Decentralized Threshold Signatures for Blockchains with Non-Interactive and Transparent Setup

Kwangsu Lee*

Abstract

Threshold signatures are digital signatures that support the multi-party signature generation such that a number of parties initially share a signing key and more than a threshold number of parties gather to generate a signature. In this paper, we propose a non-interactive decentralized threshold signature (NIDTS) scheme that supports the non-interactive and transparent key setup based on BLS signatures. Our NIDTS scheme has the following properties. 1) The key setup process is completely non-interactive and does not require message exchange between parties since the transfer of the register keys of parties is enough for a combiner to generate a compact verification key. 2) The register key of a party is compact since the size is independent of the number of group parties. 3) The signing process of parties is non-interactive. 4) The final threshold signature as well as partial signatures are succinct. We prove the security of our NIDTS scheme under computational assumptions in the random oracle model. Furthermore, we implement our NIDTS scheme in Rust and compare its performance with other scheme to show that the key setup of our scheme is more efficient. For example, in the unweighted setting of 1000 parties, the key setup process of the NIDTS scheme takes 164 seconds, which is 5.9 times faster than the key setup process of the multiverse threshold signature (MTS) scheme.

Keywords: Threshold signatures, BLS signatures, Non-interactive setup, Bilinear maps, Blockchain.

*Sejong University, Seoul, South Korea. Email: kwangsu@sejong.ac.kr.

1 Introduction

A threshold signature scheme is a special type of digital signatures in which a group of n parties shares a secret signing key through a key setup process, and any subset of parties greater than t can generate a valid signature for a message M through the signing process [16, 17]. A threshold signature scheme is secure if a valid signature cannot be generated when less than t parties are involved. An ideal threshold signature scheme is that the key setup process is efficient in terms of computation and communication, a verification key is compact, the signing process that derives a threshold signature from partial signatures is non-interactive, and the final threshold signature is succinct. Threshold signatures have been studied for a long time [4,9,14,23–25,31], and they can be applied for user wallet protection, distributed random services, and consensus protocols in decentralized blockchains [23, 37, 38]. And recently, due to the importance of threshold cryptosystems, the standardization of threshold cryptosystems has begun [33].

The method of constructing a threshold signature scheme from a digital signature scheme consists of a key setup process in which a group of parties share a secret signing key and a signing process in which partial signatures that are generated by some parties are combined into a final threshold signature. The key setup process is usually done by using a distributed key generation (DKG) protocol [26], which is a multi-party protocol that distributes a secret signing key to the secret shares of all parties. However, it is difficult to use this DKG protocol if the number of parties increases since the amount of computation and communication of each party is quite large. Recently, a non-interactive DKG (NI-DKG) scheme that reduces the rounds of the DKG protocol was proposed [28], but this scheme is still inefficient since individual parties are required to generate zero-knowledge proofs and solve the discrete logarithm of small exponents. The signing process of a threshold signature scheme can be non-interactive if it is based on BLS signatures [11]. The threshold signature scheme from BLS signatures (TS-BLS) [9] is attractive since it supports non-interactive signing, but it requires an interactive DKG protocol. The multi-signature scheme based on BLS signatures (MS-BLS) [9] also can be used as threshold signatures if the final signature contains a set of signers and the aggregation of public keys is done by a verifier, but the size of a verification key increases in proportion to the number of parties.

Recently, multiverse threshold signature (MTS) and silent threshold signature (STS) schemes [2, 21], that support the efficient key setup without exchanging messages between parties and the non-interactive signing based on BLS signatures, have been proposed. In the MTS scheme [2], each party generates an online message based on the public keys of all parties which are organized by a combiner and delivers it to the combiner, and then the combiner creates the verification key of MTS. Thus, the key setup process of the MTS scheme is two rounds since the set of parties should be determined in the first round, and it is transparent because anyone who has online messages and all public keys can check the validity of this key setup process. In the STS scheme [21], each party generates a hint if the number of all parties and the index of the party are given, and the combiner creates the verification key of STS by using all hints of parties. The hint of a party only can be generated when the number of all parties and the position of the party are known which can be determined after the combiner fixes the group of parties. Although, the authors of STS offer some ideas to get rid of it, it doesn't completely solve the problem without communication and computation overhead or errors due to collisions. Thus, the key setup process of the basic STS scheme can be regarded as two rounds. Additionally, the STS scheme requires the trusted setup for structured reference strings.

As described above, both MTS and STS schemes remove message exchanges between individual parties, but they cannot be non-interactive key setup since each party must wait for a combiner to determine the dynamic group of parties before generating an online message or a hint of the party. Therefore, we ask the following question:

Table 1: Comparison of unweighted threshold signatures based on BLS signatures

Scheme	PK	VK	PS	TS	Sign	Verify	Key Setup
TS-BLS [9]	$\mathbb{G}$	$\mathbb{G}$	$\hat{\mathbb{G}}$	$\hat{\mathbb{G}}$	E + H	2P + H	DKG
MS-BLS [9]	$\mathbb{G}$	$m\mathbb{G}$	$\hat{\mathbb{G}}$	$\hat{\mathbb{G}}+S$	E + H	2P + H	NI, TP
ASM [10]	$\mathbb{G}$	$\mathbb{G}$	$\hat{\mathbb{G}}$	$\mathbb{G}+\hat{\mathbb{G}}+S$	E + H	3P + $(t+1)$H	2R, TP
Groth [28]	$\mathbb{G}$	$\mathbb{G}$	$\hat{\mathbb{G}}$	$\hat{\mathbb{G}}$	E + H	2P + H	NI-DKG
MTS [2]	$\mathbb{G}$	$\mathbb{G}+\hat{\mathbb{G}}$	$\hat{\mathbb{G}}$	$2\mathbb{G}+\hat{\mathbb{G}}$	E + H	4P + H	2R, TP
STS [21]	$\mathbb{G}$	$2\mathbb{G}+\hat{\mathbb{G}}$	$\hat{\mathbb{G}}$	$7\mathbb{G}+2\hat{\mathbb{G}}+5\mathbb{Z}_p$	E + H	10P + H	2R, SRS
NIDTS	$\mathbb{G}$	$\mathbb{G}+\hat{\mathbb{G}}$	$\hat{\mathbb{G}}$	$\mathbb{G}+2\hat{\mathbb{G}}$	E + H	4P + H	NI, TP

Let m be the number of parties, t be a threshold, and S be a set of signers with m bits. We use abbreviations PK for public key, VK for verification key, PS for partial signature, and TS for threshold signature. We use symbols H for map-to-hash, E for exponentiation, and P for pairing. In the key setup, we use DKG for distributed key generation, NI-DKG for non-interactive DKG, 2R for two-round setup, SRS for structured reference string, NI for non-interactive (or one-round) setup, and TP for transparent setup.

