More Efficient Two-Round Multi-Signature Scheme with Provably Secure Parameters for Standardized Elliptic Curves *

Kaoru Takemure[1,2], Yusuke Sakai[2], Bagus Santoso[1], Goichiro Hanaoka[2], and Kazuo Ohta[1,2]

[1]The National Institute of Advanced Industrial Science and Technology
[2]The University of Electro-Communications
kaoru.takemure@gmail.com

Abstract

The existing discrete-logarithm-based two-round multi-signature schemes without using the idealized model, i.e., the Algebraic Group Model (AGM), have quite large reduction loss. This means that an implementation of these schemes requires an elliptic curve (EC) with a very large order for the standard 128-bit security when we consider concrete security. Indeed, the existing standardized ECs have orders too small to ensure 128-bit security of such schemes. Recently, Pan and Wagner proposed two two-round schemes based on the Decisional Diffie-Hellman (DDH) assumption (EUROCRYPT 2023). For 128-bit security in concrete security, the first scheme can use the NIST-standardized EC P-256 and the second can use P-384. However, with these parameter choices, they do not improve the signature size and the communication complexity over the existing non-tight schemes. Therefore, there is no two-round scheme that (i) can use a standardized EC for 128-bit security and (ii) has high efficiency.

In this paper, we construct a two-round multi-signature scheme achieving both of them from the DDH assumption. We prove that an EC with at least a 321-bit order is sufficient for our scheme to ensure 128-bit security. Thus, we can use the NIST-standardized EC P-384 for 128-bit security. Moreover, the signature size and the communication complexity per one signer of our proposed scheme under P-384 are 1152 bits and 1535 bits, respectively. These are most efficient among the existing two-round schemes without using the AGM including Pan-Wagner's schemes and non-tight schemes which do not use the AGM. Our experiment on an ordinary machine shows that for signing and verification, each can be completed in about 65 ms under 100 signers. This shows that our scheme has sufficiently reasonable running time in practice.

1 Introduction

In a *multi-signature* scheme [1], for a single common message m, multiple parties cooperatively generate a signature, known as a multi-signature, which is basically a combination of multiple individual signatures on m where each is created by each party using its own signing key. An essential property of the multi-signature is that its size is kept constant independently of the number of parties. Multi-signature schemes based on several hardness problems are proposed so far, e.g., the discrete logarithm (DL)-based schemes [2, 3, 4, 5, 6, 7, 8, 9, 10], pairing-based schemes [11, 12, 13, 14, 15], and lattice-based schemes [16, 17, 18, 19, 20].

In this research, we focus on DL-based multi-signature schemes which can be implemented under the elliptic curves used to implement standard digital signature schemes, e.g., the ECDSA [21] and the Schnorr signature scheme [22], used in cryptocurrencies, e.g., Bitcoin.

For multi-signatures, the primary desirable features are the followings. The first one is the security in the plain public-key (PPK) model, which allows an adversary to make cosigners' public keys maliciously without knowing the signing keys. The second is key aggregation property, which allows aggregating a set of public keys into a single short key when signatures are verified. The third is a small number of rounds of communication for signing. Moreover, a scheme achieving a small signature size (independent from the number of signers) while achieving these three properties is desirable.

For DL-based multi-signature schemes, Bellare and Neven proposed the first *three-round* scheme, which is the scheme with *three* rounds of communication for signing, proven secure in the PPK model [3]. Moreover, Maxwell et al. proposed the first three-round scheme with key aggregation [4]. In recent years, several *two-round* schemes, achieving security in the PPK model and the support of key aggregation, are proposed [5, 6, 7, 8, 9, 10, 23, 24].

1.1 Importance of Concrete Security for Parameter Choice

Theoretically, a security proof of a cryptosystem consists of a reduction from solving some computational problem to breaking the cryptosystem under a defined adversarial model. From the security proof, usually, we can derive a relation between $T_A = t_A/\varepsilon_A$ and $T_P = t_P/\varepsilon_P$, where t_A and ε_A are the adversary's running time and success probability for breaking the cryptosystem and t_P and ε_P are the algorithm's running time and success probability for solving the computational problem. A typical relation between T_P and T_A is as follows: $T_A \geq T_P/\Phi$, where Φ is often referred to as the *reduction loss*.

When we derive the size of parameters, e.g., an order of the underlying group in a DL-based scheme, for guaranteeing the security of the cryptosystem in practice, Φ was often disregarded. However, this disregard sometimes makes schemes vulnerable. An example of this vulnerability is shown by the recent work of Kales and Zaverucha [25] which demonstrated an attack on the MQDSS signature scheme [26]. Their attack exploits the fact that the parameter of MQDSS was derived without considering Φ. Therefore, it is important to derive the size of parameters by considering Φ based on the security proof, and thus we should implement cryptosystems with *provable secure parameters*.

A small Φ is preferable in practice because it does not let the problem mentioned above occur in the first place. Also, if Φ is large, we need to ensure that T_P is sufficiently large so that the derived lower bound of T_A, i.e., T_P/Φ, is not too small to have a practical meaning. Usually, the only way to make T_P larger is by setting larger parameters, which means higher costs for implementation in practice. If Φ is a relatively small constant value independent of the parameters of the cryptosystem and the adversary, we say that the security proof is *tight*.

Also, it is difficult for a scheme with a large Φ to use the standardized cryptographic tools. Specifically, for the DL-based schemes, a elliptic curve (EC) with a 256-bit prime order is required to guarantee 128-bit security, but it is only applied to tightly secure schemes. The scheme with a large Φ requires an EC with an order larger than 256-bit. We have standardized ECs with such an order, e.g., NIST P-384 and P-521. However, for the scheme with very large Φ, such ECs are not sufficient for 128-bit security, and then, we need to design a new desireable EC. This makes the implementation difficult and less reliable.

For schemes with a small Φ, it is easy to use the standardized tools because we do not have to care about the above-mentioned problems. Specifically, for the DL-based schemes, we can use the standardized ECs, e.g., NIST P-256 and Secp256k1, for 128-bit security. Also if Φ is small, even though Φ is not constant, we may be able to avoid the inconvenient situation where there is no suitable standardized tool.

1.2 Concrete Security of Existing DL-Based Multi-Signature Schemes

Here, we review the existing DL-based two-round multi-signature schemes in terms of the tightness of a reduction.

Most of them can be categorized into two types: The first type is the schemes with a non-tight reduction (namely, having a large reduction loss) and the second type is the schemes with a tight reduction. The

[1]For MuSig2, ν is a unique parameter. In this comparison, mBCJ is a variant that is secure in the plain public-key model, not the original one proposed in [5].

Table 1: Detailed performance comparison among two-round multi-signature schemes.[1]

Scheme	Assumption	Model	Loss	Signature Size	Communication Complexity	Pub. Key Size	$\|q\|_{128}$ (bit) Curve	$\|\widetilde{\sigma}\|_{128}$ (bit) $\|\widetilde{\sigma}\|_{\mathrm{EC}}$ (bit)	$\|\mathrm{CC}\|_{128}$ (bit) $\|\mathrm{CC}\|_{\mathrm{EC}}$ (bit)	Key Agg.
MuSig2 ($\nu = 2$) [7]	AOMDL	AGM, ROM	$O(1)$	$\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	$2\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	$\|\mathbb{G}\|$	257 P-256	515 513	773 770	Yes
DWMS [8]	OMDL		$O(1)$	$\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	$2\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	$\|\mathbb{G}\|$	257 P-256	515 513	773 770	
HBMS-AGM [9]	DL		$O(1)$	$\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$\|\mathbb{G}\|$	257 P-256	772 769	772 769	
LK [10]	DL	AGM, NPROM	$O(1)$	$3\|\mathbb{Z}_q\|$	$\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$\|\mathbb{G}\|$	258 P-256	774 768	775 769	
MuSig-DN [6]	DL, DDH, PRNG zk-SNARKs, PRF	ROM	$O(q_H^3/\varepsilon^3)$	$\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	$2\|\mathbb{G}\| + \|\mathbb{Z}_q\| + \|\pi\|$	$\|\mathbb{G}\|$	740 Not Exist	1481 -	- -	Yes
MuSig2 ($\nu \geq 4$) [7]	AOMDL		$O(q_H^3/\varepsilon^3)$	$\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	$\nu\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	$\|\mathbb{G}\|$	750 Not Exist	1501 -	3754 -	
HBMS [9]	DL		$O(q_S^4 q_H^3/\varepsilon^3)$	$\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$\|\mathbb{G}\|$	986 Not Exist	2959 -	2959 -	
TZ [24]	DL		$O(q_H^3/\varepsilon^3)$	$\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$4\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$\|\mathbb{G}\|$	742 Not Exist	2227 -	4456 -	
mBCJ [5]	DL		$O(q_S^2 q_H/\varepsilon)$	$\|\mathbb{G}\| + 3\|\mathbb{Z}_q\|$	$2\|\mathbb{G}\| + 3\|\mathbb{Z}_q\|$	$\|\mathbb{G}\|$	544 Not Exist	2177 -	2722 -	No
PW-1 [23]	DDH	ROM	$O(1)$	$6\|\mathbb{G}\| + 8\|\mathbb{Z}_q\| + N$	$6\|\mathbb{G}\| + 8\|\mathbb{Z}_q\| + 1$	$2\|\mathbb{G}\|$	260 P-256	3646+N 3590+N	3647 3591	No
PW-2 [23]	DDH		$O(q_S)$	$5\|\mathbb{Z}_q\|$	$3\|\mathbb{G}\| + 4\|\mathbb{Z}_q\|$	$2\|\mathbb{G}\|$	322 P-384	1610 1920	2257 2691	Yes
Ours	DDH	ROM	$O(q_S)$	$3\|\mathbb{Z}_q\|$	$2\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$2\|\mathbb{G}\|$	321 P-384	963 1152	1286 1538	Yes

* Column 2 shows the security assumptions. Column 3 shows whether idealized models are used for a cyclic group and hash functions. Column 4 shows the reduction loss. Columns 5, 6 and 7 show the size of a multi-signature, elements sent in the signing protocol per a signer, and a public key, respectively. Column 8 shows the required underlying group size $|q|_{128}$ and the NIST standardized EC that enables a parameter choice with 128-bit security, which is called the recommended EC hereafter. Column 9 shows the signature sizes $|\widetilde{\sigma}|_{128}$ and $|\widetilde{\sigma}|_{\mathrm{EC}}$ under the $|q|_{128}$-bit EC group and the recommended EC, respectively. Column 10 shows the communication complexities $|\mathrm{CC}|_{128}$ and $|\mathrm{CC}|_{\mathrm{EC}}$ under the $|q|_{128}$-bit EC group and the recommended EC. Column 11 shows whether each scheme allows key aggregation. $\mathbb{G}$ and $\mathbb{Z}_q$ indicate the underlying group $\mathbb{G}$ of a prime order q and the ring of integers modulo q, respectively. We assume that the sizes of $|\mathbb{G}|$ and $|\mathbb{Z}_q|$ over a q-bit EC are $q+1$ and q bits, respectively. ROM and NPROM indicate the random oracle model and the non-programmable random oracle model. q_H and q_S indicate the number of random oracle queries and signing oracle queries, respectively. ε indicates the advantage of an adversary against the scheme. N indicates the number of signers. $|\pi|$ is the size of the zk-SNARK proof. For MuSig-DN, we write "-" in Column 10 because the size of $|\pi|$ considering concrete security is explicitly unknown.

schemes of the first type include MuSig-DN [6], MuSig2 ($\nu \geq 4$) [7], HBMS [9], TZ [24], and mBCJ [5], while the schemes of the second type include MuSig2 ($\nu = 2$) [7], DWMS [8], HBMS-AGM [9], LK [10].

When taking tightness into consideration, even for 128-bit security, the first-type schemes require an elliptic curve (EC) with an very large order. Table 1 shows how large the order of the group should be in order to provably ensure 128-bit security. As Table 1 shows, MuSig-DN, MuSig2 ($\nu \geq 4$), HBMS, TZ [24], and mBCJ, respectively, require 740-bit, 750-bit, 986-bit, 742-bit, and 544-bit groups. Importantly, these schemes no longer have standardized curves that provably ensure 128-bit security.

The cause of the large reduction losses of these schemes is that to prove the security based on the DL assumption, the reduction performs the *rewinding* of the adversary, like the security proof of the Schnorr signature scheme. Moreover, in schemes with key aggregation, the number of rewindings has to be increased. Thus, these schemes have larger reduction losses than that of the Schnorr signature scheme.

The schemes of the second type achieve tight security by using the Algebraic Group Model (AGM) [27], which is a very idealized model of computation. The schemes allow us to use an EC of a small order, e.g., 256-bit. However, the reliance on the AGM needs more care because recent research [28, 29] shows that the reliability of the AGM is still not well-understood. Indeed, Zhandry showed the one-time message authentication code that is secure in the AGM but insecure in the standard model [28].

The random oracle model is also an idealized model of hash functions, but the situation is rather different from the AGM. Much cryptanalytic literature investigates the possibility of distinguishing a concrete hash function from a random oracle by finding a dedicated input-output correlation beyond (target) collision resistance or preimage resistance [30, 31, 32, 33]. These lines of research provide a more fine-grained understanding of how far (or near) concrete hash functions are from a random oracle.

Recently, Pan and Wagner proposed two two-round multi-signature schemes which can guarantee 128-bit security under standardized ECs. The first scheme PW-1 achieves tight security but does not support key

aggregation. The second scheme PW-2 has a small but non-tight reduction loss, e.g., $O(q_S)$ where q_S is the number of the signing queries. The second scheme is more efficient than the first one and supports key aggregation. Both are proven secure under the decisional Diffie-Hellman (DDH) assumption in the random oracle model.

However, under provably secure parameters, the two schemes do not improve the signature size and the communication complexity over the existing non-tight secure schemes even though those achieve tight security or a small reduction loss. Indeed, as shown in Table 1, the signature size of PW-1 is largest among the existing schemes without using AGM and the size of PW-2 is larger than theirs except for HBMS. Therefore, there is no scheme without using the AGM that (i) can use a standardized EC for 128-bit security and (ii) has high efficiency.

1.3 Our Contribution

In this paper, we propose a two-round multi-signature scheme that achieves (i) and (ii) mentioned above. We construct it from the DDH assumption and the random oracle model without using the AGM. This scheme guarantees 128-bit security under a standardized EC, e.g., NIST P-384. The signature size and the communication complexity under provable secure parameters are the most efficient among the existing two-round schemes without using the AGM. Moreover, our scheme is proven secure in the PPK model and supports key aggregation.

To achieve a scheme with the following properties, two rounds of communication for signing and a small reduction loss, we base our scheme on the Katz-Wang signature scheme [34] by applying the technique of HBMS. HBMS uses a certain tool like a homomorphic equivocal commitment scheme to achieve a two-round signing protocol. However, this tool is specialized to use together with the Schnorr identification scheme. Therefore, we cannot apply it to our case. To overcome this, we introduce a new technique to tailor a certain tool for use with the Katz-Wang DDH-based signature scheme.

Our scheme has a small reduction loss $O(q_S)$ where q_S is the number of signing queries of a forger. As the result of the estimation of provable secure parameters, it only needs an EC with at least 321-bit order to ensure 128-bit security.[2] Therefore, the curve P-384 is sufficient. Under P-384, the signature size is 1152 bits and the communication complexity per one signer is 1538 bits. These values achieve the shortest size among the existing schemes without using the AGM. Below, we compare our scheme with the existing scheme in detail.

Firstly, we compare our scheme with the existing non-tight schemes without using the AGM, e.g., MuSig-DN, MuSig2 ($\nu \geq 4$), HBMS, TZ, and mBCJ. Our signature size is reduced by more than 22%, 23%, 60%, 45%, and 47%, respectively. Compared to MuSig2 ($\nu \geq 4$), HBMS, TZ, and mBCJ, our communication complexity is reduced by more than 59%, 48%, 65%, and 43%, respectively. On the other hand, while the public key of our scheme consists of two group elements, theirs consist of only one group element. However, the public key size of our scheme under P-384 is 770 bits. This size is almost the same as theirs except for mBCJ. While the key size of mBCJ is smaller than ours, it does not support key aggregation. Therefore, we conclude that our scheme is more efficient compared to the existing non-tight schemes when we consider concrete security.

Secondly, we compare ours with PW-1 and PW-2. The signature size of our scheme is reduced by more than 67% and 40% , respectively. The communication complexity of our scheme is also reduced by more than 57% and 41%. The public keys of them are two group elements. The public key size of PW-1 is 514 bits but this scheme does not support key aggregation. The key size of PW-2 is 770 bits as same as ours. Thus, we also conclude that our scheme is more efficient than Pan-Wagner's schemes.

We implement our scheme on an ordinary machine and measure the running time of our implementation. We set the number of signers $N = 3$, 5, 10, and 15, as typical numbers of signers in a real-world Multi-Sig Wallet, and $N = 50$ and 100 as large-scale settings. For more details of the setting and the environment, see Section 6. Both the running time of the signing protocol and that of the verification under $N = 15$ are less than 10 ms. For large-scale settings, both the running time of the signing protocol and that of the

[2] We explain the way to estimate provable secure parameters in Section 5.1.

verification are about 30 ms under $N = 50$, and those are about 65 ms under $N = 100$. Moreover, since our proposed scheme also supports key aggregation, by precomputing a aggregated key, both the running time of signing and that of verification can be shortened to less than 2 ms irrelevantly to N. Thus, we can conclude that our scheme has a realistic running time in practice.

1.4 Related Works

Bellare and Neven proposed the first Schnorr-based three-round multi-signature scheme [3] which is secure in the PPK model. In the document [35], they also proposed a DDH-based scheme which is built from the Katz-Wang signature scheme. Maxwell et al. proposed a variant of the Bellare-Neven scheme that supports key aggregation [4]. Fukumitsu and Hasegawa proposed a DDH-based scheme with key aggregation [36].

Drijvers et al. proposed a secure Schnorr-based two-round multi-signature scheme mBCJ [5]. They constructed this scheme by applying a patch to the insecure two-round scheme BCJ [37] which uses a homomorphic (special) equivocal commitment scheme. Bellare and Dai proposed an improvement of mBCJ as HBMS [9]. Lee and Kim proposed a two-round scheme LK [10] based on HBMS and the Okamoto identification scheme [38]. The security of these schemes is proven under the DL assumption. MuSig-DN [6] is a two-round multi-signature scheme with a different approach from the above schemes. Specifically, this scheme achieves a two-round signing protocol by using a pseudorandom functions (PRF), a pseudorandom number generators (PRNG), and a succinct non-interactive arguments of knowledge (SNARKs) [39] to make the signing protocol deterministic. MuSig2 [7] and DWMS [8] are also two-round schemes with different approaches from the above-mentioned schemes. They are proven secure under the algebraic one-more DL (AOMDL) and one-more DL (OMDL) assumptions, respectively. Tessaro and Zhu proposed a two-round scheme [24]. This scheme is similar to MuSig2 but is proven secure under the DL assumption.

1.5 Concurrent Work

In concurrent and independent work, Pan and Wagner proposed two two-round multi-signature schemes based on the DDH assumption [23]. The first scheme PW-1 achieves tight security but key aggregation is not supported. The second scheme PW-2 supports key aggregation and achieves a small but non-tight reduction loss. As shown in Table 1, our scheme has a security assumption, a reduction loss, and a standardized EC for 128-bit security which are similar to those of PW-2. However, ours achieves a more efficient signature size and communication complexity than PW-2. Below, we briefly explain the cause of this improvement.

Whereas Pan and Wagner dedicated the effort to present their construction in a generic and modular way, we trade genericness and modularity for more efficiency. Our improvement is a benefit of this. PW-2 is an instantiation of their generic construction. PW-2 is constructed by combining a special commitment scheme and other building blocks in a generic manner. The scheme requires the binding property of the commitment scheme for proving the unforgeability. On the other hand, we construct our scheme in a specific way to be able to directly prove the unforgeability without using the binding property of the commitment scheme. In short, in our scheme, the binding property is not a necessary condition. Thus, the commitment scheme no longer needs to have the binding property. Then, we can reduce the size of the commitment key, the commitment, and the decommitment. This gives a smaller signature size and communication complexity. However, our scheme cannot be captured by Pan-Wagner's generic construction because the commitment scheme used in ours deviates from their syntax of the special commitment scheme.

2 Technical Overview

Our goal is to construct a two-round multi-signature scheme with a small reduction loss. Our technique is based on a tightly secure DDH-based variant of the Schnorr signature scheme (i.e., a DDH-based lossy identification). Before describing our techniques in detail, we explain the difficulty to construct a two-round multi-signature scheme from the basic Schnorr signature scheme in Section 2.1. Next, in order to explain the idea of our technique to construct a two-round multi-signature scheme from a tightly secure signature

scheme, first we review the DDH-based lossy identification in Section 2.2. And then, we explain in detail the difficulties we face if we only naively combine already existing techniques in Section 2.3. Finally, we explain our solutions to overcome those difficulties in Section 2.4.

