Threshold EdDSA Signature for Blockchain-based Decentralized Finance Applications

Yang Shi
shiyang@tongji.edu.cn
Tongji University
China

Junqing Liang
2031556@tongji.edu.cn
Tongji University
China

Mianhong Li
1931545@tongji.edu.cn
Tongji University
China

Tianchen Ma
m_tttttc@tongji.edu.cn
Tongji University
China

Guodong Ye
yegd@gdou.edu.cn
Guangdong Ocean University
China

Jiangfeng Li
lijf@tongji.edu.cn
Tongji University
China

Qinpei Zhao
qinpeizhao@tongji.edu.cn
Tongji University
China

ABSTRACT

The threshold digital signature technique is important for decentralized finance (DeFi) applications such as asset custody and cross-chain interoperations. The Edwards-curve digital signature algorithm (EdDSA) is widely used in blockchains, e.g., Libra/Diem; however, no suitable threshold solution exists. Therefore, to bridge this gap, we propose a threshold EdDSA that allows n parties to generate keys in a decentralized and distributed manner. Any $t+1$-of-n parties can generate standard EdDSA signatures. This scheme supports an arbitrary threshold (t, n) and has been proven to be secure against at most t malicious adversaries. The theoretical analysis (computation complexity and communication footprints) and experimental results demonstrate that the proposed scheme performs efficiently on cloud servers and embedded devices. Furthermore, the proposed scheme is integrated with Tendermint, a blockchain framework that uses EdDSA, to generate keys and sign transactions in a decentralized manner, which indicates that this scheme is compatible with blockchains for supporting DeFi applications.

KEYWORDS

decentralized finance, blockchain, EdDSA, threshold signature, asset custody, cross-chain

ACM Reference Format:

Yang Shi, Junqing Liang, Mianhong Li, Tianchen Ma, Guodong Ye, Jiangfeng Li, and Qinpei Zhao. 2022. Threshold EdDSA Signature for Blockchain-based Decentralized Finance Applications . In *25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2022), October 26–28, 2022, Limassol, Cyprus.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3545948.3545977

Jiangfeng Li is the corresponding author.

RAID 2022, October 26–28, 2022, Limassol, Cyprus

ACM ISBN 978-1-4503-9704-9/22/10…$15.00
https://doi.org/10.1145/3545948.3545977

1 INTRODUCTION

Decentralized finance (DeFi) [50], which is one of the most discussed emerging technological changes in global financial systems, has been increasingly mentioned when discussing the future evolution of finance and its regulation. Owing to the assistance of blockchain technology, DeFi services can be used to expand financial inclusion, promote open access, encourage permission-free innovation, and generate new opportunities for entrepreneurs.

DeFi offers several potential solutions to inherent problems in the traditional financial infrastructure. Moreover, Defi is also expected to offer a way to innovate economic models in the metaverse [26]. Major business models in DeFi include decentralized currencies, contracts, and payments[17, 37, 42]. Blockchain-based asset custody and cross-chain applications are two main business types in DeFi. On the one hand, a decentralized asset custody scheme[18] can be used to support several custodians and securely holds consignor assets. On the other hand, a cross-chain asset mapping service[16] (e.g., a cross-chain asset exchange) is used to map encrypted assets on one blockchain to tokens on another for chain interoperability.

The custody of crypto assets, including cryptocurrencies [19], provides convenience for digital asset applications. Since blockchain technology provides technical support to cryptocurrency, its decentralized feature ensures a trusted environment and improves the security of the asset custody business. However, safety and security issues still exist, such as key loss [30], the honesty of service providers, and the intrusion on systems, which motivates the use of threshold signature schemes [46].

A threshold signature scheme allows a group of parties to generate a digital signature collectively without obtaining information about the private signing key. Furthermore, a decentralized (also called dealer-free or dealerless) threshold signature scheme uses a distributed key generation (DKG) protocol in the setup phase to generate secret shares to sign messages without a trusted third party (i.e., a dealer). Decentralized threshold signature schemes

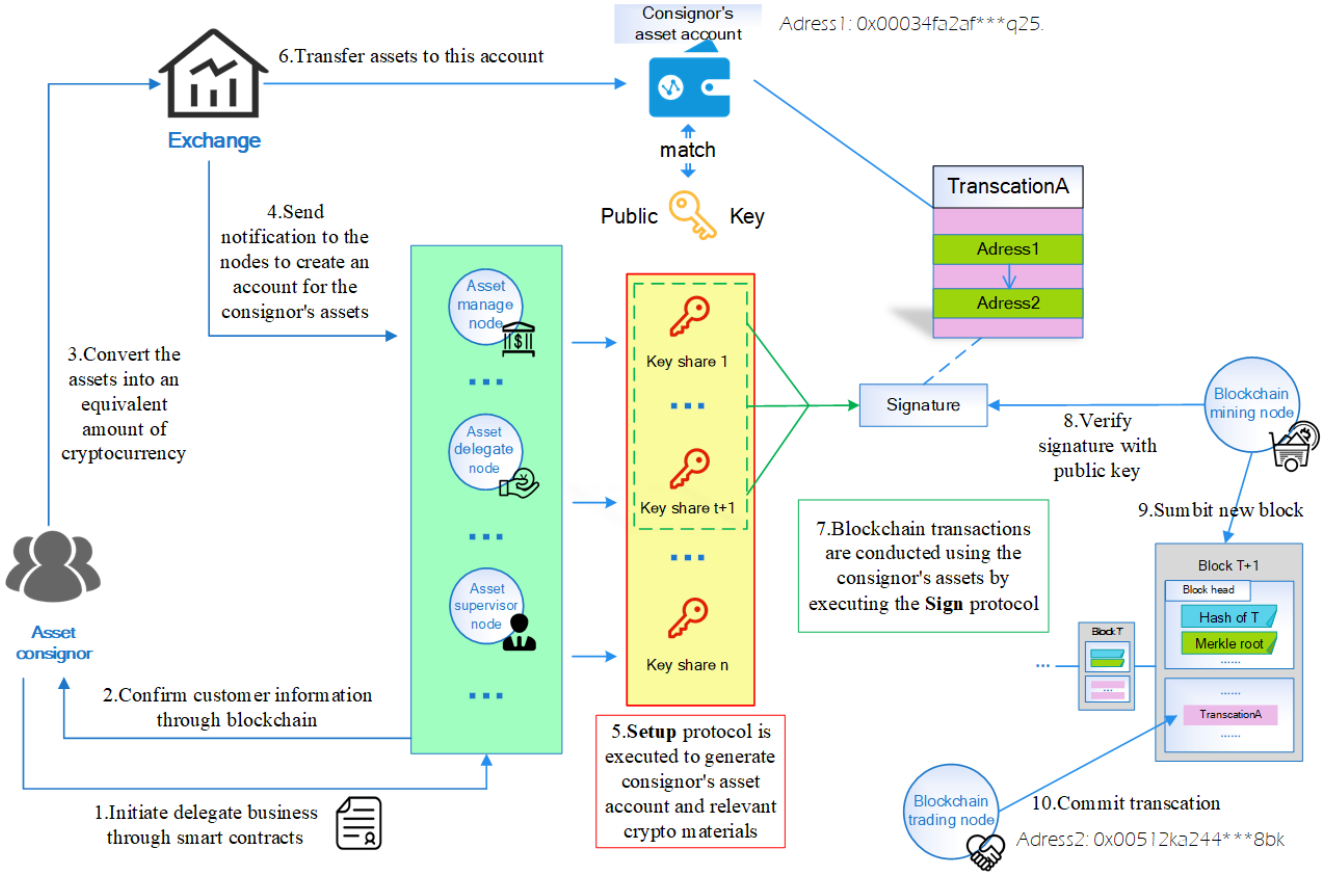


Figure 1: Application scenario of decentralized threshold signature schemes in assets custody

have been suggested for use in asset custody and cross-chain in both industry and academia.

The general application scenarios of such schemes in asset custody and cross-chain are illustrated in Figures 1 and 2, respectively.

The asset custody process can be divided into three steps: contract initiation, pre-transaction setup and asset transformation.

(i) Contract initiation (Steps 1-2 in Figure 1). The asset consignors create smart contracts to initiate asset delegation. Related nodes on the blockchain network confirm the customer information, and preparation for the following transactions begins.
(ii) Pretransaction setup (Steps 3-5 in Figure 1). The consignors convert the assets into equivalent amount of cryptocurrency. When the amount is set, notifications from the exchange are sent to the asset nodes, the setup protocol of the blockchain is started, and an account for the consignor's assets is generated.
(iii) Asset transformation (Steps 6-10 in Figure 1). When the setup protocol is finished, the equivalent assets are transferred from the exchange to the consignor's account. Then, the consignor will make transactions to the asset possessor's account. The details of the process are similar to those of normal blockchains.

Cross-chain services are usually used to map crypto assets on one blockchain to tokens on another blockchain for interchain operability. For example, in Figure 2, there are two parties named Alice and Bob. Alice wants to make a cross-chain transaction, so she sends request to the validator nodes. Then, a locked account address, which is used to store the cryptocurrency during the transaction, is sent to Alice through the validator nodes. The transactions made by Alice are sent to the locked account.

After the aforementioned transaction is confirmed, a new transaction is made from the locked account to Bob's account by the validator nodes, transferring the matching amount of cryptocurrency on Bob's blockchain, and Bob will receive the equivalent cryptocurrency on his blockchain.

Concretely in industry, there are a number of public chain and cross-chain systems that employ the decentralized threshold signature scheme. The following are representative examples:

(i) Binance[7] enables wallet providers and custodians to avoid a single point of failure in private keys with distributed key management by open-sourcing the threshold signature library.
(ii) Wanchain[2] provides a decentralized locked account mechanism that utilizes threshold signature schemes to perform account management without any third-party participation or credit endorsement of any entities.
(iii) Fusion[29] presents a decentralized control rights management framework, wherein a threshold signature scheme is

used to hold and transfer assets on behalf of the user across heterogeneous chains.

(iv) RenVM[3] is a permissionless and decentralized virtual machine protocol that utilizes a threshold signature for decentralized applications, which enables cross-chain lending, exchanges, and collateralization.

(v) Terra[27] provides a blockchain network powered by a family of coins. This technique is used to split the key generation and key management responsibilities among multiple parties.

