Received 17 October 2022, accepted 12 December 2022, date of publication 19 December 2022, date of current version 28 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3230683

RESEARCH ARTICLE

A Key Recovery Protocol for Multiparty Threshold ECDSA Schemes

MYUNGSUN KIM[1], SANGRAE CHO[2], SEONGBONG CHOI[3], YOUNG-SEOB CHO[2], SOOHYUNG KIM[2], AND HYUNG TAE LEE[3]
[1]Department of Mathematics, Gachon University, Seongnam-si 13120, Republic of Korea
[2]Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, Republic of Korea
[3]School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea

Corresponding author: Hyung Tae Lee (hyungtaelee@cau.ac.kr)

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2020-0-00321, Privacy preserving Self-controlled Decentralized Identity Management and Security Technology in 5G service environment).

ABSTRACT Recently, threshold ECDSA schemes have received much attention from the security community, due to the need of efficient key management in the blockchain system. For the practical use of threshold cryptosystem, a key recovery protocol is essential for users who lost their own secret shares to recover them. It was studied for a long time in the proactive secret sharing area, but the main aim of recent studies in that area is to achieve stronger security and so they are immoderate for the currently existing threshold ECDSA schemes. In this paper, we provide a new key recovery protocol for threshold ECDSA schemes that is secure against static corruptions by malicious adversaries, as in the common adversary model of the state-of-the-art threshold ECDSA schemes. Our proposed protocol reduces both the computational and communication costs to $O(t^2)$ from $O(t^3)$ where t is the threshold of the schemes, that is, the minimum number of users required for generating a valid signature. According to our experimental results, when $t = 2$ with 128-bit security, while the previous result takes 10.46 ms in total for all computations (excluding the transmission time on the network), our protocol takes 4.21 ms, which improves by a factor of about 2.48 times. The advantage of our protocol over the previous result is bigger when t is larger. For example, when $t = 9$ with 128-bit security, while the previous result requires 333.42 ms in total for all computations, our protocol requires 56.61 ms, which outperforms the previous result by a factor of about 5.89 times.

INDEX TERMS Recovery protocol, proactive security, threshold ECDSA, secret sharing.

I. INTRODUCTION

A (t, n) threshold signature scheme allows to distribute the right of sign generation with n parties and then to generate a signature by joining at least t parties among them.[1] It enables us to distribute and store a secret key securely and so can be applied for secure key management on the network. Thus, there were various studies [1], [2], [3], [4], [5] to design threshold signature schemes since its concept was firstly introduced by Desmedt and Frankel [6]. In particular, with the development of blockchain and cryptocurrencies, a threshold version of ECDSA schemes has received much attention from cryptography and security communities. Differently from other widely utilized signature schemes, like RSA [7] and Schnorr signatures [8], it was a difficult task to design efficient threshold signature schemes based on (DSA and) ECDSA [9], due to the form of signatures that includes an inverse of a randomly chosen element which should be secret to all parties. However, since the appearance of fascinating applications on blockchain and cryptocurrencies, recently there have been proposed various elegant results on designing threshold ECDSA schemes [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20].

Though threshold signature schemes make an adversary harder, it is not the aegis for preventing it from attaining a

The associate editor coordinating the review of this manuscript and approving it for publication was Aneel Rahim.

[1]In some references on threshold ECDSA schemes, a (t, n) threshold signature scheme indicates the scheme where a valid signature can be generated by cooperating with at least $t + 1$ parties, not t parties. However, to clarify throughout the paper, we define a (t, n) threshold scheme as in the body text.

secret key: Once it captures secret shares of at least t parties, it can generate a valid signature by itself. This type of potential threats was introduced by Ostovsky and Yung [21] and several solutions [22], [23], named *proactive* threshold signatures, were presented to avoid such type of attacks. Compared to traditional threshold signature schemes, proactive threshold signatures additionally provide two protocols, *refresh* and *recovery*: A *refresh* protocol updates secret shares of n parties without changing the corresponding secret and public keys of a group of n parties. So, if parties run the refresh protocol periodically, then an adversary should attain secret shares of at least t parties in the same period and it makes threshold signature schemes more secure. A *recovery* protocol is for a party who needs to recover his/her secret share for several reasons, e.g., an attacker's extortion and a party's simple loss. The protocol runs with the help of other honest parties and the party who needs to recover succeeds after executing the protocol.

Recently, proactivation has been also considered in threshold ECDSA schemes. In the open-source [24] of Doerner et al.'s threshold ECDSA schemes [15], the refresh protocol for $(2, 2)$ threshold ECDSA schemes was provided, though the detailed description of the protocol was not explictly given. Later, Canetti et al. [17] presented the refresh protocol for (n, n) threshold ECDSA using Paillier additive homomorphic encryption scheme [25] and zero-knowledge proof techniques. They stated that their refresh protocol can be easily extended to (t, n) threshold settings for $t \leq n$ by applying the secret sharing technique, but the detail was not given. In [26], Kondi et al. provided refresh protocols, but their protocols focused on the case where some of parties are offline. They presented the refresh protocol for $(2, n)$ threshold schemes and showed that it is impossible to design refresh protocols for (t, n) if $t > 2$ where some of n parties are offline. But, all aforementioned works did not consider a recovery protocol. To the best of our knowledge, there is only one recent result for a key recovery protocol [27] for threshold ECDSA schemes, but their protocol considered $(1, 3)$ threshold ECDSA schemes only.

Since each party in (t, n) threshold ECDSA schemes has the same type of secret shares with that of (t, n) secret sharing schemes, one may consider the approach to applying currently existing proactive secret sharing schemes [28], [29], [30] for threshold ECDSA schemes. However, recent studies on proactive secret sharing schemes mainly attempt to achieve robustness and fairness against active corruptions. So, almost all existing solutions achieve stronger security, but are rather inefficient. On the other hand, the currently existing ECDSA schemes exclude robustness and fairness, and consider static corruptions by malicious adversaries only. Thus, it is immoderate to apply the existing proactive secret sharing techniques directly to threshold ECDSA schemes.

A. OUR CONTRIBUTION

In this paper, we provide a new key recovery protocol for (t, n) threshold ECDSA schemes with $t < n$ which is secure against static corruptions by malicious adversaries without considering robustness and fairness, as in the common adversary model of the currently existing threshold ECDSA schemes. In our protocol, there are $t + 1$ parties where one party, say $\mathcal{P}_1$, lost his/her secret share and wishes to recover it, whereas other t parties join with their own secret shares to help $\mathcal{P}_1$. At the beginning of the protocol, all parties, except $\mathcal{P}_1$, generate random values which will be shared with other parties and used for hiding his/her secret share before sending it to $\mathcal{P}_1$. This step can be understood as generating secret shares for zero. In the previous work [22] which achieves the same security level with ours, this step was realized by employing secret sharing techniques for zero. That is, each party generates a random polynomial of degree $t - 1$ whose constant term is zero, evaluates it at identities of other parties, and then shares them with other parties each. However, this brings about a relatively heavy verification step to check whether the received secret shares are generated correctly using verifiable secret sharing schemes [31], [32]. We bypass this verification step by generating just random values as shares and then letting each party add them to his/her own secret appropriately. In this modification, randomly selected shares play a role of masking secret shares and then they are removed after $\mathcal{P}_1$'s simple calculation. As a result, we reduce both the computational and communication costs to $O(t^2)$ from $O(t^3)$.

We also present our implementation results of the proposed recovery protocol and the previous protocol [22] for various t and security levels. Our experimental results show that when $t = 2$ with 128-bit security, our protocol takes 4.21 ms in total for all computations excluding the data transmission time on the network, while the previous result takes 10.46 ms in total. Thus, ours reduces the required time by a factor of about 2.48 times. If t is larger, the advantage of our protocol over the previous result is bigger: When $t = 9$ with 128-bit security, our protocol requires 56.61 ms in total for all computations, while the previous result requires 333.42 ms in total. So, ours outperforms the previous result by a factor of about 5.89 times.

B. RELATED WORK

Since the concept of threshold cryptosystem was proposed by Desmedt and Frankel [6], there were various studies on constructing threshold signature schemes. However, it was a challenging problem to design efficient threshold DSA/ECDSA schemes for a long time. Due to the development of blockchain, the need of threshold ECDSA schemes was also raised. In 2016, Gennaro et al. [10] made a breakthrough using threshold additive homomorphic encryption [33], [34]. Later, there have been proposed various studies on designing threshold ECDSA schemes and they can be divided into two categories by exploited cryptographic tools.

The first category includes the schemes that use additive homomorphic encryption. Lindell [11] proposed two-party threshold ECDSA schemes using Paillier encryption [25]. Then, Lindell and Nof [13] extended it to the multi-party setting using additive ElGamal encryption [35]. Independently, Gennaro and Goldfeder [12] proposed multi-party threshold ECDSA schemes using Paillier encryption. Castagnos et al. [16] presented an efficient two-party ECDSA scheme using additive homomorphic encryption over class groups. Canetti et al. [17] proposed UC-secure and proactive multi-party ECDSA schemes using Paillier encryption. Recently, Deng et al. [19] proposed an efficient multi-party ECDSA protocol using additive homomorphic encryption over class groups by improving the corresponding sigma protocols.

The second category includes the schemes that use other cryptographic primitives. Doerner et al. [14] presented a two-party ECDSA scheme using oblivious transfer and subsequently they extended it to the multi-party setting [15]. Damgård et al. [20] proposed a multi-party ECDSA scheme using secret sharing techniques. Their protocol is efficient, but requires the honest majority assumption. Finally, Xue et al. [18] recently proposed a two-party online-friendly ECDSA scheme. Their construction can be applied for both cases that use additive homomorphic encryption and oblivious transfer each and improved the required online time. We remark that it is assumed that adversaries in all aforementioned schemes in both categories can corrupt parties at the onset of the execution. That is, they allow adversaries for static corruptions only. In addition, they did not consider robustness and fairness.