2.1 Existing Non-Tight Secure Two-Round Schemes

Here, we explain the difficulty to construct a two-round multi-signature scheme from the Schnorr signature scheme. Schnorr signatures seem possible to be aggregated by using linearity. However, we cannot do that easily because a hash function used to sign does not have linearity. The well-known approach for this obstacle is to generate a multi-signature interactively as follows. Firstly each signer broadcasts a commitment $R_i \in \mathbb{G}$ of the Schnorr protocol and computes $\widetilde{R} \leftarrow \sum_i R_i$ where $\mathbb{G}$ is an additive cyclic group of a prime order q. Each signer computes a challenge $c_i \in \mathbb{Z}_q$ of the Schnorr protocol by the random oracle $H_{\mathrm{c}}(\widetilde{R}, pk_i, m)$, generates a response $s_i \in \mathbb{Z}_q$ of the Schnorr protocol, and sends it to all the cosigners where pk_i is a public key corresponding to the signer i and m is a message to be signed. Finally, each signer computes $\tilde{s} \leftarrow \sum_i s_i$ and outputs $(\widetilde{R}, \tilde{s})$ as a multi-signature on m. The verification equation is $\widetilde{R} = \tilde{s}G - \sum_i c_i \cdot pk_i$ where G is a generator of $\mathbb{G}$. Unfortunately, this two-round multi-signature scheme is insecure. In this case, honest verifier zero-knowledge does not work because the reduction needs to return R to a forger before deciding c in the random oracle. There are attacks [40, 5] against this multi-signature scheme.

As a solution to the above problem, Drijvers et al. proposed the secure two-round multi-signature scheme mBCJ by combining the Schnorr protocol and a homomorphic (special) equivocal commitment scheme. In a nutshell, all signers broadcast their homomorphic commitment T to R in the first round, and they broadcast their decommitment d and response s in the second round. The commitment key is generated *by the random oracle on input a message* m, e.g., $H_{\mathrm{ck}}(m)$. Thus, in the security proof, the reduction can embed either a binding commitment key or an equivocal commitment key into the random oracle $H_{\mathrm{ck}}(m)$. The reduction can simulate the honest signer without the secret key exploiting (special) equivocability if the commitment keys corresponding to queried messages are equivocal keys. It can also extract the secret key of the honest signer due to the binding property and the special soundness of the Schnorr protocol if the commitment key corresponding to the forgery is a binding key.

Bellare and Dai proposed a more efficient DL-based two-round multi-signature scheme HBMS than mBCJ by using a tool like the Pedersen commitment [41] instead of the homomorphic equivocal commitment scheme.

2.2 DDH-Based Lossy Identification

As in a well-known approach to achieve tight security of (standard) signature schemes, we attempts to construct it from the Katz-Wang (standard) signature scheme [34] which employs a DDH-based lossy identification.

Here, we review the DDH-based lossy identification. The secret key is $x \in \mathbb{Z}_q$ and the public key is $(Y, Z) = x(G, H)^T$, which is a Diffie-Hellman (DH) tuple. The identification protocol is as follows. At first, the prover generates and sends $(R_1, R_2)^T = r(G, H)^T \in \mathbb{G}^2$ to the verifier where r is uniformly chosen from $\mathbb{Z}_q$. Next, the verifier uniformly chooses $c \xleftarrow{\$} \mathbb{Z}_q$ and sends it to the prover. After that, the prover computes $s \leftarrow r + cx \mod q$ and sends it to the verifier. Finally, the verifier checks $R = s(G, H)^T - c(Y, Z)^T$.

The soundness is proven under the DDH assumption as follows: First, we prove that impersonation is *statistically* hard under the *lossy key* which is a non-DH tuple, i.e., there exists no $x \in \mathbb{Z}_q$ s.t. $(Y, Z) = x(G, H)^T$. Namely, under the lossy key, even for a computationally unbounded adversary, the success probability of an adversary is negligible.[3] Next, assume that there is an adversary that can perform impersonation with a non-negligible probability under the real public key, i.e., a DH tuple. Notice that this assumption induces a non-negligible gap between the adversary's success probability under a lossy key and that under a real

[3]Indeed, the probability of an adversary outputting s s.t. $R = s(G, H)^T - c(Y, Z)^T$ after the verifier uniformly chooses c is at most $1/q$ independently of the behavior of the adversary because (s, c) is uniquely determined according to R when (G, H) and (Y, Z) are linearly independent.

public key. Based on this, we can construct an algorithm solving the DDH problem by internally running (without rewinding) the adversary given an instance of the DDH problem as the public key.

2.3 Naive Approach and Difficulty

To construct a two-round scheme with a small reduction loss, we attempt to combine the technique of HBMS and the DDH-based lossy identification. In the signing protocol of HBMS, for a commitment key $ck \in \mathbb{G}$, each signer generates a commitment T by $T \leftarrow d \cdot ck + R$, where we consider an additive cyclic group and d is a randomness for the commitment. Then, the verification equation is $T = d \cdot ck + s \cdot G - c \cdot pk$.[4] Note that one having the discrete logarithm of ck can extract the secret key from two forgeries $(T, c, (d, s))$ and $(T', c', (d', s'))$ s.t. $T = T'$ and $c \neq c'$ like the special soundness. Then, the binding property is no longer needed. Our observation means that we can replace the commitment scheme in mBCJ with a simpler and more efficient tool that has equivocability[5] and the above property like the special soundness.

We require a tool that has similar properties to HBMS to achieve our goal. More concretely, in our case, a tool needs to have the following two types of commitment keys. The first type **Type-1** ensures that forgery is statistically hard under this commitment key and a lossy key like the lossy identification. The second type **Type-2** has (special) equivocability.

We need to newly construct such a tool tailored to the DDH-based lossy identification because we cannot reuse the tool used in HBMS. In contrast to the Schnorr protocol, R of the DDH-based lossy identification consists of two group elements. Thus, we cannot just apply the tool of HBMS to the DDH-based lossy identification.

One may think that the following naive way is sufficient, but it is not true. Below, we explain why the naive way fails. Each signer generates two keys ck_1 and ck_2 of the tool used in HBMS by hashing the message and generates a commitment T_1 of R_1 and a commitment T_2 of R_2 by using ck_1 and ck_2, respectively. Then, the verification equation is $(T_1, T_2)^T = d_1(ck_1, O)^T + d_2(O, ck_2)^T + s(G, H)^T - c(Y, Z)^T$ where O is the identity element in $\mathbb{G}$, $c = H_c(T_1, T_2, m, pk)$, and $d_1, d_2 \in \mathbb{Z}_q$. The important observation is that as long as the verification equation includes the term $d_1(ck_1, O)^T + d_2(O, ck_2)^T$, we cannot prove that forgery under the lossy key is statistically hard. Note that a forger can maliciously choose d_1 and d_2 so that $d_1(ck_1, O)^T + d_2(O, ck_2)^T$ is a non-DH tuple. This means that a computationally unbounded forger can forge by generating d_1, d_2, and s so that they cancel out the lossy key even if (Y, Z) is a non-DH tuple. Therefore, we must modify the term to ensure the property of **Type-1**.

2.4 Our Solutions

Our solution is aggregating two terms $d_1(ck_1, O)^T$ and $d_2(O, ck_2)^T$ into $d(ck_1, ck_2)^T$ and programming the random oracle to let (ck_1, ck_2) be a random DH tuple. This programming is validated by the DDH assumption. When (ck_1, ck_2) is a DH tuple, we can rewrite $(ck_1, ck_2)^T = a(G, H)^T$ where $a \in \mathbb{Z}_q$, and we obtain $(T_1, T_2)^T = (ad + s)(G, H)^T - c(Y, Z)^T$ as the verification equation. Notice that because (G, H) and (Y, Z) are linearly independent, c satisfying the equation is determined uniquely at the point when (T_1, T_2) is determined. Since c is uniformly chosen from $\mathbb{Z}_q$ by the random oracle on input (T_1, T_2), we can prove that forgery is statistically hard. Remind that (ck_1, ck_2) is generated by the random oracle, and thus (ck_1, ck_2) uniformly distributes over $\mathbb{G}$ in the real environment. Then, due to the DDH assumption, we can program the random oracle to output a random DH tuple in $\mathbb{G}^2$ instead of a random tuple in $\mathbb{G}^2$.

We can construct keys of **Type-2** by embedding the public key into a DH tuple. Specifically, we program the random oracle to output $(ck_1, ck_2)^T \leftarrow \rho(G, H)^T + (Y, Z)^T$ where ρ is a trapdoor uniformly chosen from $\mathbb{Z}_q$. Due to the above approach, all terms in the verification equation are DH tuples. Then, The simulator can generate (T_1, T_2) with embedded (Y, Z) in the first round and can generate (d, s) by exploiting ρ and the state information in the first round after c is determined.

[4] For simplicity, we consider the case where there is only one signer.

[5] The equivocal key is generated by embedding the public key, and one having a trapdoor can simulate the honest signer without using the secret key.

We need to guarantee that the forgery is valid under the commitment key **Type-1** and embedded those two types of commitment keys into the same random oracle to make our solution work well. Then, we use the technique of the security proof of the RSA-FDH signature scheme by Coron [42]. Consequently, our scheme has the security loss $O(q_S)$.

Consequently, we can construct a two-round multi-signature scheme with a small reduction loss.

3 Preliminaries

3.1 Notation

Unless noted otherwise, any algorithm is probabilistic. For an algorithm $\mathcal{A}$, we write $b \xleftarrow{\$} \mathcal{A}(\alpha_1, \ldots)$ to mean that $\mathcal{A}$ on inputs $\alpha_1, \ldots$ and a uniformly chosen random tape outputs b. For algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_n$, we write $b \xleftarrow{\$} \langle \{\mathcal{A}_i(\alpha_{i1}, \ldots)\}_{i=1}^n \rangle$ to mean that each algorithm $\mathcal{A}_i$ on inputs $\alpha_{i1}, \ldots$ and a uniformly chosen random tape executes a protocol with the others, and eventually all algorithms obtain b. For a list L, we write the i-th element in L as $L[i]$. For any value a, we write $a \leftarrow b$ means the assignment of a into b. We denote the security parameter by λ.

3.2 Hardness Assumption

For a prime integer q, we denote the ring of integers modulo q by $\mathbb{Z}_q$. Let $\mathbb{G}$ be an additive cyclic group of order q and let G be a generator of $\mathbb{G}$. We denote the identity element of $\mathbb{G}$ by O.

In this paper, we use the following notations. For $A,\ B,\ G,\ H \in \mathbb{G}$ and $x \in \mathbb{Z}_q$, we write $(A, B)^T \leftarrow x(G, H)^T$ to mean that A and B are computed by xG and xH, respectively. Also, for $A,\ B,\ G,\ H,\ Y,\ Z \in \mathbb{G}$, we write $(A, B)^T \leftarrow (G, H)^T + (Y, Z)^T$ to mean that A and B are computed by $G+Y$ and $H+Z$, respectively.

Below, we recall the definitions of the discrete logarithm (DL) assumption and the decisional Diffie-Hellman (DDH) assumption.

Definition 3.1. *The advantage* $\mathrm{Adv}^{\mathrm{dl}}_{\mathbb{G}}(\mathcal{A})$ *of an algorithm* $\mathcal{A}$ *is defined as*

$$\mathrm{Adv}^{\mathrm{dl}}_{\mathbb{G}}(\mathcal{A}) = \Pr[xG = X : X \xleftarrow{\$} \mathbb{G}, x \xleftarrow{\$} \mathcal{A}(X)].$$

We say that an algorithm $\mathcal{A}$ (t, ε)*-solves the DL problem in* $\mathbb{G}$ *if it runs in time at most* t *and satisfies* $\mathrm{Adv}^{\mathrm{dl}}_{\mathbb{G}}(\mathcal{A}) \geq \epsilon$*. We also say that the DL problem in* $\mathbb{G}$ *is* (t, ε)*-hard if there is no algorithm that* (t, ε)*-solves it.*

Definition 3.2. *The advantage* $\mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A})$ *of an algorithm* $\mathcal{A}$ *is defined as*

$$\mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}) = \left| \Pr[\mathcal{A}(G, xG, yG, xyG) = 1 : x, y \xleftarrow{\$} \mathbb{Z}_q] - \Pr\left[\mathcal{A}(G, xG, yG, zG) = 1 : \begin{array}{l} x, y \xleftarrow{\$} \mathbb{Z}_q, \\ z \xleftarrow{\$} \mathbb{Z}_q \backslash \{xy\} \end{array} \right] \right|.$$

We say that an algorithm $\mathcal{A}$ (t, ε)*-solves the DDH problem in* $\mathbb{G}$ *if* $\mathcal{A}$ *runs in time at most* t *and satisfies* $\mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}) \geq \varepsilon$*. We also say that* $\mathbb{G}$ *is a* (t, ε)*-DDH group if there is no algorithm that* (t, ε)*-solves the DDH problem in* $\mathbb{G}$*.*

3.3 Randomizing Algorithm of (non-)DH tuple

Bellare et al. proposed a randomizing algorithm of a (non-)DH tuple in [43]. Their algorithm on input a (non-)DH tuple outputs a re-randomized (non-)DH tuple. More concretely, the algorithm is given a tuple $(G, H, P, Q) \in \mathbb{G}^4$ as input and outputs a tuple $(G, H', P', Q') \in \mathbb{G}^4$. If (G, H, P, Q) is a DH tuple, (G, H', P', Q') satisfies that (H', P') is uniformly distributed over $\mathbb{G}^2$ and (G, H', P', Q') is a DH tuple. If (G, H, P, Q) is a non-DH tuple, (G, H', P', Q') satisfies that (H', P', Q') is uniformly distributed over $\mathbb{G}^3$.

In this paper, we use the subtly modified algorithm to prove Proposition 4.5. This algorithm on input a (non-)DH tuple outputs a re-randomized (non-)DH tuple of which the second element is also the same as

the second one of a tuple given as input. Specifically, if (G, H, P, Q) is a DH tuple, (G, H, P', Q') satisfies that P' is uniformly distributed over $\mathbb{G}$ and (G, H, P', Q') is a DH tuple. If (G, H, P, Q) is a non-DH tuple, (G, H, P', Q') satisfies that (P', Q') is uniformly distributed over $\mathbb{G}^2$.

Below we show our randomizing algorithm **Rand**.

Rand$(G, H, P, Q) \to (P', Q')$

Choose $s, t \xleftarrow{\$} \mathbb{Z}_q$.

Compute $P' \leftarrow sG + tP$ and $Q' \leftarrow sH + tQ$.

Output (P', Q').

3.4 Multi-Signatures

In this section, we show the definition and security model of the multi-signature scheme.

Definition 3.3. *A multi-signature scheme consists of the following three algorithms and an interactive protocol. Let n be the number of signers.*

Pg$(1^\lambda) \to pp$**.** *On input the security parameter 1^λ, the public parameter generation algorithm outputs a public parameter pp.*

Kg$(pp) \to (pk, sk)$**.** *On an input pp, the key generation algorithm outputs a public key pk and a secret key sk.*

$\langle \{\mathcal{S}(pp, i, sk_i, L, m)\}_{i=1}^n \rangle \to \widetilde{\sigma}$**.** *The signing protocol is executed by multiple signers who intend to sign on the common message m. On inputs pp, an index of signers i, sk_i, a public-key list $L = \{pk_1, \ldots, pk_n\}$, and m, each signer executes the signing protocol with cosigners. Finally, it outputs a multi-signature $\widetilde{\sigma}$ on m.*

Vf$(pp, L, \widetilde{\sigma}, m) \to \{0, 1\}$**.** *On inputs pp, L, $\widetilde{\sigma}$, and m, the verification algorithm deterministically outputs 1 (Accept) or 0 (Reject).*

A multi-signature scheme must satisfy the following correctness property: For any message m, any integer n, $pp \xleftarrow{\$}$ **Pg**(1^λ), $(pk_i, sk_i) \xleftarrow{\$}$ **Kg**(pp) for all $i \in \{1, n\}$, $L \leftarrow \{pk_i\}_{i=1}^n$, and $\widetilde{\sigma} \xleftarrow{\$} \langle \{\mathcal{S}(pp, i, sk_i, L, m)\}_{i=1}^n \rangle$, **Vf**$(pp, L, \widetilde{\sigma}, m) = 1$ holds.

3.4.1 Security Definition of Multi-Signatures

Here, we show the security definition of multi-signatures. Our security model is very similar to Bellare-Neven's model [3]. In the security model, corruption of cosigners by a forger is modeled by allowing a forger to freely choose cosigners' public keys involving a forgery and signing queries. Note that there are two subtle differences from Bellare-Neven's model in the winning conditions.

The first one is that, in our model, a forger's output $(m^*, \widetilde{\sigma}^*, L^*)$ does not count as a successful forgery if the forger has received a signature on the message m^* from the signing oracle. In Bellare-Neven's model, $(m^*, \widetilde{\sigma}^*, L^*)$ counts as a forgery even if the forger has ever received a signature on m^* as long as the pair of the message and the public-key list (m^*, L^*) has never been queried. The second one is that, in our model, $(m^*, \widetilde{\sigma}^*, L^*)$ counts as a successful forgery even if a signing protocol for m^* is opened but not completed. In Bellare-Neven's model, $(m^*, \widetilde{\sigma}^*, L^*)$ does not count as a forgery if a signing protocol for that pair (m^*, L^*) is opened but not completed. The latter modification captures adversaries who exploit the interruption of the signing protocol. To see the difference, let us consider the following example. A forger sends a message m^* to the signing oracle as a signing query and receives a response of the first round from the oracle. Then, it outputs a forgery on m^* without completing the signing protocol. In our model, the forgery is valid even if m^* is queried to the signing oracle (as long as the signature on m^* has never been received).

Our security model is formally defined by the following three-phase game.

Setup. The challenger generates a public parameter pp by $\mathbf{Pg}(1^\lambda)$ and a key pair (pk, sk) by $\mathbf{Kg}(pp)$. It initializes tables $T_M[\cdot]$ and $T_\Sigma[\cdot]$ to $\emptyset$. It sends pp and pk to a forger $\mathcal{F}$. $\mathcal{F}$ is allowed to access random oracles and a signing oracle.

Signing Oracle. The challenger receives a message m, a session identifier I_s, and a public-key list L as a signing query from $\mathcal{F}$. Let $n_h = \{i | L[i] = pk\}$. The challenger responses as follows.

Case $|n_h| = 0$**.** The challenger returns $\perp$ to $\mathcal{F}$.

Case $|n_h| \geq 1$**.** The challenger executes the signing protocol by behaving as any honest signer corresponding to indices in n_h. Note that it behaves as each honest signer by maintaining its state and using its random tape. Let st_ℓ be the state information, kept during the protocol, of the honest signer corresponding to $\ell \in n_h$. At the end of each round of the protocol, the challenger stores $T_\Sigma[I_s] \leftarrow \{st_\ell\}_{\ell \in n_h}$.

$\mathcal{F}$ is allowed to make multiple signing queries concurrently. Note that, under the same session identifier, $\mathcal{F}$ is not allowed to make multiple signing queries in the same round of the signing protocol. If the signing protocol for a queried m is completed, the challenger assigns $T_M[m] \leftarrow 1$.

Check. Finally, $\mathcal{F}$ outputs a public-key list L^*, a message m^*, and a forgery $\widetilde{\sigma}^*$. $\mathcal{F}$ is said to win the game if $\mathcal{F}$'s output satisfies that $pk \in L^*$, $T_M[m^*] \neq 1$, and $\mathbf{Vf}(pp, L^*, \widetilde{\sigma}^*, m^*) = 1$.

Definition 3.4. *Let* $\mathrm{Adv}_{\mathbf{MS}}(\mathcal{F})$ *be the probability that* $\mathcal{F}$ *wins the above game. We say that* $\mathcal{F}$ $(t, q_S, q_H, N, \varepsilon)$*-breaks multi-signature scheme if* $\mathcal{F}$ *runs in at most* t *time, it makes at most* q_S *signing queries and* q_H *random oracle queries, the numbers of public keys included in* L *for any signing queries and a forgery are at most* N*, and* $\mathrm{Adv}_{\mathbf{MS}}(\mathcal{F}) \geq \varepsilon$*. We say a multi-signature scheme is* $(t, q_S, q_H, N, \varepsilon)$*-secure if there is no* $\mathcal{F}$ *that* $(t, q_S, q_H, N, \varepsilon)$*-breaks that scheme.*

4 Proposed Scheme

We show the construction of our proposed two-round multi-signature scheme and its security. We construct our scheme by combining the Katz-Wang DDH-based signature scheme [34] to avoid rewinding and the technique of HBMS [9], a DL-based two-round multi-signature scheme.

4.1 Our Proposed Scheme

Below, we show the construction of our two-round multi-signature scheme.

$\mathbf{Pg}(1^\lambda) \to pp$. On input the security parameter 1^λ, the public parameter generation algorithm sets up $(\mathbb{G}, q, G)$. It chooses a random element $H \in \mathbb{G}$, hash functions $H_{\mathrm{c}} : \{0,1\}^* \to \mathbb{Z}_q$, $H_{\mathrm{ck}} : \{0,1\}^* \to \mathbb{G}^2$, and $H_{\mathrm{agg}} : \{0,1\}^* \to \mathbb{Z}_q$, and then it outputs $pp = (\mathbb{G}, q, G, H, H_{\mathrm{c}}, H_{\mathrm{ck}}, H_{\mathrm{agg}})$.