Since the threshold signature scheme is applied to the blockchain signature process, the multiparty participation feature of the threshold signature can be used to increase resistance to collusion attacks [49], make the signature process more reliable, and improve the credibility of data. In academia, blockchain applications have regained interest in threshold signature schemes in the last few years [10, 14, 19–21, 24, 25, 32, 33, 39, 47]. These studies focused on combining threshold signature schemes with signature schemes used in blockchain protocols, namely, the elliptic curve digital signature algorithm (ECDSA) [38] and EdDSA [11, 12].

Currently, EdDSA, which is second only to ECDSA, is one of the most widely used digital signatures in blockchain technology. Furthermore, because of the potential of EdDSA, many projects have adopted EdDSA as their main signature scheme, with many more projects starting to consider a technological migration towards EdDSA. Some well-known blockchain systems, including Etherum (ETH)[48], Hyperledger Fabric[9], Tendermint[40], Cardano (ADA)[6], and Tezos[36], use EdDSA as one of their basic signature schemes. In addition, some blockchain projects, such as Libra/Diem[8] and Monero[1], use EdDSA as their only signature algorithm. Thus, we believe that EdDSA will play an increasingly important role in blockchains.

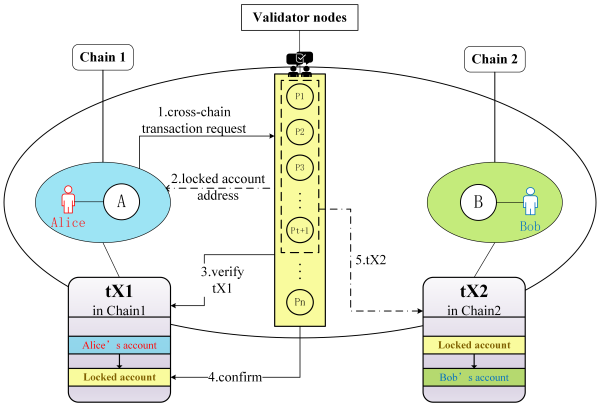


Figure 2: Application scenario of decentralized threshold signature schemes in cross-chain transactions

Lindell et al.[41], Gennaro et al.[32], and Doerner et al.[25] proposed several threshold ECDSA signature schemes. However, these schemes can only be integrated into blockchains using ECDSA. In contrast, our scheme essentially differs from those threshold ECDSA schemes because EdDSA differs from ECDSA, as we expand on in Section 2.3. In terms of threshold schemes related to EdDSA [10, 15, 28], there are limitations such as: (i) Feng et al.'s scheme [28] only supports the (2,2)-threshold, and Battagliola et al.'s scheme [10] only supports the (2,3)-threshold; and (ii) [15] uses time-consuming techniques, such as garbled circuits (GC). For a detailed analysis and comparison of these schemes, refer to Section 6. To summarize, the challenges are as follows.

(i) Existing threshold signature schemes of ECDSA are not suitable for a number of blockchains that use the EdDSA signature scheme, which is not easy to change.

(ii) Although related works on threshold signature schemes of EdDSA have been proposed, there are considerable limitations with respect to flexibility, applicability and efficiency.

Our contributions are summarized as follows:

- In this paper, a threshold signature scheme is proposed that is fully compatible with EdDSA and supports arbitrary thresholds (t, n) without using generic multiparty computation approaches, such as GC, which cost a considerable computation load and have a large communication footprint. This scheme enables n parties to generate keys in a decentralized, dealer-free, and distributed manner, and any $t+1$-of-n parties can generate standard EdDSA signatures. To achieve these features, the proposed scheme significantly distincts from existing constructions of threshold EdDSAs.
- Theoretically, we formally prove the correctness of the proposed scheme (in Propositions 3.1 and 3.2) and security (in Theorem 4.2).
- Practically, we implement the scheme and evaluate its performance using various parameters on two categories of computing platforms, i.e., cloud servers and embedded devices. The performance in terms of the computation and communication is considerably promising, as the computation on cloud servers is of the order of milliseconds, and the communication cost is of the order of kilobytes. Furthermore, we integrate the proposed scheme with a blockchain framework using EdDSA, i.e., Tendermint; the results of this experimental study show that the proposed scheme can effectively work with a blockchain.

The remainder of this paper is organized as follows. In Section 2, the preliminaries are presented. In Section 3, the proposed scheme is discussed and its correctness is verified. In Section 4, the scheme's security is analyzed and proven. In Section 5, the theoretical analysis and experimental evaluation of the efficiency are presented. In Section 6, related works and differences are discussed. Finally, in Section 7, this paper is concluded.

2 PRELIMINARIES

2.1 EdDSA

The EdDSA signature scheme has the following seven system parameters [11, 12]: integer $b \geq 10$, $2b$-bit cryptographic hash function H, prime power $q(=1 \bmod 4)$, $(b-1)$-bit encoding of the elements in the finite field of q (i.e., $\mathbb{F}_q$), nonsquare element $d \in \mathbb{F}_q$, prime $l \in [2^{b-4}, 2^{b-3}]$, and element $B \in \mathbb{E} - \{(0,1)\}$.

$$\mathbb{E} = \{(x, y) \in (\mathbb{F}_q)^2 \mid x^2 + y^2 = 1 + dx^2y^2\} \quad (1)$$

where d is not a square. Let $\mathbb{E}$ be a twisted Edwards curve, "+" be the addition operator, and $0 = (0, 1) \in \mathbb{E}$ be the zero element, we have that $(\mathbb{E}, +, 0)$ is an additive group.

The EdDSA secret key is a b-bit string κ. The hash $H(\kappa) = (h_0, h_1, \cdots, h_{2b-1})$ determines an integer $s = 2^{b-2} + \sum_{i=3}^{b-3} 2^i h_i$ as a secret scale. The public verification key is $A = sB$, and $\underline{A}$ denotes the Little-Endian encoding on A.

For an arbitrary message M, the signature under secret key s is generated as follows. First, we compute

$$r = H(h_b, \ldots, h_{2b-1}, M) \in \{0, 1, \ldots, 2^{2b} - 1\}, \tag{2}$$

where the output of H is a $2b$-bit string, which is used as an integer in the Little-Endian form. Next, we compute

$$R = rB \tag{3}$$

and

$$S = \left(r + H\left(\underline{R}, \underline{A}, M\right) s\right) \bmod l. \tag{4}$$

The string $(\underline{R}, \underline{S})$ is the signature, where the underline denotes a b-bit Little-Endian encoding.

To verify a message M and its corresponding signature $(\underline{R}, \underline{S})$, the verifier decodes the signature $(\underline{R}, \underline{S})$ and checks whether equation (5) holds; here, 2^3 denotes the cofactor for Ed25519 [11].

$$2^3 SB = 2^3 R + 2^3 H\left(\underline{R}, \underline{A}, M\right) A \tag{5}$$

2.2 Threshold signature

Threshold signature schemes remove the burden of the single atomic private key and split the responsibility between multiple parties. The threshold signature schemes originate from Shamir's[45] (t, n) secret sharing scheme, which splits a secret into n shares, among which more than t shares can recover the secret, whereas any t or less than t shares reveal no information about the secret.

Threshold signature schemes are divided into dealer-based and dealer-free schemes. When using a dealer-based threshold signature scheme, verification on whether shares received by shareholders are sent by the dealer is required. The participation of a dealer and noninteraction between parties is required in this scheme. In a dealer-free threshold signature scheme, participants need to verify the validity of shares before performing secret calculations when they receive shares from others during the secret reconstruction process. This process requires the interaction of all parties without the involvement of the dealer.

2.3 Comparison of related signatures

Threshold signatures have progressed on many algorithms, including RSA signing[46], ElGamal signing[23], Schnorr signing[39] and more. However, EdDSA and ECDSA still require much effort to construct simple and efficient protocols for threshold signing.

Table 1 shows the difference between EC-Schnorr, EdDSA and ECDSA. In all cases, the public verification key is an elliptic curve point Q, and the private key is x such that $q = xG$, where G is the generator point of an elliptic curve group of order q.

It is easy to convert the Schnorr signature scheme into a distributed version since the private key x and the value k are both used in a linear equation. Thus, two parties with shares x_1 and x_2 such that $Q = (x_1 + x_2)G$ can each locally choose k_1, k_2. Then, set $R = k_1 G + k_2 G = (k_1 + k_2)G$. Then each party can locally compute r and $s_i = k_i - x_i r \bmod q$, send the results to each other, sum $s = s_1 + s_2 \bmod q$ and output a valid Schnorr signature (r, s). Even in the case of malicious adversaries, some highly efficient proofs of knowledge, such as discrete logarithms, can be used to ensure that R is uniformly distributed.

In contrast, due to the difficulty of computing r without knowing k, there is no simple approach to convert these two schemes into a threshold variant. In EdDSA signing, the equation for computing s, including the hash value on k, is complex to compute in a distributed way. Additionally, in ECDSA signing, the equation for computing s includes k^{-1}. Given shares k_1, k_2 such that $k_1 + k_2 = k \bmod q$, it is very difficult to compute k_1'. k_2' such that $k_1' + k_2' = k^{-1} \bmod q$.

3 PROPOSED SCHEME

3.1 Overview

We propose a dealerless threshold EdDSA signature scheme and prove the correctness of this scheme. The frequently used symbols are listed in Table 2.

As illustrated in Figure 3, the proposed scheme comprises two protocols: the **Setup** protocol for generating shares of signing key and secret vectors among parties in a distributed manner and the **Sign** protocol for generating signatures on given messages, where the generated signatures can be verified by the standard verification algorithm of EdDSA. We assume that the EdDSA system parameters have already been shared among all parties.

When initially addressing this problem, we found that the distributed computation of the SHA-512 hash function is required in both the distributed key generation and the signing. Unfortunately, it is difficult to efficiently and distributively compute typical cryptographic hash functions, such as the family of MD* (e.g., MD5) and SHA* (e.g., SHA256, SHA512, and even SHA-3 algorithms), regardless of whether they use the Merkle Damgård structure [22, 43] or the sponge structure [13]. Although distributing the computing of such a cryptographic hash function could be done using generic approaches from secure multiparty computation (MPC) theory, its computation and communication costs are considerably high for practical applications. For example, Charlotte et al. [15] computed the hash function (SHA512) via a form of MPC to create a signature of EdDSA. In [15] , the computation cost was 116 ms to complete the **Sign** protocol (HSS) in LAN, while using our scheme, the cost is only 2.3 ms.