> *"Can we design a threshold signature scheme with a compact verification key that supports the completely non-interactive key setup for the dynamic group of parties without requiring message exchanges between parties?"*

In the completely non-interactive key setup of threshold signatures, except that individual parties generate their own public keys, individual parties forward only one message to the combiner without waiting for the combiner to determine the dynamic group of parties[1], and then the combiner completes the key setup process by generating a compact verification key.

1.1 Our Contributions

In this paper, we show that there is a positive answer to the above question. To this end, we first define a non-interactive decentralized threshold signature (NIDTS) scheme that supports the non-interactive key setup. In the NIDTS scheme, each party generates a secret key and a public key by running a key generation algorithm, and registers the public key to a blockchain (on-chain). After that, each party generates a register key with a public group identifier *GID* as an input to join a group associated with *GID* before a combiner determines group parties, and transmits the register key to the combiner. The untrusted combiner collects the public keys and register keys of group parties to establish a group and generates the combine key and verification key of this group. Note that this key setup process is transparent since the combine key and verification key are deterministically derived from public keys and register keys. In order to generate a threshold signature for any message, individual parties create partial signatures with their secret keys and transmit them to the combiner. The untrusted combiner generates a threshold signature by combining partial signatures of sufficient number of parties, and anyone can verify this threshold signature by using the verification key registered in the blockchain (on-chain).

[1] In existing threshold signatures with DKG protocols, the key setup assume that the set of parties and the threshold are fixed. In this work, we require that threshold signatures should support the dynamic choice of the set of parties and the threshold for blockchain applications.

Next, we propose an efficient NIDTS scheme with a compact verification key that supports the non-interactive key setup, transparent setup, and non-interactive signing from bilinear groups, and prove its security under complexity assumptions in the random oracle model. Basically, our NIDTS scheme is built from the BLS signature scheme [11] to support the non-interactive signing. In addition, the key setup of our scheme is similar to that of the MTS scheme [2] in that it matches the public keys of group parties with points of a polynomial and additionally exposes some public points of the polynomial to enable the threshold verification. However, in order to support the non-interactive key setup, we use a mechanism that synchronizes a group element for all parties by using the map-to-hash function $H_0(GID)$ when a group identifier GID is given as an input. Because of this synchronization device, individual parties can generate register keys immediately without waiting for the combiner to determine all parties of a group, and it can prevent some parties from maliciously setting exponent values of group elements. More details on our NIDTS scheme are described in the next section. And we use the widely used virtualization technique to support threshold signatures with weights for individual parties. The comparison of our NIDTS scheme with other threshold signature schemes based on BLS signatures is given in Table 1.

Finally, we implement our NIDTS scheme in Rust and analyze the performance of our scheme. The key setup process of the NIDTS scheme maps the public keys of individual parties to polynomial pointers, and multiple public pointers of the polynomial must be generated for threshold functionality using polynomial interpolation. If this is implemented in a simple way, it requires $O(n^3)$ scalar multiplications to calculate the Lagrange coefficients required for polynomial interpolation where n is the degree of the polynomial, so the running time of this algorithm increases rapidly when the number of group parties increases. To overcome this problem, we apply a method that efficiently handles the calculation of Lagrange coefficients by using the barycentric form of the Lagrange interpolation. We can confirm that our NIDTS scheme performed better in most algorithms by comparing the performance with the MTS scheme. In particular, in the case of the key setup process, the computation of individual parties is very efficient, and the computation of the combiner is also more efficient than before.

1.2 Our Techniques

We construct our NIDTS scheme that supports the non-interactive key setup by modifying the MTS scheme of Baird et al. [2]. The main idea of the MTS scheme is that the pubic keys of parties are mapped to polynomial points to eliminate message exchange between parties and expose additional polynomial points to support the threshold signing. First, the MTS scheme initially sets an $(n-1)$-degree polynomial $f(x)$ by using the public keys of n parties. That is, when the public keys of individual parties are given as $g^{s_1},\ldots,g^{s_n}$ where s_i is the secret key of i-th party, the $(n-1)$-degree polynomial $f(x)$ is implicitly defined such that $f(i)=s_i$ for each $i\in[n]$. However, this method has a problem of requiring n polynomial points instead of t points to reconstruct an element $g^{f(0)}$. The idea of the MTS scheme to overcome this problem is to additionally reveal $(n-t)$ polynomial points $g^{f(-(n-t))},\ldots,g^{f(-1)}$ as public parameters. In this case, if t polynomial points obtained from signing parties and $(n-t)$ public points are combined, then the polynomial can be reconstructed since $t+(n-t)=n$ pieces of polynomial points are obtained.