$\mathbf{Kg}(pp) \to (pk, sk)$. On an input pp, the key generation algorithm chooses $x \xleftarrow{\$} \mathbb{Z}_q$, computes $(Y, Z)^T \leftarrow x(G, H)^T$ and outputs a public key $pk = (Y, Z)$ and a secret key $sk = x$.

$\langle \{\mathcal{S}(pp, i, sk_i, L, m)\}_{i=1}^n \rangle \to \tilde{\sigma}$. Each signer proceeds with the signing protocol as follows.

Round 1: Each signer computes $t_j \leftarrow H_{\mathrm{agg}}((Y_j, Z_j), L)$ for all $j \in [1, n]$ and $\widetilde{pk} \leftarrow \sum_{j=1}^n t_j (Y_j, Z_j)^T$. It computes $(U_1, U_2) \leftarrow H_{\mathrm{ck}}(m)$, chooses $r_i, z_i \xleftarrow{\$} \mathbb{Z}_q$ and computes $T_i \leftarrow z_i (U_1, U_2)^T + r_i (G, H)^T$. It broadcasts T_i to the cosigners.

Round 2: Each signer receives $\{T_j\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$ from the cosigners. It computes $\widetilde{T} \leftarrow \sum_{i=1}^n T_i$, $c \leftarrow H_{\mathrm{c}}(\widetilde{T}, \widetilde{pk}, m)$, and $s_i \leftarrow x_i t_i c + r_i \mod q$. It broadcasts (z_i, s_i) to the cosigners.

Aggregate: Each signer receives $\{(z_j, s_j)\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$ from the cosigners. It computes $\tilde{z} \leftarrow \sum_{j=1}^n z_j \mod q$ and $\tilde{s} \leftarrow \sum_{j=1}^n s_j \mod q$ and outputs $\tilde{\sigma} = (c, \tilde{z}, \tilde{s})$.

Vf$(pp, L, \tilde{\sigma}, m) \to \{0,1\}$**.** On inputs $pp, L, \tilde{\sigma}$, and m, the verification algorithm computes $t_j \leftarrow H_{\mathrm{agg}}((Y_j, Z_j), L)$ for all $j \in [1,n]$ and $\widetilde{pk} \leftarrow \sum_{j=1}^{n} t_j (Y_j, Z_j)^T$. It computes $(U_1, U_2) \leftarrow H_{\mathrm{ck}}(m)$ and $\widetilde{T} \leftarrow \tilde{z}(U_1, U_2)^T + \tilde{s}(G, H)^T - c \cdot \widetilde{pk}$. It outputs 1 if $c = H_{\mathrm{c}}(\widetilde{T}, \widetilde{pk}, m)$ holds. Otherwise, it outputs 0.

4.2 Security

We show that our proposed scheme is secure under the DDH assumption in the random oracle model.

Theorem 4.1. *If $\mathbb{G}$ is a (t', ε')-DDH group, then our scheme is $(t_{\mathcal{F}}, q_S, q_H, N, \varepsilon_{\mathcal{F}})$-secure s.t.*

$$\varepsilon_{\mathcal{F}} \geq e(q_S+1)(2\varepsilon' + (2q_H + q_S + 2)/q) \text{ and}$$
$$t_{\mathcal{F}} \leq \min(t_1, t_2) \text{ where}$$
$$t_1 = t' - (4q_H + 6q_S N + 2N + 12)t_{\mathrm{mul}} - O(q_H + q_S N),$$
$$t_2 = t' - (3q_H + 6q_S N + 3q_S + 2N + 6)t_{\mathrm{mul}} - O(q_H + q_S N),$$

where e is the base of the natural logarithm, and t_{mul} is the time of a scalar multiplication in $\mathbb{G}$.

Before showing the full proof, we show a proof sketch.

As in the Katz-Wang signature scheme, we prove the unforgeability of our scheme by replacing the public key with a non-DH tuple due to the DDH assumption and proving that forgery is statistically hard under such the public key. The main strategy is that we ensure an situation where we can statistically evaluate the forger's success probability $\varepsilon_{\mathcal{F}}$. To enable this, we need to ensure a situation where we can statistically evaluate the forger's success probability $\varepsilon_{\mathcal{F}}$ even if the forger is computationally unbounded when the public key is a non-DH tuple. To ensure such a situation, we replace (U_1, U_2) generated by the random oracle $H_{\mathrm{ck}}(m)$ with a random DH tuple. The effect of this replacement is guaranteed to be negligible by the DDH assumption. However, if we replace all (U_1, U_2) with DH tuples, we cannot simulate the honest signer without the secret key sk. To solve this issue, we provide another way to generate (U_1, U_2) which allows simulating the honest signer without sk. Then, to make these two contrasting ways compatible, we use the technique of Coron [42], which is to prove the security of the RSA Full Domain Hash (RSA-FDH) signature scheme [44], as in mBCJ and HBMS.

Our proof is a game-hopping proof. We start with the game of the security definition and sequentially change it into a game in which forgery is statistically hard. Specifically, we consider the following game-hopping.

Game G_1 **(**Game_1**-**Game_3**) :** We change the game of the security game as follows: The challenger generates *two* types of (U_1, U_2) instead of uniformly choosing from $\mathbb{G}^2$ and assigns one of them to the random oracle table of $H_{\mathrm{ck}}(m)$ according to a biased coin which comes out heads with a certain probability, like the technique of Coron [42]. The first type (**Type-1**) is to statistically evaluate the success probability of a forger in the final game. The other type (**Type-2**) is to simulate the honest signer without sk in the signing oracle.

Game G_2 **(**Game_4**) :** We change the above game as follows: The challenger simulates the honest signer without sk by using the property of (U_1, U_2) of **Type-2**.

Game G_3 **(**Game_5**-**Game_6**) :** We change the above game as follows: The challenger embeds a non-DH tuple into pk, like the security proof of the Katz-Wang signature scheme.

In a nutshell, as first, we show our procedure to prove that Game G_1 and Game G_3 are computationally indistinguishable under the DDH assumption. First, we prove that Game G_1 and Game G_2 are perfectly indistinguishable by proving that the distribution of responses of the signing oracle with sk and that of the response of the signing oracle using the property of (U_1, U_2) of **Type-2** are perfectly indistinguishable. Next, we prove that Game G_2 and Game G_3 are computationally indistinguishable by proving that pk generated by **Kg** in Game G_2 which is a DH tuple and pk in Game G_3 which is a non-DH tuple are indistinguishable under the DDH assumption.

Using the property of (U_1, U_2) of **Type-1**, in Game G_3, we can statistically evaluate $\varepsilon_{\mathcal{F}}$ and then prove that forgery is statistically hard. Then, to complete the explanation of this proof sketch, it remains to show the construction of two types of (U_1, U_2) and the indistinguishability between the game of the security definition and Game G_1. We explain these below.

First, we explain the way to generate (U_1, U_2) of **Type-1**. The challenger generates it satisfying that it is uniformly distributed in the span of (G, H). Specifically, the challenger chooses $\rho \xleftarrow{\$} \mathbb{Z}_q$ and computes $(U_1, U_2)^T \leftarrow \rho(G, H)^T$. To explain why this is necessary, we consider the simple case where there is only one signer and key aggregation is not supported. Then, the verification equation is $T_1 = z_1(U_1, U_2)^T + s_1(G, H)^T - c(Y_1, Z_1)^T$ where $c = H_{\mathrm{c}}(T_1, (Y_1, Z_1), m)$. Notice that, when (G, H, Y_1, Z_1) is a non-DH tuple and (U_1, U_2) is in the span of (G, H), c satisfying the above equation is determined uniquely at the point when T_1 is determined. Because c is uniformly chosen from $\mathbb{Z}_q$ by the random oracle, the probability that c satisfies the above equation is at most $1/q$.[6]

Next, we explain the way to generate (U_1, U_2) of **Type-2**. The challenge key (Y_1, Z_1) is embedded in this type of (U_1, U_2). More concretely, the challenger chooses $\rho \xleftarrow{\$} \mathbb{Z}_q$ and computes $(U_1, U_2)^T \leftarrow \rho(G, H)^T + (Y_1, Z_1)^T$. Then, the challenger has ρ as the trapdoor. The equivocal commitment T_1 is generated by $\alpha(G, H)^T + \beta(Y_1, Z_1)^T$ where α and β are uniformly chosen from $\mathbb{Z}_q$. We need to produce z_1 and s_1 satisfying $T_1 = z_1(U_1, U_2)^T + s_1(G, H)^T - c(Y_1, Z_1)^T$ given c. Notice that sk is no longer required since T_1 and $(U_1, U_2)^T$ is expressed by linear combinations of $(G, H)^T$ and $(Y_1, Z_1)^T$. Specifically, by using ρ, α, and β, we can produce z_1 and s_1 satisfying $T_1 = \alpha(G, H)^T + \beta(Y_1, Z_1)^T = z_1(\rho(G, H)^T + (Y_1, Z_1)^T) + s_1(G, H)^T - c(Y_1, Z_1)^T$. For indistinguishability between the distribution of responses of the signing oracle with sk and that of responses of the signing oracle using ρ, see Proposition 4.6.

Finally, we explain that the game of the security definition and Game G_1 are computationally indistinguishable under the DDH assumption. Notice that, for both types of (U_1, U_2), the challenger generates (U_1, U_2) by producing a random DH tuple $\rho(G, H)^T$. Then, we can prove that the distribution of (U_1, U_2) uniformly chosen from $\mathbb{G}^2$ and the one of (U_1, U_2) generated by $H_{\mathrm{ck}}(m)$ in Game G_1 are computationally indistinguishable under the DDH assumption. Therefore, the game of the security definition and Game G_1 are computationally indistinguishable under the DDH assumption.

As a result, we can show that our scheme is secure under the DDH assumption in the random oracle model.

Remark. Since we need to guarantee that the forger only produces a forgery which is valid under (U_1, U_2) of **Type-1**, we need to add a condition that a forgery is valid under (U_1, U_2) of **Type-1** into the winning condition of the forger in Game G_1. In the full proof, we consider intermediate games between the original game of the security definition and Game G_1 with the above additional winning condition. Thus, our scheme has the reduction loss $e(q_S + 1)$, which is the same as the reduction loss of the RSA-FDH signature scheme proven by Coron [42].

Below, we show the formal security proof.

proof of Theorem 4.1. The game Game_0 is the unforgeability game for our scheme. Game_0 is as follows.

Setup: The challenger generates pp by $(\mathbb{G}, q, G, H, H_{\mathrm{c}}, H_{\mathrm{ck}}, H_{\mathrm{agg}}) \leftarrow \mathbf{Pg}(1^\lambda)$ and (pk, sk) by $((Y^*, Z^*), x) \xleftarrow{\$} \mathbf{Kg}(pp)$ where H_{c}, H_{ck}, and H_{agg} are random oracles. For H_{c}, H_{ck}, and H_{agg}, it initializes tables $T_c[\cdot]$, $T_{\mathrm{ck}}[\cdot]$, and $T_{\mathrm{agg}}[\cdot]$ to $\emptyset$. It also initializes tables $T_M[\cdot]$ and $T_\Sigma[\cdot]$ to $\emptyset$. It sends pp and pk to $\mathcal{F}$ who is allowed to access the signing oracle and the following random oracles.

Random Oracle $H_{\mathrm{ck}}(m)$: The challenger chooses $(U_1, U_2) \xleftarrow{\$} \mathbb{G}^2$ and assigns $T_{\mathrm{ck}}[m] \leftarrow (U_1, U_2)$ if $T_{\mathrm{ck}}[m]$ is undefined. It returns $T_{\mathrm{ck}}[m]$.

Random Oracle $H_{\mathrm{c}}(\widetilde{T}, \widetilde{pk}, m)$: The challenger chooses $c \xleftarrow{\$} \mathbb{Z}_q$ and assigns $T_c[\widetilde{T}, \widetilde{pk}, m] \leftarrow c$ if $T_c[\widetilde{T}, \widetilde{pk}, m]$ is undefined. It returns $T_c[\widetilde{T}, \widetilde{pk}, m]$.

[6] Because our scheme supports the key aggregation, we need to consider a more complex setting. For more details, see Proposition 4.11.

Random Oracle $H_{\mathrm{agg}}((Y,Z),L)$**:** The challenger chooses $t \xleftarrow{\$} \mathbb{Z}_q$ and assigns $T_{\mathrm{agg}}[(Y,Z),L] \leftarrow t$ if $T_{\mathrm{agg}}[(Y,Z),L]$ is undefined. It returns $T_{\mathrm{agg}}[(Y,Z),L]$.

Signing Oracle: The challenger receives (I_s, m, L) as a query. It returns $\perp$ if $pk \notin L$. It executes the signing protocol by behaving the honest signers $\{\mathcal{S}(pp, i, sk, L, m)\}_{i \in n_h}$ where n_h is the set of the indices of the signers who have pk as the public keys. At the end of **Round 1** of the signing protocol, it stores $T_\Sigma[I_s] \leftarrow (m, L, \widetilde{pk}, n_h, \{(T_i, z_i, r_i, t_i)\}_{i \in n_h})$. If **Round 2** of the signing protocol is completed, it sets $T_M[m] \leftarrow 1$.

Check: $\mathcal{F}$ wins the game if $\mathcal{F}$'s output $(L^*, m^*, \tilde{\sigma}^*)$ satisfies $pk \in L^*$, $T_M[m^*] \neq 1$, and $\mathbf{Vf}(pp, L^*, \tilde{\sigma}^*, m^*) = 1$.

Now we change the above game Game_0. Below, we describe only the changes.

$\underline{\mathsf{Game}_1}$**:** We change Game_0 as follows.

Random Oracle $H_{\mathrm{c}}(\widetilde{T}, \widetilde{pk}, m)$**:** The challenger makes a query $H_{\mathrm{ck}}(m)$ first if $T_{\mathrm{ck}}[m]$ undefined.

Signing Oracle: The challenger makes a query $H_{\mathrm{ck}}(m)$ first if $T_{\mathrm{ck}}[m]$ undefined.

$\underline{\mathsf{Game}_2}$**:** We change Game_1 as follows.

Setup: The challenger additionally initializes a table $T_{\mathrm{b}}[\cdot] \leftarrow \emptyset$.

Random Oracle $H_{\mathrm{ck}}(m)$**:** If $T_{\mathrm{ck}}[m]$ is undefined, the challenger firstly chooses a bit $b_K \in \{0,1\}$ which becomes 1 with probability $\delta = q_S/(q_S+1)$. Note that the way to generate (U_1, U_2) is unchanged. It additionally assigns $T_{\mathrm{b}}[m] \leftarrow b_K$.

Signing Oracle: If $T_{\mathrm{b}}[m] = 0$ holds for a queried m, the challenger terminates the game when it starts **Round 2** of the signing protocol. Otherwise, it continues the game.

Check: The condition $T_{\mathrm{b}}[m^*] = 0$ is added to the winning conditions.

$\underline{\mathsf{Game}_3}$**:** We change Game_2 as follows.

Setup: The challenger additionally initializes a table $T_{\mathrm{td}}[\cdot] \leftarrow \emptyset$.

Random Oracle $H_{\mathrm{ck}}(m)$**:** Instead of $(U_1, U_2) \xleftarrow{\$} \mathbb{G}^2$, the challenger chooses $\rho \xleftarrow{\$} \mathbb{Z}_q$, computes $(U_1, U_2)^T \xleftarrow{\$} \rho(G, H)^T$ when $T_{\mathrm{b}}[m] = 0$, and computes $(U_1, U_2)^T \xleftarrow{\$} \rho(G, H)^T + (Y^*, Z^*)^T$ when $T_{\mathrm{b}}[m] = 1$. It additionally assigns $T_{\mathrm{td}}[m] \leftarrow \rho$.

$\underline{\mathsf{Game}_4}$**:** We change Game_3 as follows.

Signing Oracle: Only in the case where $T_{\mathrm{b}}[m] = 1$, the challenger executes the following modified protocol instead of $\mathcal{S}(pp, i, sk, L, m)$.

Round 1: The challenger computes $\widetilde{pk}$ as in $\mathcal{S}(pp, i, sk, L, m)$. For all $i \in n_h$, it chooses $(\alpha_i, \beta_i) \xleftarrow{\$} \mathbb{Z}_q^2$, computes $T_i \leftarrow \alpha_i(G, H)^T + \beta_i(Y^*, Z^*)^T$. It stores $T_\Sigma[I_s] \leftarrow (m, L, \widetilde{pk}, n_h, \{(T_i, \alpha_i, \beta_i, t_i)\}_{i \in n_h})$ and returns $\{T_i\}_{i \in n_h}$.

Round 2: The challenger receives $(I_s, \{T_i\}_{i \in [1,n] \setminus n_h})$, looks up $(m, L, \widetilde{pk}, n_h, \{(T_i, \alpha_i, \beta_i, t_i)\}_{i \in n_h})$ from $T_\Sigma[I_s]$, (U_1, U_2) from $T_{\mathrm{ck}}[m]$, and ρ from $T_{\mathrm{td}}[m]$. It computes $\widetilde{T}$ and c as in $\mathcal{S}(pp, i, sk, L, m)$. For all $i \in n_h$, computes $z_i \leftarrow \beta_i + c \mod q$ and $s_i \leftarrow \alpha_i - z_i\rho \mod q$. It returns $\{(z_i, s_i)\}_{i \in n_h}$.

$\underline{\mathsf{Game}_5}$**:** We change Game_4 as follows.

Setup: The challenger generates pk by choosing $x \xleftarrow{\$} \mathbb{Z}_q$ and $y \xleftarrow{\$} \mathbb{Z}_q \setminus \{x\}$ and computing $Y \leftarrow xG$ and $Z \leftarrow yH$.

$\underline{\mathsf{Game}_6}$**:** We change Game_5 as follows.

Random Oracle $H_{\mathrm{agg}}((Y,Z),L)$**:** For all $i \in [1,n]$ where n is a number of public keys in L, the challenger chooses $t_i \xleftarrow{\$} \mathbb{Z}_q$ and assigns $T_{\mathrm{agg}}[L[i],L] \leftarrow t_i$ if $T_{\mathrm{agg}}[L[i],L]$ is undefined.

We write $\Pr[\mathsf{Game}_i = 1]$ to mean the probability that a forger wins the game Game_i. From Propositions 4.2, $\cdots$, 4.11, which we will prove later, we obtain

$$\begin{aligned}
\varepsilon_{\mathcal{F}} &= \Pr[\mathsf{Game}_1 = 1] && \text{(Propositions 4.2 and 4.3)}\\
&\leq e(q_S+1)\Pr[\mathsf{Game}_2 = 1] && \text{(Proposition 4.4)}\\
&\leq e(q_S+1)(\varepsilon' + \Pr[\mathsf{Game}_4 = 1]) && \text{(Propositions 4.5 and 4.6)}\\
&\leq e(q_S+1)(2\varepsilon' + \Pr[\mathsf{Game}_6 = 1]) && \text{(Propositions 4.9 and 4.10)}\\
&\leq e(q_S+1)(2\varepsilon' + (2q_H + q_S + 2)/q). && \text{(Proposition 4.11)}
\end{aligned}$$

From Propositions 4.4 and 4.9, we also have

$$\begin{aligned}
&t_{\mathcal{F}} \geq t' - (4q_H + 6q_S N + 2N + 12)t_{\mathrm{mul}} - O(q_H + q_S N),\\
&\text{and} \quad t_{\mathcal{F}} \geq t' - (3q_H + 6q_S N + 3q_S + 2N + 6)t_{\mathrm{mul}} - O(q_H + q_S N).
\end{aligned}$$

□

From here, we prove Propositions 4.2, $\cdots$,4.11.

Proposition 4.2. $\varepsilon_{\mathcal{F}} = \Pr[\mathsf{Game}_0 = 1]$

proof of Proposition 4.2. Because Game_0 is the unforgeability game of our scheme, the probability that $\mathcal{F}$ wins this game is identical to $\varepsilon_{\mathcal{F}}$. □

Proposition 4.3. $\Pr[\mathsf{Game}_0 = 1] = \Pr[\mathsf{Game}_1 = 1]$

proof of Proposition 4.3. The difference between Game_0 and Game_1 is that, in Game_1, the challenger makes a query $H_{\mathrm{ck}}(m)$ first in H_{c} and **Signing Oracle**. These newly added steps do not affect the probability of $\mathcal{F}$ winning the game. Therefore, $\Pr[\mathsf{Game}_0 = 1] = \Pr[\mathsf{Game}_1 = 1]$ holds. □

Proposition 4.4. $\Pr[\mathsf{Game}_1 = 1] \leq e(q_S+1)\Pr[\mathsf{Game}_2 = 1]$

proof of Proposition 4.4. First, we show that the added steps of H_{ck} in Game_2 do not affect the probability of $\mathcal{F}$ winning the game. Since (U_1, U_2) in Game_2 is generated by uniformly choosing from $\mathbb{G}^2$ independently of the value of $T_{\mathrm{b}}[m]$, the distributions of the responses of H_{ck} in both games are identical. Therefore, the added steps do not affect the probability of $\mathcal{F}$ winning the game.