We realize that obtaining the values of s and r via distributively computing cryptographic hash functions in distributed key generation (i.e., Setup protocol) and signature generation (i.e., Signing protocol) are two prominent obstacles to design an efficient threshold EdDSA scheme. Therefore, we avoid these obstacles by using distributed random element generation approaches in our scheme (see Eqs. (22) and (23) and Steps (III) and (IV) of the Sign protocol). The signatures generated by using our scheme can be verified using the corresponding public key, which indicates the full compatibility with the original one-party EdDSA signature schemes.

3.2 Protocols

The Setup protocol is as follows.

(I) Let $\mathbb{Q}_1 = \{P_i | i \in \{1, \ldots, n\}\}$ be the initial set of qualified parties. For each $i \in \{1, \ldots, n\}$, party P_i:

Table 1: Comparison between signature schemes.

EC-Schnorr	EdDSA	ECDSA
Choose a random k	**Compute** $k = H(x\|\|m)$	Choose a random k
Compute $R = kG$	Compute $R = kG$	Compute $R = kG$
Compute $r = H(m\|\|R)$	Compute $r = H(R)$	Compute $r = H(R)$
Compute $s = k - xr \bmod q$	Compute $s = k - xH(m\|\|r) \bmod q$	**Compute** $s = k^{-1}(H(m) + xr) \bmod q$
Output (r, s)	Output (r, s)	Output (r, s)

Table 2: Frequently used symbols.

Symbol	Description
q	A prime ($q = 1 \bmod 4$).
$\mathbb{F}_q$	The finite field of a character q.
b	The length parameter of EdDSA.
b'	The auxiliary length parameter.
H	The hash function used in EdDSA that outputs a $2b$-bit value.
H'	A hash function that outputs a b'-bit value.
$\mathbb{E}$	The set of points on the Edwards curve.
B	An element of $\mathbb{E}$, which generates the group $\mathbb{G}$.
$\mathbb{G}$	$(\mathbb{G}, +, 0)$ is the additive group generated by B. ($\mathbb{G} \subset \mathbb{E}$)
V	Another generator of $\mathbb{G}$.
C	An element of $\mathbb{E}$.
l	The order of B, where l is a prime.
n	The number of parties.
p	The number of parties participating in the signing process.
t	The threshold.
$\mathbb{Z}_l$	The finite filed of character l.
f	A polynomial over $\mathbb{Z}_l$ (used with superscript and subscript).
k	A cursor value.
a	A coefficient (used with superscript/subscript, e.g. $a_{i,u}^{[k]}$).
A	The public verification key.
$\mathbb{Q}_{index}$	The set of qualified parties in different phases, $index \in \{1, 2, 3, 4, 5\}$.
$L_j^{\mathbb{Q}_{index}}$	The Lagrangian coefficient corresponding to the ordinal number of qualified party members.
M	A message.
$(R, S)/(\underline{R}, \underline{S})$	A signature (underline denotes the encoded form).

i. Generate $2b' + 2$ t-degree polynomials over $\mathbb{Z}_l$ as

$$f_i(x) = \sum_{u=0}^{t} a_{i,u}x^u, g_i(x) = \sum_{u=0}^{t} b_{i,u}x^u, \tag{6}$$

$$f_i^{[k]}(x) = \sum_{u=0}^{t} a_{i,u}^{[k]}x^u, g_i^{[k]}(x) = \sum_{u=0}^{t} b_{i,u}^{[k]}x^u, \tag{7}$$

where $k \in \{0, \ldots, b' - 1\}$;

ii. For each $j \in \{1, \ldots, n\}$, compute

$$s_{i,j} = f_i(j) \bmod l, s'_{i,j} = g_i(j) \bmod l, \tag{8}$$

$$s_{i,j}^{[k]} = f_i^{[k]}(j) \bmod l, s_{i,j}^{'[k]} = g_i^{[k]}(j) \bmod l, \tag{9}$$

where $k \in \{0, \ldots, b' - 1\}$.

iii. Send $s_{i,j}, s'_{i,j}$ and $s_{i,j}^{[k]}, s_{i,j}^{'[k]} (k \in \{0, \cdots, b' - 1\})$ to each P_j ($j \in \{1, \ldots, n\}, j \neq i$) via a private secure channel. The coefficients in $f_i(x)$, $g_i(x)$, $f_i^{[k]}(x)$, and $g_i^{[k]}(x)$ are uniformly random in $\mathbb{Z}_l$,

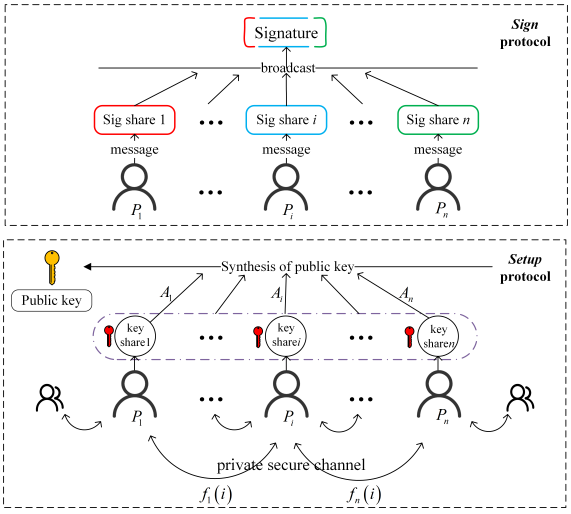


Figure 3: Scheme overview

iv. Compute and broadcast

$$C_{i,u} = a_{i,u}B + b_{i,u}V, C_{i,u}^{[k]} = a_{i,u}^{[k]}B + b_{i,u}^{[k]}V, \tag{10}$$

where $k \in \{0, \ldots, b' - 1\}$.

(II) For each $j \in \{1, \ldots, n\}$, party P_j verifies the secret value received from all other parties by checking whether (11) and (12) hold. If the check on i fails, P_j broadcasts a complaint message $< P_i, \{idx\} >$ against P_i, where the set $\{idx\}$ contains b' if (11) does not hold; further, the set contains k if (12) does not hold on k.

$$s_{i,j}B + s'_{i,j}V = \sum_{u=0}^{t} j^u C_{i,u} \tag{11}$$

$$s_{i,j}^{[k]}B + s_{i,j}^{'[k]}V = \sum_{u=0}^{t} j^u C_{i,u}^{[k]}, k \in \{0, \ldots, b' - 1\} \tag{12}$$

(III) For each $i \in \{1, \ldots, n\}$, party P_i broadcasts shares $(s_{i,j}, s'_{i,j})$ matching (11) and/or $\left(s_{i,j}^{[k]}, s_{i,j}^{'[k]}\right)$ matching (12) correspondingly for each complaining party (e.g., P_j).

(IV) Each party P_j proceeds as follows:

i. For any party P_i, if any of the revealed shares do not satisfy equation (11)/(12) or more than t complaints regarding P_i are received, P_i is disqualified and removed from $\mathbb{Q}_1$; and

ii. Let $\mathbb{Q}_2$ be the refined set of qualified parties. If $|\mathbb{Q}_2| \leq t$, the protocol is terminated.

(V) Each party $P_i \in \mathbb{Q}_2$ computes and broadcasts commitments as

$$A_{i,u} = a_{i,u}B, u \in \{0, \ldots, t\}; \tag{13}$$

$$A_{i,u}^{[k]} = a_{i,u}^{[k]} B, u \in \{0, \ldots, t\}, k \in \{0, \ldots, b' - 1\}; \quad (14)$$

(VI) Each party $P_j \in \mathbb{Q}_2$
i. verifies the secret value received from all other parties in $\mathbb{Q}_2$ by checking whether (15) holds. If the check on P_i fails, P_j broadcasts a complaint message $< P_i, \{b'\}, s_{i,j}, s'_{i,j} >$, where $(s_{i,j}, s'_{i,j})$ satisfies (11) but does not satisfy (15)

$$s_{i,j} B = \sum_{u=0}^{t} j^u A_{i,u} \quad (15)$$

ii. verifies the secret value received from all other parties in $\mathbb{Q}_2$ by checking whether (15) holds. If the check on P_i fails, P_j broadcasts a complaint message $< P_i, \{k\}, s_{i,j}^{[k]}, s_{i,j}^{'[k]} >$ for each $k \in \{0, \ldots, b' - 1\}$, where $\left(s_{i,j}^{[k]}, s_{i,j}^{'[k]}\right)$ satisfies (12) but does not satisfy (16)

$$s_{i,j}^{[k]} B = \sum_{u=0}^{t} j^u A_{i,u}^{[k]} \quad (16)$$

(VII) Parties in $\mathbb{Q}_2$ proceed as follows when a party P_i has been complained.
i. Reconstruct f_i with $f_i(j) = s_{i,j}$, where $s_{i,j}$ satisfies (6) among parties.
ii. Reconstruct $a_{i,0} = f_i(0)$.

(VIII) Let $\mathbb{Q}_2$ be the refined set of qualified parties. If $|\mathbb{Q}_2| \leq t$, the protocol is terminated; otherwise, the following calculations are conducted.
- The public key is computed as $A = \sum_{i \in \mathbb{Q}_2} A_{i,0}$.
- The public verification values are computed as $A_u = \sum_{i \in \mathbb{Q}_2} A_{i,u}$ for each $u \in \{0, \ldots, t\}$.
- The public verification vectors are computed as

$$\overline{A_u} = \{A_u^{[k]} | k \in \{0, \ldots, b' - 1\}\}, \quad (17)$$

for each $u \in \{0, \ldots, t\}$, where

$$A_u^{[k]} = \sum_{i \in \mathbb{Q}_2} A_{i,u}^{[k]}. \quad (18)$$

- For each $j \in \{0, \ldots, n-1\}$, party P_j sets his/her share of the signing key as in (19) and the share of secret vector $\overline{s_i}$ as in (20).

$$s_j = \sum_{i \in \mathbb{Q}_2} s_{i,j} \quad (19)$$

$$\overline{s_j} = \{s_j^{[k]} | k \in \{0, \ldots, b' - 1\}\}, s_j^{[k]} = \sum_{i \in \mathbb{Q}_2} s_{i,j}^{[k]}, \quad (20)$$

where $k \in \{0, \ldots, b' - 1\}$;

The public output of the **Setup** protocol is the public key A, public verification values $\{A_u, u \in \{0, \ldots, t\}\}$ and public verification vectors $\{\overline{A_u}, u \in \{0, \ldots, t\}\}$. The private output of the **Setup** protocol for a given party (e.g., P_i) is the secret share of the private key s_i and the secret vector $\overline{s_i}$.