However, since the partial signature of the MTS scheme has a form of $H_1(M)^{f(s_i)}$ but public pointers have a form of $g^{f(k)}$, it is difficult to derive the final signature $H_1(M)^{f(0)}$ by combining partial signatures and public pointers. To overcome this problem, the MTS scheme modifies the signature verification equation so that the verification can be done even when the final signature is derived by combining only partial signatures. From the Lagrange interpolation method, we can obtain the following equation from the private

keys and public pointers of the parties.

$$\begin{aligned} f(0) &= \big(f(-(n-t))L_{-(n-t)}(0) + \ldots + f(-1)L_{-1}(0)\big) + \big(f(1)L_1(0) + \ldots + f(t)L_t(0)\big) \\ &= f^{pub}(0) + f^{par}(0) \end{aligned}$$

where $\{L_i(0)\}$ are the Lagrange coefficients, $f^{pub}(0)$ is for the public points part, and $f^{par}(0)$ is for the partial signature part. By rearranging the equation $e(g, H_1(M)^{f(0)}) = e(g, H_1(M)^{f^{pub}(0)}) \cdot e(g, H_1(M)^{f^{par}(0)})$, the new verification equation can be derived as follows:

$$e(g^{f(0)}/g^{f^{pub}(0)}, H_1(M)) = e(g, H_1(M)^{f^{par}(0)})$$

when the combiner computes $\sigma^{par} = H_1(M)^{f^{par}(0)}$ and $\sigma^{pub} = g^{f^{pub}(0)}$. That is, the verification key consists of $g^{f(0)}$, and the threshold signature consists of $H_1(M)^{f^{par}(0)}$ derived by combining the partial signatures and $g^{f^{pub}(0)}$ obtained by linearly combining public points. To ensure that $g^{f^{pub}(0)}$ is the linear combination of public points, the MTS scheme requires an additional mechanism such that each party generates an online message by selecting a random k_i, and deliver it to the combiner. To guarantee that at least one party correctly created this online message with a random k_i, the combiner additionally computes $g^{kf^{pub}(0)}$ by combining all online messages of parties. Thus, the final signature is $(H_1(M)^{f^{par}(0)}, g^{f^{pub}(0)}, g^{kf^{pub}(0)})$.

In order to overcome the problem that the key setup process of the MTS scheme is two rounds such that the first round is the distribution of all public keys of group parties and the second round is the delivery of the online messages of parties, we use a map-to-hash function $H_0(x)$ in our NIDTS scheme instead of combining individual online messages. That is, we set $H_0(GID) = \hat{g}^k$ to fix a random k when a group identifier GID is given since $H_0(x)$ is modeled as a random oracle. Note that GID can be known in advance even before group parties are determined by a combiner. Thus, since all parties belonging to the same group can use the same $H_0(GID)$ when GID is known, each party generates a public key g^{s_i} and additionally generates a register key $H_0(GID)^{s_i}$ to join a group associated with GID. After that, the combiner can perform the key setup process by calculating public points $g^{f(-(n-t))}, \ldots, g^{f(-1)}, g^{f(0)}$ and $H_0(GID)^{f(-(n-t))}, \ldots, H_0(GID)^{f(-1)}$ by collecting all public keys and register keys of parties in the group. The modified final signature of our NIDTS scheme is $(H_1(M)^{f^{par}(0)}, g^{f^{pub}(0)}, H_0(GID)^{f^{pub}(0)})$.

The main advantage of this approach is that when a party generates a register key, the group identifier GID is required, but the public keys of other parties are unnecessary. This makes the key setup process of the NIDTS scheme to be non-interactive because it only requires the transfer of a register key to a combiner. In addition, the online message of the MTS scheme consists of $g^{k_i f(-(n-t))}, \ldots, g^{k_i f(-1)}, g^{k_i}$, whereas the register key of our NIDTS scheme consists of only $H_0(GID)^{s_i}$. Thus, this approach is very efficient when comparing the amount of computation and communication performed by individual parties and the combiner in the key setup process since the combiner should verify the online messages or register keys before it generates the combine key and verification key. Furthermore, if GID is disclosed at the initial stage of the system and each party publishes a public key with a register key at the same time, then the key setup process can be zero round.

1.3 Related Work

Research on threshold signatures has been carried out for a long time as part of threshold cryptography [9, 16, 17, 24, 25]. Recently interest in threshold signatures without a trusted center has increased rapidly due to various attempts to apply threshold signatures to decentralized blockchains [23, 38]. The main step of threshold signatures is the distributed key generation (DKG) protocol in which shares of a signing key are

securely distributed among individual parties by performing a verifiable secret sharing protocol in parallel [26]. However, it is very expensive to perform the DKG protocol when the number of parties increases due to the computation and communication overhead of individual parties. Since then, research has been conducted to improve the performance of the DKG protocol [29, 39], and a DKG protocol in an asynchronous network environment has also been proposed [30]. Recently, a non-interactive DKG protocol was proposed except for the setting of a public key to reduce the rounds of the DKG protocol [28]. However, this protocol has the disadvantage that the group of parties and a threshold must be fixed in advance.

Threshold signatures based on Schnorr signatures [36] and BLS signatures [11] are attracting a lot of attention recently. The FROST scheme was proposed as an efficient discrete-logarithm based threshold signature scheme based on Schnorr signatures [31]. The FROST scheme uses the DKG protocol to setup the secret share key of parties, and the signature generation consists of two rounds of a message independent preprocessing process and a message dependent signing process, and the resulting signature is the same as the original Schnorr signature. In the FROST scheme, if the preprocessing step can be done in advance, then the scheme supports the non-interactive signing. Recently, some study were conducted to improve the performance and security of the FROST scheme [4,14,35]. The TS-BLS scheme has been proposed as a pairing based threshold signature based on BLS signatures [9]. The TS-BLS scheme also uses the DKG protocol for the key setup of all parties, and generates a threshold signature non-interactively in which individual parties generate partial signatures and a combiner compresses them as the final threshold signature. This scheme has the greatest advantage since it supports non-interactive signing without preprocessing. Recently, some studies have been conducted to analyze the security the TS-BLS scheme [1].

Multi-signatures can be seen as the special type of threshold signatures because all parties generate partial signatures for the same message and these partial signatures are combined into a multi-signature [32]. A simple threshold signature scheme can be built from a multi-signature scheme based on BLS signatures (MS-BLS) if a verifier checks the threshold of signing parties and aggregates the public keys of these parties [9]. The advantage of this scheme is that the key setup is simple and non-interactive, but the size of a verification key increases in proportion to the number of parties since all public keys of parties are required for verification. An efficient accountable-subgroup multi-signature (ASM) scheme based on BLS signatures was proposed [10], but the key setup process is two rounds and requires message exchanges between parties. It is also possible to construct a threshold signature scheme from succinct non-interactive arguments of knowledge (SNARK) [22, 27]. The basic idea is a combiner to produce a proof that it witnessed a sufficient number of signatures generated by enough parties. This method has the advantage that weights and general access structures can be easily applied. However, in general, the signing algorithm that uses efficient SNARKs is rather slow if the number of parties increases and it requires the trusted setup to generate structured reference strings.