Second, we show that $\Pr[\mathsf{Game}_1 = 1] \leq e(q_S+1)\Pr[\mathsf{Game}_2 = 1]$. For Game_2, let $E_1^{\mathsf{Game}_2}$ be the event where the game does not terminate in **Signing Oracle**, $E_2^{\mathsf{Game}_2}$ be the event where $\mathcal{F}$'s output satisfies the added condition $T_{\mathrm{b}}[m^*] = 0$, and $E_3^{\mathsf{Game}_2}$ be the event where $\mathcal{F}$'s output satisfies the winning conditions as same as Game_1. Then, we have

$$\begin{aligned}
\Pr[\mathsf{Game}_2 = 1] &= \Pr[E_1^{\mathsf{Game}_2} \wedge E_2^{\mathsf{Game}_2} \wedge E_3^{\mathsf{Game}_2}]\\
&= \Pr[E_1^{\mathsf{Game}_2}]\Pr[E_3^{\mathsf{Game}_2} | E_1^{\mathsf{Game}_2}]\Pr[E_2^{\mathsf{Game}_2} | E_1^{\mathsf{Game}_2} \wedge E_3^{\mathsf{Game}_2}].
\end{aligned}$$

Firstly, we evaluate $\Pr[E_1^{\mathsf{Game}_2}]$. The game terminates before the challenger starts **Round 2** in **Signing Oracle** if $T_{\mathrm{b}}[m] = 0$ holds for some queried message. Thus, $E_1^{\mathsf{Game}_2}$ occurs when $T_{\mathrm{b}}[m] = 1$ holds at that point for all messages queried to **Signing Oracle**. Since the responses of the random oracles and the responses of the signing oracle until **Round 1** leak no information on the value of $T_{\mathrm{b}}[m]$ for any m, $\mathcal{F}$ can know $T_{\mathrm{b}}[m]$ only when it observes whether the game continues or not. Also, $\mathcal{F}$ can only know $T_{\mathrm{b}}[m] = 1$ for all messages queried to **Signing Oracle** as long as the game continues. Therefore, the probability that $T_{\mathrm{b}}[m] = 0$ holds for a queried message is equal to $(1-\delta)$. In consequence, because $\mathcal{F}$ can make at most q_S

signing queries, we have $\Pr[E_1^{\mathsf{Game}_2}] \geq \delta^{q_S}$. Setting $\delta = q_S/(q_S+1)$, we have $\Pr[E_1^{\mathsf{Game}_2}] \geq \delta^{q_S} \geq 1/e$. The last inequality holds because of the fact that $(1+1/q_S)^{q_S} < e$ for $q_S > 0$.

Next, we evaluate $\Pr[E_3^{\mathsf{Game}_2}|E_1^{\mathsf{Game}_2}]$. Conditioned on $E_1^{\mathsf{Game}_2}$, Game_2 does not terminate, and the distribution of the view of $\mathcal{F}$ in Game_2 is identical to the distribution of the view of $\mathcal{F}$ in Game_1. Thus, $\mathcal{F}$'s output in Game_2 satisfies the winning conditions of Game_1 with the same probability as in Game_1. Namely, we have $\Pr[E_3^{\mathsf{Game}_2}|E_1^{\mathsf{Game}_2}] = \Pr[\mathsf{Game}_1 = 1]$.

Finally, we evaluate $\Pr[E_2^{\mathsf{Game}_2}|E_1^{\mathsf{Game}_2} \wedge E_3^{\mathsf{Game}_2}]$. Conditioned on $E_1^{\mathsf{Game}_2}$ and $E_3^{\mathsf{Game}_2}$, since **Round 2** of the signing protocol on m^* has never been executed in **Signing Oracle**, $\mathcal{F}$ cannot know $T_{\mathrm{b}}[m^*]$. Then, $\Pr[E_2^{\mathsf{Game}_2}|E_1^{\mathsf{Game}_2} \wedge E_3^{\mathsf{Game}_2}] = (1-\delta)$ holds. Setting $\delta = q_S/(q_S+1)$, we obtain $\Pr[E_2^{\mathsf{Game}_2}|E_1^{\mathsf{Game}_2} \wedge E_3^{\mathsf{Game}_2}] = 1/(q_S+1)$.

From the above, we obtain $\Pr[\mathsf{Game}_1 = 1] \leq e(q_S+1)\Pr[\mathsf{Game}_2 = 1]$. □

Proposition 4.5. *If $\mathbb{G}$ is a (t', ε')-DDH group, then*

$$|\Pr[\mathsf{Game}_2 = 1] - \Pr[\mathsf{Game}_3 = 1]| \leq \varepsilon', \text{ and}$$
$$t_{\mathcal{F}} \geq t' - (4q_H + 6q_S N + 2N + 12)t_{\mathrm{mul}} - O(q_H + q_S N)$$

where t_{mul} is the time for a scalar multiplication in $\mathbb{G}$.

proof of Proposition 4.5. To prove this proposition, we construct a distinguisher $\mathcal{A}_{\mathrm{DDH}}$ against the DDH problem from a forgery $\mathcal{F}$ as follows.

Setup: $\mathcal{A}_{\mathrm{DDH}}$ receives a tuple (G, H, P, Q) as input. It assigns $(\mathbb{G}, q, G, H)$ of input to $(\mathbb{G}, q, G, H)$ of a public parameter pp. It sets up pk and tables as in Game_2 or Game_3. It runs $\mathcal{F}$ on inputs pp and pk.

Random Oracle $H_{\mathrm{ck}}(m)$: $\mathcal{A}_{\mathrm{DDH}}$ responds similarly to in Game_3 with the following difference in the way to generate (U_1, U_2). It generates $(P', Q') \leftarrow \mathbf{Rand}(G, H, P, Q)$. If $T_{\mathrm{b}}[m] = 0$, it assigns $(U_1, U_2) \leftarrow (P', Q')$. If $T_{\mathrm{b}}[m] = 1$, it computes $(U_1, U_2)^T \leftarrow (P', Q')^T + (Y^*, Z^*)^T$.

Random Oracle $H_{\mathrm{c}}(\widetilde{T}, \widetilde{pk}, m)$: $\mathcal{A}_{\mathrm{DDH}}$ responds as in Game_2 or Game_3.

Random Oracle $H_{\mathrm{agg}}((Y, Z), L)$: $\mathcal{A}_{\mathrm{DDH}}$ responds as in Game_2 or Game_3.

Signing Oracle: $\mathcal{A}_{\mathrm{DDH}}$ responds as in Game_2 or Game_3.

Output: $\mathcal{A}_{\mathrm{DDH}}$ returns 1 if $\mathcal{F}$ wins the game. Otherwise, it returns 0.

We show that $|\Pr[\mathsf{Game}_2 = 1] - \Pr[\mathsf{Game}_3 = 1]| = \mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}_{\mathrm{DDH}})$. We can prove this equality by proving the followings.

(i): $\Pr[\mathsf{Game}_2 = 1]$ is equal to the probability that $\mathcal{A}_{\mathrm{DDH}}$ outputs 1 conditioned on (G, H, P, Q) is a non-DH tuple.

(ii): $\Pr[\mathsf{Game}_3 = 1]$ is equal to the probability that $\mathcal{A}_{\mathrm{DDH}}$ outputs 1 conditioned on (G, H, P, Q) is a DH tuple.

The differences between the behavior of $\mathcal{A}_{\mathrm{DDH}}$ and the behavior of the challenger in Game_2 or Game_3 are the way to generate pp in **Setup** and the way to generate (U_1, U_2) in $H_{\mathrm{ck}}(m)$. It is clear that the difference in **Setup** does not affect the probability of $\mathcal{F}$ winning the game. Therefore, to prove the above (i) and (ii), it is sufficient to prove the following (I) and (II), respectively.

(I): The distribution of the responses of $H_{\mathrm{ck}}(m)$ in Game_2 is identical to the distribution of the responses of $H_{\mathrm{ck}}(m)$ in $\mathcal{A}_{\mathrm{DDH}}$ conditioned on (G, H, P, Q) is a non-DH tuple.

(II): The distribution of the responses of $H_{\mathrm{ck}}(m)$ in Game_3 is identical to the distribution of the responses of $H_{\mathrm{ck}}(m)$ in $\mathcal{A}_{\mathrm{DDH}}$ conditioned on (G, H, P, Q) is a DH tuple.

Below, we prove (I) and (II).

(I): In Game_2, the challenger chooses $(U_1, U_2) \xleftarrow{\$} \mathbb{G}^2$ independently of $T_{\mathrm{b}}[m]$. $\mathcal{A}_{\mathrm{DDH}}$ generates (P', Q') by $\mathbf{Rand}(G, H, P, Q)$, assigns $(U_1, U_2)^T \leftarrow (P', Q')^T$ if $T_{\mathrm{b}}[m] = 0$, and computes $(U_1, U_2)^T \leftarrow (P', Q')^T + (Y^*, Z^*)^T$ if $T_{\mathrm{b}}[m] = 1$. Because of the property of $\mathbf{Rand}$, conditioned on (G, H, P, Q) is a non-DH tuple, (P', Q') is uniformly distributed over $\mathbb{G}^2$. Then, $(P', Q')^T + (Y^*, Z^*)^T$ is also uniformly distributed over $\mathbb{G}^2$. Therefore, the responses of $H_{\mathrm{ck}}(m)$ of $\mathcal{A}_{\mathrm{DDH}}$ are uniformly distributed over $\mathbb{G}^2$ in both cases where $T_{\mathrm{b}}[m] = 0$ and $T_{\mathrm{b}}[m] = 1$. Therefore, (I) holds.

(II): In Game_3, the challenger chooses $\rho \xleftarrow{\$} \mathbb{Z}_q$, assigns $(U_1, U_2)^T \leftarrow \rho(G, H)^T$ if $T_{\mathrm{b}}[m] = 0$, and computes $(U_1, U_2)^T \leftarrow \rho(G, H)^T + (Y^*, Z^*)^T$ if $T_{\mathrm{b}}[m] = 1$. $\mathcal{A}_{\mathrm{DDH}}$ generates (P', Q') by $\mathbf{Rand}(G, H, P, Q)$, assigns $(U_1, U_2)^T \leftarrow (P', Q')^T$ if $T_{\mathrm{b}}[m] = 0$, and assigns $(U_1, U_2)^T \leftarrow (P', Q')^T + (Y^*, Z^*)^T$ if $T_{\mathrm{b}}[m] = 1$. Because of the property of $\mathbf{Rand}$, conditioned on (G, H, P, Q) is a DH tuple, (P', Q') satisfies that P' is uniformly distributed over $\mathbb{G}$ and (G, H, P', Q') is a DH tuple. Thus, the distribution of $(P', Q')^T$ is identical to the distribution of $\rho(G, H)^T$ where ρ is uniformly chosen from $\mathbb{Z}_q$. Therefore, (II) holds.

From the above, we obtain $|\Pr[\mathsf{Game}_2 = 1] - \Pr[\mathsf{Game}_3 = 1]| = \mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}_{\mathrm{DDH}})$. Since $\mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}_{\mathrm{DDH}}) \leq \varepsilon'$ holds when $\mathbb{G}$ is a (t', ε')-DDH group, then we obtain $|\Pr[\mathsf{Game}_2 = 1] - \Pr[\mathsf{Game}_3 = 1]| \leq \varepsilon'$.

Now we consider the running time of $\mathcal{A}_{\mathrm{DDH}}$. We assume that t_{mul} time is required for one scalar multiplication in $\mathbb{G}$, and unit time is required for the other non-cryptographic operations. In $\mathbf{Setup}$, $\mathcal{A}_{\mathrm{DDH}}$ computes 2 scalar multiplications to generate pk. For time to answer random oracle queries, we consider only the case of H_{c} because H_{c} takes a longer time than H_{ck} and H_{agg}. To respond to a query to H_{c}, $\mathcal{A}_{\mathrm{DDH}}$ makes one query to H_{ck} and executes $O(1)$ other non-cryptographic operations. 4 scalar multiplications and $O(1)$ other non-cryptographic operations are required for one query to H_{ck}. Thus, in total, $\mathcal{A}$ executes 4 scalar multiplications and $O(1)$ other non-cryptographic operations to respond to one query to H_{c}. For each signing query, there are at most $6N$ scalar multiplications, one query to H_{ck}, one query to H_{c}, N queries to H_{agg}, and $O(N)$ other non-cryptographic operations, thus totally $6q_S N t_{\mathrm{mul}} + O(q_S N)$ time is required for responding to all signing queries. In $\mathbf{Check}$, there are $2N + 6$ scalar multiplications, one query to H_{ck}, one query to H_{c}, N queries to H_{agg}, and $O(N)$ other non-cryptographic operations. From these evaluations and the fact that $\mathcal{A}_{\mathrm{DDH}}$ runs $\mathcal{F}$ once, we obtain $t_{\mathcal{F}} \geq t' - (4q_H + 6q_S N + 2N + 12)t_{\mathrm{mul}} - O(q_H + q_S N)$. □

Proposition 4.6. $\Pr[\mathsf{Game}_3 = 1] = \Pr[\mathsf{Game}_4 = 1]$.

To prove Proposition 4.6, we use the following lemma.

Lemma 4.7. *Let $\mathsf{Game}_{\mathrm{eqv}}$ be the following game between a challenger and a distinguisher $\mathcal{A}$.*

Setup: *The challenger chooses $(\mathbb{G}, q, G)$. It sends $(\mathbb{G}, q, G)$ to $\mathcal{A}$ and receives $H \in \mathbb{G}$ and $x \in \mathbb{Z}_q$ from $\mathcal{A}$. It computes $(Y, Z)^T \leftarrow x(G, H)^T$ and initializes a table $T_S[\cdot]$. It chooses a bit $b \xleftarrow{\$} \{0, 1\}$.*

Oracles: *The challenger allows $\mathcal{A}$ to access to the following oracles concurrently at most q_S times. Note that $\mathcal{A}$ is allowed to make only one query for each session identifier I, which is included in each query to oracles.*

- $\Sigma_{\mathrm{eqv1}}(b, \cdot, \cdot)$**:** *As a query, the challenger receives a session identifier I and $\rho \in \mathbb{Z}_q$. It computes $(U_1, U_2)^T \leftarrow \rho(G, H)^T + (Y, Z)^T$. It responds as follows.*
  - **Case** $b = 0$**:** *It chooses $r, z \xleftarrow{\$} \mathbb{Z}_q$ and computes $T \leftarrow z(U_1, U_2)^T + r(G, H)^T$. It stores $T_S[I] \leftarrow ((U_1, U_2), \rho, r, z, T)$ and returns T.*
  - **Case** $b = 1$**:** *It chooses $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$ and computes $T \leftarrow \alpha(G, H)^T + \beta(Y, Z)$. It stores $T_S[I] \leftarrow ((U_1, U_2), \rho, \alpha, \beta, T)$ and returns T.*
- $\Sigma_{\mathrm{eqv2}}(b, \cdot, \cdot)$**:** *As a query, the challenger receives a session identifier I and $c \in \mathbb{Z}_q$. If $T_S[I]$ is empty, then it return $\perp$. Otherwise, it responds as follows.*
  - **Case** $b = 0$**:** *The challenger looks up $((U_1, U_2), \rho, r, z, T)$ from $T_S[I]$, computes $s \leftarrow xc + r \mod q$ and returns (z, s).*

Case $b = 1$**:** *The challenger looks up* $((U_1, U_2), \rho, \alpha, \beta, T)$ *from* $T_S[I]$*, computes* $z \leftarrow \beta + c \mod q$ *and* $s \leftarrow \alpha - z\rho \mod p$ *and returns* (z, s)*.*

Guess: *Finally,* $\mathcal{A}$ *outputs a guess* $b' \in \{0, 1\}$*. If* $b = b'$ *holds, then* $\mathcal{A}$ *wins this game.*

The advantage of $\mathcal{A}$ *is defined as*

$$\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A}) = \left|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]\right|.$$

For any computationally unbounded distinguisher $\mathcal{A}$*,* $\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A}) = 0$ *holds.*

Before we prove Lemma 4.7, we explain the intuition of the proof.

To prove this lemma, we should prove that, in $\mathsf{Game}_{\mathrm{eqv}}$, the statistical distance between the distribution of a distinguisher's view in the case where $b = 0$ and the distribution of a distinguisher's view in the case where $b = 1$ is equal to 0. However, it is hard to prove it directly because a distinguisher can *concurrently* access *stateful* oracles.

To overcome this difficulty, we prove Lemma 4.7 step by step. We resolve the difficulty arising from concurrently accessing by using the hybrid argument. To carry out this strategy, we consider the intermediate game $\mathsf{Game}_{\mathrm{eqv},k}$ in which a distinguisher is allowed to access the *stateful* oracles that switch behavior on the k-th query. Moreover, to evaluate the advantage of a distinguisher in this game, we consider the simple game $\mathsf{Game}_{\mathrm{eqv0}}$ in which a distinguisher needs to make all queries to the interactive oracles first of all the game. We prove the advantage of a distinguisher in $\mathsf{Game}_{\mathrm{eqv0}}$ is 0 (in Lemma 4.8), and by using this, we prove the advantage of a distinguisher in $\mathsf{Game}_{\mathrm{eqv},k}$ is also 0.

Now we start the proof of Lemma 4.7. First, we prove the following lemma.

Lemma 4.8. *We consider the following game* $\mathsf{Game}_{\mathrm{eqv0}}$ *between a challenger and a distinguisher* $\mathcal{A}$*.*

Setup: *The challenger chooses* $(\mathbb{G}, q, G)$*. It sends* $(\mathbb{G}, q, G)$ *to* $\mathcal{A}$ *and receives* $H \in \mathbb{G}$ *and* $x, \rho, c \in \mathbb{Z}_q$ *from* $\mathcal{A}$*. It computes* $(Y, Z)^T \leftarrow x(G, H)^T$ *and* $(U_1, U_2)^T \leftarrow \rho(G, H)^T + (Y, Z)^T$*. It chooses a bit* $b \xleftarrow{\$} \{0, 1\}$*. It allows* $\mathcal{A}$ *to access to the following oracle only once.*

Oracle $\Sigma_{\mathrm{eqv0}}(b)$**:** *The challenger responds as follows.*

- **Case** $b = 0$**:** *The challenger chooses* $r, z \xleftarrow{\$} \mathbb{Z}_q$*, computes* $T \leftarrow z(U_1, U_2)^T + r(G, H)^T$ *and* $s \leftarrow xc + r \mod q$ *and returns* (T, z, s)*.*
- **Case** $b = 1$**:** *The challenger chooses* $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$ *and computes* $T \leftarrow \alpha(G, H)^T + \beta(Y, Z)$*,* $z \leftarrow \beta + c \mod q$*, and* $s \leftarrow \alpha - z\rho \mod q$ *and returns* (T, z, s)*.*

Guess: *Finally,* $\mathcal{A}$ *outputs a guess* $b' \in \{0, 1\}$*. If* $b = b'$ *holds, then* $\mathcal{A}$ *wins this game.*

The advantage of $\mathcal{A}$ *is defined as*

$$\mathrm{Adv}_{\mathrm{eqv0}}(\mathcal{A}) = \left|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]\right|.$$

For any computationally unbounded algorithm $\mathcal{A}$*,* $\mathrm{Adv}_{\mathrm{eqv0}}(\mathcal{A}) = 0$ *holds.*

proof of Lemma 4.8. For $W \in \{V \in \mathbb{G}^2 | V = v(G, H)^T, v \in \mathbb{Z}_q\}$, let $\log_{(G,H)} W$ be the element $w \in \mathbb{Z}_q$ s.t. $W = w(G, H)^T$. Below, we write Σ_{eqv0}'s response (T, z, s) using matrices and vectors with $\mathbb{Z}_q$ coefficients.

- In the case $b = 0$, the response of Σ_{eqv0} satisfies $T = z(U_1, U_2)^T + r(G, H)^T = (r + (\rho + x)z)(G, H)^T$ and $s = xc + r \pmod q$ where $r, z \xleftarrow{\$} \mathbb{Z}_q$. Thus, we obtain

$$\begin{pmatrix} \log_{(G,H)} T \\ z \\ s \end{pmatrix} = \begin{pmatrix} 1 & \rho + x \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r \\ z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ xc \end{pmatrix}. \tag{1}$$

- In the case $b = 1$, the response of Σ_{eqv0} satisfies $T = \alpha(G,H)^T + \beta(Y,Z)^T = (\alpha + \beta x)(G,H)^T$, $z = \beta + c \pmod q$ and $s = \alpha - z\rho \pmod q$ where $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$. Thus, we obtain

$$\begin{pmatrix} \log_{(G,H)} T \\ z \\ s \end{pmatrix} = \begin{pmatrix} 1 & x \\ 0 & 1 \\ 1 & -\rho \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} + \begin{pmatrix} 0 \\ c \\ -c\rho \end{pmatrix}. \tag{2}$$

The advantage of $\mathcal{A}$ in $\mathsf{Game}_{\mathrm{eqv0}}$ is equal to the statistical distance between the distribution of the response of Σ_{eqv0} in the case $b = 0$ and that in the case $b = 1$. Therefore, to prove $\mathrm{Adv}_{\mathrm{eqv0}}(\mathcal{A}) = 0$ for any computationally unbounded algorithm $\mathcal{A}$, we prove that the distribution of $(\log_{(G,H)} T, z, s)^T$ in Eq. (1) is identical to that in Eq. (2) when $r, z \xleftarrow{\$} \mathbb{Z}_q$ and $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$.