At the end of the **Setup** protocol, we implicitly construct a polynomial as

$$f_{\{\mathbb{Q}_2\}}(x) = \sum_{i \in \mathbb{Q}_2} (f_i(x)) \quad (21)$$

where each party P_j has $s_{i,j} = f_i(j)$ for all $i \in \mathbb{Q}_2$. In fact, P_j has a Shamir share of $f_{\{\mathbb{Q}_2\}}(0)$ because $s_j = \sum_{i \in \mathbb{Q}_2} s_{i,j} = f_{\{\mathbb{Q}_2\}}(j)$.

Moreover, for each $k \in \{0, \ldots, b' - 1\}$, we implicitly construct a polynomial as

$$f_{\{\mathbb{Q}_2\}}^{[k]}(x) = \sum_{i \in \mathbb{Q}_2} \left(f_i^{[k]}(x)\right) \quad (22)$$

where each party P_j has $s_{i,j}^{[k]} = f_i^{[k]}(j)$ for all $i \in \mathbb{Q}_2$. In fact, P_j has a Shamir share of $f_{\{\mathbb{Q}_2\}}^{[k]}(0)$ because $s_j^{[k]} = \sum_{i \in \mathbb{Q}_2} s_{i,j}^{[k]} = f_{\{\mathbb{Q}_2\}}^{[k]}(j)$.

The shared signing key and shared secret vector are given in (23) and (24), respectively. These two items can be recovered by $t+1$ parties with their shares using the Lagrange interpolation. However, they should never be explicitly recovered because the whoever recovers these values can generate signatures verifiable under the public key A.

$$s = \sum_{i \in \mathbb{Q}_2} a_{i,0} = f_{\{\mathbb{Q}_2\}}(0) \quad (23)$$

$$s^{[k]} = \sum_{i \in \mathbb{Q}_2} a_{i,0}^{[k]}, k \in \{0, \ldots, b' - 1\} = f_{\{\mathbb{Q}_2\}}^{[k]}(0) \quad (24)$$

The Sign protocol assumes that parties in a subset $\mathbb{Q}_3 \subseteq \mathbb{Q}_2$ want to issue a signature. If $|\mathbb{Q}_3| > t$, the protocol works as follows; otherwise, the protocol does not start.

(I) For each party in $\mathbb{Q}_3$, the party that is supposed to be P_i proceeds as follows.
- Compute the b'-bit hash value of M: $(m_0, \ldots, m_{b'-1}) = H'(M)$.
- Generate $\mathfrak{M} = \{k | k \in \{0, \ldots, b' - 1\} \wedge m_k \neq 0\}$.
- Compute $r_i = \sum_{k \in \mathfrak{M}} s_i^{[k]}, R_i = r_i B$; broadcast R_i.

(II) For each party in $\mathbb{Q}_3$, the party that is supposed to be P_i proceeds as follows.
- Check for each $P_j \in \mathbb{Q}_3$:

$$R_j = \sum_{k \in \mathfrak{M}} \left(\sum_{u=0}^{t} j^u A_u^{[k]}\right); \quad (25)$$

- Remove parties that do not satisfy (25) from $\mathbb{Q}_3$.

(III) Let the refined set of parties be $\mathbb{Q}_4$. If $|\mathbb{Q}_4| \leq t$, then the protocol is terminated; otherwise, for each party $P_i \in \mathbb{Q}_4$, the party works as follows.
- Compute $R = \sum_{j \in \mathbb{Q}_4} L_j^{\mathbb{Q}_4} R_j$, where $L_j^{\mathbb{Q}_4}$ is the Lagrangian coefficient.
- Compute and broadcast

$$S_i = \left(r_i + H\left(\underline{R}, \underline{A}, M\right) s_i\right) \bmod l. \quad (26)$$

(IV) For each party in $\mathbb{Q}_4$, the party that is supposed to be P_i proceeds as follows.
- Check for each $P_j \in \mathbb{Q}_4$

$$S_j B = R_j + H\left(\underline{R}, \underline{A}, M\right) \sum_{u=0}^{t} \left(j^u A_u\right) \quad (27)$$

- Remove parties that do not satisfy (27) from $\mathbb{Q}_4$.

(V) Let the refined set of parties be $\mathbb{Q}_5$. If $|\mathbb{Q}_5| \leq t$, the protocol is terminated; otherwise, parties in $\mathbb{Q}_5$ generate the signature $(\underline{R}, \underline{S})$ as follows.
- Compute $S = \sum_{j \in \mathbb{Q}_5} L_j^{\mathbb{Q}_5} S_j$.
- Encode (R, S) and output the encoded value $(\underline{R}, \underline{S})$.

3.3 Correctness

Proposition 3.1. *The generation of R_j and S_j by an honest party P_j satisfy (25) and (27), respectively.*

Remark 1. *The addition and multiplication of numbers are performed in $\mathbb{Z}_l$, and "*mod$l$*" is omitted.*

Proof. If R_j is generated by an honest party P_j, we have

$$\begin{aligned} R_j &= \sum_{k \in \mathfrak{M}} \left(s_j^{[k]} B\right) = \sum_{k \in \mathfrak{M}} \left(\sum_{i \in \mathbb{Q}_2} s_{i,j}^{[k]} B\right) \\ &= \sum_{k \in \mathfrak{M}} \left(\sum_{i \in \mathbb{Q}_2} \left(\sum_{u=0}^{t} j^u A_{i,u}^{[k]}\right)\right) \\ &= \sum_{k \in \mathfrak{M}} \left(\sum_{u=0}^{t} j^u \left(\sum_{i \in \mathbb{Q}_2} A_{i,u}^{[k]}\right)\right) = \sum_{k \in \mathfrak{M}} \left(\sum_{u=0}^{t} j^u A_u^{[k]}\right) \end{aligned}$$

If S_j is generated by an honest party P_j, we have

$$\begin{aligned} S_jB &= \left(\left(r_j + H\left(\underline{R},\underline{A},M\right)s_j\right) \bmod l\right) B = R_j + \left(H\left(\underline{R},\underline{A},M\right)s_j\right) B \\ &= R_j + \left(H\left(\underline{R},\underline{A},M\right)\sum_{i\in\mathbb{Q}_2} s_{i,j}\right) B \\ &= R_j + H\left(\underline{R},\underline{A},M\right)\sum_{i\in\mathbb{Q}_2}\sum_{u=0}^{t} j^u A_{i,u} \\ &= R_j + H\left(\underline{R},\underline{A},M\right)\sum_{u=0}^{t}\left(j^u \sum_{i\in\mathbb{Q}_2} A_{i,u}\right) \\ &= R_j + H\left(\underline{R},\underline{A},M\right)\sum_{u=0}^{t}\left(j^u A_u\right) \end{aligned}$$

Therefore, (25) and (27) hold. □

Proposition 3.2. *The signature generated by the proposed scheme satisfies the verification equation of EdDSA, and the signature is unique for each message.*

Proof. Let $r_{\{H'(M)\}} = \sum_{k\in\mathfrak{M}} s^{[k]}$ and $s = \sum_{j\in\mathbb{Q}_5} L_j^{\mathbb{Q}_5} s_i$. Then, we have

$$\begin{aligned} R &= \textstyle\sum_{j\in\mathbb{Q}_4} L_j R_j = \sum_{j\in\mathbb{Q}_4} L_j \sum_{k\in\mathfrak{M}} s_j^{[k]} B \\ &= \textstyle\sum_{k\in\mathfrak{M}} \sum_{j\in\mathbb{Q}_4} L_j s_j^{[k]} B = \sum_{k\in\mathfrak{M}} s^{[k]} B = r_{\{H'(M)\}} B \end{aligned}$$

$$\begin{aligned} S &= \textstyle\sum_{j\in\mathbb{Q}_5} L_j^{\mathbb{Q}_5} S_j = \sum_{j\in\mathbb{Q}_5} L_j^{\mathbb{Q}_5}\left(\left(r_j + H\left(\underline{R},\underline{A},M\right)s_i\right) \bmod l\right) \\ &= \textstyle\sum_{j\in\mathbb{Q}_5} L_j^{\mathbb{Q}_5}\left(r_j\right) + \sum_{j\in\mathbb{Q}_5} L_j^{\mathbb{Q}_5}\left(H\left(\underline{R},\underline{A},M\right)s_j\right) \\ &= \textstyle\sum_{j\in\mathbb{Q}_5} L_j^{\mathbb{Q}_5}\left(\sum_{k\in\mathfrak{M}} s_j^{[k]}\right) + H\left(\underline{R},\underline{A},M\right)\sum_{j\in\mathbb{Q}_5} L_j^{\mathbb{Q}_5} s_j \\ &= \textstyle\sum_{k\in\mathfrak{M}}\left(\sum_{j\in\mathbb{Q}_5} L_j^{\mathbb{Q}_5} s_j^{[k]}\right) + H\left(\underline{R},\underline{A},M\right)s \\ &= \textstyle\sum_{k\in\mathfrak{M}} s^{[k]} + H\left(\underline{R},\underline{A},M\right)s = r_{\{H'(M)\}} + H\left(\underline{R},\underline{A},M\right)s \end{aligned}$$

The signature is unique because $r_{\{H'(M)\}}$ is unique for each message. It is trivial to verify that the encoded form of

$$(R,S) = \left(r_{\{H'(M)\}}B, r_{\{H'(M)\}} + H\left(\underline{R},\underline{A},M\right)s\right) \tag{28}$$

satisfies the verification equation of EdDSA. □

4 SECURITY ANALYSIS

4.1 Security model and assumption

The system comprises a set of n parties $P_1,\ldots,P_n$ that can be modeled by probabilistic polynomial-time (PPT) Turing machines. At most, t of the n parties can be corrupted, and other parties are honest. Furthermore, these parties are connected by a complete network of private (i.e., confidential and authenticated) party-to-party channels, and have a dedicated broadcast channel.

