable Commitment Scheme. The equivocable commitment scheme additionally contains KeyGen' and Equiv. $\mathsf{KeyGen}'(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{tk})$ generates a public key pk and a trapdoor tk. $\mathsf{Equiv}(\mathsf{pk}, \mathsf{tk}, C, M') \rightarrow D'$ generates decommitment string D' using trapdoor tk such that $\mathsf{Decom}(\mathsf{pk}, C, D') = M'$.

The additional algorithm is required to be $\mathsf{equivocable}$ and $\mathsf{non\text{-}malleable}$. A commitment scheme is called for equivocable if for all messages M, M', $(\mathsf{pk}, \mathsf{tk}) \leftarrow \mathsf{KeyGen}'(1^\lambda)$, $(C, D) \leftarrow \mathsf{Com}(\mathsf{pk}, M)$ and $D' \leftarrow \mathsf{Equiv}(\mathsf{pk}, \mathsf{tk}, C, M')$, then $M' \leftarrow \mathsf{Decom}(\mathsf{pk}, C, D')$. Non-malleable means that no adversary $\mathcal{A}$ could generate C' related to C such that the decommitment of C' is computed from M.

B Trapdoor Commitments and Its ZK Proofs

We instantiate the trapdoor commitment Com_z as the Pedersen commitment $\mathsf{Com}(\mathsf{pk}, m) \rightarrow (C, D)$ for $C = mG + rH$ and $D = (m, r)$. The ZK proof in Phase 5 could be instantiated directly following the Algorithm 6 of [22]. The ZK proofs in Phase 7 and 8 follow the ZK proof in Sect. 3.3 of [15].

C Zero-Knowledge Proof for MtA(wc)

We give an informal description of assumptions used in HSM group here and refer to [22] for the complete definition. These hard assumptions are defined on prime number $q > 2^\lambda$ and HSM group $\mathcal{G}_{\mathrm{HSM}} = (\mathbb{G}, \mathbb{G}^q, \mathbb{F}, g, g_q, f, \tilde{s}, q)$ for $\mathcal{G}_{\mathrm{HSM}} \leftarrow \mathsf{GGen}_{\mathrm{HSM}}(1^\lambda)$. If we denote H as a generator in the ECC group with prime order q, then

$$\mathcal{R}_{\mathsf{Aff}\underline{\mathsf{wc}}} = \left\{ \begin{array}{r|l} (\mathsf{pk}, C_1, C_2, \tilde{C}_1, \tilde{C}_2); & \mathsf{pk}, C_2 \in G^q, C_1 \in G \setminus F, \gamma\beta \in \mathbb{Z}_q, \rho \in [0, S] : \\ (\gamma, \beta, \rho) & \tilde{C}_1 = C_1^\gamma f^\beta \mathsf{pk}^\rho \wedge \tilde{C}_2 = C_2^\gamma g_q^\rho \wedge \underline{H' = \gamma H} \end{array} \right\}.$$

We have 2 important facts in HSM group. The first one if Adaptive root subgroup hardness. Given q and HSM group $\mathcal{G}_{\text{HSM}}$, it's hard to find $u^\ell = w$ and $w^q \neq 1$ for specific $\ell \leftarrow \mathsf{Primes}(\lambda)$. The other one is Non-trivial order hardness, which states that given q and $\mathcal{G}_{\text{HSM}}$, it's hard to find $h \neq 1 \in \mathbb{G}$ such that $h^d = 1$ and $d < q$.

