Experimental Analysis of the Recent Key Recovery Protocol with respect to Commitment Schemes

Seongbong Choi, Yongseok Son, Jeongyeup Paek, and Hyung Tae Lee
School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea
{welq2st, sysganda, jpaek, hyungtaelee}@cau.ac.kr

***Abstract*—Recently, Kim et al. proposed a key recovery protocol for (t, n)-threshold ECDSA schemes that enables a user who lost his secret share to recover with the aid of other t users among n users [1]. In their protocol, a commitment scheme was employed to commit to messages, but they provided implementation results of their protocol by employing the Feldman commitment scheme only. In this paper, we examine the efficiency of their protocol with respect to commitment schemes by implementing them. More precisely, we implement the protocol by employing hash-based and Pedersen commitment schemes each as well as the Feldman commitment scheme. Our experimental results show that the hash-based commitment scheme provides the most efficient protocol than other commitment schemes. For example, when $t = 3$ with 128-bit security, the protocol with the hash-based commitment requires 0.485 ms in total for all computations, while the protocols with Feldman and Pedersen commitment schemes take 7.713 ms and 15.228 ms in total, which improve by factors of about 15.90 and 31.40, respectively.**

***Index Terms*—Key recovery protocol, commitment schemes, hash-based commitment, Pedersen commitment, threshold ECDSA**

I. Introduction

A (t, n)-threshold ECDSA scheme allows to distribute secret shares of the signing key to n parties and generate a valid signature when at least t parties join the signing procedure. It enables the secret key owner to distribute and store its share securely. Recently, due to powerful application scenarios in Blockchain, there have been proposed various threshold ECDSA schemes [2]–[12] for the secure key management. However, despite its advantages, it remains a possible security issue in practice if secret shares are lost or malicious users capture other parties' secret shares.

To handle this issue, proactive threshold signature schemes were proposed [13], [14]. Unlike traditional threshold signature schemes, proactive threshold signature schemes introduce additional protocols, *refresh* and *recovery* protocols. On the one hand, in a key refresh protocol, secret shares of entire parties are updated to new values without changing the corresponding secret and public keys of a group. This feature makes the adversary harder to reconstruct the secret key. On the other hand, in a recovery protocol, it allows to re-generate a secret share of a user who lost it. To that end, t other parties jointly participate in the protocol. However, secret shares of users participating in the protocol are also updated to other values, so it causes computational overheads.

Recently, there were several studies for designing efficient key recovery protocols for threshold ECDSA schemes. In [15], Bae et al. presented a key recovery protocol for threshold ECDSA schemes, but their protocol focused on $(1, 3)$-threshold ECDSA schemes only. Furthermore, at the end of executing the recovery protocol, all of the shares are replaced with other values, thus it also occurs additional computational overheads.

Later, Kim et al. [1] proposed a key recovery protocol for (t, n)-threshold ECDSA schemes. Compared to other key recovery protocols, their protocol does not make computational overheads since it maintains secret shares of all users. In the protocol, users who join the protocol with their own secret shares to help the recovery of other user's secret share, first select random values and share them each other through the secure channel. In this process, they exploit a commitment scheme to commit to selected random values. Then, by following the pre-determined computation rule, they generate masking values for their secret shares so that those are cancelled in the end. Finally, the authors of [1] provided implementation results of their proposed protocol by employing the Feldman commitment scheme [16].

In this paper, we re-evaluate the efficiency of Kim et al.'s protocol by substituting the Feldman commitment scheme for two well-known commitment schemes in the implementation: One is the hash-based commitment scheme and the other is the Pedersen commitment scheme. We point out that the Feldman commitment scheme does not achieve the requirement of the commitment scheme. In Kim et al.'s protocol, it is assumed that the exploited commitment scheme satisfies binding and hiding properties. However, the Feldman commitment scheme cannot satisfy the hiding property. On the other hand, the hash-based construction and Pedersen commitment schemes satisfy binding and hiding properties. Our implementation results show that the hash-based commitment scheme provides the most efficient experimental result, compared to other two commitment schemes. For example, when $t = 3$ with 128-bit security, Kim et al.'s key recovery protocol with the hash-based commitment requires 0.485 ms in total for all computations, whereas the protocols with Feldman and Pedersen commitment schemes take 7.713 ms and 15.228 ms in total, which improve by factors of about 15.90 and 31.40, respectively.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A4A5034130), and also by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2023-RS-2022-00156353) supervised by the IITP (Institute for Information Communications Technology Planning Evaluation). H. T. Lee is the corresponding author.