We assume that computations proceed in synchronized rounds and messages are received by recipients within some specified time bound, which is known as a partially synchronous communication model [35]. In this model, the messages sent from honest parties to corrupted parties can still be delivered relatively fast. Therefore, in every round of communication, adversaries can wait for messages from honest parties to arrive and then decide the computations and communications for that round. Finally, adversaries can obtain their messages that are sent to the honest parties on time.

This model, and the resultant stronger adversarial model, are more realistic compared to the assumed fully synchronous model, wherein messages in a given round are assumed to be sent simultaneously by all parties and delivered to the intended recipients. This model represents the worst case scenario, wherein the adversaries speak last in every communication round.

Malicious adversaries that can cause (corrupted parties) to divert from the specified protocol are allowed in the security model. In addition, we assume that corrupted parties are fixed at the beginning of the protocol and that the computational power of an adversary is adequately modeled by a PPT Turing machine. We assume that $t < n/2$, which is the best achievable threshold value that can provide both secrecy and robustness [35] in the presence of malicious adversaries, where t is the theoretical security threshold; in practice, the threshold of the proposed scheme (i.e., the required number of parties to generate a signature) can be larger than t under the assumption that there are at most t corrupted parties.

In addition, we assume that the EdDLP (given in Definition 4.1) is computationally intractable, similar to the existing schemes based on Edwards curves.

Definition 4.1. Edwards curve discrete logarithm problem [31] (EdDLP): Let $\mathbb{E}$ be an Edwards curve over a finite field $\mathbb{F}_q$. Points $A, B \in \mathbb{E}(\mathbb{F}_q)$ are provided to determine if integer β exists such that $A = \beta B$.

4.2 Security proof

The main result of this section is the following theorem.

Theorem 4.2. *The proposed threshold signature scheme is secure under the EdDLP assumption.*

Proof. It is trivial to prove this theorem based on a composition of **Lemmas** 4.3 and 4.4, which are given as follows. □

Lemma 4.3. *The one-party instance of the proposed scheme is a signature scheme that is secure under the EdDLP assumption.*

Proof. Let this instance be a signature scheme named "*EdDSA**" that is comprised of the following three algorithms.

(I) The key generation algorithm *EdDSA**.KGen, which operates as follows.
  (i) Uniquely select $b'+1$ random elements $s, s^{[0]},\ldots,s^{[b'-1]}$ from $\mathbb{Z}_l$.
  (ii) Compute $A = sB, A^{[0]} = s^{[0]}B,\ldots,A^{[b'-1]} = s^{[b'-1]}B$
  (iii) Return the private key $(s, s^{[0]},\ldots,s^{[b'-1]}) \in \mathbb{Z}_l^{b+1}$ and the public key $(\underline{A},\underline{A^{[0]}},\ldots,\underline{A^{[b'-1]}})$.

(II) The signing algorithm *EdDSA**.Sign, which works as follows on a given message M.
  (i) Compute $h' = H'(M)$ and $r = \sum_{k\in\{0,\ldots,b'-1\},h'_k=1} s^{[k]} \bmod l$, where h'_k represents the k^{th} bit of h'.
  (ii) Compute $R = rB$ and encode it into $\underline{R}$.
  (iii) Compute $S = (r + H(\underline{R},\underline{A},M)s) \bmod l$.
  (iv) Encode S and return the signature $(\underline{R},\underline{S})$.

(III) The verification algorithm *EdDSA**.Verify. The description of this algorithm is omitted because this algorithm is the same as the verification algorithm of the standard EdDSA.

Based on their descriptions, we compare the *EdDSA** signature scheme induced by the proposed threshold signature scheme with the Schnorr signature scheme in the same group $\mathbb{G}$. The main difference is that the "r" in our scheme is generated by $r = \sum_{k\in\{0,\ldots,b'-1\},h'_k=1} s^{[k]} \bmod l \in \mathbb{Z}_l$, while the "$r$" in the Schnorr

signature scheme is a uniformly random value in $\mathbb{Z}_l$. Because the "r" in $EdDSA^*$ is indeed a sum (mod l) of uniquely random elements of $\mathbb{Z}_l$, it has the same distribution as the "r" in the Schnorr signature scheme. Therefore, the result regarding the security of the Schnorr signature scheme, e.g., Theorem 4 in [44], can be applied to $EdDSA^*$. Let $\mathcal{A}$ be an attacker that performs an existential forgery under an adaptively chosen-message attack against the $EdDSA^*$ signature scheme within a time bound T with probability ε. We denote the number of queries that $\mathcal{A}$ can ask to the random oracle and the number of queries that $\mathcal{A}$ can ask to the signing oracle by q_O and q_S, respectively. Assume that $\varepsilon \geq 10(q_S+1)(q_S+q_O)/l$; then, the discrete logarithm in $\mathbb{G}$ can be solved within an expected time that is less than $120686 q_O T/\varepsilon$.

This result contradicts the assumption that EdDLP is hard; thus, $EdDSA^*$ is secure. This completes the proof. □

Lemma 4.4. *Assume that there exist t PPT adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_t$ who can forge a valid signature for the proposed scheme in a chosen message attack; then, there exists a PPT adversary $\mathcal{A}_0$ who can successfully forge a signature of $EdDSA^*_{\{A, A_0^{[0]}, \ldots, A_0^{[b'-1]}\}}$ in a chosen message attack under an arbitrarily given verification key.*

Remark 2. *An underline is used to denote an encoded value, and the encoding/decoding operations are omitted in the lemma and the following proof.*

Proof. Given a public key $\{A, A_0^{[0]}, \ldots, A_0^{[b'-1]}\}$ and the corresponding sign oracle of the $EdDSA^*$ signature $O^{Sign}_{\{A, A_0^{[0]}, \ldots, A_0^{[b'-1]}\}}$, we construct $\mathcal{A}_0$, which interacts with $\mathcal{A}_i (i \in \{1, \ldots, t\})$ as follows:

Suppose that $\mathcal{A}_0$ controls parties $P_i (i \in \{t+1, \ldots, n\})$, where the other parties are $\mathcal{A}_i (i \in \{1, \ldots, t\})$. Here, $\mathcal{A}_0$ uses a series of simulators $\{\mathcal{S}^{[k]} | k \in 0, \ldots, b'\}$ to interact with adversaries in the **Setup** protocol. Each $\mathcal{S}^{[k]}$ works the same as that in the "Algorithm of Simulator S" in [35] for secure distributed key generation. Here, $\mathcal{S}^{[b']}$ is used to perform operations that do not involve symbols with superscripts in square brackets, such as generating f_i and g_i, computing and sending $s_{i,j}$, and computing and broadcasting $C_{i,u}$. Each $\mathcal{S}^{[k]}$ $(k \in 0, \ldots, b'-1)$ is used to perform operations that involve symbols with superscripts in square brackets, such as generating $f_i^{[k]}$ and $g_i^{[k]}$, computing and sending $s_{i,j}^{[k]}$, and computing and broadcasting $C_{i,u}^{[k]}$. In Section 4.3 [35], it was proven that each $\mathcal{S}^{[k]}$ can interact with these t adversaries to distributively generate a pair $(s \in \mathbb{Z}_l, A = sB \in \mathbb{G})$, where s denotes the shared secret and A is the specified public point in $\mathbb{G}$.

Thus, the **Setup** protocol's output includes the public verification key A (of the threshold signature scheme) and b' public generators $\{A_0^{[k]} | k \in \{0, \ldots, b'-1\}\}$ (used in the threshold signature), which are exactly the same as those for the given public verification key (of the given $EdDSA^*_{\{A, A_0^{[0]}, \ldots, A_0^{[b'-1]}\}}$ scheme) and the b' public generators (used in the given $EdDSA^*_{\{A, A_0^{[0]}, \ldots, A_0^{[b'-1]}\}}$ scheme), respectively. The output also includes other public information and secret shares distributed equally to a normal **Setup** phase.

At this stage, adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_t$ perform a chosen message attack against $\mathcal{A}_0$. When an adversary $\mathcal{A}_i$ $(i \in \{1, \ldots, t\})$ issues a query to the signing oracle (provided by $\mathcal{A}_0$) on message M, $\mathcal{A}_0$ works as follows.

(I) Obtain the $EdDSA^*$ signature (R, S) on the same message by issuing a query to $O^{Sign}_{\{A, A_0^{[0]}, \ldots, A_0^{[b'-1]}\}}$ ($EdDSA^*$'s signing oracle).

(II) Simulate the behavior of parties $P_i (i \in \{t+1, \ldots, n\})$ in the **Sign** protocol.

(III) Construct R_{t+1} as in (29), where $L_i^{(t+1)} = \prod_{h \neq i, h=1}^{h=t+1} (h/(h-j))$ is the coefficient of the Lagrange interpolation.

$$R_{t+1} = \left(L_{t+1}^{(t+1)}\right)^{-1} \left(R - \sum_{i=1}^{t+1} \left(L_i^{(t+1)} R_i\right)\right) \tag{29}$$

(IV) Construct S_{t+1} as in (30) in $\mathbb{Z}_l$.

$$S_{t+1} = \left(L_{t+1}^{(t+1)}\right)^{-1} \left(S - \sum_{i=1}^{t+1} \left(L_i^{(t+1)} S_i\right)\right) \tag{30}$$

(V) Send the $(t+1)^{th}$ share of the signature (R_{t+1}, S_{t+1}).

To summarize, $\mathcal{A}_0$ is used to simulate the roles of the uncorrupted parts in the entire **Setup** protocol. The output of the protocol fits the given public key $\{A, A_0^{[0]}, \ldots, A_0^{[b'-1]}\}$. In addition, $\mathcal{A}_0$ is used to simulate the roles of the uncorrupted parts in the **Sign** protocol during the chosen message attack. Therefore, if adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_t$ successfully forge a valid signature, then $\mathcal{A}_0$. This completes the proof. □

5 EFFICIENCY ANALYSIS AND PERFORMANCE EVALUATION

5.1 Efficiency analysis

Table 3 provides a summary of the communication costs of the **Setup** and **Sign** protocols on the most widely used Edwards curve, i.e., the Ed25519 curve [11]. Here, n, p, and k represent the number of parties, the number of parties participating in the signing process, and a cursor value, respectively. In Table 4, we provide the mathematical operations involved in both the protocols and the efficiency of the algorithms in terms of the numbers of operations. We also analyse the storage cost of each signing party, which requires $256(b'+2)$ bits to keep the public key A, its own secret key and secret vector.