For a matrix C, let $\mathrm{Im}(C)$ denote the column space of C. Let D_0 and D_1 be the column spaces as follows.

$$D_0 = \mathrm{Im} \begin{pmatrix} 1 & \rho + x \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad D_1 = \mathrm{Im} \begin{pmatrix} 1 & x \\ 0 & 1 \\ 1 & -\rho \end{pmatrix}.$$

Note that the distribution of $(\log_{(G,H)} T, z, s)^T$ in Eq. (1) and that in Eq. (2) are identical if and only if

$$D_0 + (0, 0, xc)^T = D_1 + (0, c, -c\rho)^T$$

holds, where the above equality means the equality of the left and the right affine subspaces. Furthermore, the above equality holds when the followings hold.

- $D_0 = D_1$.
- $(0, 0, xc)^T - (0, c, -c\rho)^T \in D_0$.

Now, we prove $D_0 = D_1$ by showing $D_0 \subseteq D_1$ and $D_1 \subseteq D_0$. For any $d_0 \in D_0$, we can write d_0 as follows:

$$\begin{aligned} d_0 &= r \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + z \begin{pmatrix} \rho + x \\ 1 \\ 0 \end{pmatrix} \\ &= r \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + z \begin{pmatrix} \rho \\ 0 \\ \rho \end{pmatrix} - z \begin{pmatrix} \rho \\ 0 \\ \rho \end{pmatrix} + z \begin{pmatrix} \rho + x \\ 1 \\ 0 \end{pmatrix} \\ &= (r + z\rho) \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + z \begin{pmatrix} x \\ 1 \\ -\rho \end{pmatrix} \in D_1. \end{aligned}$$

where $r, z \in \mathbb{Z}_q$. Thus, any $d_0 \in D_0$ is in D_1. This implies $D_0 \subseteq D_1$. On the other hand, for any $d_1 \in D_1$, we can write d_1 as follows:

$$\begin{aligned} d_1 &= \alpha \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \beta \begin{pmatrix} x \\ 1 \\ -\rho \end{pmatrix} \\ &= \alpha \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} - \beta \begin{pmatrix} \rho \\ 0 \\ \rho \end{pmatrix} + \beta \begin{pmatrix} \rho \\ 0 \\ \rho \end{pmatrix} + \beta \begin{pmatrix} x \\ 1 \\ -\rho \end{pmatrix} \\ &= (\alpha - \beta\rho) \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \beta \begin{pmatrix} \rho + x \\ 1 \\ 0 \end{pmatrix} \in D_0. \end{aligned}$$

where $\alpha, \beta \in \mathbb{Z}_q$. Thus, any $d_1 \in D_1$ is in D_0. This implies $D_1 \subseteq D_0$.

Next, we show $(0,0,xc)^T - (0,c,-c\rho)^T \in D_0$. This holds because, for $(0,0,xc)^T$ and $(0,c,-c\rho)^T$, we have

$$\begin{aligned}\begin{pmatrix}0\\0\\xc\end{pmatrix} - \begin{pmatrix}0\\c\\-c\rho\end{pmatrix} &= \begin{pmatrix}0\\0\\xc\end{pmatrix} + \begin{pmatrix}c(\rho+x)\\0\\0\end{pmatrix} - \begin{pmatrix}c(\rho+x)\\0\\0\end{pmatrix} - \begin{pmatrix}0\\c\\-c\rho\end{pmatrix} \\ &= c(x+\rho)\begin{pmatrix}1\\0\\1\end{pmatrix} - c\begin{pmatrix}\rho+x\\1\\0\end{pmatrix} \in D_0.\end{aligned}$$

□

Now we show Lemma 4.7 from the above lemma.

Proof of Lemma 4.7. We consider the following $\mathsf{Game}_{\mathrm{eqv},k}$ where $k \in [1, q_S]$.

Setup: The challenger chooses $(\mathbb{G}, q, G)$. It sends $(\mathbb{G}, q, G)$ to $\mathcal{A}_{\mathrm{eqv},k}$ and receives $H \in \mathbb{G}$ and $x \in \mathbb{Z}_q$ from $\mathcal{A}_{\mathrm{eqv},k}$. It computes $(Y,Z)^T \leftarrow x(G,H)^T$. It initializes tables $T_S[\cdot]$. It chooses a bit $b \xleftarrow{\$} \{0,1\}$.

Oracles: The challenger allows $\mathcal{A}_{\mathrm{eqv},k}$ to access to the following oracles concurrently at most q_S times. Note that $\mathcal{A}$ is allowed to make only one query for each session identifier I, which is included in each query to oracles.

$\Sigma_{\mathrm{eqv1},k}(b,\cdot,\cdot)$**:** As a query, the challenger receives a session identifier I and $\rho \in \mathbb{Z}_q$. It computes $(U_1,U_2)^T \leftarrow \rho(G,H)^T + (Y,Z)^T$. It responds as follows.

Case $I > k$ **or** $(I = k) \wedge (b = 0)$**:** The challenger responds as $\Sigma_{\mathrm{eqv1}}(0,\cdot,\cdot)$ in $\mathsf{Game}_{\mathrm{eqv}}$.

Case $I < k$ **or** $(I = k) \wedge (b = 1)$**:** The challenger responds as $\Sigma_{\mathrm{eqv1}}(1,\cdot,\cdot)$ in $\mathsf{Game}_{\mathrm{eqv}}$.

$\Sigma_{\mathrm{eqv2},k}(b,\cdot,\cdot)$ The challenger receives a session identifier I and $c \in \mathbb{Z}_q$ as a query. If $T_S[I]$ is empty, then it return $\perp$. Otherwise, it responds as follows.

Case $I > k$ **or** $(I = k) \wedge (b = 0)$**:** The challenger responds as $\Sigma_{\mathrm{eqv2}}(0,\cdot,\cdot)$ in $\mathsf{Game}_{\mathrm{eqv}}$.

Case $I < k$ **or** $(I = k) \wedge (b = 1)$**:** The challenger responds as $\Sigma_{\mathrm{eqv2}}(1,\cdot,\cdot)$ in $\mathsf{Game}_{\mathrm{eqv}}$.

Guess: Finally, $\mathcal{A}_{\mathrm{eqv},k}$ outputs a guess $b' \in \{0,1\}$. If $b = b'$ holds, then $\mathcal{A}_{\mathrm{eqv},k}$ wins this game.

Then, the advantage of $\mathcal{A}_{\mathrm{eqv},k}$ in the above game is defined as

$$\mathrm{Adv}_{\mathrm{eqv},k}(\mathcal{A}_{\mathrm{eqv},k}) = \left|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]\right|.$$

We show that $\mathrm{Adv}_{\mathrm{eqv},k}(\mathcal{A}_{\mathrm{eqv},k}) = 0$ for any computationally unbounded algorithm $\mathcal{A}_{\mathrm{eqv},k}$ and any $k \in [1, q_S]$ from Lemma 4.8. To show this, we construct a distinguisher $\mathcal{A}_{\mathrm{eqv0}}$ in $\mathsf{Game}_{\mathrm{eqv0}}$ in Lemma 4.8 from $\mathcal{A}_{\mathrm{eqv},k}$ as follows.

Setup: $\mathcal{A}_{\mathrm{eqv0}}$ receives $(\mathbb{G}, q, G)$, sends it to $\mathcal{A}_{\mathrm{eqv},k}$ and receives (H, x) from $\mathcal{A}_{\mathrm{eqv},k}$. It computes $(Y,Z)^T \leftarrow x(G,H)^T$. It initializes a table $T_S[\cdot]$ to $\emptyset$.

Oracles: The challenger allows $\mathcal{A}_{\mathrm{eqv},k}$ to access to the following oracles concurrently at most q_S times.

$\Sigma'_{\mathrm{eqv1},k}(\cdot,\cdot,\cdot)$**:** $\mathcal{A}_{\mathrm{eqv0}}$ receives (I,ρ) as a query and computes $(U_1,U_2)^T \leftarrow \rho(G,H)^T + (Y,Z)^T$. It responds as follows.

Case $I > k$**:** $\mathcal{A}_{\mathrm{eqv0}}$ responds as $\Sigma_{\mathrm{eqv1}}(0,\cdot,\cdot)$ in $\mathsf{Game}_{\mathrm{eqv}}$.

Case $I < k$**:** $\mathcal{A}_{\mathrm{eqv0}}$ responds as $\Sigma_{\mathrm{eqv1}}(1,\cdot,\cdot)$ in $\mathsf{Game}_{\mathrm{eqv}}$.

Case $I = k$**:** $\mathcal{A}_{\mathrm{eqv0}}$ chooses $c' \xleftarrow{\$} \mathbb{Z}_q$ and outputs (H, x, ρ, c'). It obtains (T, z, s) by accessing to Σ_{eqv0}, stores $T_S[I] \leftarrow (z, s, c')$ and returns T.

$\Sigma'_{\mathrm{eqv2},k}(\cdot,\cdot)$**:** $\mathcal{A}_{\mathrm{eqv0}}$ receives (I,c) as a second query. If $T_S[I]$ is empty, it returns $\perp$. Otherwise, it responds as follows.

Case $I > k$**:** $\mathcal{A}_{\mathrm{eqv0}}$ responds as $\Sigma_{\mathrm{eqv2}}(0,\cdot,\cdot)$ in $\mathsf{Game}_{\mathrm{eqv}}$.

Case $I < k$**:** $\mathcal{A}_{\mathrm{eqv0}}$ responds as $\Sigma_{\mathrm{eqv2}}(1,\cdot,\cdot)$ in $\mathsf{Game}_{\mathrm{eqv}}$.

Case $I = k$**:** $\mathcal{A}_{\mathrm{eqv0}}$ looks up (z,s,c) from $T_S[I]$. If $c \neq c'$, then it halts with output 0. Otherwise, it returns (z,s)

Output: Eventually, it receives a guess b' from $\mathcal{A}_{\mathrm{eqv},k}$ and returns b'.

$\mathcal{A}_{\mathrm{eqv0}}$ outputs $\mathcal{A}_{\mathrm{eqv},k}$'s guess b' if it does not halt because of $c = c'$ in $\Sigma'_{\mathrm{eqv2},k}$ in the case where $I = k$. So, we get the following equation.

$$\mathrm{Adv}_{\mathrm{eqv0}}(\mathcal{A}_{\mathrm{eqv0}}) = |\Pr[b' = 1 \wedge c = c' | b = 1] - \Pr[b' = 1 \wedge c = c' | b = 0]| \tag{3}$$

where b is a bit which Σ_{eqv0} has.

Since Σ_{eqv0} generates T without using c' in both cases where $b = 0$ and $b = 1$, $\mathcal{A}_{\mathrm{eqv},k}$ obtain no information about c' before $\mathcal{A}_{\mathrm{eqv},k}$ makes a query (k,c) to $\Sigma'_{\mathrm{eqv2},k}$. Also, c' is uniformly chosen from $\mathbb{Z}_q$. Therefore, we have $\Pr[c = c' | b = 1] = \Pr[c = c' | b = 0] = 1/q$. Then, we obtain

$$\begin{aligned} &|\Pr[b' = 1 \wedge c = c' | b = 1] - \Pr[b' = 1 \wedge c = c' | b = 0]| \\ &= \frac{1}{q} \left| \Pr[b' = 1 | c = c' \wedge b = 1] - \Pr[b' = 1 | c = c' \wedge b = 0] \right|. \end{aligned} \tag{4}$$

In the k-th query to $\Sigma'_{\mathrm{eqv1},k}$, conditioned on $c = c'$, Σ_{eqv0} generates (T,z,s) in the same way to Σ_{eqv1} and Σ_{eqv2}. Thus, for all $b \in \{0,1\}$, conditioned on $c = c'$, the distribution of the responses of $\Sigma'_{\mathrm{eqv1},k}$ and $\Sigma'_{\mathrm{eqv2},k}$ is identical to the distribution of the responses of $\Sigma_{\mathrm{eqv1},k}(b,\cdot,\cdot)$ and $\Sigma_{\mathrm{eqv1},k}(b,\cdot,\cdot)$, respectively. Then, from Eqs. (3) and (4), we have

$$\mathrm{Adv}_{\mathrm{eqv0}}(\mathcal{A}_{\mathrm{eqv0}}) = \frac{1}{q} \mathrm{Adv}_{\mathrm{eqv},k}(\mathcal{A}_{\mathrm{eqv},k}). \tag{5}$$

From Lemma 4.8, $\mathrm{Adv}_{\mathrm{eqv0}}(\mathcal{A}_{\mathrm{eqv0}}) = 0$ holds. Therefore, for any computationally unbounded algorithm $\mathcal{A}_{\mathrm{eqv},k}$ and any $k \in [1, q_S]$, $\mathrm{Adv}_{\mathrm{eqv},k}(\mathcal{A}_{\mathrm{eqv},k}) = 0$.

From here, we evaluate $\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A})$, which is the advantage of $\mathcal{A}$ in $\mathsf{Game}_{\mathrm{eqv}}$. By the hybrid argument, we get

$$\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A}) \leq \sum_{i=1}^{q_S} \mathrm{Adv}_{\mathrm{eqv},k}(\mathcal{A}).$$

Since $\mathrm{Adv}_{\mathrm{eqv},k}(\mathcal{A}) = 0$ for all $k \in [1, q_S]$, we have $\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A}) \leq 0$. Also $\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A}) \geq 0$ because the advantage is a non-negative real number. Therefore, we obtain $\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A}) = 0$. □

Now we prove Proposition 4.6 by using this lemma.

proof of Proposition 4.6. We construct a distinguisher $\mathcal{A}_{\mathrm{eqv}}$ in the game $\mathsf{Game}_{\mathrm{eqv}}$ in Lemma 4.7 from $\mathcal{F}$ as follows.

Setup: $\mathcal{A}_{\mathrm{eqv}}$ receives $(\mathbb{G}, q, G)$. It chooses $H \xleftarrow{\$} \mathbb{G}$ and assigns $(\mathbb{G}, q, G, H)$ to $(\mathbb{G}, q, G, H)$ of a public parameter pp. It sets up pk and tables as in Game_3 or Game_4. It returns (H, sk). It initializes a counter $ctr \leftarrow 1$. It runs $\mathcal{F}$ on inputs pp and pk.

Random Oracle $H_{\mathrm{ck}}(m)$**:** $\mathcal{A}_{\mathrm{eqv}}$ responds as in Game_3 or Game_4.

Random Oracle $H_{\mathrm{c}}(\widetilde{T}, \widetilde{pk}, m)$ **:** $\mathcal{A}_{\mathrm{eqv}}$ responds as in Game_3 or Game_4.

Random Oracle $H_{\mathrm{agg}}((Y,Z),L)$: $\mathcal{A}_{\mathrm{eqv}}$ responds as in Game_3 or Game_4.

Signing Oracle: In the case where $T_{\mathrm{b}}[m]=0$, $\mathcal{A}_{\mathrm{eqv}}$ responds as in Game_3 or Game_4. In the case where $T_{\mathrm{b}}[m]=1$, instead of $\mathcal{S}(pp,i,sk,L,m)$, it executes **Round 1** and **Round 2** as follows.

Round 1: $\mathcal{A}_{\mathrm{eqv}}$ generates $\widetilde{pk}$ as in $\mathcal{S}(pp,i,sk,L,m)$. It looks up ρ from $T_{\mathrm{td}}[m]$. For all $i\in n_h$, it repeats the following steps. It sets $I_i \leftarrow ctr$, obtains T_i by querying (I_i,ρ) to Σ_{eqv1}, computes $ctr \leftarrow ctr+1$. After completing steps for all $i\in n_h$, it stores $T_\Sigma[I_s] \leftarrow (m,L,\widetilde{pk},n_h,\{(T_i,t_i,I_i)\}_{i\in n_h})$ and returns $\{T_i\}_{i\in n_h}$.

Round 2: $\mathcal{A}_{\mathrm{eqv}}$ looks up $(m,L,\widetilde{pk},n_h,\{(T_i,t_i,I_i)\}_{i\in n_h})$ from $T_\Sigma[I_s]$, computes $\widetilde{T}$ and c as in $\mathcal{S}(pp,i,sk,L,m)$. For all $i\in n_h$, it obtains (s_i,z_i) by querying (I_i,t_ic) to Σ_{eqv2}. It returns $\{(z_i,s_i)\}_{i\in n_h}$.

Output: $\mathcal{A}_{\mathrm{eqv}}$ returns 1 if $\mathcal{F}$ wins the game. Otherwise, it returns 0.

For **Signing Oracle** of $\mathcal{A}_{\mathrm{eqv}}$, the distribution of responses is the same as in Game_3 when $b=0$ where b is a bit chosen by the challenger in $\mathsf{Game}_{\mathrm{eqv}}$. That also is the same as that in Game_4 when $b=1$. Thus, we have

$$\begin{aligned}\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A}_{\mathrm{eqv}}) &= |\Pr[b'=1|b=0]-\Pr[b'=1|b=1]| \\ &= |\Pr[\mathsf{Game}_3=1]-\Pr[\mathsf{Game}_4=1]|\end{aligned}$$

where b' is $\mathcal{A}_{\mathrm{eqv}}$'s output. From Lemma 4.7, we have $\mathrm{Adv}_{\mathrm{eqv}}(\mathcal{A}_{\mathrm{eqv}})=0$. Thus, we obtain $\Pr[\mathsf{Game}_3=1]=\Pr[\mathsf{Game}_4=1]$. □

Proposition 4.9. *If $\mathbb{G}$ is a (t',ε')-DDH group, then*

$$\begin{aligned}&|\Pr[\mathsf{Game}_4=1]-\Pr[\mathsf{Game}_5=1]|\le\varepsilon', \text{ and}\\ &t_{\mathcal{F}}\ge t'-(3q_H+6q_SN+3q_S+2N+6)t_{\mathrm{mul}}\\ &\qquad\qquad -O(q_H+q_SN)\end{aligned}$$

where t_{mul} is the time for a scalar multiplication in $\mathbb{G}$.

proof of Proposition 4.9. We construct a distinguisher $\mathcal{A}'_{\mathrm{DDH}}$ who solves the DDH problem by running $\mathcal{F}$ internally as follows.

Setup: $\mathcal{A}'_{\mathrm{DDH}}$ receives $(G,H,Y,Z)\in\mathbb{G}^4$. For a public parameter pp, it uses H instead of uniformly choosing H from $\mathbb{G}$. For a public key pk, it assigns $(pk,sk)\leftarrow((Y,Z),\perp)$. The rest of pp and tables are set up as in Game_4 or Game_5. It runs $\mathcal{F}$ on inputs pp and pk.

Random Oracle $H_{\mathrm{ck}}(m)$**:** $\mathcal{A}'_{\mathrm{DDH}}$ responds as in Game_4 or Game_5.

Random Oracle $H_{\mathrm{c}}(\widetilde{T},\widetilde{pk},m)$: $\mathcal{A}'_{\mathrm{DDH}}$ responds as in Game_4 or Game_5.

Random Oracle $H_{\mathrm{agg}}((Y,Z),L)$: $\mathcal{A}'_{\mathrm{DDH}}$ responds as in Game_4 or Game_5.

Signing Oracle: $\mathcal{A}'_{\mathrm{DDH}}$ responds as in Game_4 or Game_5.

Output: $\mathcal{A}'_{\mathrm{DDH}}$ returns 1 if $\mathcal{F}$ wins the game. Otherwise, it returns 0.

In the case where $T_{\mathrm{b}}[m]=0$, $\mathcal{A}'_{\mathrm{DDH}}$ can respond to **Signing Oracle** though $\mathcal{A}'_{\mathrm{DDH}}$ does not have the secret key because the secret key is unnecessary for executing **Round 1** and **Round 2** is not executed.

If (G,H,Y,Z) is a DH tuple, the distribution of pk is the same as that in Game_4. If (G,H,Y,Z) is a non-DH tuple, the distribution of pk is the same as that in Game_5. Then, we have

$$\begin{aligned}\mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}'_{\mathrm{DDH}}) &= |\Pr[b'=1|(G,H,P,Q)\text{ is a DH tuple}]-\Pr[b'=1|(G,H,P,Q)\text{ is a non-DH tuple}]|\\ &= |\Pr[\mathsf{Game}_4=1]-\Pr[\mathsf{Game}_5=1]|\end{aligned}$$

where b' is $\mathcal{A}'_{\mathrm{DDH}}$'s output. From the assumption that $\mathbb{G}$ is a (t', ε')-DDH group, we have $\mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}) \leq \varepsilon'$. Therefore, we obtain $|\Pr[\mathsf{Game}_4 = 1] - \Pr[\mathsf{Game}_5 = 1]| \leq \varepsilon'$.