Outline of the Paper. In Section II, we review a cryptographic assumption, definitions of commitment scheme, and its concrete instantiations that will be utilized throughout the paper. Section III recalls Kim et al.'s key recovery protocol. We provide experimental results of their key recovery protocol with respect to commitment schemes in Section IV.

II. Preliminaries

In this section, we briefly review commitment schemes and introduce cryptographic assumptions required to achieve their security properties.

Notations. Throughout the paper, $x \in_R X$ denotes that x is sampled from a set X uniformly at random. We denote by $a \leftarrow A$ that a is an outcome of algorithm A. The concatenation of two strings a and b is denoted by $a\|b$.

A. Cryptographic Assumptions

In this subsection, we first look at the formal definition of the discrete logarithm (DL) assumption which will be required to guarantee the security properties of Feldman and Pedersen commitment schemes. Then, we introduce a collision resistant property of hash functions required for hash-based commitment schemes.

Definition 1 (Discrete Logarithm Assumption)**.** *Let $\mathbb{G}$ be an additive cyclic group of prime order $q = q(\lambda)$ with the security parameter λ. Let P be a random generator of $\mathbb{G}$. The discrete logarithm problem (DLP) with respect to $(\mathbb{G}, P)$ is to find $a \in \mathbb{Z}_q$ such that $Q = aP$ where Q is an element of $\mathbb{G}$.*

We say that the discrete logarithm (DL) assumption holds if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$

$$\Pr[\mathcal{A}(P, Q) = a \mid a \in_R \mathbb{Z}_q \text{ and } Q = aP]$$

is negligible in the security parameter λ.

Definition 2 (Collision Resistance)**.** *A hash function $H : \{0,1\}^* \to \{0,1\}^\ell$ with $\ell = \ell(\lambda)$ for the security parameter λ is collision-resistant if for any PPT adversary $\mathcal{A}$*

$$\Pr[(x, x') \leftarrow \mathcal{A} \text{ such that } H(x) = H(x') \text{ and } x \neq x']$$

is negligible in λ.

B. Commitment Schemes

Now, we first provide the definition of commitment schemes and formally define two properties of the security for commitment schemes. Next, we review three commitment schemes, Feldman, Pedersen, and hash-based commitment schemes.

Definition 3 (Commitment scheme)**.** *A commitment scheme consists of three polynomial time algorithms (*KeyGen*,* Commit*,* Reveal*), which are defined as follows:*

- $ck \leftarrow \mathsf{KeyGen}(1^\lambda)$*: It takes a security parameter λ as an input and returns a commitment key ck.*
  Throughout the paper, we denote a message space, a randomness space, and a commitment space by $\mathcal{M}_{ck}$, $\mathcal{R}_{ck}$, and $\mathcal{C}_{ck}$, respectively, and it is assumed that those information are included in ck.
- $C \leftarrow \mathsf{Commit}(ck, u; r)$*: Given the commitment key ck, a value $u \in \mathcal{M}_{ck}$, and a randomness $r \in \mathcal{R}_{ck}$, it outputs a commitment $C \in \mathcal{C}_{ck}$.*
- $1/\perp \leftarrow \mathsf{Reveal}(ck, C, u; r)$*: Given the commitment key ck, the commitment $C \in \mathcal{C}_{ck}$, the value $u \in \mathcal{M}_{ck}$, and the randomness $r \in \mathcal{R}_{ck}$, it returns 1 indicating $C = C'$ or $\perp$ indicating $C \neq C'$ where $C' \leftarrow \mathsf{Commit}(ck, u; r)$.*

For simplicity, we omit ck and the randomness r if no confusions arise.

The commitment scheme should satisfy the following two properties.

Definition 4 (Hiding property)**.** *A commitment scheme is* hiding *if for all interactive algorithms $\mathcal{A}$, the following probability is negligible in the security parameter λ :*

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

Definition 5 (Binding property)**.** *A commitment scheme is* binding *if for all interactive algorithms $\mathcal{A}$, the following probability is negligible in the security parameter λ :*

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

In the above definitions for hiding and binding properties of commitment schemes, if an adversary $\mathcal{A}$ is restricted to be a PPT algorithm, then we say that the commitment scheme is computationally hiding/binding. On the other hand, if there is no restriction on $\mathcal{A}$, then we say that the commitment scheme is unconditionally hiding/binding.