Though secret keys for threshold schemes are distributively stored, they still have a security issue once at least t secret shares for (t, n) threshold schemes are revealed. This type of threats was introduced by Ostrovsky and Yung [21]. To avoid such an attack, proactive threshold schemes were introduced in [22] and [23]. Following the original works, there have been proposed various studies [28], [29], [30] on proactive secret sharing that additionally provides refresh and recovery protocols, to enhance the security and improve the efficiency. However, the direction for recent proactive secret sharing schemes focuses on achieving stronger security, e.g., robustness and fairness against adaptive corruptions in the dynamic setting. Thus, they are excessive to apply directly for threshold ECDSA schemes.

Recently, there have been proposed several studies on providing the proactive security to threshold ECDSA schemes. In the open-source [24] of Doerner et al.'s protocol [15], a function for the refresh protocol was given, but it works only for (2, 2) threshold ECDSA schemes. Canetti et al. [17] proposed a refresh protocol using Paillier encryption and zero-knowledge proofs, but they provided a description of the refresh protocol for (n, n) threshold ECDSA only and just stated that one can easily obtain a refresh protocol for (t, n) threshold ECDSA schemes with $t < n$ by applying secret sharing techniques. In [26], Kondi et al. presented refresh protocols where some of parties are offline. However, the aforementioned works did not consider a recovery protocol. As far as we know, the authors in [27] only considered the recovery protocol, but the proposed construction was for $(1, 3)$ threshold ECDSA schemes only, not for general cases.

C. OUTLINE OF THE PAPER

Section II reviews the definitions and models for the problem that we consider in this paper and introduces building blocks for our construction. We provide our key recovery protocol with analysis in Section III. Several implementation results and concluding remarks are given in Section IV and Section V, respectively.

II. DEFINITIONS AND BUILDING BLOCKS

In this section, we provide our communication model and definitions for key recovery protocols. Then, we review commitment schemes and secret sharing schemes which are main building blocks of our recovery protocol as well as key generation for (t, n) threshold ECDSA schemes.

A. MODELS AND DEFINITIONS

Now, we briefly introduce our communication model, adversary types, and definitions for key recovery protocols.

1) COMMUNICATION MODEL

It is assumed that our computational model consists of a set of $t + 1$ parties, $\mathcal{P}_1, \ldots, \mathcal{P}_{t+1}$, and they are connected by a complete network of private point-to-point channels as well as a broadcast channel. For simplicity, we assume that a semantically secure encryption scheme [36] is employed for establishing a secure channel. We also assume that the network is partially synchronous and so an adversary speaks last in every communication round.

2) ADVERSARY TYPES

We assume that an adversary can corrupt up to $t - 1$ parties among $t + 1$ parties. (Since we focus on (t, n) threshold schemes, if t parties are corrupted, then the adversary can learn the secret information. On the other hand, we note that our protocol does not consider robustness and so does not require the honest majority assumption.) The adversary is a probabilistic polynomial-time (PPT) Turing machine and sets corrupted parties at the onset of the execution. In other words, the adversary is *static*. We consider a malicious adversary and so the parties corrupted by the adversary can behave in any way.

3) CRYPTOGRAPHIC ASSUMPTION

We use the discrete logarithm (DL) assumption [37, §3.5]. Throughout the paper, for any set X, we use $x \overset{\$}{\leftarrow} X$ to denote a uniform random sampling of x from X.

Definition 1: Let $\mathbb{G}$ be an additive cyclic group of prime order $q = q(\lambda)$ where λ is the security parameter. Let P be a random generator of $\mathbb{G}$. Given an instance (P, aP) with randomly chosen $a \in \mathbb{Z}_q$, the discrete logarithm problem (DLP) is to find $a \in \mathbb{Z}_q$.

We say that the DL assumption holds if for any PPT adversary $\mathcal{A}$

$$\Pr[\mathcal{A}(P, aP) = a : a \xleftarrow{\$} \mathbb{Z}_q]$$

is negligible in the security parameter λ.

4) (t, n)-SECRET SHARING

A secret sharing scheme allows a dealer to share a secret sk among n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, such that any authorized subset of parties can use all their shares to reconstruct the secret, whereas any other non-authorized subset learns nothing about the secret from their shares. More formally, we give a definition of (t, n) secret sharing. We begin by introducing a useful new notation: For an algorithm A, the expression $a \leftarrow A(\cdots)$ denotes that A is executed on the elided inputs and a is assigned its output, throughout the paper.

Definition 2: A (t, n)-secret sharing scheme over message space M is a pair of algorithms $(\mathsf{Share}, \mathsf{Recon})$ *such that*:

$(sk_1, \ldots, sk_n) \leftarrow \mathsf{Share}(sk)$ *is run by a dealer to distribute shares. It is a randomized algorithm that takes as input a secret $sk \in M$ and outputs an n-tuple of shares $(sk_1, sk_2, \ldots, sk_n)$.*

$sk \leftarrow \mathsf{Recon}(sk_{i_1}, \ldots, sk_{i_t})$ *is run by a subset of t or more parties to reconstruct the secret sk. It is a deterministic algorithm that given a t-tuple of shares outputs a secret $sk \in M$.*

There have been various (t, n) secret sharing schemes, e.g., see References [38], [39], [40], [41]. We will further discuss Shamir's secret sharing scheme among them because most of threshold ECDSA schemes rely on it (see Section II-C for the details).

5) A KEY RECOVERY PROTOCOL

In this paper, we consider the following scenario: There are $t + 1$ parties, $\mathcal{P}_1, \ldots, \mathcal{P}_{t+1}$, where each party has a secret share sk_i for secret sk of (t, n) threshold ECDSA scheme with $t < n$.[2] Suppose a party, say $\mathcal{P}_1$, lost his/her secret share and wishes to recover it. The aim of a recovery protocol is to recover sk_1 with the help of other parties, $\mathcal{P}_2, \ldots, \mathcal{P}_{t+1}$, without revealing secret information of other parties and secret sk.

We formalize the syntax and security notion of key recovery protocol below.

Definition 3: A key recovery protocol $\mathcal{K}$ is a pair of polynomial time algorithms $(\mathsf{Group}, \mathsf{Recover})$ *such that*:

$\mathsf{G} \leftarrow \mathsf{Group}(n, t, i)$ *is run by a party $\mathcal{P}_i$ which lost his secret share sk_i and wishes to recover it. This algorithm takes as input a pair of threshold parameters (t, n) and the index of the party $\mathcal{P}_i$, and outputs an ad-hoc group $\mathsf{G} \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \backslash \{\mathcal{P}_i\}$ of cardinality $\geq t$ in which each party $\mathcal{P}_j$ owns its secret share sk_j.*

[2]In fact, our proposed protocol can be applied for other threshold cryptosystems, beyond threshold ECDSA schemes. However, we mainly pay attention to threshold ECDSA schemes in this paper.

$sk_i \leftarrow \mathsf{Recover}(pk, \{sk_j\}_{j \in \mathsf{G}})$ *is run by the parties $\mathcal{P}_i$ and $\mathcal{P}_j \in \mathsf{G} \backslash \{i\}$. The algorithm takes as input the public key pk and a set of secret shares $\{sk_j\}$ of the parties in G, denoted by $\{sk_j\}_{j \in \mathsf{G}}$, and outputs the restored secret share sk_i to the party $\mathcal{P}_i$ and nothing to the parties in G. We remark that the public key pk is implicitly given from an algorithm at a higher level.*

Moreover, it satisfies the following correctness requirement:

- *Correctness: For all $sk \in M$ and for all $\widehat{I} = \{i_1, \ldots, i_k\} \subsetneq \{1, 2, \ldots, n\} \backslash \{i\}$ of cardinality at least $t - 1$,*

$$\Pr_{sk_i \leftarrow \mathsf{Recover}(pk, \{sk_j\}_{j \in \mathsf{G}})}[\mathsf{Recon}(sk_i, \{sk_l\}_{l \in \widehat{I}}) = sk] = 1.$$

We next define the security for such a key recovery protocol. Intuitively, we require that if the adversary is trying to learn any secret information about honest parties' secret share sk_i and the secret sk, it will fail.

Definition 4 Real Model: Let $\mathcal{K}$ be a key recovery protocol where each party $\mathcal{P}_i$ has a secret input x_i and a public input y_i. Each $\mathcal{P}_i$ returns an output z_i after the execution of the protocol $\mathcal{K}$. Let $\mathcal{A}$ be an adversary that can corrupt up to $t - 1$ parties. Let $\boldsymbol{x} = (x_1, y_1, \ldots, x_{t+1}, y_{t+1})$, let $\boldsymbol{r} = (r_1, \ldots, r_{t+1}, r^*)$ be the random inputs the parties and the adversary respectively, let I^* be the index set of the corrupted parties, and let $a \in \{0, 1\}^*$ be the auxiliary input. We denote the output of the adversary after the execution of the protocol as $\mathsf{Real}^{(\mathcal{A})}_{\mathcal{K},\mathcal{A}}(\boldsymbol{x}, I^*, a, \boldsymbol{r})$ and the output of the party $\mathcal{P}_i$ as $\mathsf{Real}^{(i)}_{\mathcal{K},\mathcal{A}}(\boldsymbol{x}, I^*, a, \boldsymbol{r})$. Then let

$$\begin{aligned}\mathsf{Real}_{\mathcal{K},\mathcal{A}}(\boldsymbol{x}, I^*, a, \boldsymbol{r}) = \langle &\mathsf{Real}^{(\mathcal{A})}_{\mathcal{K},\mathcal{A}}(\boldsymbol{x}, I^*, a, \boldsymbol{r}),\\ &\mathsf{Real}^{(1)}_{\mathcal{K},\mathcal{A}}(\boldsymbol{x}, I^*, a, \boldsymbol{r}), \ldots,\\ &\mathsf{Real}^{(t)}_{\mathcal{K},\mathcal{A}}(\boldsymbol{x}, I^*, a, \boldsymbol{r})\rangle.\end{aligned}$$

We define the $\mathsf{Real}_{\mathcal{K},\mathcal{A}}(\boldsymbol{x}, I^*, a, \boldsymbol{r})$ to be the random variable for a uniformly chosen $\boldsymbol{r}$. Finally we define a distribution ensemble $\mathsf{Real}_{\mathcal{K},\mathcal{A}}$ indexed by $(\boldsymbol{x}, I^*, a)$ as

$$\{\mathsf{Real}_{\mathcal{K},\mathcal{A}}(\boldsymbol{x}, I^*, a, \boldsymbol{r})\}_{\boldsymbol{x} \in (\{0,1\}^*)^{2\times(t+1)}, I^* \subsetneq I, a \in \{0,1\}^*}$$

where an index set $I = \{i_1, \ldots, i_t\}$.