2 Preliminaries

For any integer $n > 0$, let $[n]$ be the set of integers $\{1, \ldots, n\}$. For any integer $m < n$, let $[m : n]$ be the set of integers $\{m, m+1, \ldots, n-1, n\}$. Let $\vec{a} = (a_1, \ldots, a_n)$ and $\vec{b} = (b_1, \ldots, b_n)$ be vectors over $\mathbb{Z}_p^n$. We use $\vec{a}[i]$ to denote the ith element a_i of $\vec{a}$. We use $\langle \vec{a}, \vec{b} \rangle$ to denote the inner product $\sum_{i=1}^{n} a_i b_i$ of two vectors. We use $g^{\vec{a}}$ to denote the vector $(g^{a_1}, \ldots, g^{a_n})$. We use $\langle g^{\vec{a}}, \vec{b} \rangle$ to denote the vector inner product over the exponents as $\prod_{i=1}^{n} (g^{a_i})^{b_i} = g^{\sum_{i=1}^{n} a_i b_i}$. A function $f(\lambda)$ is negligible if for all polynomial $p(\lambda)$, $f(\lambda) < 1/p(\lambda)$ for all large enough security parameter λ.

2.1 Bilinear Groups

A bilinear group generator $\mathcal{BG}$ takes as input a security parameter λ and outputs a bilinear group $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})$ where p is a random prime, $\mathbb{G}, \hat{\mathbb{G}}$, and $\mathbb{G}_T$ are three cyclic groups of prime order p, and g, $\hat{g}$ are random generators of $\mathbb{G}$, $\hat{\mathbb{G}}$, respectively. The bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ has the following properties:

1. Bilinearity: $\forall u \in \mathbb{G}, \forall \hat{v} \in \hat{\mathbb{G}}$ and $\forall a, b \in \mathbb{Z}_p$, $e(u^a, \hat{v}^b) = e(u, \hat{v})^{ab}$.
2. Non-degeneracy: $\exists g \in \mathbb{G}, \hat{g} \in \hat{\mathbb{G}}$ such that $e(g, \hat{g})$ has order p in $\mathbb{G}_T$.

We say that $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})$ is a bilinear group with no efficiently computable isomorphisms if the group operations in $\mathbb{G}, \hat{\mathbb{G}}$, and $\mathbb{G}_T$ as well as the bilinear map e are all efficiently computable, but there are no efficiently computable isomorphisms between $\mathbb{G}$ and $\hat{\mathbb{G}}$.

2.2 Complexity Assumptions

In this section, we introduce complexity assumptions for the security proof of our NIDTS scheme.

Assumption 1 (Discrete Logarithm, DL [18])**.** Let $\mathbb{G}$ be a cyclic group of the prime order p and g be a random generator of $\mathbb{G}$. The experiment of DL assumption is defined as follows:

$$\begin{aligned}&\text{Experiment } \mathbf{Exp}_{\mathcal{A}}^{DL}(p, \mathbb{G}, g)\\ &\quad \alpha \leftarrow \mathbb{Z}_p^*; D \leftarrow \left(p, \mathbb{G}, g, g^{\alpha}\right); x \leftarrow \mathcal{A}(D);\\ &\quad \text{If } g^x = g^{\alpha} \text{ then return 1 else return 0.}\end{aligned}$$

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{DL}(\lambda) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{DL}(p, \mathbb{G}, g) = 1]$. We say that DL assumption holds if for every PPT adversary $\mathcal{A}$ the advantage $\mathbf{Adv}_{\mathcal{A}}^{DL}(\lambda)$ is negligible.

Assumption 2 (Computational co-Diffie-Hellman, co-CDH [11])**.** Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})$ be a bilinear group generated by $\mathcal{BG}(1^{\lambda})$. The experiment of co-CDH assumption is defined as follows:

$$\begin{aligned}&\text{Experiment } \mathbf{Exp}_{\mathcal{A}}^{co\text{-}CDH}(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})\\ &\quad \alpha, \beta \leftarrow \mathbb{Z}_p^*; D \leftarrow \left((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}), g^{\alpha}, \hat{g}^{\alpha}, \hat{g}^{\beta}\right); X \leftarrow \mathcal{A}(D);\\ &\quad \text{If } X = \hat{g}^{\alpha\beta} \text{ then return 1 else return 0.}\end{aligned}$$

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{co\text{-}CDH}(\lambda) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{co\text{-}CDH}(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}) = 1]$. We say that co-CDH assumption holds if for every PPT adversary $\mathcal{A}$ the advantage $\mathbf{Adv}_{\mathcal{A}}^{co\text{-}CDH}(\lambda)$ is negligible.