Now, we review three main commitment schemes that will be utilized in our implementation.

Feldman Commitment Scheme [16]. The Feldman commitment scheme consists of three polynomial-time algorithms (Fel.KeyGen, Fel.Commit, Fel.Reveal).

- $ck \leftarrow \mathsf{Fel.KeyGen}(1^\lambda)$: Given a security parameter λ,
  1) Generate a cyclic group $\mathbb{G}$ of prime order $q = q(\lambda)$.
  2) Select a random generator P of $\mathbb{G}$.
  3) Output a commitment key $ck = (P, \mathbb{G}, q)$.
- $C \leftarrow \mathsf{Fel.Commit}(u)$: Given a message $u \in \mathbb{Z}_q$,
  1) Compute $C = uP$.
  2) Output a commitment C to a message u.
- $1/\perp \leftarrow \mathsf{Fel.Reveal}(C, u)$: Given a commitment C and a message u,
  1) Compute $C' = uP$.
  2) Check whether $C' = C$. If it holds, return 1. Otherwise, return $\perp$.

It is known that the Feldman commitment scheme is computationally binding under the DL assumption in $\mathbb{G}$, but does not achieve the hiding property.

Pedersen Commitment Scheme [17]. The Pedersen commitment scheme consists of three polynomial-time algorithms ($\mathsf{Ped.KeyGen}$, $\mathsf{Ped.Commit}$, $\mathsf{Ped.Reveal}$).

- $ck \leftarrow \mathsf{Ped.KeyGen}(1^\lambda)$: Given a security parameter λ,
  1) Generate a cyclic group $\mathbb{G}$ of prime order $q = q(\lambda)$.
  2) Select two random generators P and Q of $\mathbb{G}$.
  3) Output a commitment key $ck = (P, Q, \mathbb{G}, q)$.
- $C \leftarrow \mathsf{Ped.Commit}(u; r)$: Given a message $u \in \mathbb{Z}_q$,
  1) Select a randomness $r \in_R \mathbb{Z}_q$.
  2) Compute $C = uP + rQ$.
  3) Output a commitment C to a message u.
- $1/\perp \leftarrow \mathsf{Ped.Reveal}(C, u, r)$: Given a commitment C, a message u and a randomness r,
  1) Compute $C' = uP + rQ$.
  2) Check whether $C = C'$. If it holds, return 1. Otherwise, return $\perp$.

It is known that the Pedersen commitment scheme satisfies computationally binding and unconditionally hiding properties under the DL assumption in $\mathbb{G}$ and the assumption that the DL of Q to the base P is unknown.

Hash-Based Commitment Scheme. The hash-based commitment scheme consists of three polynomial-time algorithms ($\mathsf{Hash.KeyGen}$, $\mathsf{Hash.Commit}$, $\mathsf{Hash.Reveal}$).

- $ck \leftarrow \mathsf{Hash.KeyGen}(1^\lambda)$: Given a security parameter λ,
  1) Generate a cryptographic hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\ell$ for a parameter $\ell = \ell(\lambda)$.
  2) Output a commitment key $ck = \mathcal{H}$.
- $C \leftarrow \mathsf{Hash.Commit}(u; r)$: Given a message $u \in \{0,1\}^*$,
  1) Select a random $r \in_R \{0,1\}^\ell$.
  2) Compute $C = \mathcal{H}(u \| r)$.
  3) Output a commitment C to a message u.
- $1/\perp \leftarrow \mathsf{Hash.Reveal}(C, u, r)$: Given a commitment C, a message u, and a randomness r,
  1) Compute $C' = \mathcal{H}(u \| r)$.
  2) Check whether $C' = C$. If it holds, return 1. Otherwise, return $\perp$.

It is known that the hash-based commitment scheme is computationally hiding and computationally binding if the exploited hash function is collision-resistant.

III. Review of Kim et al.'s Key Recovery Protocol

In this section, we review a key recovery protocol for a (t, n)-threshold ECDSA, recently proposed by Kim et al. [1]. We assume that the party $\mathcal{P}_1$ lost his/her own secret share and other at least t parties join the protocol to recover $\mathcal{P}_1$'s secret share.