Theorem 2. *The protocol* $\mathsf{ZKPoKAff}\underline{\mathsf{wc}}$ *is an argument of knowledge in the generic group model.*

Proof. We rewind the adversary on fresh challenges ℓ so that each accepting transcript outputs an $(Q_1, Q_2, R_1, R_2, P_1, r_\rho, r_\gamma, \ell)$. Recall that we have $C_2 \in G^q$. By the PoKRepS protocol in [22], with overwhelming probability there exists $\rho^*, \gamma^* \in \mathbb{Z}$ s.t. $\rho^* = r_\rho \bmod \ell$ and $\gamma^* = r_\gamma \bmod \ell$, and $g_q^{\rho^*} C_2^{\gamma^*} = S_2 \tilde{C}_2^c$. Since $S_2 \tilde{C}_2^c = (D_2 E_2)^q g_q^{e_\rho} C_2^{e_\gamma}$, it implies $\rho^* = e_\rho \bmod q$ and $\gamma^* = e_\gamma \bmod q$. Considering 2 cases, $\mathsf{pk}^{\rho^*} C_1^{\gamma^*} f^{u_\beta} = S_1 \tilde{C}_1^c$ is at overwhelming probability.

Next we consider the rewinding of c. The extractor obtains a pair of accepting transcripts with $(\rho^*, \gamma^*, u_\beta, c)$ and $(\rho', \gamma', u'_\beta, c')$. The extractor can compute $\Delta_\rho = \rho^* - \rho'$, $\Delta_\gamma = \gamma^* - \gamma'$ and $\Delta_{u_\beta} = u_\beta - u'_\beta \bmod q$. We denote $\rho = \frac{\Delta_\rho}{\Delta_c}, \gamma = \frac{\Delta_\gamma}{\Delta c}$ and $\beta = \frac{\Delta_{u_\beta}}{\Delta_c} \bmod q$. Hence we have $\tilde{C}_1^{\Delta_c} = (\mathsf{pk}^\rho C_1^\gamma f^\beta)^{\Delta_c}$. If $\tilde{C}_1 \neq \mathsf{pk}^\rho C_1^\gamma f^\beta$, then $\frac{\mathsf{pk}^\rho f^\beta C_1^\gamma}{\tilde{C}_1}$ is a non-trivial element of order $\Delta_c < q$ which contradicts with the non-trivial element and its order in the generic group model.

As our scheme includes a sub-protocol ZKPoKRepS on input $\tilde{C}_2$ w.r.t. bases $g_q \in G \setminus F$. Since ZKPoKRepS is an argument of knowledge, there exists an extractor to extract the same (γ, ρ) such that $\tilde{C}_2 = C_2^\gamma g_q^\rho$. <u>Similar argument applies to H. There exists an extractor to extract the sameγ such that $H' = \gamma H$.</u> Hence the extractor can output (β, γ, ρ) such that $\tilde{C}_1 = C_1^\gamma f^\beta \mathsf{pk}^\rho$, $\tilde{C}_2 = C_2^\gamma g_q^\rho$ <u>and $H' = \gamma H$</u>. □

Theorem 3. *The protocol* $\mathsf{ZKPoKAff}\underline{\mathsf{wc}}$ *is an honest-verifier statistically zero-knowledge argument of knowledge for relation* $\mathcal{R}_{\mathsf{Aff}\underline{\mathsf{wc}}}$ *in the generic group model.*

Proof. The simulator Sim randomly picks a challenge $c' \in [0, q-1]$ and a prime $\ell' \in \mathsf{Prime}(\lambda)$. It picks a random $u'_\beta \in \mathbb{Z}_q$, $q'_\rho, q'_\gamma \in [0, B-1]$ and $r'_\rho, r'_\gamma \in [0, \ell'-1]$.

It finds $d'_\rho, d'_\gamma \in \mathbb{Z}$ and $e'_\rho, e'_\gamma \in [0, q-1]$ such that $d'_\rho q + e'_\rho = q'_\rho \ell' + r'_\rho, \quad d'_\gamma q + e'_\gamma = q'_\gamma \ell' + r'_\gamma$.