Now we consider the running time of $\mathcal{A}'_{\mathrm{DDH}}$. In one signing query, there are at most $6N$ scalar multiplications, $O(N)$ other non-cryptographic operations, a query to H_{c}, a query to H_{ck}, and $O(N)$ queries to H_{agg}. For random oracle queries, we only evaluate the cost of H_{ck} because H_{ck} is more expensive than H_{c} and H_{agg}. In one query to H_{ck}, there are 2 scalar multiplications and $O(1)$ other non-cryptographic operations. In **Check**, it computes $2N + 6$ scalar multiplications and $O(N)$ other non-cryptographic operations. From these evaluations and the fact that $\mathcal{A}'_{\mathrm{DDH}}$ runs $\mathcal{F}$ once, we obtain $t_{\mathcal{F}} \geq t' - (3q_H + 6q_S N + 3q_S + 2N + 6)t_{\mathrm{mul}} - O(q_H + q_S N)$. □

Proposition 4.10. $\Pr[\mathsf{Game}_5 = 1] = \Pr[\mathsf{Game}_6 = 1]$

proof of Proposition 4.10. The difference between Game_5 and Game_6 is that, in Game_6, the challenger defines $H_{\mathrm{agg}}(L[i], L)$ for all $i \in [1, n]$. Since the challenger gives $\mathcal{F}$ only $T_{\mathrm{agg}}[(Y, Z), L]$, where $((Y, Z), L)$ is queried, this change does not affect the probability of $\mathcal{F}$ winning the game. Thus, we have $\Pr[\mathsf{Game}_5 = 1] = \Pr[\mathsf{Game}_6 = 1]$. □

Proposition 4.11. $\Pr[\mathsf{Game}_6 = 1] \leq (2q_H + q_S + 2)/q$

proof of Proposition 4.11. At the end of Game_6, $\mathcal{F}$ outputs m^*, L^*, and $\tilde{\sigma}^* = (c^*, \tilde{z}^*, \tilde{s}^*)$. If $\mathcal{F}$ wins, then $(Y^*, Z^*) \in L^*$, $T_{\mathrm{b}}[m^*] = 0$, and $\tilde{\sigma}^*$ is a valid forgery on m^* under $(U_1, U_2) = H_{\mathrm{ck}}(m^*)$. Then, there exists $c^* = H_{\mathrm{c}}(\widetilde{T}^*, \widetilde{pk}^*, m^*)$ in T_{c} s.t.

(a) $\widetilde{T}^* = \tilde{z}^*(U_1, U_2)^T + \tilde{s}^*(G, H)^T - c^* \cdot \widetilde{pk}^*$,

(b) $\widetilde{pk}^*$ is the aggregated key computed from L^*,

(c) $(U_1, U_2)^T = \rho^*(G, H)^T$.

Below, we show that $\mathcal{F}$ can make such a query with probability at most $(2q_H + q_S + 2)/q$.

To evaluate the probability, we rewrite the right-hand of the equation in (a) by $(G, H)^T$ and $(Y^*, Z^*)^T$. Since (G, H, Y^*, Z^*) in Game_6 is a non-DH tuple, $(G, H)^T$ and $(Y^*, Z^*)^T$ are linearly independent. Then, we can denote the aggregated key $\widetilde{pk}^*$ as $\phi^* (G, H)^T + \psi^* (Y^*, Z^*)^T$ where $\phi^*, \psi^* \in \mathbb{Z}_q$. Substituting the above and (c) in the equation in (a), we have

$$\widetilde{T}^* = (\tilde{z}^*\rho^* + \tilde{s}^* - c^*\phi^*)(G, H)^T - c^*\psi^* (Y^*, Z^*)^T . \tag{6}$$

Since $(G, H)^T$ and $(Y^*, Z^*)^T$ are linearly independent, the values of coefficients $(\tilde{z}^*\rho^* + \tilde{s}^* - c^*\phi^*)$ and $c^*\psi^*$ which make Eq. (6) hold are uniquely determined at the point where $(\widetilde{T}^*, \widetilde{pk}^*, m^*)$ is queried to H_{c}. Moreover, the values of ϕ^* and ψ^* are uniquely determined at the same point since the query includes $\widetilde{pk}^*$. For the coefficient $c^*\psi^*$, c^* is determined by H_{c} and ψ^* is also determined by H_{agg}.

Here, we evaluate $\Pr[\mathsf{Game}_6 = 1]$. For $R \in \mathbb{G}^2$, let $\phi(R)$ and $\psi(R)$ be the elements in $\mathbb{Z}_q$ s.t. $R = \phi(R)(G, H)^T + \psi(R)(Y^*, Z^*)^T$. Let E_{agg} be the event where there exists at least one random oracle query $H_{\mathrm{agg}}((Y', Z'), L')$ s.t. $(Y^*, Z^*) \in L'$ and $\psi(\widetilde{pk}') = 0$ for the aggregated key $\widetilde{pk}'$ computed from L'. Then, we have

$$\begin{aligned} \Pr[\mathsf{Game}_6 = 1] &= \Pr[\mathsf{Game}_6 = 1 \wedge E_{\mathrm{agg}}] + \Pr[\mathsf{Game}_6 = 1 \wedge \overline{E_{\mathrm{agg}}}] \\ &\leq \Pr[E_{\mathrm{agg}}] + \Pr[\mathsf{Game}_6 = 1 \wedge \overline{E_{\mathrm{agg}}}]. \end{aligned} \tag{7}$$

Also, let E_{chal} be the event where there exists at least one random oracle query $c' = H_{\mathrm{c}}(\widetilde{T}', \widetilde{pk}', m')$ s.t. $\psi(\widetilde{pk}') \neq 0$, $T_{\mathrm{b}}[m'] = 0$, and $\psi(\widetilde{T}') = c'\psi(\widetilde{pk}')$. If $\mathsf{Game}_6 = 1$ occurs, $T_{\mathrm{b}}[m^*] = 0$ holds from the winning conditions. Also, if $\mathsf{Game}_6 = 1$ occurs, there exists at least one random oracle query to H_{c} making $\psi(\widetilde{T}^*) =$

$c^*\psi(\widetilde{pk}^*)$ hold since $\mathcal{F}$'s output satisfies Eq. (6). There is no query $H_{\mathrm{agg}}((Y', Z'), L')$ s.t. $(Y^*, Z^*) \in L'$ and $\psi(\widetilde{pk}') = 0$ when $\overline{E_{\mathrm{agg}}}$ occurs. Thus, if $\overline{E_{\mathrm{agg}}}$ occurs, then $\psi(\widetilde{pk}^*) \neq 0$ holds. Therefore, if $\mathsf{Game}_6 = 1$ and $\overline{E_{\mathrm{agg}}}$ occur, then E_{chal} occurs. Then, we have $\Pr[\mathsf{Game}_6 = 1 \wedge \overline{E_{\mathrm{agg}}}] \leq \Pr[E_{\mathrm{chal}}]$. Applying this fact to Eq. (7), we obtain

$$\Pr[\mathsf{Game}_6 = 1] \leq \Pr[E_{\mathrm{agg}}] + \Pr[E_{\mathrm{chal}}]. \tag{8}$$

First, we evaluate $\Pr[E_{\mathrm{agg}}]$. For an aggregate key $\widetilde{pk}'$ computed from L', $\psi(\widetilde{pk}') = \sum_{i=1}^{n'} t'_i \psi(L'[i])$ holds where n' is the number of the public keys in L' and $t'_i = H_{\mathrm{agg}}(L'[i], L')$. Since the challenger defines the value t'_i for all $i \in [1, n']$ when L' is first queried to H_{agg}, $\{t'_i\}_{i=1}^{n'}$ is uniformly chosen from $\mathbb{Z}_q^{n'}$ after $\{\psi(L'[i])\}_{i=1}^{n'}$ is fixed. If L' includes (Y^*, Z^*), there exists at least one i s.t. $\psi(L'[i]) \neq 0$. Thus, per one query to H_{agg}, $\sum_{i=1}^{n'} t'_i \psi(L'[i]) = 0$ holds with probability at most $1/q$. Since at most $q_H + q_S + 1$ public key lists appear in T_{agg}, we obtain

$$\Pr[E_{\mathrm{agg}}] \leq (q_H + q_S + 1)/q. \tag{9}$$

Next, we evaluate $\Pr[E_{\mathrm{chal}}]$. Let $E_{\mathrm{chal},j}$ be the event where the j-th random oracle query $c'_j = H_{\mathrm{c}}(\widetilde{T}'_j, \widetilde{pk}'_j, m'_j)$ satisfies following conditions.

$$E_{j,1} : \psi(\widetilde{pk}'_j) \neq 0,\ E_{j,2} : T_{\mathrm{b}}[m'_j] = 0,\ \text{and}\ E_{j,3} : \psi(\widetilde{T}'_j) = c'_j \psi(\widetilde{pk}'_j).$$

Note that there are at most $q_H + 1$ queries $H_{\mathrm{c}}(\widetilde{T}', \widetilde{pk}', m')$ s.t. $T_{\mathrm{b}}[m'] = 0$. From this fact and the union bound, we have

$$\begin{aligned}
\Pr[E_{\mathrm{chal}}] &\leq \sum_{i=1}^{q_H+1} \Pr[E_{\mathrm{chal},j}] \\
&= \sum_{i=1}^{q_H+1} \Pr[E_{j,1} \wedge E_{j,2} \wedge E_{j,3}] \\
&\leq \sum_{i=1}^{q_H+1} \Pr[E_{j,3} | E_{j,1} \wedge E_{j,2}].
\end{aligned} \tag{10}$$

As described previously, at the point where $(\widetilde{T}'_j, \widetilde{pk}'_j, m'_j)$ is queried to H_{c}, the value of $\psi(\widetilde{T}'_j)$ and $\psi(\widetilde{pk}'_j)$ are fixed. Also, conditioned on $E_{j,1}$ $\psi(\widetilde{pk}'_j) \neq 0$ holds. Thus, conditioned on $E_{j,1}$ and $E_{j,2}$, before c'_j is chosen, c'_j making $\psi(\widetilde{T}'_j) = c'_j \psi(\widetilde{pk}'_j)$ hold is determined uniquely. Since c'_j is uniformly chosen from $\mathbb{Z}_q$ independently of the j-th random oracle query, $\Pr[E_{j,3} | E_{j,1} \wedge E_{j,2}]$ is at most $1/q$. From this and Eq. (10), we have

$$\Pr[E_{\mathrm{chal}}] \leq \sum_{i=1}^{q_H+1} 1/q = (q_H + 1)/q. \tag{11}$$

From Eqs. (8), (9), and (11), we obtain

$$\Pr[\mathsf{Game}_6 = 1] \leq (2q_H + q_S + 2)/q.$$

□

5 Performance Comparison

In this section, we compare our scheme with other related two-round multi-signature schemes, which are proven secure in the PPK model based on the DL, DDH, or OMDL assumptions, e.g., MuSig2 [7], DWMS [8],

HBMS [9], LK [10], MuSig-DN [6], TZ [24], mBCJ [5], PW-1 [23], and PW-2 [23]. We remark the followings on HBMS and mBCJ. For HBMS, in [9], Bellare and Dai showed the security proof of HBMS under and without using the AGM. Especially, we call the former HBMS-AGM. For mBCJ, instead of the original mBCJ, we use a modified mBCJ which is proven secure *in the PPK model*. This is because the original mBCJ is proven secure in *the key verification model*. For more details, see B.

We compare the underlying group sizes for 128-bit security. Thus, we need to estimate the requirements of the sizes of the underlying groups considering the reduction loss under 128-bit security for all schemes. We also compare whether there exists the NIST standardized EC that enables a parameter choice with 128-bit security, which is called the recommended EC hereafter. The way to estimate the size of the underlying group considering the reduction loss for 128-bit security is described in Section 5.1. Table 1 summarizes the comparison.

5.1 Estimation of the Underlying Group Size

Here, we explain how to estimate $|q|_{128}$ which is the size of the underlying group $\mathbb{G}$ for 128-bit security.

We estimated $|q|_{128}$ by the following steps:

Step 1. We obtained inequalities $\varepsilon_P \geq B_p(\varepsilon_{\mathcal{F}}, q_s, q_H, N, q)$ and $t_P \leq B_t(t_{\mathcal{F}}, q_s, q_H, N, q)$ from the security proof, where B_p and B_t are functions derived by the security proof, ε_P and t_P are the success probability and the running time of an algorithm for solving an underlying problem P, respectively, and $\varepsilon_{\mathcal{F}}$ and $t_{\mathcal{F}}$ are the success probability and the running time of a forger, respectively.

Step 2. We derived the inequality $t_P/\varepsilon_P \leq B_t(t_{\mathcal{F}}, q_s, q_H, N, q)/B_p(\varepsilon_{\mathcal{F}}, q_s, q_H, N, q) =: B_{t/p}(t_{\mathcal{F}}, \varepsilon_{\mathcal{F}}, q_s, q_H, N, q)$ from the previous step.

Step 3. We solved $\sqrt{q} = B_{t/p}(2^{128}, 1, 2^{30}, 2^{80}, 2^{15}, q)$ for q and set $|q|_{128} \leftarrow \lceil \log_2 q \rceil$.[7]

In Step 3, we assumed $t_{DL}/\varepsilon_{DL} = t_{DDH}/\varepsilon_{DDH} = t_{OMDL}/\varepsilon_{OMDL} = \sqrt{q}$. This assumption is natural because of the following two facts. The first fact is that the best-known attack for solving the DDH problem and the OMDL problem is to solve the DL problem. The second one is that the known fastest algorithm for solving the DL problem is Pollard's ρ algorithm [45], which requires $O(\sqrt{q})$ scalar multiplications in $\mathbb{G}$. Also, in the same step, we consider the setting where $q_H = 2^{80}$, $q_s = 2^{30}$, and $N = 2^{15}$. We set $q_H = 2^{80}$ referring to a recent collision attack [46] to SHA-1 with complexity $2^{61.2}$ with a margin. We set $q_S = 2^{30}$ for a large scenario as in [47]. We set $N = 2^{15}$ for a large-scale setting.[8]

Remarks for Estimation. We estimate $|q|_{128}$ according to the steps described above and show the results of this estimation in Column 8 in Table 1. Here, we should remark on the following points for this estimation.

For MuSig2 ($\nu \geq 4$), we suppose $\nu = 4$ where ν is a unique parameter.

For MuSig2 and DWMS, we obtained B_p and B_t from [9, Appendix A]. For HBMS-AGM, we obtained B_p and B_t from [9, Theorem 7.1]. For LK, we obtained B_p and B_t from [10, Theorem 4.1]. For B_t of this, we suppose $t_P = t_{\mathcal{F}}$ because there is no evaluation of the running time of the reduction and the fact that the reduction runs a forger only one time. For MuSig-DN, we obtained B_p and B_t from [9, Appendix A]. For B_p and B_t of this scheme, the terms except for constants and the ones related to the DL assumption were ignored. For HBMS, we obtained B_p and B_t from [9, Theorem 3.2, 3.4, and 7.2]. For TZ, we obtained B_p and B_t from [24, Theorem 2]. For mBCJ, we obtain B_p and B_t from Theorem B.1. For PW-1 and PW-2, we obtained B_p and B_t from [23, Theorem 3.5 and 3.3], respectively.

For MuSig2 ($\nu = 2$), DWMS, HBMS-AGM, LK, and PW-1 the results of their estimation of $|q|_{128}$ are 257, 258, or 260. We chose the P-256 curve as the recommended EC, even though the order of this curve is slightly smaller for 128-bit security. We ignore the very small exceedance of the group size, whose effects on concrete security are small.

[7] To simplify the calculation, we ignore non-dominant terms in $B_{t/p}$.

[8] This large-scale setting had little effect on the estimation here because the terms related to N in $B_{t/p}$ are not dominant.

5.2 Comparison

We compare the efficiency of the related two-round multi-signature schemes in Table 1 under provably secure parameters.

First, we compare our scheme to the schemes having large reduction loss which are proven secure without using the AGM, i.e., MuSig2 ($\nu \geq 4$), MuSig-DN, HBMS, TZ, and mBCJ. Among these schemes, our scheme has the most efficient signature size and communication complexity. More concretely, $|\widetilde{\sigma}|_{128}$ of ours is reduced by about 22% from MuSig2 ($\nu \geq 4$) and MuSig-DN, about 60% from HBMS, and about 45% from TZ and mBCJ. Moreover, we can use NIST standardized P-384 to ensure 128-bit security for our scheme, while other schemes have no such standardized EC. These benefits are because the DDH assumption enables us to prove the security without the rewinding technique. However, remind that the DDH assumption is a stronger (not weaker) computational assumption than the DL assumption. For MuSig2 ($\nu \geq 4$), the AOMDL assumption is also stronger than the DL assumption. Multi-signatures of MuSig2 ($\nu \geq 4$) and MuSig-DN consist of only an element in $\mathbb{G}$ and an element in $\mathbb{Z}_q$, whose form is the same as the ordinary Schnorr signature. Thus, these schemes are more compatible with a currently deployed scheme than the other schemes. For MuSig2 ($\nu \geq 4$) and TZ, the first round of signing protocols can be executed before a message to be signed is determined.

Next, we compare our scheme to the schemes proven secure in the AGM, i.e., MuSig2 ($\nu = 2$), DWMS, HBMS-AGM, and LK. The signature size and the communication complexity of these schemes are more efficient than ours. Concretely, $|\widetilde{\sigma}|_{128}$ of our scheme is 2.2 times longer than MuSig2 ($\nu = 2$) and DWMS and 1.5 times longer than HBMS-AGM and LK. This is because these schemes are proven secure without rewinding by using AGM and achieve tight security.[9] Our scheme also does not require rewinding to prove the security because of the DDH assumption, while ours has the reduction loss yielded from the technique of the proof of the RSA-FDH signature scheme by Coron. Thus, $|q|_{128}$ of ours is larger than the other schemes. Note that our scheme does not require the AGM. For MuSig2 ($\nu = 2$) and DWMS, the signature size is most efficient among all the two-round schemes.

Finally, we compare our scheme to PW-1 and PW-2. To ensure 128-bit security, PW-1 can use P-256, and PW-2 and our scheme can use P-384. The signature size and communication complexity of our scheme are the most efficient among these schemes. The signature size of ours is reduced by about 67% and 40% from PW-1 and PW-2, respectively. The communication complexity of ours is reduced by about 57% and 41% from PW-1 and PW-2, respectively. PW-1 does not support key aggregation. All schemes are proven secure under the DDH assumption in the random oracle model. Thus, our scheme can be considered an improvement on PW-1 and PW-2 under provably secure parameters.

Conclusion of Comparison. The above comparison shows a trade-off between the efficiency and the strength of underlying assumptions and one between the efficiency and the necessity of the AGM.

Among schemes that do not need the AGM to prove their security, in concrete security, our scheme achieves the smallest signature size and the communication complexity. Moreover, our scheme has a recommended EC, i.e., P-384, for 128-bit security. This fact makes the implementation of our scheme easier because we do not need to design a new suitable EC.

6 Implementation Results

In this section, we explain our machine implementation of the proposed scheme and the evaluation of the running time of our implementation. The result of our evaluation shows that our proposed scheme can be implemented easily in a real-world environment with reasonable running time in practice. We show the detailed results of our evaluation in Table 2.

Environment. Our implementation is written in C++. We implemented our scheme by using the mcl library [48] and P-384 for the EC. We used g++ version 9.4.0 for compilation. We evaluated the running time of algorithms of our scheme on a computer provided with a 1.30GHz Intel(R) Core(TM) i7-1065G7 CPU and 16.0 GB of RAM and running WSL2 on Windows 10 Home version 21H2.

[9] HBMS-AGM can eliminate the reduction loss caused by the technique of Coron [42] due to the AGM. For more details, see [9, Appendix I].

Table 2: Execution Time Evaluation of Our Scheme under P-384 (in milliseconds).

	$N=3$	$N=5$	$N=10$	$N=15$	$N=50$	$N=100$
Key Generation.						
Kg	4.6×10^{-1}	4.7×10^{-1}	4.8×10^{-1}	4.9×10^{-1}	5.1×10^{-1}	5.2×10^{-1}
Signing Protocol.						
Round 1 (Computing $\widetilde{pk}$)	1.4	2.5	5.0	8.0	29	64
Round 1 (Others)	1.0	1.1	1.1	1.1	1.1	1.2
Round 2	1.7×10^{-2}	2.0×10^{-2}	2.7×10^{-2}	3.5×10^{-2}	9×10^{-2}	1.7×10^{-1}
Aggregate	1.8×10^{-4}	2.2×10^{-4}	3.0×10^{-4}	4.1×10^{-4}	1×10^{-3}	2×10^{-3}
Verification.						
Vf without $\widetilde{pk}$	3.0	4.1	6.9	9.6	31	66
Vf with $\widetilde{pk}$	1.5	1.6	1.6	1.6	1.7	1.7

Settings. Here, we describe the details of the setting of the evaluation. In Table 2, we show the average time of the 1000 loops of executions under a fixed public parameter. As a message to be signed, we generated a random alphabet string of 100 characters for each loop by using the command mt19937 in the random library. We set the size of a message as above considering the size of the hash value (256 bits) of a transaction to be signed in Bitcoin with a margin. We evaluated the running time for the setting where N are 3, 5, 10, 15, 50, and 100. The cases where N are 3, 5, 10, and 15 are the typical numbers of signers for Multi-Sig Wallets, and the cases where N are 50 and 100 are larger-scale settings, respectively.

We measured **Round 1** of the signing protocol in two phases. Specifically, one phase is computing the aggregated key $\widetilde{pk}$ from a public-key list L, and the other phase is computing other computations. For the verification, we measured the time for the verification algorithm without $\widetilde{pk}$ shown in Section 4.1 and for the one given an aggregated key $\widetilde{pk}$ instead of a public-key list L.

Results. The key generation took about 0.5 ms. This can be regarded as the time of two scalar multiplications in $\mathbb{G}$.