Assumption 3 (n-Augmented Knowledge of Exponent Assumption, n-AugKEA)**.** Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})$ be a bilinear group generated by $\mathcal{BG}(1^{\lambda})$. Let $h_1 = g^{b_1}, \ldots, h_n = g^{b_n} \in \mathbb{G}$, $\hat{h}_1 = \hat{g}^{b_1}, \ldots, \hat{h}_n = \hat{g}^{b_n} \in \hat{\mathbb{G}}$ where $b_1, \ldots, b_n \in \mathbb{Z}_p$. The experiment of n-AugKEA with an auxiliary input z is defined as follows:

$$\begin{aligned}&\text{Experiment } \mathbf{Exp}_{\mathcal{A},\mathbf{Ext}}^{n\text{-}AugKEA}((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}), h_1, \ldots, h_n, \hat{h}_1, \ldots, \hat{h}_n, z)\\ &\quad a \leftarrow \mathbb{Z}_p^*; D \leftarrow \left((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}), h_1, \ldots, h_n, \hat{g}^a, \hat{h}_1^a, \ldots, \hat{h}_n^a\right);\\ &\quad (X, Y) \leftarrow \mathcal{A}(D, z); (c_1, \ldots, c_n) \leftarrow \mathbf{Ext}(\mathcal{A}, D, z);\\ &\quad \text{If } e(X, \hat{g}^a) = e(g, Y) \wedge X \neq \prod_{i=1}^{n} g^{b_i c_i} \text{ then return 1 else return 0.}\end{aligned}$$

The advantage of $\mathcal{A}$ related to $\mathbf{Ext}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{n\text{-}AugKEA}(\lambda) = \Pr[\mathbf{Exp}_{\mathcal{A},\mathbf{Ext}}^{n\text{-}KEA}((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}), z) = 1]$. We say that n-AugKEA holds if for every PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathbf{Ext}$ such that the advantage $\mathbf{Adv}_{\mathcal{A}}^{n\text{-}AugKEA}(\lambda)$ is negligible for all $h_1, \ldots, h_n, \hat{h}_1, \ldots, \hat{h}_n$ and any auxiliary input z.

The n-AugKEA is a similar to the n-KEA [8] which is a natural generalization of the knowledge-of-exponent assumption (KEA) [6, 15] except that it holds for all $h_1, \ldots, h_n, \hat{h}_1, \ldots, \hat{h}_n$.

2.3 BLS Signature

The BLS signature scheme is described as follows:

BLS.Setup(1^λ): Let λ be the security parameter. Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})$ be a bilinear group generated by $\mathcal{BG}(1^\lambda)$. It chooses a hash function $H_1 : \{0,1\}^* \to \hat{\mathbb{G}}$. It outputs public parameters $PP = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}), H_1\big)$.

BLS.GenKey(PP): It selects a random exponents $s \in \mathbb{Z}_p$. It outputs a secret key $SK = s$ and a public key $PK = X = g^s$.

BLS.Sign(M, SK, PP): Let $M \in \{0,1\}^*$ and $SK = s$. It outputs a signature $\sigma = H_1(M)^s$.

BLS.Verify(σ, M, PK, PP): Let $PK = X$. It checks that $e(g, \sigma) \stackrel{?}{=} e(X, H_1(M))$. If the equation holds, then it outputs 1. Otherwise, it outputs 0.

Theorem 2.1 ([11]). *The BLS signature scheme is unforgeable in the random oracle model if the co-CDH assumption holds.*

2.4 NIZKPoK for DL

An NIZKPoK system of discrete logarithm problem consists of algorithms **Setup**, **Prove**, and **Verify** that satisfy the following properties:

- Correctness. For all $x \in \mathbb{Z}_p$, we have that $\Pr[\textbf{Verify}(CRS, g^x, \pi) = 1 : CRS \leftarrow \textbf{Setup}(1^\lambda), \pi \leftarrow \textbf{Prove}(CRS, g^x, x)] = 1$.
- Zero-knowledge. There exists a simulator **Sim** such that for all g^x, the output of $\textbf{Sim}(g^x)$ is indistinguishable from the output of $\textbf{Prove}(CRS, g^x, x)$.
- Simulation Extraction. For all PPT $\mathcal{A}$ and $x \in \mathbb{Z}_p$, there exists a simulator **Sim** and an extractor **Ext** such that $\Pr[\textbf{Verify}(CRS, g^x, \pi) = 1 \wedge (\textbf{Ext}(\mathcal{A}, g^x, \pi) \neq x) : CRS \leftarrow \textbf{Setup}(1^\lambda), \pi \leftarrow \mathcal{A}^{\textbf{Sim}}(CRS, g^x)] \leq negl(\lambda)$.

The Schnorr NIZKPoK system which is one instance of NIZKPoK for DL is described as follows:

SchPoK.Setup(1^λ): Let λ be the security parameter. It generates a cyclic group $\mathbb{G}$ of prime order p of bit size $\Theta(\lambda)$ with a random generator g of $\mathbb{G}$. It chooses a hash function $H : \mathbb{G}^3 \to \mathbb{Z}_p$. It outputs $CRS = (p, \mathbb{G}, g, H)$.

SchPoK.Prove(CRS, g^x, x): It selects a random exponent $r \in \mathbb{Z}_p$ and computes $R = g^r$. It derives $h = H(g, g^x, R)$ and computes $z = r + hx$. It outputs $\pi = (R, z)$.

SchPoK.Verify(CRS, g^x, π): Let $\pi = (R, z)$. It computes $h = H(g, g^x, R)$ and checks that $g^z \stackrel{?}{=} R(g^x)^h$. If the equation holds, then it outputs 1. Otherwise, it outputs 0.

Theorem 2.2 ([34, 36]). *The Schnorr NIZKPoK system is correct, zero-knowledge, and simulation extractable in the random oracle model if the DL assumption holds.*

Remark 1. To efficiently extract the knowledge, we can use the NIZKPoK system of Fischlin [19] since it is an online extractor.

3 Non-Interactive Decentralized Threshold Signatures

In this section, we define the concept of NIDTS and propose an efficient NIDTS scheme that supports the non-interactive key setup.

3.1 Definition

To define the syntax of NIDTS, we first define a weighted group of parties in which each party has an individual weight.

Definition 3.1 (Weighted Group)**.** Let $\mathcal{P} = \{P_1, P_2, \ldots\}$ be the set of all parties in a system. Let m, n, and t be positive integers such that $2 \leq m \leq n$ and $t \leq n$. Let $w_1, \ldots, w_m$ be positive integers such that $n = \sum_{i=1}^{m} w_i$. A group of weighted parties is specified by a group description $GD = (GID, \{(PK_i, RK_i, P_i, w_i)\}_{i=1}^{m}, n, t)$ which consists of a group identifier string $GID \in \{0,1\}^*$, a set of tuples $\{(PK_i, RK_i, P_i, w_i)\}_{i=1}^{m}$ which specifies group parties in $P_G = \{P_1, \ldots, P_m\} \subseteq \mathcal{P}$ where a party P_i with a weight w_i is associated with a public key PK_i and a register key RK_i, the total weight n of all parties, and a threshold t. We assume that parties in P_G are indexed by some canonical ordering. We say that any subset $S \subseteq P_G$ is authorized if and only if $\sum_{P_i \in S} w_i \geq t$.