Definition 5 Ideal Model: Let $f : (\{0, 1\}^*)^{t+1} \rightarrow (\{0, 1\}^*)^{t+1}$ be a $(t+1)$-ary functionality. We define the output of f as:

$$f((x_1, y_1), (x_2, y_2), \ldots, (x_{t+1}, y_{t+1})) = (z_1, \ldots, z_{t+1}).$$

In the ideal model, parties send their input (x_i, y_i) to a trusted party which computes f and returns $\mathcal{P}_i$ its output value z_i. The ideal model adversary $\mathcal{B}$ can see $(y_1, \ldots, y_{t+1})$ and secret values $\{x_j\}_{j \in I^*}$ for the corrupted parties, and further replace $\{(x_j, y_j)\}_{j \in I^*}$ with $\{(x_j^*, y_j^*)\}_{j \in I^*}$ of its own choice. We denote the output of the ideal model adversary after the evaluation as $\mathsf{Ideal}^{(\mathcal{B})}_{f,\mathcal{B}}(\boldsymbol{x}, I^*, a, \boldsymbol{r})$ and the output of the party $\mathcal{P}_i$ as $\mathsf{Ideal}^{(i)}_{f,\mathcal{B}}(\boldsymbol{x}, I^*, a, \boldsymbol{r})$. Then let

$$\mathsf{Ideal}_{f,\mathcal{B}}(\boldsymbol{x}, I^*, a, \boldsymbol{r}) = \langle \mathsf{Ideal}^{(\mathcal{B})}_{f,\mathcal{B}}(\boldsymbol{x}, I^*, a, \boldsymbol{r}),$$

$$\mathsf{Ideal}^{(1)}_{f,\mathcal{B}}(\boldsymbol{x}, I^*, a, \boldsymbol{r}), \ldots,$$
$$\mathsf{Ideal}^{(t)}_{f,\mathcal{B}}(\boldsymbol{x}, I^*, a, \boldsymbol{r})\rangle.$$

Similarly to the real model, we define a distribution ensemble $\mathsf{Ideal}_{f,\mathcal{B}}$ *indexed by* $(\boldsymbol{x}, I^*, a)$ *as*

$$\{\mathsf{Ideal}_{f,\mathcal{B}}(\boldsymbol{x}, I^*, a)\}_{\boldsymbol{x}\in(\{0,1\}^*)^{2\times(t+1)}, I^*\subsetneq I, a\in\{0,1\}^*}.$$

Definition 6: Let f be as in Definition 5 and $\mathcal{K}$ be a $(t+1)$-party key recovery protocol for computing f. Then we say that $\mathcal{K}$ securely computes f in the statically corrupt setting if for all PPT algorithm $\mathcal{A}$, there exists an ideal model adversary $\mathcal{B}$ whose running time is polynomial in the running time of $\mathcal{A}$, and such that

$$\mathsf{Ideal}_{f,\mathcal{B}} \overset{c}{\approx} \mathsf{Real}_{\mathcal{K},\mathcal{A}}$$

where $\overset{c}{\approx}$ means computationally indistinguishable between two ensembles.

In addition to the above requirements, proactive secret sharing schemes generally are required to achieve the robustness. This property means that the adversary cannot deny their outputs to the honest parties. However, to the best of our knowledge, all currently existing ECDSA schemes do not consider the robustness. So, we exclude it from the security requirements for the recovery protocol.

We remark that more than t parties who have their own secret shares may join the key recovery protocol for (t, n) threshold schemes if $n > t+1$. We can easily adjust our proposed protocol for this case. However, for simple explanation, we assume that exact $t+1$ parties join the protocol where t parties among them have their own secret shares. In addition, two or more parties may lose their secret shares. In this case, we can recover secret shares by running the recovery protocol sequentially if there exist at least t parties who have their own secret shares. So, we assume that just a party loses his/her own secret share in the protocol.

B. COMMITMENT

We briefly review a commitment scheme to make our key recovery protocol secure even in the malicious model. Roughly, we let each party commit to a randomness to mask its secret share before it provides the secret share masked by the randomness.

1) COMMITMENT

A commitment scheme consists of two algorithms Kg and COM, which are defined as follows:

- $ck \leftarrow \mathsf{Kg}(1^\lambda)$ is called the key generation algorithm that takes the security parameter λ and outputs the commitment key ck. The message space M_{ck}, the randomness space R_{ck}, and the commitment space C_{ck} are specified in ck.
- $B \leftarrow \mathsf{COM}(ck, b; r)$ is called the commitment algorithm that takes $b \in M_{ck}$ and outputs a commitment $B \in C_{ck}$ using randomness $r \in R_{ck}$. To commit to a message $b \in M_{ck}$, the sender selects $r \overset{\$}{\leftarrow} R_{ck}$ and computes the commitment $B = \mathsf{COM}(ck, b; r)$. We omit ck when no confusions arise.

We formally define two properties of the security for the commitment scheme.

Definition 7 Hiding property: A commitment scheme is hiding *if for all polynomial-time interactive algorithms $\mathcal{A}$, the following probability is negligible in the security parameter λ:*

$$\left| \Pr\left[\beta = \beta^* \,\middle|\, \begin{array}{l} ck \leftarrow \mathsf{Kg}(1^\lambda);\ (b_0, b_1) \leftarrow \mathcal{A}(ck); \\ \beta \overset{\$}{\leftarrow} \{0,1\};\ r \overset{\$}{\leftarrow} R_{ck}; \\ B \leftarrow \mathsf{COM}(b_\beta; r);\ \beta^* \leftarrow \mathcal{A}(ck, B) \end{array} \right] - \frac{1}{2} \right|.$$

Definition 8 Binding property: A commitment scheme is binding *if for all polynomial-time interactive adversaries $\mathcal{A}$, the following probability is negligible in the security parameter λ:*

$$\Pr\left[\begin{array}{c} \mathsf{COM}(b_0; r_0) = \mathsf{COM}(b_1; r_1) \\ \textit{and} \\ b_0 \neq b_1 \end{array} \,\middle|\, \begin{array}{l} ck \leftarrow \mathsf{Kg}(1^\lambda); \\ \binom{b_0, b_1}{r_0, r_1} \leftarrow \mathcal{A}(ck) \end{array} \right].$$

2) INSTANTIATIONS

Let P and Q be two random generators of a group $\mathbb{G}$ of prime order q. Consider a commitment scheme simply defined as $\mathsf{COM}(b; r) := bP$. It is known as the Feldman commitment [31], and satisfies unconditionally binding and computationally hiding properties under the assumption that the DL problem is hard in $\mathbb{G}$. Another scheme, known as the Pedersen commitment [32], is of the form $\mathsf{COM}(b; r) := bP + rQ$, where $r \in \mathbb{Z}_q$. Pedersen commitments are unconditionally hiding and computationally binding under the DL assumption. Finally, a committer may use $H(b)$ or $H(b \parallel r)$ to commit to b for a collision-resistant hash function H. Refer to Section 3.5 of [42] that covers commitment schemes for details.

Remark 1: A popular way to achieve the malicious security is to combine a commitment scheme and zero-knowledge proof protocols. However, this approach is clearly quite effective at the price of computation and communication overheads. For this performance reason, we let a sender open a commitment to a value over a secure channel later. Then a receiver can test if the sender honestly committed to the value only by a simple calculation and more importantly, it can directly use the value in subsequent computation. This approach is more suitable for our setting.

C. SHAMIR SECRET SHARING

Secure key distribution is one of key components to design threshold cryptosystems. Among various techniques for key distribution, a Shamir secret sharing scheme [43] is the most well-known and widely utilized technique. It is based on polynomial evaluation and interpolation, and consists of two phases: key distribution and reconstruction.

Suppose n parties, $\mathcal{P}_1, \ldots, \mathcal{P}_n$, would like to share a secret key sk with threshold t, that is, $t-1$ or fewer parties cannot recover the secret key sk using their own secret shares sk_i's, whereas at least t parties can recover it jointly. In the key distribution phase, a dealer first generates a random polynomial $f(x)$ of degree $t-1$ with the constant term $f(0) = sk$. Then, he evaluates $f(x)$ at i for each party $\mathcal{P}_i$ and forwards $sk_i = f(i)$ to $\mathcal{P}_i$ as his/her own secret share via a secure channel between the dealer and $\mathcal{P}_i$.