Table 3: Communication costs

Protocol	Communication costs (Bytes)
Setup	$128n(k+1) - 96(k+1)$
Sign	$128p - 64$

To determine the costs, it is assumed that there is no node downtime or malicious node. When there is a node check failure, the node broadcasts a complaint message, which increases communication costs. If the protocol is terminated midway, the abovementioned overhead calculation is not met.

As observed from Tables 3 and 4, the verification failure of the parameters transmitted by the node is not included in the calculation of the communication cost and the algorithm efficiency of the **Setup** protocol. Here, the node broadcasts an additional complaint message for the node that fails the verification, which increases the communication overhead. When this scenario occurs, the number of nodes in the node set is reduced, and the total costs of the subsequent communication and computation are reduced.

Table 4: Efficiency of the algorithms

Protocol	Space	Operation	Number of Operations
Setup	In $\mathbb{Z}_l$	*Rand*	$2t(k+1)$
		Add	$2t(k+1)+(n-1)(k+1)$
		Mult	$2t(k+1)$
		Exp	$4t(k+1)$
	In G	*Mult*	$t(k+1)+n(t+1)(k+1)$
		Exp	$(3t+6)(k+1)+t$
Sign	In $\mathbb{Z}_l$	*Add*	$k+4p(p-1)+p-1$
		Inv	$4p(p-1)+2p$
		Mult	$4p(p-2)+2p+1$
		Exp	$t(k-1)+t$
	$M \to \{0,1\}^{2b}$	H	1
	$M \to \{0,1\}^{b'}$	H'	2
	In G	*Mult*	n
		Exp	$k+p+2$

Here, *Rand*, *Mult*, *Inv*, and *Exp* represent the operations of selecting a random element, multiplication, computing the multiplicative inversion, and exponentiation, respectively.

For the **Sign** protocol, nodes calculate R and S through Lagrange interpolations. The calculation process needs to meet only the number of participating nodes p satisfying $n \geq p > t$. Therefore, in Table II, the independent variable in the total communication cost equation should be p instead of n. Similarly, in the algorithm efficiency analysis in Table III, the number of operations in $\mathbb{Z}_l$ and G are related to only the actual number of participating nodes p.

For $k = 256$, the communication cost of the **Setup** protocol is $32n - 24$ KB, and the communication cost of the **Sign** protocol is approximately $64(2n-1)$ bytes. For example, when the number of nodes $n = 10$, the communication cost of the **Setup** protocol is 297.2 KB. When the number of nodes participating in the sign protocol $p = n = 10$, the communication cost of completing a signature through the **Sign** protocol is 1.2 KB.

5.2 Implementation and experimental evaluation

We implement the **Setup** and **Sign** protocols in Golang, using the Ed25519 curve and calls the ed25519[1] package in the Golang standard library. Here, k is set as 256. Our protocol uses the same concrete hash function as specified in the EdDSA standard.

In our experiment, the parameters include the total number of nodes n, the threshold t, and the number of nodes p participating in the **Sign** process. We use the average time for the nodes to execute the protocol as a measure of efficiency in the experiment.

These experiments are aimed at evaluating the efficiency of the **Setup** and **Sign** protocols on different platforms, i.e., cloud servers and embedded devices. Cloud servers are a set of Alibaba Cloud Elastic Compute Service (ECS) ecs.g6a.4xlarge nodes, each running Ubuntu 16.04 64-bit with kernel ecs-616, having 16 physical cores clocked at 2.6 GHz. Among these nodes, the bandwidth is approximately 10 Gbits/s. The embedded devices are a group of Raspberry Pi model 4B, which are equipped with a quad-core ARMv7 Processor rev 3 (v7l) processor clocked at 1.5 GHz. The boards are loaded with Raspberry Pi OS and connected to one another via Ethernet.

[1] golang.org/x/crypto/ed25519

5.2.1 Experimental Setup. For the experiment of the **Setup** protocol, we create nine nodes under two benchmark conditions. The computation cost depends on t, n and the bandwidth and not on p. Thus, we change n and t to test the effect on the execution efficiency of the **Setup** protocol. As shown in Figure 4, we test the efficiency of the different numbers of nodes n on the **Setup** when t is the same. Experimental results with the same number of nodes n and different thresholds t are depicted in Figure 5. The experimental results in Figs. 4 and 5 indicate that changing the threshold has a negligible effect on efficiency when fixing the number of participants.

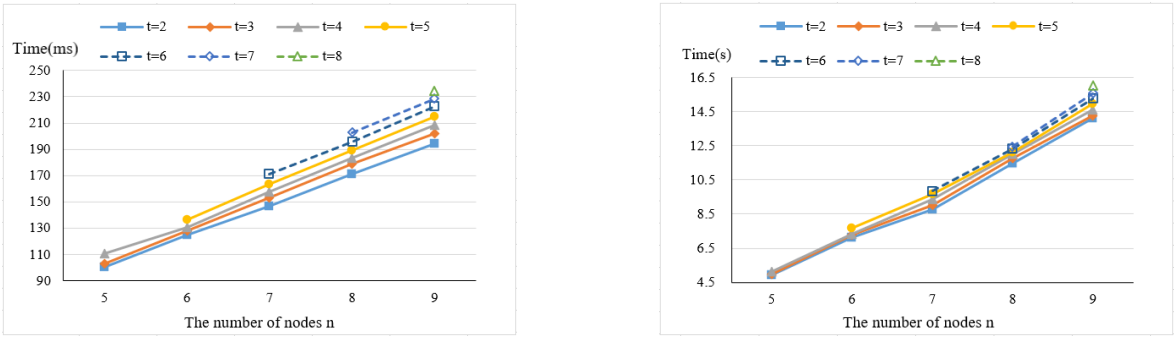
5.2.2 Sign Experiments. For the **Sign** protocol, Figure 6 shows the execution time (in milliseonds) with respect to the number of participating nodes p under different combinations of (t, n).

The linear relationship shown by the experimental results meets our expected efficiency calculation formula in Table 4. The analysis of the experimental results indicates that the execution of the scheme is very efficient on cloud servers. In the case that all nine identical nodes participate in the **Sign** protocol, the signing time remains considerably less than 10 ms, while the **Setup** protocol requires only a few hundred milliseconds. In the experiment with embedded devices, we observe that the **Sign** protocol requires considerably less than a second, and the **Setup** protocol requires less than 20 s, despite the limitations of these devices. Our scheme has good portability because the standard EdDSA signature that is used by many blockchain systems can be output.

5.2.3 Blockchain Experiments. We verify the applicability of the proposed scheme based on Tendermint, which is a blockchain system comprising the Tendermint Core consensus engine and Application Blockchain Interface (ABCI), to evaluate the compatibility of our scheme with the blockchain. In our experiment, we implement the ABCI to store $(key, value)$ pairs as transactions in a Tendermint blockchain. The experiments involves two types of clients: *Signers* and the *Proposer*, where the *Signers* are responsible for the implementation of our scheme at each Tendermint node, and the *Proposer* is responsible for initiating the Setup/Sign task and verifying the results. The experimental platform includes cloud servers similar to the previous experiment. There are two APIs involved in our experiments. **Query** is used to query the transaction in the blockchain, and **write**, which includes "write:Setup", "write:Sign", "write:PubKey" (public key) and "write:Signature", is used to write a new transaction (e.g., $(key, value)$ pair) into the blockchain.

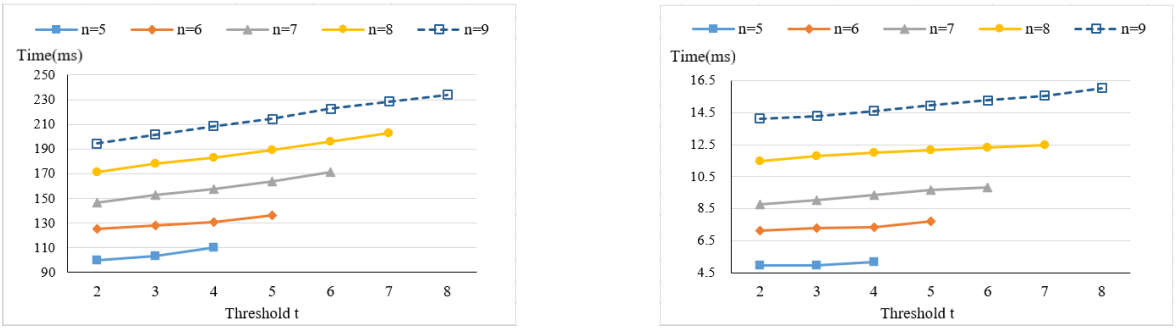
The specific procedure of the blockchain experiment is illustrated in Figure 7. We count the time for the four steps marked in the figure, which consists of the time of the **Setup**(resp. **Sign**) protocol and the time of "write:PubKey"(resp. "write:Signature"). The detailed experimental procedure is

(1) The *proposer* delivers a transaction containing the "Setup" or "Sign" signal to the blockchain by "write:Setup" or "write:Sign," where the "Sign" signal contains the corresponding message.
(2) *Signers* listen for the signal by using the **query** API.
(3) *Signers* detect the signal in the blockchain and implement the *Setup* or *Sign* protocol. Then, one of the *Signers* sends the "public key" or "signature" as a transaction to the blockchain by "write:PubKey" or "write:Signature."



(a) Cloud servers **(b) Embedded devices**

Figure 4: Experimental results of the Setup protocol under the condition that the independent variable is n.



(a) Cloud servers **(b) Embedded devices**

Figure 5: Experimental results of the Setup protocol under the condition that the independent variable is t.

(4) The *proposer* detects the "public key" or "signature" in the blockchain using the **query** API.

The experimental results indicate that the *Signer* of each blockchain node can correctly respond to the task of the *Proposer* and obtain the correct EdDSA "public key" or "signature." Furthermore, we evaluate the time cost of different steps in the experiments for different n and t settings, which is shown in Figure 8. The markings in each step of Figure 8 correspond to the markings in Figure 7. As shown in the figure, it takes approximately 2.8 s to complete the experiments. The change in (t, n) has a negligible effect on the time because it only affects the time of the **Setup** (approximately 100 ms) and the **Sign** (approximately 2.3 ms) protocol, and the time required for the **Setup** or the **Sign** protocol is significantly lower than that for writing a new transaction into the blockchain by API (approximately 1.3 s), which is a very small proportion.