We define the syntax of NIDTS by modifying the syntax of MTS [2]. In NIDTS, each party generates a secret key SK and a public key PK by running the key generation algorithm and adds PK to a blockchain. After that, the party generates a register key RK by running the register key generation algorithm with a group identifer GID as input and send it to a combiner. The external untrusted combiner collects all public keys and register keys of group parties with GID, and then runs the group setup algorithm to create a combination key CK and a verification key VK of this group. At this time, the combiner can add VK in the blockchain. If individual parties generate partial signatures $\{\sigma_i\}$ for a message M by using their secret keys, then the combiner collects a sufficient number of partial signatures and runs the combining algorithm to derive a final threshold signature σ_τ. The verification of σ_τ can be done by running the verification algorithm. The detailed syntax of NIDTS is defined as follows:

Definition 3.2 (Non-Interactive Decentralized Threshold Signature, NIDTS)**.** A non-interactive decentralized threshold signature (NIDTS) scheme for the set of parties $\mathcal{P} = \{P_1, P_2, \ldots\}$ that supports decentralized, non-interactive, and transparent key setup with non-interactive threshold signing consists of eight PPT algorithms **Setup**, **GenKey**, **GenRegKey**, **GroupSetup**, **Sign**, **VerifyPart**, **Combine**, and **Verify**, which are defined as follows:

Setup(1^λ). The setup algorithm takes as input the security parameter λ in unary, and outputs public parameters PP.

GenKey(PP). The key generation algorithm takes as input public parameters PP. It outputs a secret key SK and a public key PK.

GenRegKey(GID, SK, PK, PP). The register key generation algorithm takes as input a group identifiers GID in which a party will join, a secret key SK, a public key PK, and public parameters PP. It outputs a register key RK.

GroupSetup(GD, PP). The group setup algorithm takes as input a group description GD and public parameters PP. It *deterministically* outputs a combine key CK and a verification key VK.

Sign(M, SK, PP). The signing algorithm takes as input a message M, a secret key SK, and public parameters PP. It outputs a partial signature σ.

VerifyPart(σ, M, PK, PP). The partial signature verification algorithm takes as input a partial signature σ, a message M, a public key PK, and public parameters PP. It outputs 1 if the partial signature is valid and 0 otherwise.

Combine(SS, M, GD, CK, PP). The combining algorithm takes as input a set of partial signatures $SS = \{\sigma_i\}$, a message M, a group description GD, a combine key CK, and public parameters PP. It outputs a threshold signature σ_τ.

Verify(σ_τ, M, VK, PP). The verification algorithm takes as input a threshold signature σ_τ, a message M, a verification key VK, and public parameters PP. It outputs 1 if the signature is valid and 0 otherwise.

Remark 2. If the group identifier GID of a group is known in advance, each party can generate a public key PK and a register key RK simultaneously. Thus, if each party creates an extended public key $EPK = (PK, RK)$ that contains the public key and the register key, the key setup process of NIDTS can be zero round.

Definition 3.3 (Correctness)**.** The correctness of an NIDTS scheme is defined as the following experiment $\mathbf{Exp}^{CO}_{\mathcal{A}}(1^\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. The challenger $\mathcal{C}$ obtains PP by running **Setup**(1^λ) and gives PP to the adversary $\mathcal{A}$. Let P_H be the set of honest parties, P_C be the set of corrupted parties, and $P_{H,M}$ be the set of honest parties which have signed a message M. $\mathcal{C}$ initializes P_H, P_C, and $P_{H,M}$ as empty.

2. $\mathcal{A}$ can access oracles **OGenKey**, **OGenRegKey**, **OCorrupt**, and **OSign**, which are defined as follows:

   - **OGenKey**(P_i): If $P_i \notin P_H \cup P_C$, then it generates SK_i, PK_i by running **GenKey**(PP), updates $P_H = P_H \cup \{P_i\}$, and responds PK_i. Otherwise, it responds $\perp$.
   - **OGenRegKey**(P_i, GID): If $P_i \in P_H$, then it generates RK_i by running **GenRegKey**(GID, SK_i, PK_i, PP) and responds RK_i. Otherwise, it responds $\perp$.
   - **OCorrupt**(P_i): It updates $P_C = P_C \cup \{P_i\}$. If $P_i \in P_H$, then it updates $P_H = P_H \setminus \{P_i\}$, and responds SK_i. Otherwise, it responds $\perp$.
   - **OSign**(P_i, M): If $P_i \in P_H$, then it updates $P_{H,M} = P_{H,M} \cup \{P_i\}$ and responds **Sign**(M, SK_i, PP). Otherwise, it responds $\perp$.

3. $\mathcal{A}$ outputs a group description $GD^* = (GID, \{(PK_i, RK_i, P_i, w_i)\}_{i=1}^m, n, t)$, a message M^*, and partial signatures $SS^* = \{\sigma_j\}_{P_j \in S}$ for a set of parties S. Let P_G be the set of all parties in GD^*. Let $S_H \subseteq S$ be the subset of honest partial signatures where all partial signatures $\{\sigma_j\}_{P_j \in S_H}$ are honestly generated by calling **OSign**(P_j, M^*). $\mathcal{C}$ derives a combine key CK^* and a verification key VK^* by running **GroupSetup**(GD^*, PP).

4. The output of this experiment is 1 if and only if at least one of the following three conditions is satisfied:

   1) All parties in the group description are corrupted, i.e., $P_G \subseteq P_C$.
   2) The subset of honest partial signatures is not an authorized subset, i.e., $\sum_{P_i \in S_H} w_i < t$.
   3) The combined threshold signature is valid, i.e, **Verify**$(\sigma^*_\tau, M^*, VK^*, PP) = 1$ where $\sigma^*_\tau =$ **Combine** $(SS^*, M^*, GD^*, CK^*, PP)$.