Kim et al.'s Key Recovery Protocol. Let G be the set of parties that participate in the protocol. To simplify the description, we assume that $G = \{2, 3, \ldots, t+1\}$. Kim et al.'s key recovery protocol consists of four steps as below.

1) For each $i \in G$, $\mathcal{P}_i$ performs as follows: For each $j \in G$ and $j \neq i$,
   a) Choose a random element $b_{ij} \in_R \mathbb{Z}_q^*$ and a randomness $r_{ij} \in_R \mathcal{R}_{ck}$.
   b) Compute B_{ij} by running $B_{ij} \leftarrow \mathsf{Commit}(b_{ij}; r_{ij})$.
   c) Send B_{ij} to $\mathcal{P}_j$.
2) For each $i \in G$, $\mathcal{P}_i$ sends (b_{ij}, r_{ij}) to $\mathcal{P}_j$ for revealing B_{ij}.
3) For each $i \in G$, $\mathcal{P}_i$ performs as follows: For each $j \in G$ and $j \neq i$,
   a) Run $\mathsf{Reveal}(B_{ij}, b_{ij}, r_{ij})$ to check whether B_{ij} is the commitment to b_{ij}. If the output is $\perp$, abort.
   b) Compute
   $$s_i = \ell_i^G(1) sk_i + \sum_{j \in G \setminus \{i\}} (b_{ij} - b_{ji}),$$
   where $\ell_i^G(1) = \prod_{j \in G \setminus \{i\}} \frac{1-j}{i-j}$.
   c) Send s_i to $\mathcal{P}_1$ through a secure channel.
4) $\mathcal{P}_1$ performs the following steps:
   a) Compute $sk_1' = \sum_{i \in G} s_i$.
   b) Check if
   $$sk_1' P = X_1.$$
   Abort if it does not hold.
   c) Output sk_1'.

We remark that Kim et al.'s protocol 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. Though the authors of [1] provided implementation results by employing the Feldman commitment scheme, it does not achieve the hiding property. So, we need to replace the commitment schemes by other candidates that satisfy hiding and binding properties.

IV. Performance Anaylsis of Kim et al.'s Protocol with respect to Commitment Schemes

In this section, we evaluate the efficiency of Kim et al.'s protocol for several commitment schemes under various parameter settings.

A. Experimental Environments

In our implementation, the source codes were written in C++ and compiled using the g++ 9.4.0 compiler. We used the OpenSSL library [18] for using the symmetric key encryption, AES-256-GCM, to establish private peer-to-peer channels, for implementing arithmetic of large numbers and elliptic curve operations in the protocols, and for using hash

TABLE I
OPENSSL SHA3 PERFORMANCE COMPARISON FOR 10,000 TIMES. (UNIT: MS)

Hash Funtions				
ℓ	SHA3-224	SHA3-256	SHA3-384	SHA3-512
64	4.362	4.120	4.043	4.551
128	4.501	4.222	6.982	6.919
256	8.023	7.563	10.615	13.577
512	12.226	11.942	14.653	22.013
1024	23.677	23.366	28.994	42.097
2048	44.009	46.257	57.124	81.108
4096	75.286	79.550	102.839	144.362

TABLE II
OPERATION TIMES ON CURVES FOR 1,000 TIMES. (UNIT: MS)

Curves				
Operation	SECP224K1	SECP256K1	SECP384R1	SECP521R1
Scalar Multiplication	266.417	301.582	692.660	115.173
Addition	1.133	1.185	1.767	2.198

functions, SHA3-224, 256, 384, and 512, to realize a hash-based commitment scheme. However, we do not implement data transmission on the network. Instead, we store and read them in memory once the 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 11 pro 64 bits.