Next, once a need arises, t parties join the reconstruction phase to recover the secret key sk. In the reconstruction phase, each party computes the Lagrange coefficient

$$\ell_i^J(0) = \prod_{J\setminus\{i\}} \frac{-i}{j-i} \tag{1}$$

where J is the set of indices of t parties, e.g., $J = \{1, 2, 4\}$ if $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_4$ join the key reconstruction phase for $t = 3$. Then, each party computes and shares

$$sk_i' = \ell_i^J(0) \cdot sk_i$$

with all other parties. Once receiving sk_i''s, each party succeeds in recovering the secret key sk by computing

$$\sum_{i\in J} sk_i' = \sum_{i\in J} \ell_i^J(0) \cdot sk_i.$$

In fact, if t parties, excluding $\mathcal{P}_1$, join the key reconstruction phase with their own secret shares sk_i, they can jointly recover the secret share of $\mathcal{P}_1$ by replacing $\ell_i^J(0)$ and $sk_i' = \ell_i^J(0) \cdot sk_i$ with

$$\ell_i^J(1) = \prod_{J\setminus\{i\}} \frac{1-i}{j-i} \quad \text{and} \quad sk_i' = \ell_i^J(1) \cdot sk_i, \tag{2}$$

respectively.

In addition, there are several issues on using the Shamir secret sharing scheme. First, in the description of the scheme above, there is a dealer who knows the secret key sk. To remove the help of the dealer, each party $\mathcal{P}_i$ performs as the dealer to share a secret s_i which is selected by $\mathcal{P}_i$ him/herself and privately sends $s_{i,j}$ to $\mathcal{P}_j$ for $1 \le j \le n$ as the secret share of $\mathcal{P}_j$ for s_i. Then, each party $\mathcal{P}_i$ sets his secret share sk_i to $\sum_{j=1}^n s_{j,i}$ and regards sk as $\sum_{i=1}^n s_i$, which is secret to all. This holds from the relation $f(a) + g(a) = (f+g)(a)$ where f, g are polynomials and a is a point in the domain of f and g.

Second, after finishing the key reconstruction phase, one obtains the secret key sk. However, in the practical schemes that exploit the secret sharing scheme, the final result is the same as an outcome of algorithms which take the secret key as an input, not the secret key itself. For example, the party who join the signing protocol of threshold ECDSA schemes obtains a valid ECDSA signature of an input message, not the secret key itself. Thus, we can repeatedly use shared secrets in practice.

D. KEY GENERATION RESULT IN THRESHOLD ECDSA SCHEMES

Since this paper focuses on a key recovery protocol for (t, n) threshold ECDSA schemes, it is assumed that parties already share a secret by jointly executing a key generation protocol. The detailed descriptions of key generation protocols in threshold ECDSA schemes are not our interest, so we omit them. However, it is important to know that each party has which type of secret shares after the key generation protocol.

Let $\mathbb{G}$ be an additive cyclic group of prime order q over elliptic curve E and P be a generator of $\mathbb{G}$. We assume that a threshold ECDSA scheme is defined over $\mathbb{G}$. To execute a key recovery protocol, $t+1$ parties already shared a secret with threshold t. So, n should be larger than t in (t, n) threshold ECDSA schemes. In this case, the Shamir secret sharing scheme is used for secret key sharing in all currently existing ECDSA schemes. As a result, each party has a secret share sk_i where $sk_i = f(i)$ for a random polynomial $f(x)$ which is hidden to all parties and whose constant term is a secret $sk = f(0)$. Furthermore, $X_i = sk_iP$ is a public key for party $\mathcal{P}_i$ and $X = skP$ is the common public key for n parties. We note that the relations

$$sk = \sum_{j\in J} \ell_j^J(0) sk_j \quad \text{and} \quad X = \sum_{j\in J} \ell_j^J(0) X_j$$

hold where J is a subset of indices of n parties with the size t or larger, and $\ell_j^J(0)$ is the Lagrange coefficient defined as in Equation (1) of Section II-C.

III. OUR KEY RECOVERY PROTOCOL

In this section, we first present the description of our key recovery protocol for threshold ECDSA schemes. We then provide security and efficiency analysis of the proposed construction.

A. PROTOCOL DESCRIPTION

The aim of the proposed protocol is to re-issue the secret share of the party who lost it for (t, n)-threshold ECDSA schemes where the re-issued share is the same as the previously issued one. More concretely, in our protocol there are $t+1$ parties, $\mathcal{P}_1, \ldots, \mathcal{P}_{t+1}$, where one party, say $\mathcal{P}_1$, lost his secret share sk_1 and wishes to recover it with the help of all other parties $\mathcal{P}_i$'s who join the protocol with their own secret shares sk_i's each. We assume that $X_i = sk_iP$ and $X = skP$ are public to all parties, where P is a generator of an underlying additive cyclic group $\mathbb{G}$ over an elliptic curve E, as described in Section II-D.

1) PROTOCOL ABSTRACTION

We begin with presenting how our protocol works in the abstract level. To recover $\mathcal{P}_1$'s secret share, $\mathcal{P}_1$ should obtain

$$sk_1 = \sum_{j\in \mathsf{G}} \ell_j^{\mathsf{G}}(1) \cdot sk_j$$

where $\mathsf{G} = \{2, 3, \ldots, t+1\}$ and $\ell_j^{\mathsf{G}}(1)$ is the Lagrange coefficient defined as in Equation (2) of Section II-D. At the same time, sk_i should be hidden to $\mathcal{P}_1$. To hide sk_i, at the beginning of the protocol, each party $\mathcal{P}_i$ for $i \in \mathsf{G}$ generates random values b_{ij}'s for $j \in \mathsf{G}\setminus\{i\}$ and passes b_{ij} to $\mathcal{P}_j$ with the commitment to b_{ij} via a secure channel. Then, each party $\mathcal{P}_i$ for $i \in \mathsf{G}$ adds the shares generated by him/herself to

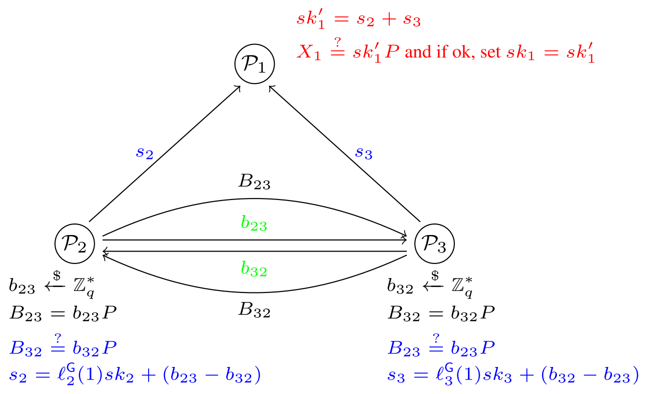
$sk_1' = s_2 + s_3$
$X_1 \overset{?}{=} sk_1' P$ and if ok, set $sk_1 = sk_1'$
$\mathcal{P}_1$
s_2
s_3
B_{23}
b_{23}
b_{32}
B_{32}
$\mathcal{P}_2$
$\mathcal{P}_3$
$b_{23} \xleftarrow{\$} \mathbb{Z}_q^*$
$B_{23} = b_{23}P$
$B_{32} \overset{?}{=} b_{32}P$
$s_2 = \ell_2^{\mathsf{G}}(1)sk_2 + (b_{23} - b_{32})$
$b_{32} \xleftarrow{\$} \mathbb{Z}_q^*$
$B_{32} = b_{32}P$
$B_{23} \overset{?}{=} b_{23}P$
$s_3 = \ell_3^{\mathsf{G}}(1)sk_3 + (b_{32} - b_{23})$

FIGURE 1. Illustrative toy example of our key recovery protocol with the Feldman commitment scheme [31] when $t = 2$. The black, blue, green, and red letters correspond to Steps 1), 2), 3), and 4), respectively. $\ell_i^{\mathsf{G}}(x)$ is the Lagrange coefficient polynomial of a party $\mathcal{P}_i$ where $\mathsf{G} = \{2, 3\}$. The symbol $a \overset{?}{=} b$ indicates to check equality of a and b; thus unless the equality holds, then the process will be terminated.

$\ell_i^{\mathsf{G}}(1) \cdot sk_i$, subtracts all the received shares from it, and sends the final result to $\mathcal{P}_1$. Lastly, $\mathcal{P}_1$ obtains sk_1 by adding all received values from other parties.

2) PROTOCOL DESCRIPTION

We are ready to describe our key recovery protocol in detail. Recall that a key recovery protocol consists of a pair of polynomial time algorithms–Group, Recover (see Definition 3).

a: THE AD-HOC GROUPING ALGORITHM Group($n, t, 1$)

A subset of n parties groups together by aggregating the identities of parties that wish to engage in a key recovery process. Conceptually, an easy way to obtain such a subset is a public bulletin board for parties who wish to create an ad-hoc group.

We should notice that a single party that is allowed to set up a group may have a power that can easily be abused. For example, some corrupted party can always group a single honest party with other $t-1$ parties that it can control over. Thus we consider a secure way to prevent the party from grouping the parties as it wishes. Informally, all parties run a sort of joint coin tossing protocol so that the parties controlled by the malicious adversary are uniformly distributed within all the groups running the protocol.

The algorithm starts with a request from party $\mathcal{P}_1$ which wishes to recover its secret share. Then our grouping algorithm directly uses the group setup protocol proposed by Lindell and Waisbard in [44, §5.2]. At the end of the execution, the grouping protocol outputs a group of t parties $\mathsf{G} = \{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_t}\}$. For completeness, we provide a description of Lindell and Waisbard's protocol in Appendix A.

b: THE RECOVERY ALGORITHM Recover($pk, \{sk_j\}_{j\in\mathsf{G}}$)

The secret share recovery algorithm jointly works as follows. For notational simplicity, we think of the group G as a set of indices rather than a set of identities, and thus we write simply $\mathsf{G} = \{2, 3, \ldots, t+1\}$. As mentioned before, we assume that the party $\mathcal{P}_1$ requests a recovery process.