It computes:

$$
\begin{aligned}
&D'_1 = \mathsf{pk}^{d'_\rho}, \quad D'_2 = g_q^{d'_\rho}, \quad E'_1 = C_1^{d'_\gamma}, \quad E'_2 = C_2^{d'_\gamma},\\
&Q'_1 = \mathsf{pk}^{q'_\rho}, \quad Q'_2 = g_q^{q'_\rho}, \quad R'_1 = C_1^{q'_\gamma}, \quad R'_2 = C_2^{q'_\gamma}, \quad \underline{P'_1 = q'_\gamma H},\\
&S'_1 = (Q'_1 R'_1)^{\ell'} \mathsf{pk}^{r'_\rho} C_1^{r'_\gamma} f^{u'_\beta} \tilde{C}_1^{-c'}, \quad S'_2 = (Q'_2 R'_2)^{\ell'} g_q^{r'_\rho} C_2^{r'_\gamma} \tilde{C}_2^{-c'},\\
&\underline{S'_3 = \ell' P'_1 + r'_\gamma H + -c' H'}.
\end{aligned}
$$

We argue that The simulated transcript is indistinguishable from a real one $(S_1, S_2, \underline{S_3}, c, u_\beta, D_1, D_2, E_1, E_2, e_\rho, \ell, Q_1, Q_2, R_1, R_2, \underline{P_1}, r_\rho, r_\gamma)$ between a prover and a verifier. Sim chooses (ℓ', c') identically to the honest verifier. Both u_β and u'_β are uniformly distributed in $\mathbb{Z}_q$. $(S'_1, S'_2, \underline{S'_3}, D'_1, D'_2, E'_1, E'_2, e'_\rho, e'_\gamma)$ is uniquely defined by the other values such that the verification holds.

We compare the simulated transcript $(Q'_1, Q'_2, R'_1, R'_2, \underline{P'_1}, r'_\rho, r'_\gamma)$ and the real transcript $(Q_1, Q_2, R_1, R_2, \underline{P_1}, r_\rho, r_\gamma)$. We need to prove that, in the real protocol, independent of ℓ and c, the either r_ρ or r_γ has a negligible statistical distance from the uniform distribution over $[0, \ell - 1]$ and each one of $\mathsf{pk}^{q_\rho}, g_q^{q_\rho}, \overline{C_1^{q_\gamma}, C_2^{q_\gamma}}, \underline{q_\gamma H}$ has negligible statistical from uniform over $G_k = \langle \mathsf{pk} \rangle, G^q$, $\overline{G_1 = \langle C_1 \rangle, G_2} = \langle C_2 \rangle, \langle h \rangle$ respectively. In addition, each of $Q_1, Q_2, R_1, R_2, \underline{P_1}, r_\rho$, r_γ are independent from others. Then, the simulator produces statistically indistinguishable transcripts. The complete proof is as follows.

Consider fixed values of c, ρ and ℓ. In the real protocol, the prover computes $u_\rho = c\rho + s_\rho$ where s_ρ is uniform in $[-B, B]$ and sets $r_\rho = u_\rho \bmod \ell$. By Fact 1, the value of u_ρ is distributed uniformly over a range of $2B + 1$ consecutive integers, thus r_ρ has a statistical distance at most $\ell/(2B + 1)$ from uniform over $[0, \ell - 1]$. This bounds the distance between the real r_ρ and the simulated r'_ρ, which is uniform over $[0, \ell - 1]$. Similarly, $\ell/(2B + 1)$ also bounds the distance between r_γ and r'_γ