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}^{CO}_{\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{CO}_{\mathcal{A}}(1^\lambda) = 1]$ where the probability is taken over all the randomness of the experiment. An NIDTS scheme satisfies correctness if for all PPT adversaries the advantage is 1 except with negligible probability.

We define the security model of NIDTS by following the security model of the MTS as follows:

Definition 3.4 (Unforgeability)**.** The unforgeability of an NIDTS scheme is defined as the following experiment $\mathbf{Exp}^{UF}_{\mathcal{A}}(1^\lambda)$ between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:

1. The challenger $\mathcal{C}$ obtains PP by running $\mathbf{Setup}(1^\lambda)$ and gives PP to the adversary $\mathcal{A}$. Let P_H be the set of honest parties, P_C be the set of corrupted parties, and $P_{H,M}$ be the set of honest parties which have signed a message M. $\mathcal{C}$ initializes P_H, P_C, and $P_{H,M}$ as empty.

2. $\mathcal{A}$ can access oracles **OGenKey**, **OGenRegKey**, **OCorrupt**, and **OSign**, which are defined as follows:

   - $\mathbf{OGenKey}(P_i)$: If $P_i \notin P_H \cup P_C$, then it generates SK_i, PK_i by running $\mathbf{GenKey}(PP)$, updates $P_H = P_H \cup \{P_i\}$, and responds PK_i. Otherwise, it responds $\perp$.
   - $\mathbf{OGenRegKey}(P_i, GID)$: If $P_i \in P_H$, then it generates RK_i by running $\mathbf{GenRegKey}(GID, SK_i, PK_i, PP)$ and responds RK_i. Otherwise, it responds $\perp$.
   - $\mathbf{OCorrupt}(P_i)$: It updates $P_C = P_C \cup \{P_i\}$. If $P_i \in P_H$, then it updates $P_H = P_H \setminus \{P_i\}$, and responds SK_i. Otherwise, it responds $\perp$.
   - $\mathbf{OSign}(P_i, M)$: If $P_i \in P_H$, then it updates $P_{H,M} = P_{H,M} \cup \{P_i\}$ and responds $\mathbf{Sign}(M, SK_i, PP)$. Otherwise, it responds $\perp$.

3. $\mathcal{A}$ outputs a group description $GD^* = (GID, \{(PK_i, RK_i, P_i, w_i)\}_{i=1}^m, n, t)$, a message M^*, and a threshold signature σ^*_τ. Let P_G be the set of all parties in GD^*. $\mathcal{C}$ derives a combine key CK^* and a verification key VK^* by running $\mathbf{GroupSetup}(GD^*, PP)$.

4. The output of this experiment is 1 if all of the following two conditions are satisfied:

   1) The adversary has not acquired sufficient partial signatures, i.e., $\sum_{P_i \in P_G \cap (P_C \cup P_{H,M^*})} w_i < t$.
   2) The threshold signature is valid, i.e., $\mathbf{Verify}(\sigma^*_\tau, M^*, VK^*, PP) = 1$.

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}^{UF}_{\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{UF}_{\mathcal{A}}(1^\lambda) = 1]$ where the probability is taken over all the randomness of the game. An NIDTS scheme satisfies unforgeability if for all PPT adversaries the advantage is negligible.

3.2 Construction in the Unweighted Setting

In this section, we propose an unweighted NIDTS scheme based on bilinear maps.

Definition 3.5 (Unweighted Group)**.** Let $\mathcal{P} = \{P_1, P_2, \ldots\}$ be the set of all parties in a system. Let n and t be positive integers such that $2 \leq n$ and $t \leq n$. A group with unweighted parties is specified by a group description $GD = (GID, \{(PK_i, RK_i, P_i)\}_{i=1}^m, n = m, t)$ which consists of a group identifier string $GID \in \{0,1\}^*$, a set of tuples $\{(PK_i, RK_i, P_i)\}_{i=1}^m$ which specifies group parties $P_G = \{P_1, \ldots, P_m\} \subseteq \mathcal{P}$ where a party P_i is associated with a public key PK_i and a register key RK_i, the total number m of all parties, and a threshold t. We assume that parties in P_G are indexed by some canonical ordering. We say that any subset $S \subseteq P_G$ is authorized if and only if $|S| \geq t$.

NIDTS.Setup(1^λ)**:** It first obtains a bilinear group $(p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e,g,\hat{g})$ by running $\mathcal{BG}(1^\lambda)$. It prepares CRS of NIZKPoK for DL on the cyclic group $\mathbb{G}$. It chooses two hash functions $H_0:\{0,1\}^*\to\hat{\mathbb{G}}$ and $H_1:\{0,1\}^*\to\hat{\mathbb{G}}$ and outputs public parameters $PP=\big((p,\mathbb{G},\hat{\mathbb{G}},\mathbb{G}_T,e,g,\hat{g}),H_0,H_1,CRS\big)$.

NIDTS.GenKey(PP)**:** It selects a random exponent $s\in\mathbb{Z}_p^*$ and computes $X=g^s$. It outputs a secret key $SK=s$ and a public key $PK=X$.

NIDTS.GenRegKey(GID,SK,PK,PP)**:** Let $SK=s$ and $PK=X$. It computes $Y=H_0(GID)^s$ and $\pi=$ **NIZKPoK.Prove**(CRS,X,s). It outputs a register key $RK=\big(GID,Y,\pi\big)$.

NIDTS.GroupSetup(GD,PP)**:** Let $GD=(GID,\{(PK_i,RK_i,P_i)\}_{i=1}^m,n=m,t)$.