1) Each $\mathcal{P}_i$, $i \in \mathsf{G}$, performs the following steps:

   For each $j \in \mathsf{G}$ and $j \neq i$,
   a) Select a random element b_{ij} from $\mathbb{Z}_q^*$ and a randomizer r_{ij} from R_{ck}.
   b) Compute and send $B_{ij} = \mathsf{COM}(b_{ij}; r_{ij})$ to $\mathcal{P}_j$.
2) Each $\mathcal{P}_i$, $i \in \mathsf{G}$, opens B_{ij} by sending (b_{ij}, r_{ij}) to $\mathcal{P}_j$.
3) Each $\mathcal{P}_i$, $i \in \mathsf{G}$, performs the following steps:

   For each $j \in \mathsf{G}$ and $j \neq i$,
   a) Check whether B_{ij} is the commitment to b_{ij}. Abort if it does not hold.
   b) Compute
   $$s_i = \ell_i^{\mathsf{G}}(1)sk_i + \sum_{j\in\mathsf{G}\setminus\{i\}} (b_{ij} - b_{ji})$$
   where $\ell_i^{\mathsf{G}}(1) = \prod_{j\in\mathsf{G}\setminus\{i\}} \dfrac{1-j}{i-j}$.
   c) Send s_i to $\mathcal{P}_1$ over a secure channel.
4) $\mathcal{P}_1$ performs the following steps:
   a) Compute $sk_1' = \sum_{i\in\mathsf{G}} s_i$.
   b) Check if
   $$sk_1'P = X_1.$$
   Abort if it does not hold.
   c) Output sk_1'.

We briefly check over the security property of our protocol first. For the correctness, while $\mathcal{P}_1$ adds all received values, random values b_{ij}'s that are added or subtracted by other parties are all cancelled since addition and subtraction of each value are performed once each. So, the correctness of the proposed protocol is guaranteed.

Let us move on the secrecy. We notice that $\mathcal{P}_1$ needs the help of $t-1$ parties to get rid of random values from the received values. However, because we allow the corruption of up to $t-1$ parties including $\mathcal{P}_1$, $\mathcal{P}_1$ cannot learn the information on sk. If $\mathcal{P}_1$ is not corrupted, other parties cannot receive the secret information related to the target honest party because the values related to secret share are passed to $\mathcal{P}_1$ only via a secure channel. So, the secrecy of the proposed protocol is also guaranteed.

In the next subsection, we will provide a detailed analysis of security evaluation.

c: Toy Example for $t = 2$

In Figure 1, we provide a toy example for better understanding how our key recovery protocol works, where we consider three parties, $\mathcal{P}_1, \mathcal{P}_2$, and $\mathcal{P}_3$, and suppose that the party $\mathcal{P}_1$ lost his/her secret share sk_1. Recall that for the secret share sk_i of $\mathcal{P}_i$, the corresponding public key is written as $X_i = sk_iP$.

We assume that the Feldman commitment scheme [31] is employed as a commitment scheme for simple explanation.

At Step 1, the party $\mathcal{P}_2$ begins with picking a random integer $b_{23} \in \mathbb{Z}_q^*$ and computes a commitment to b_{23} in $B_{23} = b_{23}P$. Then $\mathcal{P}_2$ sends B_{23} to $\mathcal{P}_3$. At the same time, $\mathcal{P}_3$ picks a random integer $b_{32} \in \mathbb{Z}_q^*$, computes $B_{32} = b_{32}P$, and sends B_{32} to $\mathcal{P}_2$. Subsequently, at Step 2, $\mathcal{P}_2$ opens B_{23} by sending b_{23} to $\mathcal{P}_3$ via a secure channel, while $\mathcal{P}_3$ opens B_{32} by sending b_{32} to $\mathcal{P}_2$ via a secure channel.

At Step 3, upon receiving b_{32} from $\mathcal{P}_3$, $\mathcal{P}_2$ checks if $b_{23}P$ equals to B_{32}. If the two values are different, then it terminates the recovery process. Using the Lagrange coefficient polynomial $\ell_2^{\mathsf{G}}(z)$, $\mathcal{P}_2$ computes

$$s_2 = \ell_2^{\mathsf{G}}(1)sk_2 + (b_{23} - b_{32}),$$

where $\mathsf{G} = \{2, 3\}$ and sends s_2 to $\mathcal{P}_1$ via a secure channel. Similarly, $\mathcal{P}_3$ sends

$$s_3 = \ell_3^{\mathsf{G}}(1)sk_3 + (b_{32} - b_{23}).$$

to $\mathcal{P}_1$ via a secure channel if B_{23} is valid.

Finally, at Step 4, $\mathcal{P}_1$ is ready to find his/her secret share sk_1. Firstly, it computes $sk_1' = s_2 + s_3$ and then it tests if the public key X_1 equals to the value $sk_1'P$. If the two values are different, then abort the protocol. Otherwise $\mathcal{P}_1$ sets sk_1 to $sk_1 = sk_1'$.

d: DIFFERENCES BETWEEN OURS AND THE HJKY PROTOCOL IN [22]

The HJKY protocol presented by Herzberg et al. [22] achieves the similar security level to ours under the adversary model that we have considered. In fact, the construction idea is very similar to ours, but the main difference is that the HJKY protocol exploits verifiable secret sharing schemes [31], [32] to generate values for masking the secret share sk_i. More precisely, in their protocol, each party $\mathcal{P}_i$ for $i \in \mathsf{G}$ first generates a random polynomial $r_i(x)$ of degree $t-1$ with $r_i(0) = 0$, evaluates it at j for $j \in \mathsf{G}\backslash\{i\}$, and passes $r_i(j)$ to $\mathcal{P}_j$ with homomorphic commitments to coefficients. Then, after receiving values, each party $\mathcal{P}_i$ checks whether the received values $r_j(i)$'s are correctly computed with the committed values. If all verification passes, each party adds the received values to his/her secret share and then sends the result to $\mathcal{P}_1$. Finally, $\mathcal{P}_1$ obtains sk_1 by applying the Lagrange interpolation to the received values.

In the above process, to verify values $r_i(j)$'s, each party sends $O(t)$ commitments to other parties each. So, the total communication cost for this step is $O(t^3)$. Moreover, it takes $O(t)$ group operations to check a value $r_i(j)$ using homomorphic commitment schemes, like Feldman and Pedersen commitment schemes [31], [32]. So, the total computational cost for this step is $O(t^3)$ group operations as well.

B. SECURITY ANALYSIS

In this subsection, we look into the correctness and secrecy of our proposed construction.

1) CORRECTNESS

Now, we investigate the correctness of our proposed key recovery protocol. That is, we check that the party $\mathcal{P}_1$ obtains his/her own secret share sk_1 once other parties $\mathcal{P}_2, \ldots, \mathcal{P}_{t+1}$ follow the protocol as described faithfully.

The following theorem shows the correctness of our proposed protocol.

Theorem 1: In our recovery algorithm $\mathsf{Recover}()$, *$\mathcal{P}_1$ obtains his/her exact secret share sk_1 that he/she lost if all parties behave as described faithfully. Therefore, our protocol satisfies the correctness of key recover protocol in Definition 3.*

Proof: First, at Step 3, each party $\mathcal{P}_i$ has b_{ji}'s for all $j \in \mathsf{G}$ with $j \neq i$ and the relation $B_{ji} = \mathsf{Com}(b_{ji}; r_{ji})$ holds for each j if the party $\mathcal{P}_j$ generates B_{ji} honestly. Next, each party $\mathcal{P}_i$ generates

$$s_i = \ell_i^{\mathsf{G}}(1)sk_i + \sum_{j \in \mathsf{G}\backslash\{i\}} (b_{ij} - b_{ji})$$

for the set $\mathsf{G} = \{2, \ldots, t+1\}$ and sends s_i to $\mathcal{P}_1$. Once receiving s_i's from $\mathcal{P}_i$, $\mathcal{P}_1$ obtains sk_1 since

$$\begin{aligned} sk_1' &= \sum_{i \in \mathsf{G}} s_i \\ &= \sum_{i \in \mathsf{G}} \left(\ell_i^{\mathsf{G}}(1)sk_i + \sum_{j \in \mathsf{G}\backslash\{i\}} (b_{ij} - b_{ji}) \right) \\ &= \sum_{i \in \mathsf{G}} \ell_i^{\mathsf{G}}(1)sk_i + \sum_{i \in \mathsf{G}} \left(\sum_{j \in \mathsf{G}\backslash\{i\}} (b_{ij} - b_{ji}) \right) \\ &= \sum_{i \in \mathsf{G}} \ell_i^{\mathsf{G}}(1)sk_i = sk_1. \end{aligned}$$

The second equality holds if parties $\mathcal{P}_2, \ldots, \mathcal{P}_{t+1}$ follow the protocol honestly. The fourth equality holds since in $\sum_{i \in \mathsf{G}} \left(\sum_{j \in \mathsf{G}\backslash\{i\}} (b_{ij} - b_{ji}) \right)$, each b_{ij} is added and subtracted exactly once and so the result is zero. Finally, the last equality holds from the relation of the Lagrange interpolation.

Since sk_1 is the exact secret share for $\mathcal{P}_1$ that it lost, once at least t parties including $\mathcal{P}_1$ jointly run $\mathsf{Recon}()$ algorithm, it returns the secret key sk. Therefore, it satisfies the correctness of Definition 3. □

2) SECRECY

Next, we take a closer look at the secrecy of our proposed protocol. Since our protocol is designed to recover a secret share of a particular party in the (t, n)-threshold ECDSA scheme, all secrets are inherently revealed once t parties who have their own secret shares are corrupted. Thus, we assume that the adversary can corrupt up to $t-1$ parties among $t+1$ parties. Furthermore, a secret share sk_i of $\mathcal{P}_i$ who is not corrupted, should not be revealed to the adversary. Otherwise, the adversary can have t secret shares including them of corrupted parties and so the secret sk can be revealed.

The following theorem shows the secrecy of our proposed protocol.