Next, $g_q^{q_\rho}$ is statistically indistinguishable from uniform in G^q. By the triangle inequality, the statistical distance of $q_\rho \bmod |G^q|$ from uniform is at most $\frac{2^{\lambda+1}}{B} + \frac{2^{\lambda-1}|G^q|}{B+1-2^\lambda}$. We consider the joint distribution of $(\mathsf{pk}^{q_\rho}, g_q^{q_\rho})$ and r_ρ. Consider the conditional distribution of $q_\rho | r_\rho$. Note that $q_\rho = z$ if $(s_\rho - r_\rho)/\ell = z$. We repeat a similar argument as above for bounding the distribution of q_ρ from uniform. For each possible value of z, there always exists a unique value of s_ρ such that $\left\lfloor \frac{s_\rho}{\ell} \right\rfloor = z$ and $s_\rho = 0 \bmod \ell$, except possibly at the two endpoints E_1, E_2 of the range of q_ρ. When r_ρ disqualifies the two points E_1 and E_2, then each of the remaining points $z \notin \{E_1, E_2\}$ still have equal probability mass, and thus the probability $\Pr(q_\rho = z | r_\rho)$ increases by at most $\frac{1}{|Y|} - \frac{\ell}{2B+1}$, which also applies to the variable $(\mathsf{pk}^{q_\rho}, g_q^{q_\rho}) | r_\rho$. Similarly, the probability $\Pr(q_\gamma = z | r_\gamma)$ increases by at most $\frac{1}{|Y|} - \frac{\ell}{2B+1}$, which also applies to the variable $(\mathsf{pk}^{q_\gamma}, g_q^{q_\gamma}, \underline{h^{q_\gamma}}) | r_\gamma$.

We can compare the joint distributions $X'_\rho = (\mathsf{pk}^{q_\rho}, g_q^{q_\rho}, r_\rho)$ to the simulated distribution $Y'_\rho = (\mathsf{pk}^{q'_\rho}, g_q^{q'_\rho}, r'_\rho)$ using Fact 3.

Algorithm 1: Protocol $\mathsf{ZKPoKAff}\underline{\mathsf{wc}}$ for the relation $\mathcal{R}_{\mathsf{Aff(wc)}}$

Param: $\mathcal{G}_{\text{HSM}} \leftarrow \mathsf{GGen}_{\text{HSM},q}(1^\lambda)$, $B = 2^{\epsilon_d+\lambda+3}q\tilde{s}$, where $\epsilon_d = 80$.
Input: $C_1, C_2, \tilde{C}_1, \tilde{C}_2, \mathsf{pk} \in G^q$.
Witness: $\rho \in [0, S], \beta \in \mathbb{Z}_q$, $\gamma \in \mathbb{Z}_q$, where $S = \tilde{s} \cdot 2^{\epsilon_d}$.

1 Prover chooses $s_\rho, s_\gamma \xleftarrow{\$} [-B, B]$, $s_\beta \xleftarrow{\$} \mathbb{Z}_q$ and computes:

$$S_1 = C_1^{s_\gamma} f^{s_\beta} \mathsf{pk}^{s_\rho}, \quad S_2 = C_2^{s_\gamma} g_q^{s_\rho}, \quad \underline{S_3 = h^{s_\gamma}}.$$

Prover sends $(S_1, S_2, \underline{S_3})$ to the verifier.

2 Verifier sends $c \xleftarrow{\$} [0, q-1]$ and $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$ to the prover.

3 Prover computes:

$$u_\beta = s_\beta + c\beta \mod q, \quad u_\rho = s_\rho + c\rho, \quad u_\gamma = s_\gamma + c\gamma.$$

Prover finds $d_\rho \in \mathbb{Z}$ and $e_\rho, e_\gamma \in [0, q-1]$ s.t. $u_\rho = d_\rho q + e_\rho$ and $u_\gamma = d_\gamma q + e_\gamma$.
Prover computes:

$$D_1 = \mathsf{pk}^{d_\rho}, \quad D_2 = g_q^{d_\rho}, \quad E_1 = C_1^{d_\gamma}, \quad E_2 = C_2^{d_\gamma}.$$

Prover sends $(u_\beta, D_1, D_2, E_1, E_2, e_\rho, e_\gamma)$ to the verifier.

4 Verifier check if $e_\rho, e_\gamma \in [0, q-1]$ and:

$$(D_1E_1)^q \mathsf{pk}^{e_\rho} C_1^{e_\gamma} f^{u_\beta} = S_1\tilde{C}_1^c, \quad (D_2E_2)^q g_q^{e_\rho} C_2^{e_\gamma} = S_2\tilde{C}_2^c,$$
$$\underline{h^{e_\gamma} = S_3H^c}.$$

If so, the verifier sends $\ell \xleftarrow{\$} \mathsf{Primes}(\lambda)$.