B. Experimental Results

Before presenting our experimental results, we first investigate the performance of SHA3 hash functions and elliptic curve operations in Tables I and II, respectively, under various parameter settings. Table I shows the performance comparison of SHA3-224, 256, 384, and 512, provided by OpenSSL library [18]. In Table I, each hash function takes a randomly sampled string of ℓ-byte length as input and computes the hash digest. The numbers in the table are all running times of 10,000 tests for each parameter and all hash functions take the same string as input in each test. Table I shows that hash functions, which output longer hash digests, take less time to execute if the length ℓ of the string is less than or equal to 1,024 bytes. For example, when $\ell = 64$, SHA3-224 requires 4.362 ms while SHA3-256 and SHA3-384 require 4.120 ms and 4.043 ms, respectively. Furthermore, when $\ell = 128$, SHA3-384 takes 6.982 ms while SHA3-512 takes 6.919 ms. This affects our experimental results for Kim et al.'s key recovery protocol because elements from $\mathbb{Z}_q$ are smaller than 1,024 bytes.

Table II shows computation times of SECP224K1, SECP256K1, SECP384R1, and SECP521R1. The numbers in the table are all running times of 1,000 tests for each parameter. According to our experimental results, it shows an interesting feature that SECP521R1 is faster than other curves. This also affects our experimental results.

Now, we provide experimental results of Kim et al.'s key recovery protocol with Feldman, Pedersen, and hash-based commitment schemes for various parameter settings. Each experiment was conducted 100 times and the results in Table III and IV are averages of those running times. In Table III, we use SECP224K1, SECP256K1, SECP384R1, and SECP521R1 curves for 112, 128, 196, and 256 bits security, respectively, where SECP224K1 and SECP256K1 are specific Koblitz curves defined by Standards for Efficient Cryptography Group [19], while SECP384R1 and SECP521R1 are Weierstrass curves [20], to implement Feldman and Pedersen commitment schemes. We also exploit SHA3-224, 256, 384, and 512 [21] for 112, 128, 196, and 256 bits security, respectively, to realize the hash-based commitment scheme.

Table III demonstrates computational running times of Kim et al.'s key recovery protocol with respect to commitment schemes and security levels. It shows that the protocol with the hash-based commitment scheme outperforms that with other commitment schemes for all security levels since it does not require operations over elliptic curves, which are relatively expensive. For example, when $t = 1$ and the security level is 128, the protocol with the hash-based commitment scheme takes 0.358 ms in total for computations while the protocols with Feldman and Pedersen commitment schemes take 1.53 ms and 2.741 ms in total, which improve by factors of about 4.27 and 7.58, respectively. Compared with the Pedersen commitment scheme, the Feldman commitment scheme is about 2 times as efficient because it only requires one scalar multiplication while the Pedersen commitment scheme requires two scalar multiplications and one addition.

Table IV provides experimental results of Kim et al.'s key recovery protocol with commitment schemes for several t with

TABLE III
EXPERIMENTAL RESULTS OF KIM ET AL'S PROTOCOL FOR SMALL t AND VARIOUS SECURITY LEVELS AND COMMITMENT SCHEMES. (UNIT: MS)

t	Security level	Feldman [16]			Pedersen [17]			Hash function		
		Step 1)	Step 3)	Step 4)	Step 1)	Step 3)	Step 4)	Step 1)	Step 3)	Step 4)
1	112	0.553	0.547	0.265	1.226	1.216	0.299	0.012	0.022	0.276
	128	0.622	0.609	0.299	1.234	1.210	0.297	0.012	0.028	0.318
	192	1.483	1.461	0.707	2.775	2.745	0.678	0.015	0.041	0.717
	256	0.267	0.283	0.121	0.706	0.715	0.119	0.013	0.052	0.128
2	112	1.721	1.695	0.276	3.667	3.619	0.307	0.029	0.054	0.291
	128	1.909	1.858	0.303	3.603	3.577	0.295	0.027	0.069	0.322
	192	4.380	4.293	0.706	8.605	8.329	0.680	0.030	0.098	0.694
	256	0.770	0.797	0.119	2.139	2.127	0.121	0.031	0.142	0.135
3	112	3.348	3.316	0.278	7.771	7.653	0.315	0.052	0.098	0.281
	128	3.705	3.698	0.310	7.302	7.623	0.303	0.047	0.126	0.312
	192	8.640	8.417	0.685	17.004	16.915	0.692	0.053	0.192	0.697
	256	1.554	1.620	0.123	4.272	4.295	0.124	0.057	0.269	0.131

TABLE IV
EXPERIMENTAL RESULTS OF KIM ET AL.'S PROTOCOL FOR VARIOUS t AT 128-BIT SECURITY (UNIT: MS)