According to the experimental results, the introduced **Setup** and **Sign** protocols in our scheme are compatible with the blockchain system. Moreover, the **Setup** and **Sign** protocols have little effect on the system performance, as the running times of the two protocols are in a very small proportion in the whole computation cost.

6 RELATED WORKS AND COMPARISON

6.1 Comparison with other types of signature schemes

Most studies on threshold signature schemes for blockchains focus on ECDSA. At CCS'18, Lindell et al.[41] and Gennaro et al. [32] presented two threshold ECDSA schemes simultaneously. At S&P' 19, Doerner et al. [25] extended their previous two-party protocol [24] to the general (t, n) case, and a concrete performance gain of approximately 40% was achieved. These schemes are valuable and useful. However, their methodologies cannot be applied to EdDSA, which is the second most widely used signature scheme in the blockchain community.

Compared with the multi-signature approach, wherein the access-control policy is encoded in the transaction and eventually publicly revealed, the threshold signatures have fundamental advantages such as [34] **(i) Flexibility.** Threshold signatures are more flexible than multi-signatures in access policies that they permit and in their ability to modify access policies. **(ii) Anonymity.** Multi-signature transactions do not provide any anonymity guarantees; on the contrary, one can infer real signers from a threshold signature. **(iii) Scalability.** With multi-signature transactions, the size of the transactions grows linearly with the access policy as all valid signing keys are included in the redeeming script, whereas the size of the transaction is not affected by the threshold signature.

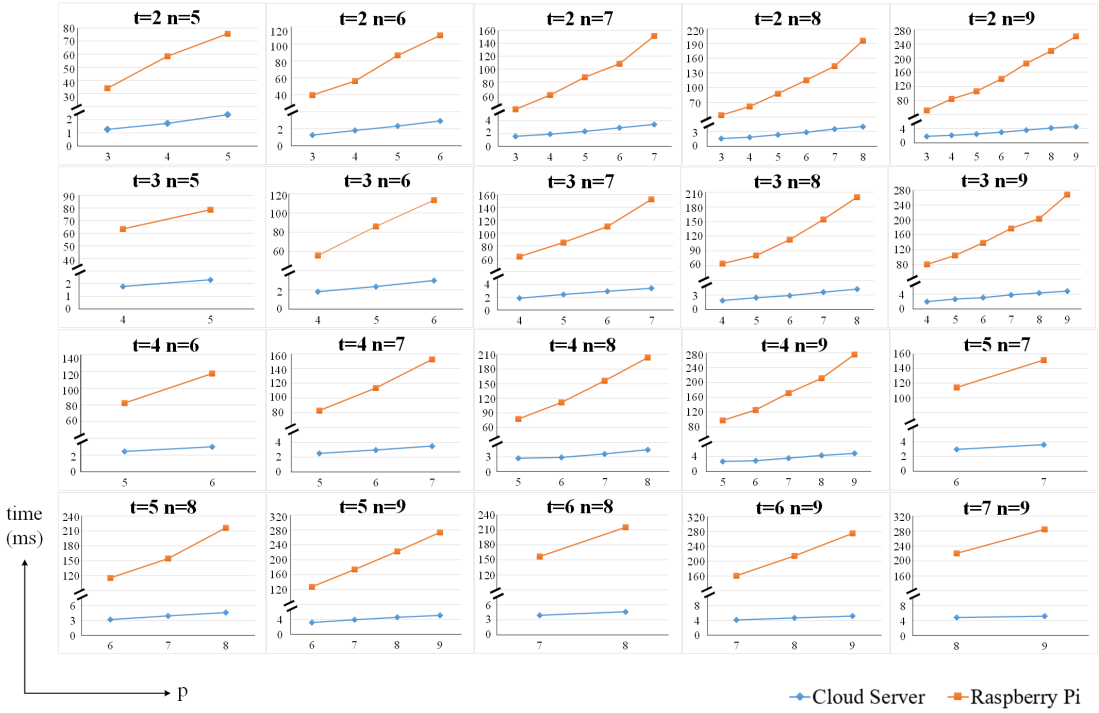


Figure 6: Experimental results of the Sign protocol with different numbers of participating nodes p.

6.2 Comparison with existing threshold EdDSA schemes

Recently, several works have discussed the threshold signature protocol for EdDSA, which provides great contributions to the state of the art. However, there are some significant limitations to these schemes, and we briefly introduce the three schemes [10, 15, 28] and compare them with our proposed scheme.

Bonte et al. [15] proposed a threshold EdDSA signature protocol with a special access structure and adopted generic secure multiparty computation (MPC) tools, such as garbled circuits, to implement distributed hashing, which requires significant time and communication overhead. Unlike Bonte et al.'s scheme, our scheme does not rely on time-consuming MPC protocols.

Feng et al. [28] proposed a practical two-party EdDSA signature protocol without heavy computational overhead. However, their protocol is suitable only for two-party scenarios. Our proposed scheme is applicable to arbitrary (t, n) threshold EdDSA signatures and is practical for various scenarios.

Battagliola et al. [10] presented a (2,3)-threshold EdDSA signature protocol that allows an offline party during the key generation. However, the involvement of the offline party makes the scheme a nonstandard threshold signature protocol and introduces security concerns. Furthermore, "the protocol involves a large utilization of ZKPs" (zero-knowledge proofs)[10], which affects the efficiency of the protocol. In contrast, all parties are treated equally in our proposed scheme, and we do not use ZKPs to achieve high efficiency.

Furthermore, in terms of experimental comparison of performance, we only compared our scheme's (2,2)-instance with [28] because of the following restrictions: (i) In [28], only two parties are supported. (ii) In [10], neither source code nor experimental data are presented. (iii) In [15], no implementation is given, and only parts of the scheme (i.e., two of five steps in the signing algorithm) have been experimentally evaluated.

We use the data from the malicious settings in [28] to compare with the proposed scheme's (2,2)-instance performance, which was specifically measured on two Alibaba Cloud ECS machines with 3.3 GHz CPU of two cores, 8.0 GB RAM and Ubuntu 18.04, through VLAN similar to the settings in [28]. As shown in Figure 9, the results indicate that even though the key generation of our scheme process is slower, our signing process is significantly faster. This means that our scheme is expected to be more efficient in applications because once the Setup phase is accomplished, only the Sign protocol is repeatedly used to generate signatures.

6.3 Comparison with nonstandard open source threshold EdDSA implementations

In addition to the academic works above, we analyze two open source threshold EdDSA implementations developed by Binance[4] and ZenGo [5]. Both implementations violate the key property that EdDSA is a unique signature scheme, and therefore, we refer to them as "nonstandard" schemes. In Section 2, we mentioned the calculation of R and r during the signature generation of EdDSA. According to equations (2), (3) and (4), for each given pair of secret

Table 5: Comparison of existing threshold EdDSA schemes/implementations and our work.

Scheme	[15]	[28]	[10]	[5]	[4]	Ours
Security proof	Yes	Yes	Yes	No	No	Yes
Without using GC	No	Yes	Yes	Yes	Yes	Yes
Without using ZKP	Yes	Yes	No	Yes	No	Yes
Full implementation	Yes	Yes	Yes	Yes	Yes	Yes
Experimental results of the entire scheme	No	Yes	No	No	No	Yes
Deterministic	Yes	Yes	Yes	No	No	Yes
Without special restriction	Yes	Only two-party	Only (2,3)-threshold	Yes	Yes	Yes

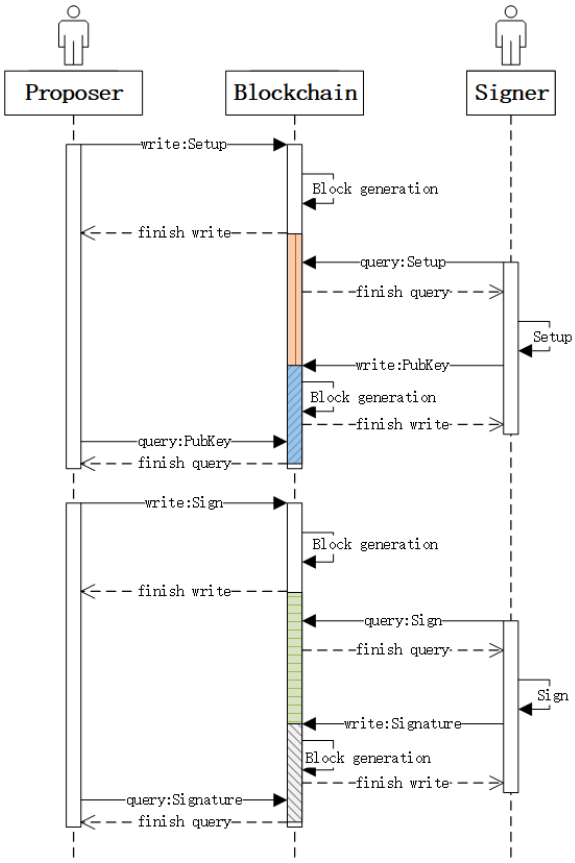


Figure 7: Detailed procedure of the blockchain experiment.

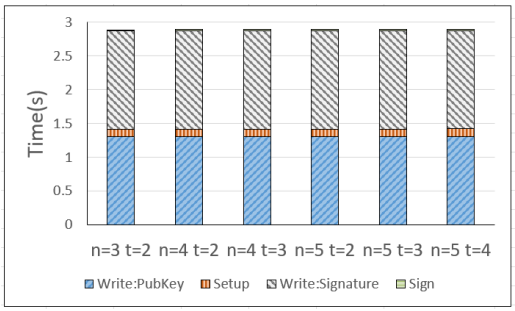


Figure 8: Running times of the main steps in the procedure of the blockchain experiment with respect to different (n, t).

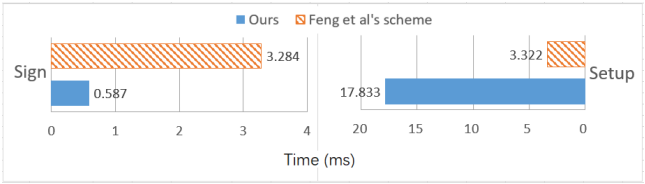


Figure 9: Comparison of the efficiency between Feng et al.'s scheme and our (2,2)-instance

```
35          // 1. select ri
36          ri := common.GetRandomPositiveInt(round.Params().EC().Params().N)
37
38          // 2. make commitment
39          pointRi := crypto.ScalarBaseMult(round.Params().EC(), ri)
```

https://github.com/bnb-chain/tss-lib/blob/master/eddsa/signing/round_1.go

Figure 10: Code in Binance's open source implementation [4]

keys κ and messages M, the valid signature is unique, which means that EdDSA is a deterministic signature scheme. However, there are some issues in these two implementations that lead to violations of this property.