The total running times of whole algorithms in the signing protocol are about 2.4, 3.6, 6.1, and 9.1 ms under the settings $N=3$, 5, 10, and 15, respectively. For the settings where $N=50$ and $N=100$, those are about 30.1 ms and 65.2 ms, respectively. From these results, notice that the time of the scalar multiplication in $\mathbb{G}$ is a dominant factor for running time. There are $2N$ scalar multiplications in **Round 1** of the signing protocol for the computation of an aggregated key $\widetilde{pk}$. By precomputing $\widetilde{pk}$, **Round 1** took only about 1 ms because it needs 4 scalar multiplications irrelevantly to N. Since there is no scalar multiplication in **Round 2** and **Aggregate**, they were completed within 0.2 ms even when $N=100$.

For **Vf** without $\widetilde{pk}$, which is the normal verification, it was completed within 10 ms when $N=15$. Also, it took about 66 ms even when $N=100$. Since the verification needs only 6 scalar multiplications by using $\widetilde{pk}$, **Vf** with $\widetilde{pk}$ took about 1.6 ms irrelevantly to N.

The above result shows that each algorithm is completed within 100ms even when $N=100$. This can be regarded as sufficiently reasonable running time in practice.

Comparison. Here, we compare the computation time with those of PW-2, which can be implemented under P-384 for 128-bit security. For this comparison, we estimate the computation time of PW-2. From the result of the above implementation, we assume that one scalar multiplication in $\mathbb{G}$ takes 0.25 ms.

Since the key generation algorithm and the computation of the aggregated key in PW-2 are the same as ours, the computation time of these is identical to ours. Moreover, because the time of computation of the aggregated key is dominant when the aggregated key is not pre-computed, the computation time of PW-2 is close to ours in large-scale settings. For example, when N is 100, the computation time of an aggregated key is about 64 ms, which is as same as that of our scheme shown in Table 2. Below, we consider the case where the aggregated key is pre-computed. Because there are 11 scalar multiplications in **Round 1** except for the computation of the aggregated key, **Round 1** (Others) takes about 2.75 ms. This time is about 2.75 times greater longer than ours. Since the verification needs 13 scalar multiplications by using $\widetilde{pk}$, **Vf** with $\widetilde{pk}$ of PW-2 takes about 3.25 ms, which is more than twice as long as ours.

From the above comparison, our scheme improves the computation time compared to PW-2 in the setting when N is small.

Communication Time. In this part, we estimate the communication time of our scheme and PW-2 and compare them. We consider the case where each signer is connected to a hub by WAN. We suppose the WAN environment whose bandwidth is 100 Mbps and latency is 30 ms.

As the result of the estimation, latency is dominant even when $N = 100$ for both schemes. In other words, there is a small difference in the communication times between both schemes. Specifically, the communication time for each round of our scheme is about 61 ms, and the communication time for each round of PW-2 is about 62 ms. For both schemes, 60 ms of these communication times is the latency.

Here, we show how we derived communication times of both schemes. In our signing protocol, in the first round, each signer sends 2 elements in $\mathbb{G}$ to the hub and receives $2(N-1)$ elements in $\mathbb{G}$ from the hub, and in the second round, it sends 2 elements in $\mathbb{Z}_q$ to the hub and receives $2(N-1)$ elements in $\mathbb{Z}_q$ from the hub. Then, when N is 100, the communication time for the first round is $770/(100 \times 10^3) + 30 + 770 \times 99/(100 \times 10^3) + 30 \approx 61$ ms, and that for the second round is $768/(100 \times 10^3) + 30 + 768 \times 99/(100 \times 10^3) + 30 \approx 61$ ms. In PW-2, in the first round, each signer sends 3 elements in $\mathbb{G}$ to the hub and receives $3(N-1)$ elements in $\mathbb{G}$ from the hub, and in the second round, it sends 4 elements in $\mathbb{Z}_q$ to the hub and receives $4(N-1)$ elements in $\mathbb{Z}_q$ from the hub. Then, when N is 100, the communication time for the first round is $1155/(100 \times 10^3) + 30 + 1155 \times 99/(100 \times 10^3) + 30 \approx 62$ ms, and that for the second round is $1536/(100 \times 10^3) + 30 + 1536 \times 99/(100 \times 10^3) + 30 \approx 62$ ms.

Acknowledgements

This work was supported by JST CREST Grant Number JPMJCR22M1, Japan JST AIP Acceleration Research JPMJCR22U5, Japan, JSPS KAKENHI Grant Numbers JP18K11292, JP18K11293, JP18H01438, JP18K18055, JP19H01109, JP22KJ1366, JP23H00479.

References

[1] K. Itakura and K. Nakamura, "A public-key cryptosystem suitable for digital multisignatures," NEC research & development, 1983.

[2] S. Micali, K. Ohta, and L. Reyzin, "Accountable-subgroup multisignatures: extended abstract," CCS 2001, pp.245–254, ACM, 2001.

[3] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," CCS 2006, pp.390–399, ACM, 2006.

[4] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple schnorr multi-signatures with applications to bitcoin," Des. Codes Cryptogr., vol.87, no.9, pp.2139–2164, 2019.

[5] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs, "On the security of two-round multi-signatures," IEEE S&P 2019, pp.1084–1101, IEEE, 2019.

[6] J. Nick, T. Ruffing, Y. Seurin, and P. Wuille, "Musig-DN: Schnorr multi-signatures with verifiably deterministic nonces," CCS 2020, pp.1717–1731, ACM, 2020.

[7] J. Nick, T. Ruffing, and Y. Seurin, "Musig2: Simple two-round schnorr multi-signatures," CRYPTO 2021, LNCS, vol.12825, pp.189–221, Springer, 2021.

[8] H.K. Alper and J. Burdges, "Two-round trip schnorr multi-signatures via delinearized witnesses," CRYPTO 2021, LNCS, vol.12825, pp.157–188, Springer, 2021.

[9] M. Bellare and W. Dai, "Chain reductions for multi-signatures and the HBMS scheme," ASIACRYPT 2021, LNCS, vol.13093, pp.650–678, Springer, 2021.

[10] K. Lee and H. Kim, "Two-Round Multi-Signatures from Okamoto Signatures." Cryptology ePrint Archive, Paper 2022/1117, 2022.

[11] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," PKC 2003, LNCS, vol.2567, pp.31–46, Springer, 2003.

[12] A. Boldyreva, C. Gentry, A. O'Neill, and D.H. Yum, "Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing," CCS 2007, pp.276–285, ACM, 2007.

[13] D. Le, A. Bonnecaze, and A. Gabillon, "Multisignatures as secure as the diffie-hellman problem in the plain public-key model," Pairing-Based Cryptography-Pairing 2009, LNCS, vol.5671, pp.35–51, Springer, 2009.

[14] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracles," EUROCRYPT2006, LNCS, vol.4004, pp.465–485, Springer, 2006.

[15] T. Ristenpart and S. Yilek, "The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks," EUROCRYPT 2007, LNCS, vol.4515, pp.228–245, Springer, 2007.

[16] R.E. Bansarkhani and J. Sturm, "An efficient lattice-based multisignature scheme with applications to bitcoins," CANS 2016, LNCS, vol.10052, pp.140–155, Springer, 2016.

[17] C. Ma and M. Jiang, "Practical lattice-based multisignature schemes for blockchains," IEEE Access, vol.7, pp.179765–179778, 2019.

[18] M. Fukumitsu and S. Hasegawa, "A lattice-based provably secure multisignature scheme in quantum random oracle model," ProvSec 2020, LNCS, vol.12505, pp.45–64, Springer, 2020.

[19] I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi, "Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices," PKC 2021, LNCS, vol.12710, pp.99–130, Springer, 2021.

[20] C. Boschini, A. Takahashi, and M. Tibouchi, "Musig-L: Lattice-based multi-signature with single-round online phase," CRYPTO 2022, LNCS, vol.13508, pp.276–305, Springer, 2022.

[21] National institute of standards and technology, "FIPS Pub 186-4 Federal Information Processing Standards Publication Digital Signature Standard (DSS)," 2013.

[22] C. Schnorr, "Efficient identification and signatures for smart cards," CRYPTO 1989, LNCS, vol.435, pp.239–252, Springer, 1989.

[23] J. Pan and B. Wagner, "Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions." Cryptology ePrint Archive, Paper 2023/198, 2023.

[24] S. Tessaro and C. Zhu, "Threshold and multi-signature schemes from linear hash functions." Cryptology ePrint Archive, Paper 2023/276, 2023.

[25] D. Kales and G. Zaverucha, "An attack on some signature schemes constructed from five-pass identification schemes," CANS 2020, LNCS, vol.12579, pp.3–22, Springer, 2020.

[26] K. Sakumoto, T. Shirai, and H. Hiwatari, "Public-key identification schemes based on multivariate quadratic polynomials," CRYPTO 2011, LNCS, vol.6841, pp.706–723, Springer, 2011.

[27] G. Fuchsbauer, E. Kiltz, and J. Loss, "The algebraic group model and its applications," CRYPTO 2018, LNCS, vol.10992, pp.33–62, Springer, 2018.

[28] M. Zhandry, "To label, or not to label (in generic groups)," CRYPTO 2022, LNCS, vol.13509, pp.66–96, Springer, 2022.

[29] C. Zhang, H. Zhou, and J. Katz, "An analysis of the algebraic group model," ASIACRYPT 2022, LNCS, vol.13794, pp.310–322, Springer, 2022.

[30] J.P. Aumasson and W. Meier, "Zero-sum distinguishers for reduced keccak-f and for the core functions of luffa and hamsi," Presented at the rump session of CHES 2009, 2009.

[31] D. Khovratovich and I. Nikolic, "Rotational cryptanalysis of ARX," FSE 2010, LNCS, vol.6147, pp.333–346, Springer, 2010.

[32] M. Lamberger, F. Mendel, C. Rechberger, V. Rijmen, and M. Schläffer, "Rebound distinguishers: Results on the full whirlpool compression function," ASIACRYPT 2009, LNCS, vol.5912, pp.126–143, Springer, 2009.

[33] H. Gilbert and T. Peyrin, "Super-sbox cryptanalysis: Improved attacks for aes-like permutations," FSE 2010, LNCS, vol.6147, pp.365–383, Springer, 2010.

[34] E. Goh, S. Jarecki, J. Katz, and N. Wang, "Efficient signature schemes with tight reductions to the diffie-hellman problems," J. Cryptol., vol.20, no.4, pp.493–514, 2007.

[35] M. Bellare and G. Neven, "New multi-signature schemes and a general forking lemma," 2005. `https://soc1024.ece.illinois.edu/teaching/ece498ac/fall2018/forkinglemma.pdf`.

[36] M. Fukumitsu and S. Hasegawa, "A tightly secure ddh-based multisignature with public-key aggregation," Int. J. Netw. Comput., vol.11, no.2, pp.319–337, 2021.

[37] A. Bagherzandi, J.H. Cheon, and S. Jarecki, "Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma," CCS 2008, pp.449–458, ACM, 2008.

[38] T. Okamoto, "Provably secure and practical identification schemes and corresponding signature schemes," CRYPTO 1992, LNCS, vol.740, pp.31–53, Springer, 1992.

[39] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," IEEE S&P 2018, pp.315–334, IEEE, 2018.

[40] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova, "On the (in)security of ROS," EUROCRYPT 2021, LNCS, vol.12696, pp.33–53, Springer, 2021.

[41] T.P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," CRYPTO 1991, LNCS, vol.576, pp.129–140, Springer, 1991.

[42] J. Coron, "On the exact security of full domain hash," CRYPTO 2000, LNCS, vol.1880, pp.229–235, Springer, 2000.

[43] M. Bellare, A. Boldyreva, and S. Micali, "Public-key encryption in a multi-user setting: Security proofs and improvements," EUROCRYPT 2000, LNCS, vol.1807, pp.259–274, Springer, 2000.

[44] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," CCS 1993, pp.62–73, ACM, 1993.

[45] J.M. Pollard, "Monte Carlo methods for index computation mod p," Mathematics of Computation, vol.32, pp.918–924, 1978.

[46] G. Leurent and T. Peyrin, "SHA-1 is a shambles: First chosen-prefix collision on SHA-1 and application to the PGP web of trust," USENIX Security Symposium 2020, pp.1839–1856, USENIX Association, 2020.

[47] R. Gay, D. Hofheinz, L. Kohl, and J. Pan, "More efficient (almost) tightly secure structure-preserving signatures," EUROCRYPT 2018, LNCS, vol.10821, pp.230–258, Springer, 2018.

[48] S. Mitsunari, "mcl - a portable and fast pairing-based cryptography library.," 2022/Apr/10 v1.60. `https://github.com/herumi/mcl`.

A General Forking Lemma

Here, we review the General Forking Lemma [3].

Lemma A.1 (General Forking Lemma [3]). *Let $Q \geq 1$ be an integer, and C a set of size $|C| \geq 2$, where $|C|$ is the size of C. Let IG be a randomized algorithm that is called the input generator and $\mathcal{A}$ be a randomized algorithm that, on input $(x, h_1, \ldots, h_Q)$ where x is an input of the forking lemma generated by IG, and $h_i \in H$ for $^\forall i \in [1, Q]$, runs at most t_A time and returns $(I, \sigma) \in [0, Q] \times \{0,1\}^*$. The accepting probability of $\mathcal{A}$, denoted acc, is defined as the probability that $J \geq 1$ in the experiment $x \xleftarrow{\$} \mathsf{IG}, \rho \xleftarrow{\$} R, h_1, \ldots, h_Q \xleftarrow{\$} H, (J, \sigma) \leftarrow \mathcal{A}(x, h_1, \ldots, h_Q; \rho)$ where R is the set of random tapes. The forking algorithm $F_{\mathcal{A}}(x)$ associated to $\mathcal{A}$ is the randomized algorithm that takes input x proceeds as follows:*

1. $\rho \xleftarrow{\$} R$
2. $h_1, \ldots, h_Q \xleftarrow{\$} H$
3. $(J, \sigma) \leftarrow \mathcal{A}(x, h_1, \ldots, h_Q; \rho)$
4. *If $J = 0$, then return $(0, \emptyset, \emptyset)$*
5. $h'_J, \ldots, h'_Q \xleftarrow{\$} H$
6. $(J', \sigma') \leftarrow \mathcal{A}(x, h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_Q; \rho)$
7. *If $(J = J' \wedge h_J \neq h_{J'})$ return $(1, \sigma, \sigma')$.*
8. *Else return $(0, \emptyset, \emptyset)$.*

Let

$$frk = \Pr[b = 1 : x \xleftarrow{\$} \mathsf{IG}, (b, \sigma, \sigma') \xleftarrow{\$} F_{\mathcal{A}}(x)].$$

Then,

$$frk \geq acc \cdot \left(\frac{acc}{Q} - \frac{1}{|C|} \right).$$

B mBCJ

Drijvers et al. proposed a secure two-round MS scheme mBCJ [5]. This scheme was constructed by applying the patch to the insecure MS scheme BCJ [37]. They showed the construction and the security proof that mBCJ is secure under the DL assumption *in the key verification model*. Moreover, they roughly described how to modify mBCJ to be secure in the PPK model. We can obtain a variant scheme of mBCJ which is secure in the PPK model by applying the way. However, there are no concrete construction and no formal security proof.

In this section, we show the construction and the security of the variant of mBCJ.

$\mathsf{Pg}(1^\lambda) \to pp$**.** On input the security parameter 1^λ, the public parameter generation algorithm chooses $(\mathbb{G}, q, G)$, hash functions $H_{\mathrm{c}} : \{0,1\}^* \to \mathbb{Z}_q$, and $H_{\mathrm{ck}} : \{0,1\}^* \to \mathbb{G}^3$, and outputs $pp = (\mathbb{G}, q, G, H_{\mathrm{c}}, H_{\mathrm{ck}})$.

$\mathsf{Kg}(pp) \to (pk, sk)$**.** On input pp, the key generation algorithm chooses $x \xleftarrow{\$} \mathbb{Z}_q$, computes $X \leftarrow xG$, and outputs a public key $pk = X$ and a secret key $sk = x$.

$\langle \{\mathcal{S}(pp, i, sk_i, L, m)\}_{i=1}^n \rangle \to \widetilde{\sigma}$**.** Each signer proceeds with the signing protocol as follows.

Round 1: Each signer computes $ck = (P, Q, R) \leftarrow H_{\mathrm{ck}}(m)$, chooses $(r_i, \alpha_i, \beta_i) \xleftarrow{\$} \mathbb{Z}_q^3$, computes $t_{i,1} \leftarrow \alpha_i G + \beta_i Q$, and $t_{i,2} \leftarrow \alpha_i P + \beta_i R + r_i G$ and broadcasts $(t_{i,1}, t_{i,2})$ to the cosigners.

Round 2 Each signer receives $\{(t_{j,1}, t_{j,2})\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$ from the cosigners. It computes $\tilde{t}_1 \leftarrow \sum_{j=1}^n t_{j,1}$, $\tilde{t}_2 \leftarrow \sum_{j=1}^n t_{j,2}$, $c_i \leftarrow H(X_i, \tilde{t}_1, \tilde{t}_2, L, m)$, and $s_i \leftarrow x_i c_i + r_i \mod q$ and broadcasts (α_i, β_i, s_i).

Aggregate Each signer receives $\{(\alpha_j, \beta_j, s_j)\}_{j \in \{1,\ldots,n\} \setminus \{i\}}$ from the cosigners. It computes $\tilde{\alpha} \leftarrow \sum_{j=1}^n \alpha_j \mod q$, $\tilde{\beta} \leftarrow \sum_{j=1}^n \beta_j \mod q$, and $\tilde{s} \leftarrow \sum_{j=1}^n s_j \mod q$. It also computes $c_i \leftarrow H(X_i, \tilde{t}_1, \tilde{t}_2, L, m)$ for all $i \in [1, n]$ and $\widetilde{A} \leftarrow \tilde{s}G - \sum_{i=1}^n c_i X_i$. It outputs $\widetilde{\sigma} = (\widetilde{A}, \tilde{s}, \tilde{\alpha}, \tilde{\beta})$.

$\mathsf{Vf}(pp, \{pk_j\}_{j=1}^n, \widetilde{\sigma}, m) \to \{0,1\}$**.** On input $pp, \{pk_j\}_{j=1}^n, \widetilde{\sigma}$, and m, the verification algorithm computes $ck = (P, Q, R) \leftarrow H_{\mathrm{ck}}(m)$, $\tilde{t}_1 \leftarrow \tilde{\alpha}G + \tilde{\beta}Q$, $\tilde{t}_2 \leftarrow \tilde{\alpha}P + \tilde{\beta}R + \widetilde{A}$, and $c_j \leftarrow H(X_j, \tilde{t}_1, \tilde{t}_2, L, m)$ for all j. It outputs 1 if $\widetilde{A} = \tilde{s}G - \sum_{i=1}^n c_i X_i$ holds. Otherwise, it outputs 0.

The following theorem states that this variant of mBCJ is secure under the DL assumption in the PPK model.

Theorem B.1. *If there exists a forger $\mathcal{F}$ who $(t_{\mathcal{F}}, q_S, q_H, N, \varepsilon_{\mathcal{F}})$-breaks mBCJ, then there exists an algorithm $\mathcal{B}$ which solves the DL problem with probability at least ε in time at most t s.t.*

$$\varepsilon \geq (1 - q_S/q)^2 \varepsilon_{\mathcal{F}}^2 / (q_H e^2 (q_S + 1)^2) - 1/q, \quad \text{and}$$
$$t \leq 2t_{\mathcal{F}} + (6q_H + 12q_S + 2N + 16)t_{exp} + O(N(q_S + q_H)),$$

where e is the base of the natural logarithm and t_{mul} is the time of an scalar multiplication in $\mathbb{G}$.

Proof. We construct $\mathcal{B}$ which solves the DL problem using $\mathcal{F}$. $\mathcal{B}$ on input $(\mathbb{G}, q, G)$ and X, which are a parameter and an instance of the DL problem, outputs x s.t. $X = xG$.

To construct $\mathcal{B}$, we construct another algorithm $\mathcal{A}$ as follows. On input $(\mathbb{G}, q, G, X)$, a random tape ρ and $h_1, \ldots, h_{q_H + q_S + 1} \in \mathbb{Z}_q$, it internally runs $\mathcal{F}$ on input $(\mathbb{G}, q, G)$ and X as a public parameter pp and a public key pk. It initiates a counter $ctr = 1$, tables $T_{\mathrm{ck}}[\cdot]$, $T_{\mathrm{c}}[\cdot]$, $T_{\mathrm{t}}[\cdot]$, $T_\Sigma[\cdot]$, and $T_M[\cdot]$ to $\emptyset$, where $T_{\mathrm{ck}}[\cdot]$ and $T_{\mathrm{c}}[\cdot]$ are random oracle tables for H_{ck} and H_{c}, respectively, and $T_{\mathrm{t}}[\cdot]$ is a table to store the trapdoor of the commitment key. Also, it responds to random oracle queries and signing queries as follows.