128-bit security (SECP256K1, SHA3-256)									
t	Feldman			Pedersen			Hash function		
	Step 1)	Step 3)	Step 4)	Step 1)	Step 3)	Step 4)	Step 1)	Step 3)	Step 4)
3	3.705	3.698	0.310	7.302	7.623	0.303	0.047	0.126	0.312
6	13.100	13.111	0.312	25.778	25.724	0.304	0.158	0.435	0.327
9	28.134	27.766	0.319	54.494	43.190	0.313	0.330	0.907	0.323
12	49.049	48.211	0.313	95.378	93.660	0.309	0.559	1.629	0.345

128-bit security. In Table IV, the hash-based commitment scheme gives the highest performance for the same reason as aforementioned. For example, when $t = 12$, the protocol with the hash-based commitment scheme requires 2.533 ms in total for computation, while the protocols with Feldman and Pedersen commitment schemes require 97.573 ms and 189.347 ms in total, which improve by factors of about 38.52 and 74.75, respectively.

V. CONCLUSION

In this paper, we provided the performance analysis of a recent key recovery protocol proposed by Kim et al., with respect to commitment schemes. More specifically, we compare the efficiency of the protocols with Feldman, Pedersen, and hash-based commitment schemes. From our experimental results, we confirm that the protocol with the hash-based commitment scheme provides the most efficient result.

REFERENCES

[1] M. Kim, S. Cho, S. Choi, Y.-S. Cho, S. Kim, and H. T. Lee, "A key recovery protocol for multiparty threshold ecdsa schemes," *IEEE Access*, vol. 10, pp. 133 206–133 218, 2022.

[2] 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 - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, ser. Lecture Notes in Computer Science, M. Manulis, A. Sadeghi, and S. A. Schneider, Eds., vol. 9696. Springer, 2016, pp. 156–174.

[3] Y. Lindell, "Fast secure two-party ECDSA signing," in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. Katz and H. Shacham, Eds., vol. 10402. Springer, 2017, pp. 613–644.

[4] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 1179–1194.

[5] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 1837–1854.

[6] J. Doerner, Y. Kondi, E. Lee, and A. shelat, "Secure two-party threshold ECDSA from ECDSA assumptions," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 980–997.

[7] ——, "Threshold ECDSA from ECDSA assumptions: The multiparty case," in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1051–1066.

[8] 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 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, ser. Lecture Notes in Computer Science, A. Boldyreva and D. Micciancio, Eds., vol. 11694. Springer, 2019, pp. 191–221.

[9] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC non-interactive, proactive, threshold ECDSA with identifiable aborts," in *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*,

J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM, 2020, pp. 1769–1787.

[10] H. Xue, M. H. Au, X. Xie, T. H. Yuen, and H. Cui, "Efficient online-friendly two-party ECDSA signature," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 558–573.

[11] 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 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds., vol. 13093. Springer, 2021, pp. 557–586.

[12] 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.

[13] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, ser. Lecture Notes in Computer Science, D. Coppersmith, Ed., vol. 963. Springer, 1995, pp. 339–352.

[14] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive public key and signature systems," in *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997*, R. Graveman, P. A. Janson, C. Neuman, and L. Gong, Eds. ACM, 1997, pp. 100–110.

[15] K. Bae, J. Park, and J. Ryou, "Secure recovery protocol of (1,3) distributed key share with trustless setup for asset management in blockchain," *Journal of the Korea Institute of Information Security & Cryptology*, vol. 31, no. 5, pp. 863–874, 2021.

[16] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*. IEEE Computer Society, 1987, pp. 427–437.

[17] T. P. Pedersen, "A threshold cryptosystem without a trusted party (extended abstract)," in *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, ser. Lecture Notes in Computer Science, D. W. Davies, Ed., vol. 547. Springer, 1991, pp. 522–526.

[18] "OpenSSL–Cryptography and SSL/TLS Toolkit, Version 1.1.1s," Available at https://www.openssl.org, 2022, online; Accessed 1 Nov 2022.

[19] "SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0," Available at https://www.secg.org/sec2-v2.pdf, 2010, online; Accessed 15 Jul 2022.

[20] "Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters," Available at https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186.pdf, 2023, online; Accessed 16 Aug 2023.

[21] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," Available at https://doi.org/10.6028/NIST.FIPS.202, 2015.