Theorem 2: Suppose that $\mathcal{P}_1$ wishes to recover the secret share of sk, and let $\mathsf{G} = \{2, 3, \ldots, t+1\}$ *be a group of parties given by algorithm* $\mathsf{Group}(n, t, 1)$*. The protocol described in Section III-A is secure against the malicious adversary which can corrupt at most $t-1$ parties, assuming that the exploited secure channel is established by a semantically secure encryption scheme, and the employed commitment scheme satisfies hiding and binding properties.*

Proof: In this simulation-based proof, we will construct an algorithm for an ideal model adversary $\mathcal{B}$ which is polynomial in the running time. The ideal model adversary $\mathcal{B}$ communicates with a set of corrupted parties $I^* \subset \mathsf{G}$, which is of cardinality at most $t-1$, pretending a set of honest parties in such a way that the corrupted parties in I^* cannot detect if he is either in the real protocol or in the ideal world. The incorruptible trusted party takes the input from $\mathcal{B}$ and the honest parties in the ideal world, and returns the output to $\mathcal{B}$ and/or the honest parties. Then $\mathcal{B}$ communicates with the malicious parties in I^*, and thus they also learn the same output. We formally describe the algorithm as follows:

1) For each simulated honest party $\delta \in \Delta = \mathsf{G} \backslash I^*$, perform the followings:
   a) Choose a random value $sk_\delta \in M$ where recall M is a message space for a secret sharing scheme.
   b) Pick random elements $b_{\delta j} \in \mathbb{Z}_q^*$ and randomizers $r_{\delta j} \in R_{ck}$ for each $j \in \mathsf{G} \backslash \{\delta\}$.
2) Perform Steps 1 and 2 of the algorithm $\mathsf{Recover}()$:
   a) Send the commitment $B_{\delta j^*}$ to $b_{\delta j^*}$ to each malicious party $j^* \in I^*$.
   b) Send an encryption of $(b_{\delta j^*}, r_{\delta j^*})$, denoted $\bar{b}_{\delta j^*}$, to each malicious party $j^* \in I^*$.
   c) Receive from each malicious party $i^* \in I^*$:
      i) commitment $B_{i^* j}$ to $b_{i^* j}$
      ii) a pair of committed values $(b_{i^* j}, r_{i^* j})$ and its encryption $\bar{b}_{i^* j}$
      iii) all pairs of committed values $\{(b_{\delta i^*}, r_{\delta i^*})\}_{\delta \in \Delta}$ and a set of corresponding encryptions $\{\bar{b}_{\delta i^*}\}$
3) $\mathcal{B}$ obtains $\{sk_{i^*}\}_{i^* \in I^*}$, $\{b_{\delta j^*}\}_{j^* \in I^*}$, and $\{b_{i^* j}\}_{i^* \in I^*}$. For all $i^* \in I^*$, build
$$s_{i^*} = \ell_{i^*}^{\mathsf{G}}(1) \cdot sk_{i^*} + \sum_{j \in \mathsf{G} \backslash \{i^*\}} (b_{i^* j} - b_{j i^*}).$$
4) $\mathcal{B}$ submits $\{s_{i^*}\}_{i^* \in I^*}$ to the trusted third party. The honest parties submit their s_δ to the trusted third party.
   - Case $1 \notin I^*$. The trusted party returns sk_1' to the honest party $\mathcal{P}_1$.
   - Case $1 \in I^*$. The trusted party returns sk_1' to $\mathcal{B}$.
5) $\mathcal{B}$ follows the rest of the protocol with the malicious parties as instructed.

This completes the description of $\mathcal{B}$. To check that $\mathcal{B}$ works as required, in the real and ideal worlds, we fix arbitrary values in the input and random tape of the environment $\mathcal{Z}$, which forces the random choices of the players and those of $\mathcal{B}$ to be the only source of randomness. We now argue that for any set of fixed values, $\mathcal{Z}$ cannot computationally determine whether it interacts with the ideal world or with the real protocol, if we use $\mathcal{B}$ in the ideal world as described above.

Note that what $\mathcal{Z}$ can observe is the output generated by the parties, plus it sees the view of the corrupted parties. As a consequence, it will be clearly sufficient to prove the following.

Firstly, until Step 1 the only messages $\mathcal{Z}$ that will see from honest parties are random elements and randomizers they hold. This is perfectly simulated: Both in simulation and in the real protocol the adversary sees $t-1$ independently random elements and values. Then it is straightforward to see that the rest of the values in the view follow in a correct way from the starting values.

At Step 2, $\mathcal{Z}$ will see results for all parties. In the ideal world, these results are computed according to the given function by functionality f from the inputs specified by $\mathcal{Z}$. In the real protocol, however it can check that all parties will compute the same function from the inputs specified by $\mathcal{Z}$. In Step 2c, $\mathcal{Z}$ will see the corrupted parties' view of commitments, encryptions, and random elements. Security of the commitment scheme guarantees that the distribution of these commitments in the ideal is identical to that of in the real protocol. Similarly, the security of the exploited encryption scheme for constructing secure channels guarantees that the distribution of these encryptions is computationally indistinguishable from that in the real protocol. If these values go to the honest parties, nothing more is revealed. This covers the case where $1 \notin I^*$. If these values go to the corrupted parties including $\mathcal{P}_1$ (i.e., the case where $1 \in I^*$), observe that in the real protocol, there are at least two or more honest parties, thus although the view of $\mathcal{P}_1$ is given by
$$\underbrace{\sum_{\delta \in \Delta} \ell_\delta^{\mathsf{G}}(1) sk_\delta + R_\Delta}_{\text{from honest parties}} + \underbrace{\sum_{i^* \in I^*} \ell_{i^*}^{\mathsf{G}}(1) sk_{i^*} + R_{I^*}}_{=\sum s_{i^*}}.$$
The above value is uniformly random because sk_δ and R_Δ are uniformly random. Here, R_Δ (resp., R_{I^*}) is obtained by adding random elements of the honest parties (resp., corrupted parties) and subtracting random elements received by the honest parties from all other parties in G. It is now clear that the procedure used by $\mathcal{B}$ to recover a lost secret share sk_1 leads to the same distribution. This concludes the proof. □

C. EFFICIENCY ANALYSIS

First, let us evaluate the computational cost of our protocol. For simplicity, we assume that the Feldman commitment scheme [31] is exploited in our key recovery protocol, and it requires a scalar multiplication of elliptic curve point for commitment generation and verification each. Then, the most expensive operation in our protocol is a scalar multiplication

TABLE 1. **Theoretical efficiency comparison of ours and the HJKY protocol [22].**

	Ours			HJKY [22]		
	$\mathcal{P}_1$	$\mathcal{P}_i$ $(i \neq 1)$	Total	$\mathcal{P}_1$	$\mathcal{P}_i$ $(i \neq 1)$	Total
Comp	1	$2(t-1)$	$2t^2 - 2t + 1$	1	$2(t-1)t$	$2t^3 - 2t^2 + 1$
Comm	–	$t\vert\mathbb{Z}_q\vert + (t-1)\vert\mathbb{G}\vert$	$t^2\vert\mathbb{Z}_q\vert + t(t-1)\vert\mathbb{G}\vert$	–	$t\vert\mathbb{Z}_q\vert + t^2\vert\mathbb{G}\vert$	$t^2\vert\mathbb{Z}_q\vert + t^3\vert\mathbb{G}\vert$

Comp: Computational cost, Comm: Communication cost

Unit of computational costs: commitment computation, Unit of communication costs: bits

$|\mathbb{Z}_q|$ and $|\mathbb{G}|$ denote the bit lengths to represent elements in $\mathbb{Z}_q$ and $\mathbb{G}$, respectively.

TABLE 2. **Experimental results of ours and the HJKY protocol for small *t* and various security levels (Unit: ms).**

t	Curves	Ours			HJKY [22]		
		Step 1)	Step 3)	Step 4)	Step 1)	Step 3)	Step 4)
1	SECP160k1	0.363	0.358	0.174	0.684	0.994	0.182
	SECP192k1	0.382	0.385	0.184	0.791	1.144	0.207
	SECP224k1	0.552	0.544	0.264	1.120	1.591	0.281
	SECP256k1	0.683	0.681	0.332	1.241	1.791	0.320
2	SECP160k1	0.995	0.975	0.160	1.453	3.675	0.180
	SECP192k1	1.147	1.136	0.186	1.717	4.402	0.224
	SECP224k1	1.684	1.654	0.276	2.508	6.426	0.305
	SECP256k1	1.968	1.926	0.317	2.842	7.267	0.358
3	SECP160k1	1.985	1.967	0.168	2.643	9.489	0.220
	SECP192k1	2.409	2.406	0.198	3.156	11.423	0.275
	SECP224k1	3.388	3.321	0.277	4.394	15.863	0.342
	SECP256k1	3.953	3.917	0.323	4.982	18.118	0.415

of elliptic curve point. At Steps 1 and 3, each party, except $\mathcal{P}_1$, takes $(t-1)$ scalar multiplications, respectively. So, $\mathcal{P}_2, \ldots, \mathcal{P}_{t+1}$ take $2(t-1)$ scalar multiplications. On the other hand, $\mathcal{P}_1$ takes only one scalar multiplication at the last step for verification. Thus, the total computational cost is $2(t-1)t+1$ scalar multiplications, which correspond to $O(t^2)$.

Next, let us consider the communication cost of our protocol. Again under assuming that the Feldman commitment scheme [31] is exploited in our key recovery protocol, at Steps 1 and 2, each party, except $\mathcal{P}_1$, sends two elements in $\mathbb{Z}_q$ and $\mathbb{G}$, respectively, to all other parties. At Step 3 each party, except $\mathcal{P}_1$, sends an element in $\mathbb{Z}_q$ to $\mathcal{P}_1$. So, the total communication cost is $t^2|\mathbb{Z}_q|+t(t-1)|\mathbb{G}|$, which corresponds to $O(t^2)$, where $|\mathbb{Z}_q|$ and $|\mathbb{G}|$ are the bit lengths to represent elements in $\mathbb{Z}_q$ and $\mathbb{G}$, respectively.