Random Oracle $H_{\mathrm{ck}}(m)$**:** It returns $T_{\mathrm{ck}}[m]$ if $T_{\mathrm{ck}}[m]$ is already defined. If $T_{\mathrm{ck}}[m]$ is undefined, it responds as follows. It chooses a bit b which becomes 1 with probability $\delta = q_S/(q_S + 1)$. If $b = 1$, it chooses $(\omega_{1,1}, \omega_{1,2}, \omega_{1,3}) \xleftarrow{\$} \mathbb{Z}_q^3$, computes $P \leftarrow \omega_{1,1}G$, $Q \leftarrow \omega_{1,2}G$, and $R \leftarrow \omega_{1,3}X$. If $b = 0$, it chooses $(\omega_{0,1}, \omega_{0,2}, \omega_{0,3}) \xleftarrow{\$} \mathbb{Z}_q^3$, computes $P \leftarrow \omega_{0,1}G$, $Q \leftarrow \omega_{0,2}X$, and $R \leftarrow \omega_{0,3}G$. It assigns $T_{\mathrm{ck}}[m] \leftarrow (P, Q, R)$, and $T_{\mathrm{t}}[m] \leftarrow (b, \omega_{b,1}, \omega_{b,2}, \omega_{b,3})$ and returns $T_{\mathrm{ck}}[m]$.

Random Oracle $H_{\mathrm{c}}(X_i, \tilde{t}_1, \tilde{t}_2, L, m)$**:** If $T_{\mathrm{ck}}[m]$ is undefined, it makes a query $H_{\mathrm{ck}}(m)$. If $T_{\mathrm{c}}[X_i, \tilde{t}_1, \tilde{t}_2, L, m]$ is already defined, it returns h where $(h, J) = T_{\mathrm{c}}[X_i, \tilde{t}_1, \tilde{t}_2, L, m]$. If $T_{\mathrm{c}}[X_i, \tilde{t}_1, \tilde{t}_2, L, m]$ is undefined, it responds as follows.

Case $X \in L$**:** For j s.t. $X \neq X_j \in L$, it chooses $c_j \xleftarrow{\$} \mathbb{Z}_q$ and assigns $T_{\mathrm{c}}[X_j, \tilde{t}_1, \tilde{t}_2, L, m] \leftarrow (c_j, 0)$. After that, for j s.t. $X = X_j \in L$, it assigns $T_{\mathrm{c}}[X_i, \tilde{t}_1, \tilde{t}_2, L, m] \leftarrow (h_{ctr}, ctr)$ and sets $ctr \leftarrow ctr + 1$. It returns h where $(h, J) = T_{\mathrm{c}}[X_i, \tilde{t}_1, \tilde{t}_2, L, m]$.

Case $X \notin L$**:** It chooses $c \xleftarrow{\$} \mathbb{Z}_q$, assigns $T_{\mathrm{c}}[X_i, \tilde{t}_1, \tilde{t}_2, L, m] \leftarrow (c, 0)$ and returns c.

Signing Queries: It receives (I_s, m, L) as a query. If $X \notin L$, it returns $\perp$. It executes the following protocol. Note that it executes the protocol multiple times if there are multiple public keys s.t. $X = X_i \in L$. At the end of **Round 1**, it stores $T_\Sigma[I_s] \leftarrow (m, L, n_h, \{(t_{i,1}, t_{i,2}, u_i, v_i, w_i)\}_{i \in n_h})$ where n_h is the set of the indies of the signers who have X as public keys. If **Round 2** is completed, it sets $T_M[m] \leftarrow 1$.

Round 1: It makes a query $H_{\mathrm{ck}}(m)$ if $T_{\mathrm{ck}}[m]$ is undefined. It looks up (P, Q, R) from $T_{\mathrm{ck}}[m]$, chooses $(u_i, v_i, w_i) \xleftarrow{\$} \mathbb{Z}_q^3$, computes $t_{i,1} \leftarrow u_i G$ and $t_{i,2} \leftarrow v_i G - w_i X$ and broadcasts $(t_{i,1}, t_{i,2})$.

Round 2: It receives $(I_s, \{(t_{j,1}, t_{j,2})\}_{j \in \{1,\ldots,n\} \setminus n_h})$, looks up $(m, L, n_h, \{(t_{i,1}, t_{i,2}, u_i, v_i, w_i)\}_{i \in n_h})$ from $T_\Sigma[I_s]$, and $(b, \omega_{b,1}, \omega_{b,2}, \omega_{b,3})$ from $T_{\mathrm{t}}[m]$. If $(b = 0)$ or $(b = 1 \wedge \omega_{1,3} = 0)$, it halts with output $(0, \emptyset)$. Otherwise, it computes $\tilde{t}_1 \leftarrow \sum_{j=1}^{n} t_{j,1}$, $\tilde{t}_2 \leftarrow \sum_{j=1}^{n} t_{j,2}$, $c_i \leftarrow H(X_i, \tilde{t}_1, \tilde{t}_2, L, m)$, $\beta_i \leftarrow (c_i - w_i)/\omega_{1,3} \mod q$, $\alpha_i \leftarrow u_i - \omega_{1,2}\beta_i \mod q$, and $s_i \leftarrow v_i - \omega_{1,1}\alpha_i \mod q$. It broadcasts (α_i, β_i, s_i).

If $\mathcal{F}$'s forgery $(m^*, L^*, \widetilde{\sigma}^* = (\widetilde{A}^*, \tilde{s}^*, \tilde{\alpha}^*, \tilde{\beta}^*))$ does not satisfy the conditions of **Check** in the game of the security definition or $b = 0$ where $(b, \omega_{b,1}, \omega_{b,2}, \omega_{b,3}) = T_{\mathrm{t}}[m^*]$, then $\mathcal{A}$ halts with output $(0, \emptyset)$. Otherwise, $\mathcal{A}$ can get a forgery $(m^*, L^*, \widetilde{\sigma}^* = (\widetilde{A}_1^*, \tilde{s}^*, \tilde{\alpha}^*, \tilde{\beta}^*))$ s.t. $X \in L^*$, $T_M[m^*] \neq 1$, $\mathsf{Vf}(pp, L^*, \widetilde{\sigma}^*, m^*) = 1$, and $(0, \omega_{0,1}, \omega_{0,2}, \omega_{0,3}) = T_{\mathrm{t}}[m^*]$. Let J be the integer s.t. $(c, J) = T_{\mathrm{c}}[X, \tilde{t}_1^*, \tilde{t}_2^*, L^*, m^*]$. It outputs $(J, \sigma = (L^*, \omega_{0,1}, \omega_{0,2}, \omega_{0,3}, \tilde{s}^*, \tilde{\alpha}^*, \tilde{\beta}^*, \{c_j\}_{j=1}^{|L^*|}))$ where $(c_j, I_j) = T_{\mathrm{c}}[X_j, \tilde{t}_1^*, \tilde{t}_2^*, L^*, m^*]$.

From Lemma B.2, the distribution of $\mathcal{A}$'s responses in **Signing Queries** is identical to the distribution of the honest signer's responses.

Let acc be the probability that $\mathcal{A}$ outputs $J > 0$. Also, we define the following events.

- E_1: $\mathcal{F}$'s forgery satisfies the conditions of **Check** in the game of the security definition.
- E_2: $\mathcal{F}$'s forgery satisfies $b = 0$ where $(b, \omega_{b,1}, \omega_{b,2}, \omega_{b,3}) = T_{\mathrm{t}}[m^*]$.
- E_3: $\mathcal{A}$ halts because of $b = 0$ in **Signing Queries**.
- E_4: $\mathcal{A}$ halts because of $(b = 1 \wedge \omega_{1,3} = 0)$ in **Signing Queries**.

Then, we obtain the following equations.

$$\begin{aligned} acc &= \Pr[E_1 \wedge E_2 \wedge \overline{E_3} \wedge \overline{E_4}] \\ &= \Pr[\overline{E_3}] \Pr[\overline{E_4} | \overline{E_3}] \Pr[E_1 | \overline{E_3} \wedge \overline{E_4}] \Pr[E_2 | E_1 \wedge \overline{E_3} \wedge \overline{E_4}] \end{aligned}$$

$\overline{E_3}$ means that, for all messages m chosen by $\mathcal{F}$ as signing queries, the bit b becomes 1 when $T_{\mathrm{ck}}[m]$ is determined. In the both cases $b = 0$ and $b = 1$, the distributions of the responses of $T_{\mathrm{ck}}[m]$ are identical. Because $b = 1$ occurs with probability δ, and $\mathcal{F}$ makes at most q_S signing queries, $\Pr[\overline{E_3}] \geq \delta^{q_S}$.

Conditioned on $\overline{E_3}$, $b = 1$ holds with probability 1 for all messages m queried in **Signing Queries**. Thus, $\Pr[\overline{E_4} | \overline{E_3}]$ is equal to the probability that $\omega_{1,3} \neq 0$ for all messages m queried in **Signing Queries**. Then, we get $\Pr[\overline{E_4} | \overline{E_3}] = (1 - 1/q)^{q_S}$.

Conditioned on $\overline{E_3} \wedge \overline{E_4}$, $\mathcal{A}$ does not halt in **Signing Queries**. Thus, $\Pr[E_1 | \overline{E_3} \wedge \overline{E_4}] = \varepsilon_{\mathcal{F}}$.

Because the distribution of $H_{\mathrm{ck}}(\cdot)$ is independent of the value of bits, and **Round 2** on m^* has never been executed in **Signing Queries**, $\mathcal{F}$ cannot know the value of the bit for m^*. Thus, the event that $b = 0$ for m^* happens with probability $1 - \delta$. Therefore, $\Pr[E_2 | E_1 \wedge \overline{E_3} \wedge \overline{E_4}] = 1 - \delta$. From the above, we obtain $acc \geq \delta^{q_S} (1 - 1/q)^{q_S} \varepsilon_{\mathcal{F}} (1 - \delta)$. Since we set $\delta = q_S/(q_S + 1)$, we have

$$\begin{aligned} acc &= \frac{1}{(1 + 1/q_S)^{q_S}} (1 - 1/q)^{q_S} \varepsilon_{\mathcal{F}} \frac{1}{q_S + 1} \\ &\geq (1 - q_S/q) \varepsilon_{\mathcal{F}} / (e(q_S + 1)) \end{aligned}$$

by using the facts that $(1 + 1/q_S)^{q_S} < e$ for $q_S \geq 0$, where e is the base of the natural logarithm, and $(1 + a)^b \geq 1 + ab$ for $a \geq -1$ and a natural number b.

Let $t_{\mathcal{A}}$ be the running time of $\mathcal{A}$. We assume that t_{mul} time is required for one scalar multiplication in $\mathbb{G}$, and unit time is required for the other non-cryptographic operations. $\mathcal{A}$ runs $\mathcal{F}$ at once.

For time to answer random oracle queries, we consider only the case of $H_{\rm c}$ because $H_{\rm c}$ takes a longer time than $H_{\rm ck}$. $\mathcal{A}$ makes one query to $H_{\rm ck}$ and executes $O(N)$ other non-cryptographic operations to respond to a query to $H_{\rm c}$. Three scalar multiplications and $O(1)$ other non-cryptographic operations are required for one random oracle query to $H_{\rm ck}$. Thus, in total, $\mathcal{A}$ executes three scalar multiplications and $O(N)$ other non-cryptographic operations to respond to one random oracle query to $H_{\rm c}$. In each signing query, $\mathcal{A}$ needs to execute one random oracle query to $H_{\rm ck}$, three scalar multiplications, and $O(N)$ other non-cryptographic operations. The verification involves at most $(N+5)$ scalar multiplications, one random oracle query to $H_{\rm ck}$, and $O(N)$ other non-cryptographic operations. Also $\mathcal{A}$ needs $O(N)$ other non-cryptographic operations to output (J, σ) after checking $\mathcal{F}$'s output. Therefore, in total $\mathcal{A}$ runs at most $t_{\mathcal{F}} + (3q_H + 6q_S + N + 8)t_{exp} + O(N(q_S + q_H))$.

$\mathcal{B}$ runs the forking algorithm $F_{\mathcal{A}}(\mathbb{G}, q, G, X)$ according to Lemma A.1. If $F_{\mathcal{A}}(\mathbb{G}, q, G, X)$ succeeds in outputting $(1, \sigma, \sigma')$, $\mathcal{B}$ can obtain $\sigma = (L^*, \omega_{0,1}, \omega_{0,2}, \omega_{0,3}, \tilde{s}^*, \tilde{\alpha}^*, \tilde{\beta}^*, \{c_j\}_{j=1}^{|L^*|})$ and $\sigma' = (L'^*, \omega'_{0,1}, \omega'_{0,2}, \omega'_{0,3}, \tilde{s}'^*, \tilde{\alpha}'^*, \tilde{\beta}'^*, \{c'_j\}_{j=1}^{|L'^*|})$ s.t. $(L^*, \omega_{0,1}, \omega_{0,2}, \omega_{0,3}, \{c_j\}_{j\in K^*}) = (L'^*, \omega'_{0,1}, \omega'_{0,2}, \omega'_{0,3}, \{c'_j\}_{j\in K'^*})$, $X \in L^*$, $(c_i = c_j) \wedge (c'_{i'} = c'_{j'}) \wedge (c_i \neq c_{i'})$ for all $i, j \in [1, |L^*|]\backslash K^*$ and $i', j' \in [1, |L'^*|]\backslash K'^*$, $\tilde{\alpha}^* G + \tilde{\beta}^*(\omega_{0,2} X) = \tilde{\alpha}'^* G + \tilde{\beta}'^*(\omega'_{0,2} X)$, and

$$\tilde{\alpha}^*(\omega_{0,1} G) + \tilde{\beta}^*(\omega_{0,3} G) + \tilde{s}^* G - \sum_{j=1}^{|L^*|} c_j X_j = \tilde{\alpha}'^*(\omega_{0,1} G) + \tilde{\beta}'^*(\omega'_{0,3} G) + \tilde{s}'^* G - \sum_{j=1}^{|L'^*|} c'_j X'_j$$

where K^* and K'^* are the sets of the indices s.t. $X \neq X_i \in L^*$ and $X \neq X_i \in L'^*$, respectively. According to the above conditions, $\mathcal{B}$ can obtain the following equations

$$\begin{aligned}
&(\tilde{\alpha}^* - \tilde{\alpha}'^*) G = (\tilde{\beta}'^* - \tilde{\beta}^*)\omega_{0,2} X, \text{ and} \\
&((\tilde{\alpha}^* - \tilde{\alpha}'^*)\omega_{0,1} + (\tilde{\beta}^* - \tilde{\beta}'^*)\omega_{0,3} + (\tilde{s}^* - \tilde{s}'^*)) G = -(|L^*| - |K^*|)(c - c') X
\end{aligned}$$

where c and c' are c_i and c'_i for some $i \in [1, |L^*|]\backslash K^*$ and some $i \in [1, |L'^*|]\backslash K'^*$, respectively. $(|L^*| - |K^*|) \neq 0$ holds because L^* includes at least one X. $\mathcal{B}$ computes and outputs the discrete logarithm x of X as follows.

Case $(\tilde{\beta}'^* \neq \tilde{\beta}^*) \wedge (\omega_{0,2} \neq 0)$**:** $\mathcal{B}$ outputs x as $(\tilde{\alpha}^* - \tilde{\alpha}'^*)/((\tilde{\beta}'^* - \tilde{\beta}^*)\omega_{0,2})$.

Case $(\tilde{\beta}'^* \neq \tilde{\beta}^*) \wedge (\omega_{0,2} = 0)$**:** In this case, $\tilde{\alpha}^* = \tilde{\alpha}'^*$ holds. Thus, $\mathcal{B}$ outputs x as $-((\tilde{\beta}^* - \tilde{\beta}'^*)\omega_{0,3} + (\tilde{s}^* - \tilde{s}'^*))/((|L^*| - |K^*|)(c - c'))$.

Case $\tilde{\beta}'^* = \tilde{\beta}^*$**:** In this case, $\tilde{\alpha}^* = \tilde{\alpha}'^*$ holds. Thus, $\mathcal{B}$ outputs x as $-(\tilde{s}^* - \tilde{s}'^*)/((|L^*| - |K^*|)(c - c'))$

We evaluate the success probability and the running time of $\mathcal{B}$. Because $\mathcal{B}$ can output the solution of the DL problem if $F_{\mathcal{A}}(\mathbb{G}, q, G, X)$ outputs $(1, \sigma, \sigma')$, $\mathcal{B}$ can solve the DL problem with the probability that $F_{\mathcal{A}}(\mathbb{G}, q, G, X)$ outputs $(1, \sigma, \sigma')$ in time as same as the running time of $F_{\mathcal{A}}(\mathbb{G}, q, G, X)$. Applying the Lemma A.1, $\mathcal{B}$ solves the DL problem with probability at least ε s.t.

$$\begin{aligned}
\varepsilon &\geq acc\,(acc/q_H - 1/q) \\
&\geq acc^2/q_H - 1/q \\
&\geq (1 - q_S/q)^2\, \varepsilon_{\mathcal{F}}^2/(q_H e^2 (q_S + 1)^2) - 1/q.
\end{aligned}$$

Because $\mathcal{B}$ runs $\mathcal{A}$ twice, the running time of $\mathcal{B}$ is at most twice as long as the running time of $\mathcal{A}$ Thus, the running of $\mathcal{B}$ is at most $2t_{\mathcal{F}} + (6q_H + 12q_S + 2N + 16)t_{exp} + O(N(q_S + q_H))$. □

The following Lemma states that, for the algorithm $\mathcal{A}$ in the above proof, the distribution of responses in **Signing Queries** is identical to th e distribution of the hoenst signer's responses. Since we can prove this lemma in the similar way used to prove Lemma 4.7 of our proposed scheme, we omit the proof of this lemma.

Lemma B.2. *Let* $\mathsf{Game}_{\mathrm{eqv}}^{\mathsf{mBCJ}}$ *be the following game between a challenger and a distinguisher* $\mathcal{A}$*.*

Setup: *The challenger chooses* $(\mathbb{G}, q, G)$*. It sends* $(\mathbb{G}, q, G)$ *to* $\mathcal{A}$ *and receives* $x \in \mathbb{Z}_q$ *from* $\mathcal{A}$*. It computes* $X \leftarrow xG$ *and initializes a table* $T_S[\cdot]$*. It chooses a bit* $b \xleftarrow{\$} \{0,1\}$*.*

Oracles: *The challenger allows* $\mathcal{A}$ *to access to the following oracles concurrently at most* q_S *times. Note that* $\mathcal{A}$ *is allowed to make only one query for each session identifier* I*, which is included in each query to oracles.*

- $\Sigma_{\mathrm{eqv1}}(b, \cdot, \cdot)$**:** *As a query, the challenger receives a session identifier* I *and* $(\omega_1, \omega_2, \omega_3) \in \mathbb{Z}_q^3$*. It computes computes* $P \leftarrow \omega_1 G$*,* $Q \leftarrow \omega_2 G$*, and* $R \leftarrow \omega_3 X$*. It responds as follows.*
  - **Case** $b = 0$**:** *It chooses* $r, \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$ *and computes* $t_1 \leftarrow \alpha G + \beta Q$*, and* $t_2 \leftarrow \alpha P + \beta R + rG$*. It stores* $T_S[I] \leftarrow ((P, Q, R), (\omega_1, \omega_2, \omega_3), r, \alpha, \beta, t_1, t_2)$ *and returns* (t_1, t_2)*.*
  - **Case** $b = 1$**:** *It chooses* $u, v, w \xleftarrow{\$} \mathbb{Z}_q^3$ *and computes* $t_1 \leftarrow uG$ *and* $t_2 \leftarrow vG - wX$*. It stores* $T_S[I] \leftarrow ((P, Q, R), (\omega_1, \omega_2, \omega_3), u, v, w, t_1, t_2)$ *and returns* (t_1, t_2)*.*
- $\Sigma_{\mathrm{eqv2}}(b, \cdot, \cdot)$**:** *As a query, the challenger receives a session identifier* I *and* $c \in \mathbb{Z}_q$*. If* $T_S[I]$ *is empty, then it return* $\perp$*. Otherwise, it responds as follows.*
  - **Case** $b = 0$**:** *The challenger looks up* $((P, Q, R), (\omega_1, \omega_2, \omega_3), r, \alpha, \beta, t_1, t_2)$ *from* $T_S[I]$*, computes* $s \leftarrow xc + r \mod q$ *and returns* (α, β, s)*.*
  - **Case** $b = 1$**:** *The challenger looks up* $((P, Q, R), (\omega_1, \omega_2, \omega_3), u, v, w, t_1, t_2)$ *from* $T_S[I]$*, computes* $\beta \leftarrow (c - w)/\omega_3 \mod q$*,* $\alpha \leftarrow u - \omega_2 \beta \mod q$*, and* $s \leftarrow v - \omega_1 \alpha \mod q$ *and returns* (α, β, s)*.*

Guess: *Finally,* $\mathcal{A}$ *outputs a guess* $b' \in \{0,1\}$*. If* $b = b'$ *holds, then* $\mathcal{A}$ *wins this game.*

The advantage of $\mathcal{A}$ *is defined as*

$$\mathrm{Adv}_{\mathrm{eqv}}^{\mathsf{mBCJ}}(\mathcal{A}) = \left|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]\right|.$$

For any computationally unbounded distinguisher $\mathcal{A}$*,* $\mathrm{Adv}_{\mathrm{eqv}}^{\mathsf{mBCJ}}(\mathcal{A}) = 0$ *holds.*