5 Prover finds $q_\rho \in \mathbb{Z}$ and $r_\rho, r_\gamma \in [0, \ell-1]$ s.t. $u_\rho = q_\rho\ell + r_\rho$ and $u_\gamma = q_\gamma\ell + r_\gamma$.
Prover computes:

$$Q_1 = \mathsf{pk}^{q_\rho}, \quad Q_2 = g_q^{q_\rho}, \quad R_1 = C_1^{q_\gamma}, \quad R_2 = C_2^{q_\gamma}, \quad \underline{P_1 = h^{q_\gamma}}.$$

Prover sends $(Q_1, Q_2, R_1, R_2, \underline{P_1}, r_\rho, r_\gamma)$ to the verifier.

6 Verifier accepts if $r_\rho, r_\gamma \in [0, \ell-1]$ and:

$$(Q_1R_1)^\ell \mathsf{pk}^{r_\rho} C_1^{r_\gamma} f^{u_\beta} = S_1\tilde{C}_1^c, \quad (Q_2R_2)^\ell g_q^{r_\rho} C_2^{r_\gamma} = S_2\tilde{C}_2^c,$$
$$\underline{P_1^\ell h^{r_\gamma} = S_3H^c}.$$

References

1. Kılınç Alper, H., Burdges, J.: Two-round trip Schnorr multi-signatures via delinearized witnesses. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 157–188. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_7
2. Bagherzandi, A., Cheon, J.H., Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In: Ning, P., Syverson, P.F., Jha, S. (eds.) CCS 2008, pp. 449–458. ACM (2008)

3. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) CCS 2006, pp. 390–399. ACM (2006)
4. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36288-6_3
5. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 435–464. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_15
6. Bresson, E., Stern, J., Szydlo, M.: Threshold ring signatures and applications to ad-hoc groups. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 465–480. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_30
7. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ECDSA from hash proof systems and efficient instantiations. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 191–221. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_7
8. Castagnos, G., Laguillaumie, F.: On the security of cryptosystems with quadratic decryption: the nicest cryptanalysis. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 260–277. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_15
9. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from DDH. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 487–505. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16715-2_26
10. Chatzigiannis, P., Chalkias, K.: Proof of assets in the diem blockchain. In: Zhou, J., et al. (eds.) ACNS 2021. LNCS, vol. 12809, pp. 27–41. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81645-2_3
11. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_28
12. Dobson, S., Galbraith, S.D.: Trustless groups of unknown order with hyperelliptic curves. IACR Cryptology ePrint Archive, p. 196 (2020). https://eprint.iacr.org/2020/196
13. Drijvers, M., et al.: On the security of two-round multi-signatures. In: 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, 19–23 May 2019, pp. 1084–1101. IEEE (2019). https://doi.org/10.1109/SP.2019.00050
14. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) CCS 2018, pp. 1179–1194. ACM (2018)
15. Gennaro, R., Goldfeder, S.: One round threshold ECDSA with identifiable abort. Cryptology ePrint Archive, Report 2020/540 (2020). https://eprint.iacr.org/2020/540
16. Itakura, K., Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC Res. Dev. **71**, 1–8 (1983)
17. Khali, H., Farah, A.: DSA and ECDSA-based multi-signature schemes. Int. J. Comput. Sci. Netw. Secur. **7**(7), 11–19 (2007)
18. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_28

19. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple Schnorr multi-signatures with applications to Bitcoin. Des. Codes Cryptogr. **87**(9), 2139–2164 (2019). https://doi.org/10.1007/s10623-019-00608-x
20. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: extended abstract. In: Reiter, M.K., Samarati, P. (eds.) CCS 2001, pp. 245–254. ACM (2001)
21. Nick, J., Ruffing, T., Seurin, Y.: MuSig2: simple two-round Schnorr multi-signatures. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 189–221. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_8
22. Yuen, T.H., Cui, H., Xie, X.: Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12710, pp. 481–511. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75245-3_18