In Table 1, we summarize the computational and communication complexities of our protocol and the HJKY protocol [22] that achieves the same security with ours.

IV. IMPLEMENTATION RESULTS

In this section, we present implementation results of ours and the HJKY protocol in [22] for various parameter settings.

A. EXPERIMENTAL ENVIRONMENTS

The source codes[3] of our implementation were written in C++ and compiled using g++ 9.4.0 compiler. We used the OpenSSL library [45] for using the symmetric key encryption, AES-256-GCM, to establish private peer-to-peer channels, and for implementing arithmetic of large numbers and elliptic curve operations in the protocols. However, we do not implement data transmission on the network. Instead, we store and read them on memory once party's computation at each step ends and begins, respectively. We have tested the programs on the modern PC with Intel(R) Core(TM) i7-11700 CPU at 2.50 GHz and 32 GB RAM. The operating system for our experiments was Ubuntu 20.04 LTS on Windows Subsystem for Linux (WSL) on Windows 10 pro 64-bits.

B. EXPERIMENTAL RESULTS

Now, we present our experimental results of ours and the HJKY protocol for various parameters. Each test was done 100 times and the results in the tables and graphs below are averages of those execution times.

Table 2 gives computation times of our protocol and the HJKY protocol for small t and several security levels. In Table 2, SECP160k1, SECP192k1, SECP224k1, and SECP256k1 denote specific Koblitz curves for 80-bit, 96-bit, 112-bit, and 128-bit security, respectively, defined by Standards for Efficient Cryptography Group [46]. The table shows that our protocol outperforms the HJKY protocol. For example, when $t = 2$ for 128-bit security, our protocol requires 4.211 ms in total for computations, while the HJKY

[3]https://github.com/CryptoLabCAU/KeyRecovery

TABLE 3. Experimental results of ours and the HJKY protocol for various *t* at 128-bit security (Unit: ms).

	SECP256k1					
t	Ours			HJKY [22]		
	Step 1)	Step 3)	Step 4)	Step 1)	Step 3)	Step 4)
3	3.953	3.917	0.323	4.982	18.118	0.415
6	12.867	12.765	0.307	15.692	103.892	0.667
9	28.413	27.885	0.316	31.752	300.614	1.051
12	47.484	47.422	0.311	52.852	657.455	1.556

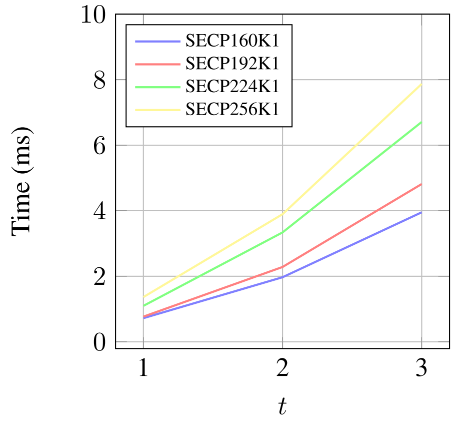

FIGURE 2. Total computation time of ours for various security level at small *t*.

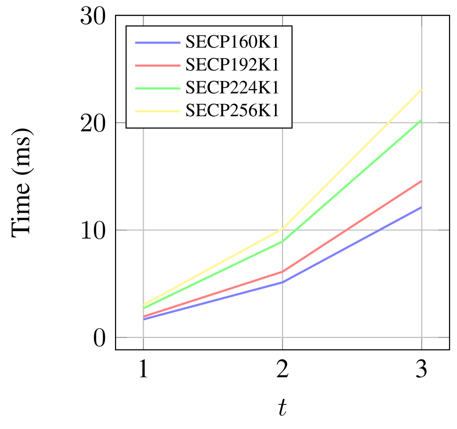

FIGURE 3. Total computation time of the HJKY Protocol for various security level at small *t*.

protocol requires 10.467 ms for the same purpose. Ours reduces the computation time by a factor of about 2.49 times. Figures 2 and 3 pictorially present computation times of ours and the HJKY protocol, respectively, and the results tend to increase when t and security levels are increased, as expected.

When t is larger, the advantage of our protocol becomes higher: Table 3 provides several implementation results of ours and the HJKY protocol for larger t at 128-bit security. In the table, when $t = 9$, while the HJKY protocol takes 333.42 ms in total for computations, ours takes 56.61 ms for the same purpose which improves by a factor of 5.89 times. Figure 4 shows a pictorial description of total computation time of ours and the HJKY protocol. It shows that the gap

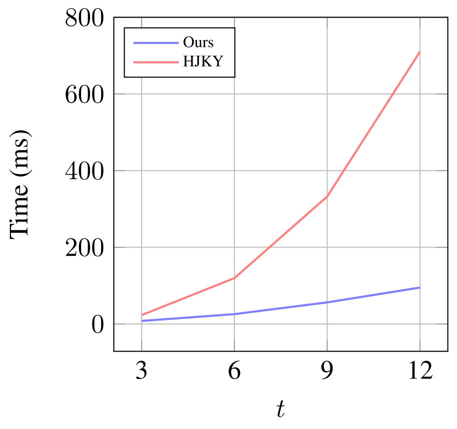

FIGURE 4. Comparison of total computation time of ours and the HJKY protocol for various *t* at 128-bit security.

between the results of ours and the HJKY protocol becomes larger as t is increased.

V. CONCLUSION

In this paper, we present a new key recovery protocol for a party who lost his/her secret share that is secure against static corruptions of up to $(t-1)$ parties by malicious adversaries in (t, n) threshold ECDSA schemes. Our proposed protocol improves the computational and communication costs from $O(t^3)$ to $O(t^2)$. We show the efficiency improvements of our proposed protocol by presenting implementation results.

Though our proposed protocol can be applied for other threshold cryptosystems, the security level that our protocol achieves is relatively weaker than those of recently proposed proactive secret sharing schemes: Our protocol has limitations that it does not achieve robustness and fairness. It does not consider the dynamic setting as well. As a future work, it may be interesting to improve the efficiency of currently existing proactive secret sharing schemes that achieve stronger security as well as to enhance the security level that threshold ECDSA schemes achieve.

APPENDIX A
GROUP SETUP PROTOCOL

Let n denote the overall number of parties in the system, $t-1$ denote the overall number of parties under the control of the adversary, and t be the size of each group running the key recovery protocol, except a party who lost his/her secret key.

The grouping protocol uses two hash functions F_1 and F_2 which will be considered as random oracles in the security proof. We provide a formal description of the protocol.

1) Each $\mathcal{P}_i$ chooses a random r_i and sends $F_1(\mathsf{ip}_i, \mathsf{pk}_i, r_i)$ to the public bulletin board (PBB) where ip_i and pk_i indicate $\mathcal{P}_i$'s IP address and public key, respectively.
2) After a short prefixed delay, each party queries the PBB for the table of parties which have registered.
3) Each party $\mathcal{P}_i$ computes $\alpha = F_2(F_1(\mathsf{ip}_1, \mathsf{pk}_1, r_1), \ldots, F_1(\mathsf{ip}_n, \mathsf{pk}_n, r_n))$ and parses the result α into chunks

of size $\log n$, denoted $\alpha_1, \ldots, \alpha_n$. Each party $\mathcal{P}_i$ is associated with α_i and the table is sorted by the α_i values.

4) Grouping is performed by taking a group of t parties according to the sorting. Namely, for some $i \xleftarrow{\$} \{1, 2, \ldots, \lfloor \frac{n}{t} \rfloor\}$, a group G is set to be the parties associated with the values $(\alpha_{n\cdot(i-1)+1}, \ldots, \alpha_{n\cdot i})$.
5) The PBB sends the IP addresses of the group members to the members of each group.
6) Members of each group send each other their IP address, public key and randomness that were used when registering with the PBB.
7) Each group member computes $F_1(\mathsf{ip}_j, \mathsf{pk}_j, r_j)$ for every party $\mathcal{P}_j$ in its group and verifies that it matches what was recorded by the center during registration. In addition, it verifies that it received the IP address of all parties that are in its group, by the computation of F_2. If no, then it sends abort to all the parties in its group.

According to the analysis in [44], the probability of a bad grouping is approximately $\left(\frac{t-1}{n}\right)^{t-2} \cdot n$. For more details of the bound, see Section 5.2.2 of [44].

REFERENCES

[1] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold DSS signatures," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1070, U. M. Maurer, Ed. Springer, 1996, pp. 354–371.

[2] P. D. MacKenzie and M. K. Reiter, "Two-party generation of DSA signatures," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 2139, J. Kilian, Ed. Santa Barbara, CA, USA: Springer, Aug. 2001, pp. 137–154.

[3] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust and efficient sharing of RSA functions," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 1109, N. Koblitz, Ed. Santa Barbara, CA, USA: Springer, Aug. 1996, pp. 157–172.

[4] D. R. Stinson and R. Strobl, "Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates," in *Proc. Inf. Secur. Privacy, 6th Australas. Conf.*, in Lecture Notes in Computer Science, vol. 2119, V. Varadharajan and Y. Mu, Eds. Sydney, NSW, Australia: Springer, Jul. 2001, pp. 417–434.

[5] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, Jan. 2007.

[6] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 435, G. Brassard, Ed. Santa Barbara, CA, USA: Springer, Aug. 1989, pp. 307–315.

[7] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[8] C.-P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, Jan. 1991.

[9] NIST, "Digital signature standard (DSS)," NIST, Gaithersburg, MD, USA, Tech. Rep. 169, Aug. 1991.

[10] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science), vol. 9696, M. Manulis, A. Sadeghi, and S. A. Schneider, Eds. Guildford, U.K.: Springer, Jun. 2016, pp. 156–174.