In Binance's open source implementation [4], as shown in Figure 10, the i-th share of r (i.e., "ri" in line 36) is generated randomly in each signing, which leads to different signatures for the same M and violates the property of the deterministic signature.

In ZenGo's open source implementation [5], as shown in Figure 11, each party generates a share of r (i.e., "r_i" in line 214) to compute the share of R (i.e., "R_i" in line 215). As in lines 209-214, the calculation of "r_i" involves "keys.prefix", which is distinct for each party. Therefore, different parties generate different signatures even for the same message M.

Nevertheless, our scheme generates a unique signature for each message, despite the different parties involved. In our protocol, differences in the share selection of different parties are eliminated since the hash of the message is the same, which is the guarantee of the same signature. In summary, compared with these two implementations, our scheme is a deterministic threshold EdDSA algorithm, regardless of the difference in the participants.

Summary of comparison: compared with the threshold EdDSA signature schemes [10, 15, 28] and implementations [4, 5], the

```
209         let r_local = HSha512::create_hash(&[
210             &keys.prefix.to_big_int(),
211             &BigInt::from_bytes(message),
212             &FE::new_random().to_big_int(),
213         ]);
214         let r_i: FE = ECScalar::from(&r_local);
215         let R_i = GE::generator() * &r_i;
216
217         EphemeralKey {
218             r_i,
219             R_i,
220             party_index: index,
221         }
```

https://github.com/ZenGo-X/multi-party-eddsa/blob/master/src/protocols/thresholdsig/mod.rs

Figure 11: Code in ZenGo's open source implementation [5]

advantages of the proposed scheme are as follows: (i) An arbitrary threshold (t, n), where $t < n$, is supposed. (ii) The GC and ZKP protocols are not used. (iii) It follows the standard model of the dealer-free threshold signature scheme. (iv) It is a deterministic signature algorithm. A qualitative comparison in Table 5 indicates that our scheme is the only one that stratifies all the desired properties.

7 CONCLUSIONS

Threshold signature techniques facilitate blockchain-based decentralized digital asset custody and cross-chain applications. EdDSA is the second most widely used digital signature scheme on existing blockchains. However, no threshold signature scheme exists that are fully compatible with EdDSA and support arbitrary threshold (t,n) without using generic MPC approaches such as garbled circuits, which require rather expensive computation and communication costs.

Therefore, we proposed a decentralized threshold EdDSA signature scheme, whose signatures can be verified by the standard EdDSA verification algorithm. The setup phase is dealer-free, which is appropriate for decentralized blockchains. Furthermore, we proved the security of the proposed scheme. The experiments on the efficiency of the scheme demonstrated that the scheme is efficient on (cloud) servers, and it performed reasonably well on embedded devices such as a Raspberry Pi. In addition, we integrated the proposed scheme with Tendermint, a blockchain framework that uses EdDSA. The experimental study showed that the scheme can effectively work with a blockchain. Finally, because EdDSA is widely used in communication security (e.g., OpenSSL) and mobile devices (e.g., Apple Watch and iPhone), the proposed approach can also be adopted for these scenarios.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China under Grant No. 62172301 and 61702371, and the Fundamental Research Funds for the Central Universities under Grant No. 22120210545, the Shanghai Municipal Science and Technology Major Project (2021SHZDZX0100), the Fundamental Research Funds for the Central Universities, and the Natural Science Foundation of Shanghai under Grant No. 20ZR1460500.

REFERENCES

[1] 2013. https://www.getmonero.org/resources/research-lab/pubs/whitepaper_annotated.pdf
[2] 2017. Wanchain:Building Super Financial Markets for the New Digital Economy. https://www.wanchain.org/files/Wanchain_White_Paper_EN.pdf
[3] 2020. RenVM project. https://docs.renproject.io/developers/z0-spec/z0-spec-draft
[4] 2021. bnb-chain/tss-lib. https://github.com/bnb-chain/tss-lib/tree/master/eddsa
[5] 2021. ZenGo-X/multi-party-eddsa. https://github.com/ZenGo-X/multi-party-eddsa
[6] 2022. https://docs.cardano.org/en/latest/
[7] 2022. Binance. https://www.binance.com/en
[8] 2022. An Introduction to Libra. https://www.diem.com/en-us/vision/
[9] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *EuroSys*. 1–15.
[10] Michele Battagliola, Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala. 2020. A Provably-Unforgeable Threshold EdDSA with an Offline Recovery Party. *arXiv preprint arXiv:2009.01631* (2020).
[11] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2011. High-speed high-security signatures. In *12th CHES*. 124–142.
[12] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of cryptographic engineering* 2, 2 (2012), 77–89.
[13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2007. Sponge functions. In *ECRYPT hash workshop*, Vol. 2007. Citeseer.
[14] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact multi-signatures for smaller blockchains. In *ASIACRYPT*. 435–464.
[15] Charlotte Bonte, Nigel P Smart, and Titouan Tanguy. 2021. Thresholdizing HashEdDSA: MPC to the rescue. *International Journal of Information Security* (2021), 1–16.
[16] Vitalik Buterin. 2016. Chain interoperability. *R3 Research Paper* (2016).
[17] Yan Chen and Cristiano Bellavitis. 2020. Blockchain disruption and decentralized finance: The rise of decentralized business models. *Journal of Business Venturing Insights* 13 (2020), e00151.
[18] Zhaohua Chen and Guang Yang. 2020. Decentralized Custody Scheme with Game-Theoretic Security. *arXiv preprint arXiv:2008.10895* (2020).
[19] Shaen Corbet, Andrew Meegan, Charles Larkin, Brian Lucey, and Larisa Yarovaya. 2018. Exploring the dynamic relationships between cryptocurrencies and other financial assets. *Economics Letters* 165 (2018), 28–34.
[20] Daniele Cozzo and Nigel P Smart. 2019. Sharing the LUOV: threshold post-quantum signatures. In *IMA International Conference on Cryptography and Coding*. Springer, 128–153.
[21] Anders Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. 2020. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *ESORICS*. 654–673.
[22] Ivan Bjerre Damgård. 1989. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*. Springer, 416–427.
[23] Yvo Desmedt and Yair Frankel. 1989. Threshold cryptosystems. In *Conference on the Theory and Application of Cryptology*. Springer, 307–315.
[24] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. 2018. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE S&P*. IEEE, 980–997.
[25] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. 2019. Threshold ECDSA from ECDSA assumptions: the multiparty case. In *IEEE S&P*. IEEE, 1051–1066.
[26] Haihan Duan, Jiaye Li, Sizheng Fan, Zhonghao Lin, Xiao Wu, and Wei Cai. 2021. Metaverse for Social Good: A University Campus Prototype. 153–161.
[27] Marco Di Maggio Nicholas Platias Evan Kereiakes, Do Kwon. 2019. Terra Money: Stability and Adoption. https://terra.money/Terra_White_paper.pdf
[28] Qi Feng, Debiao He, Min Luo, Zengxiang Li, and Kim-Kwang Raymond Choo. 2020. Practical Secure Two-Party EdDSA Signature Generation with Key Protection and Applications in Cryptocurrency. In *19th TrustCom*. IEEE, 137–147.
[29] FUSION FOUNDATION. 2017. An Inclusive Cryptofinance Platform Based on Blockchain. https://www.fusion.org/
[30] Mark Frauenfelder. 2017. I Forgot My Pin: An Epic Tale of Losing $30,000 in Bitcoin. *Wired*. (2017).
[31] Steven D Galbraith and Pierrick Gaudry. 2016. Recent progress on the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography* 78, 1 (2016), 51–72.
[32] Rosario Gennaro and Steven Goldfeder. 2018. Fast multiparty threshold ECDSA with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1179–1194.
[33] Rosario Gennaro and Steven Goldfeder. 2020. One Round Threshold ECDSA with Identifiable Abort. *IACR Cryptol. ePrint Arch.* 2020 (2020), 540.
[34] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. 2016. Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In

Applied Cryptography and Network Security, Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider (Eds.). https://eprint.iacr.org/2016/013.pdf

[35] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology* 20, 1 (2007), 51–83.

[36] LM Goodman. 2014. Tezos: A self-amending crypto-ledger position paper. *Aug* 3 (2014), 2014.

[37] Lewis Gudgeon, Sam Werner, Daniel Perez, and William J Knottenbelt. 2020. DeFi protocols for loanable funds: Interest rates, liquidity and market efficiency. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 92–112.

[38] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security* 1, 1 (2001), 36–63.

[39] Chelsea Komlo and Ian Goldberg. 2020. FROST: Flexible Round-Optimized Schnorr Threshold signatures. (2020).

[40] Jae Kwon. 2014. Tendermint: Consensus without mining. *Draft v. 0.6, fall* 1, 11 (2014).

[41] Yehuda Lindell and Ariel Nof. 2018. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *25th CCS*. 1837–1854.

[42] Qiushan Liu, Lang Yu, and Chang Jia. 2020. MovER: Stabilize Decentralized Finance System with Practical Risk Management. In *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 55–56.

[43] Ralph C Merkle. 1989. One way hash functions and DES. In *Conference on the Theory and Application of Cryptology*. Springer, 428–446.

[44] David Pointcheval and Jacques Stern. 2000. Security arguments for digital signatures and blind signatures. *Journal of cryptology* 13, 3 (2000), 361–396.

[45] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[46] Victor Shoup. 2000. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 207–220.

[47] Nigel P Smart and Younes Talibi Alaoui. 2019. Distributing any elliptic curve based protocol. In *IMA International Conference on Cryptography and Coding*. Springer, 342–366.

[48] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[49] Gan Yuan-ju. 2004. Verifiable threshold signature schemes against conspiracy attack. *Journal of Zhejiang University-SCIENCE A* 5, 1 (2004), 50–54.

[50] Dirk A Zetzsche, Douglas W Arner, and Ross P Buckley. 2020. Decentralized finance. *Journal of Financial Regulation* 6, 2 (2020), 172–203.