[11] Y. Lindell, "Fast secure two-party ECDSA signing," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 10402, J. Katz and H. Shacham, Eds. Santa Barbara, CA, USA: Springer, Aug. 2017, pp. 613–644.

[12] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. Toronto, ON, Canada: ACM, Oct. 2018, pp. 1179–1194.

[13] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. Toronto, ON, Canada: ACM, Oct. 2018, pp. 1837–1854.

[14] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Secure two-party threshold ECDSA from ECDSA assumptions," in *Proc. IEEE Symp. Secur. Privacy (SP)*. San Francisco, CA, USA: IEEE Computer Society, May 2018, pp. 980–997.

[15] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ECDSA from ECDSA assumptions: The multiparty case," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2019, pp. 1051–1066.

[16] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Two-party ECDSA from hash proof systems and efficient instantiations," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 11694, A. Boldyreva and D. Micciancio, Eds. Santa Barbara, CA, USA: Springer, Aug. 2019, pp. 191–221.

[17] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC non-interactive, proactive, threshold ECDSA with identifiable aborts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. Nov. 2020, pp. 1769–1787.

[18] H. Xue, M. H. Au, X. Xie, T. H. Yuen, and H. Cui, "Efficient online-friendly two-party ECDSA signature," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. Virtual Event, South Korea, Nov. 2021, pp. 558–573.

[19] Y. Deng, S. Ma, X. Zhang, H. Wang, X. Song, and X. Xie, "Promise $\sum$-protocol: How to construct efficient threshold ECDSA from encryptions based on class groups," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science), vol. 13093, M. Tibouchi and H. Wang, Eds. Singapore: Springer, Dec. 2021, pp. 557–586.

[20] I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter, and M. B. Østergaard, "Fast threshold ECDSA with honest majority," *J. Comput. Secur.*, vol. 30, no. 1, pp. 167–196, 2022.

[21] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks (extended abstract)," in *Proc. 10th Annu. ACM Symp. Princ. Distrib. Comput.*, L. Logrippo, Ed. Montreal, QC, Canada: ACM, Aug. 1991, pp. 51–59.

[22] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Proc. 15th Annu. Int. Cryptol. Conf.*, in Lecture Notes in Computer Science, vol. 963, D. Coppersmith, Ed. Santa Barbara, CA, USA: Springer, Aug. 1995, pp. 339–352.

[23] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive public key and signature systems," in *Proc. 4th ACM Conf. Comput. Commun. Secur. (CCS)*, R. Graveman, P. A. Janson, C. Neuman, and L. Gong, Eds. Zurich, Switzerland: ACM, Apr. 1997, pp. 100–110.

[24] J. Doerner. (2021). Open Source for mpecdsa. GitLab. Aug. 6, 2022. [Online]. Available: https://gitlab.com/neucrypt/mpecdsa/activity

[25] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1592, J. Stern, Ed. Prague, Czech Republic: Springer, May 1999, pp. 223–238.

[26] Y. Kondi, B. Magri, C. Orlandi, and O. Shlomovits, "Refresh when you wake up: Proactive threshold wallets with offline devices," in *Proc. 42nd IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2021, pp. 608–625.

[27] K. Bae, J. Park, and J. Ryou, "Secure recovery protocol of (1,3) distributed key share with trustless setup for asset management in blockchain," *J. Korea Inst. Inf. Secur. Cryptol.*, vol. 31, no. 5, pp. 863–874, 2021.

[28] V. Nikov and S. Nikova, "On proactive secret sharing schemes," in *Selected Areas in Cryptography* (Lecture Notes in Computer Science), vol. 3357, H. Handschuh and M. A. Hasan, Eds. Waterloo, ON, Canada: Springer, Aug. 2004, pp. 308–325.

[29] S. Dolev, K. E. Defrawy, J. Lampkins, R. Ostrovsky, and M. Yung, "Proactive secret sharing with a dishonest majority," in *Security and Cryptography for Networks* (Lecture Notes in Computer Science), vol. 9841, V. Zikas and R. D. Prisco, Eds. Amalfi, Italy: Springer, Aug./Sep. 2016, pp. 529–548.

[30] K. Eldefrawy, T. Lepoint, and A. Leroux, "Communication-efficient proactive MPC for dynamic groups with dishonest majorities," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science), vol. 13269, G. Ateniese and D. Venturi, Eds. Rome, Italy: Springer, Jun. 2022, pp. 565–584.
[31] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proc. 28th Annu. Symp. Found. Comput. Sci.* Los Angeles, CA, USA: IEEE Computer Society, Oct. 1987, pp. 427–437.
[32] T. P. Pedersen, "A threshold cryptosystem without a trusted party (extended abstract)," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 547, D. W. Davies, Ed. Brighton, U.K.: Springer, 1991, pp. 522–526.
[33] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 2045, B. Pfitzmann, Ed. Innsbruck, Austria: Springer, 2001, pp. 280–299.
[34] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Proc. 4th Int. Workshop Pract. Theory Public Key Cryptogr. (PKC)*, in Lecture Notes in Computer Science, vol. 1992, K. Kim, Ed. Cheju Island, South Korea: Springer, Feb. 2001, pp. 119–136.
[35] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. IT-31, no. 4, pp. 469–472, Jul. 1985.
[36] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, Apr. 1984.
[37] A. Menezes, P. V. Ooschot, and S. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1997.
[38] J. C. Benaloh and J. Leichter, "Generalized secret sharing and monotone functions," in *Proc. 8th Annu. Int. Cryptology Conf.*, in Lecture Notes in Computer Science, vol. 403, S. Goldwasser, Ed. Santa Barbara, CA, USA: Springer, Aug. 1988, pp. 27–35.
[39] E. F. Brickell and D. M. Davenport, "On the classification of ideal secret sharing schemes," *J. Cryptol.*, vol. 4, no. 2, pp. 123–134, 1991.
[40] M. Bertilsson and I. Ingemarsson, "A construction of practical secret sharing schemes using linear block codes," in *Proc. Workshop Theory Appl. Cryptograph. Techn.*, in Lecture Notes in Computer Science, vol. 718, J. Seberry and Y. Zheng, Eds. Gold Coast, QLD, Australia: Springer, Dec. 1992, pp. 67–79.
[41] M. Karchmer and A. Wigderson, "On span programs," in *Proc. 8th Annu. Struct. Complex. Theory Conf.* San Diego, CA, USA: IEEE Computer Society, May 1993, pp. 102–111.
[42] I. Damgård, "Commitment schemes and zero-knowledge protocols," in *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School* (Lecture Notes in Computer Science), vol. 1561, I. Damgård, Ed. Aarhus, Denmark: Springer, Jul. 1998, pp. 63–86.
[43] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[44] Y. Lindell and E. Waisbard, "Private web search with malicious adversaries," in *Privacy Enhancing Technologies* (Lecture Notes in Computer Science), vol. 6205, M. J. Atallah and N. J. Hopper, Eds. Berlin, Germany: Springer, Jul. 2010, pp. 220–235.
[45] *OpenSSL–Cryptography and SSL/TLS Toolkit, Version 1.1.1N.* Accessed: Apr. 7, 2022. [Online]. Available: https://www.openssl.org
[46] (2010). *SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0.* Accessed: Jul. 15, 2022. [Online]. Available: https://www.secg.org/sec2-v2.pdf

MYUNGSUN KIM received the B.S. degree in computer science and engineering from Sogang University, Seoul, South Korea, in 1994, the M.S. degree in computer science and engineering from Information and Communications University (ICU), Daejeon, in 2002, and the Ph.D. degree in mathematics from Seoul National University (SNU), Seoul, in 2012. He was working with the Department of Information Security, University of Suwon. He is currently an Associate Professor with the Department of Mathematics, Gachon University. His research interests include efficient constructions of cryptographic algorithms and their practical applications to real-world solutions.

SANGRAE CHO received the B.Eng. degree in computing from Imperial College London, in 1996, and the M.Sc. degree in information security from the Royal Holloway, University of London, in 1997. He started his career as a Researcher at LG Corporate Technology Institute, in 1997, and he worked at the Electronics and Telecommunications Research Institute (ETRI), South Korea, as a Security Researcher for more than 15 years. During that time, he has been actively involved in constructing a national PKI infrastructure project, until 2001. He is currently a Senior Researcher with the Authentication Research Team, ETRI. In 2004, he has done several projects relating to digital identity management, including SAML v2.0 and authentication technology based on fast identity online (FIDO) specifications.

SEONGBONG CHOI received the B.Sc. degree in computer science and engineering from Jeonbuk National University, Republic of Korea, in 2020, and the M.S. degree in computer science and engineering from Chung-Ang University, Republic of Korea, in 2022, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering. His research interest includes public key cryptography.

YOUNG-SEOB CHO received the Ph.D. degree in computer science from Inha University, South Korea. Since 1998, he has been working on research projects with the Electronics and Telecommunications Research Institute (ETRI) and continuously conducted numerous projects on national and international level. He is currently a Principal Researcher with the Information Security Research Division, ETRI. His current research interests include multiparty computation, blockchain identity management, and AI security.

SOOHYUNG KIM received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, South Korea, in 1996 and 1998, respectively, and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2016. He is currently a Project Leader with the Information Security Research Division, Electronics and Telecommunications Research Institute (ETRI), Daejeon. His research interests include biometrics, identity management, payment systems, and network and system security.

HYUNG TAE LEE received the B.Sc., M.Sc., and Ph.D. degrees in mathematics from Seoul National University, Republic of Korea, in 2006, 2008, and 2013, respectively. He was a Research Fellow with Nanyang Technological University, Singapore, and an Assistant Professor with Jeonbuk National University, Republic of Korea. He is currently an Assistant Professor with the School of Computer Science and Engineering, Chung-Ang University, Republic of Korea. His research interests include computational number theory, cryptography, and information security.