1. For each pair $(PK_i=X_i,RK_i=(GID',Y_i,\pi_i))\in GD$, it checks that $GID=GID'$ and $e(g,Y_i)\stackrel{?}{=}e(X_i,H_0(GID))\ \wedge$ **NIZKPoK.Verify**$(CRS,X_i,\pi_i)\stackrel{?}{=}1$. If one of these checks fail, it outputs $\perp$.
2. It defines a polynomial $f\in\mathbb{Z}_p[X]$ of degree $n-1$ such that $f(i)=s_i$ for each $i\in[1:n]$ where $\mathrm{dlog}_g(X_i)=s_i$. That is, it defines $f(x)=\sum_{i\in[n]}f(i)L_i(x)$ by using the Lagrange interpolation where $f(i)=s_i$ and $L_i(x)=\prod_{k\in[n],k\neq i}\frac{x-k}{i-k}$ is the Lagrange basis.
3. For each $k\in\{-(n-t),\ldots,-2,-1\}$, it derives the Lagrange coefficients $\{L_i(k)\}_{i\in[n]}$ and computes components as
$$V_k=g^{f(k)}=\prod_{i\in[n]}X_i^{L_i(k)},\ W_k=H_0(GID)^{f(k)}=\prod_{i\in[n]}Y_i^{L_i(k)}.$$
Next, it derives the Lagrange coefficients $\{L_i(0)\}_{i\in[n]}$ and computes a component as
$$V_0=g^{f(0)}=\prod_{i\in[n]}X_i^{L_i(0)}.$$
4. It outputs a combine key $CK=\big(\{V_k\}_{k\in[-(n-t):-1]},\{W_k\}_{k\in[-(n-t):-1]}\big)$ and a verification key $VK=\big(V_0,W=H_0(GID)\big)$.

NIDTS.Sign(M,SK,PP)**:** Let $M\in\{0,1\}^*$ and $SK=s$. It outputs a partial signature $\sigma=H_1(M)^s\in\hat{\mathbb{G}}$.

NIDTS.VerifyPart(σ,M,PK,PP)**:** Let $PK=X$. It checks that $e(g,\sigma)\stackrel{?}{=}e(X,H_1(M))$. If the equation holds, it outputs 1. Otherwise, it outputs 0.

NIDTS.Combine(SS,M,GD,CK,PP)**:** Let $SS=\{\sigma_i\}$, $GD=(GID,\{(PK_i,RK_i,P_i)\}_{i=1}^m,n=m,t)$, and $CK=(\{V_k\},\{W_k\})$. Let P_σ be the set of parties in SS and P_G be the set of parties in GD.

1. It initializes $Q=\emptyset$. For each $P_i\in P_\sigma\cap P_G$, it adds P_i to Q if **VerifyPart**$(\sigma_i,M,PK_i,PP)=1$. If $|Q|<t$, it outputs $\perp$.
2. Let X_Q be the set of t evaluation points corresponding to parties in Q since parties can be indexed by some canonical ordering. It derives the Lagrange coefficients $\{L_i(0)\}_{i\in X_Q\cup[-(n-t):-1]}$ that satisfy $f(0)=\sum_{i\in X_Q\cup[-(n-t):-1]}f(i)L_i(0)$. We simply write $\{L_i(0)\}_{i\in X_Q}$ as $\{L_i^{par}\}$ and $\{L_i(0)\}_{i\in[-(n-t):-1]}$ as $\{L_i^{pub}\}$.
3. Next, it builds signature components
$$S_0=\prod_{i\in X_Q}\sigma_i^{L_i^{par}},\ S_1=\prod_{i\in[-(n-t):-1]}V_i^{L_i^{pub}},\ S_2=\prod_{i\in[-(n-t):-1]}W_i^{L_i^{pub}}.$$

4. It outputs a threshold signature $\sigma_\tau = (S_0, S_1, S_2) \in \hat{\mathbb{G}} \times \mathbb{G} \times \hat{\mathbb{G}}$.

NIDTS.Verify(σ_τ, M, VK, PP)**:** Let $\sigma_\tau = (S_0, S_1, S_2)$ and $VK = (V_0, W)$. It checks that

$$e(g, S_0) \stackrel{?}{=} e(V_0/S_1, H_1(M)) \ \wedge \ e(S_1, W) \stackrel{?}{=} e(g, S_2).$$

If the equations hold, it outputs 1. Otherwise, it outputs 0.

3.3 Construction in the Weighted Setting

In this section, we propose a weighted NIDTS scheme based on bilinear maps. The method of modifying our unweighted NIDTS scheme to support weights is a virtualization method in which individual parties additionally have shared secrets corresponding to weights.

NIDTS.Setup(1^λ)**:** It first obtains a bilinear group $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})$ by running $\mathcal{BG}(1^\lambda)$. It prepares CRS of NIZKPoK for DL on the cyclic group $\mathbb{G}$. It chooses two hash functions $H_0 : \{0,1\}^* \to \hat{\mathbb{G}}$ and $H_1 : \{0,1\}^* \to \hat{\mathbb{G}}$, and outputs public parameters $PP = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}), H_0, H_1, CRS, w\big)$ where w is the weight bound.

NIDTS.GenKey(PP)**:** It selects a random vector $\vec{s} = (s_1, \ldots, s_w) \in \mathbb{Z}_p^w$ and computes $\vec{X} = g^{\vec{s}}$. It outputs a secret key $SK = \vec{s}$ and a public key $PK = \vec{X}$.

NIDTS.GenRegKey(GID, SK, PK, PP)**:** Let $SK = \vec{s}$ and $PK = \vec{X}$. It computes $\vec{Y} = H_0(GID)^{\vec{s}}$ and $\vec{\pi}$ by running **NIZKPoK.Prove**$(CRS, \vec{X}[j], \vec{s}[j])$ for each $j \in [w]$. It outputs a register key $RK = \big(GID, \vec{Y}, \vec{\pi}\big)$.

NIDTS.GroupSetup(GD, PP)**:** Let $GD = (GID, \{(PK_i, RK_i, P_i, w_i)\}_{i=1}^m, n, t)$ where $n = \sum_{i=1}^m w_i$ and $w_i \leq w$.

1. For each pair $(PK_i = \vec{X}_i, RK_i = (GID', \vec{Y}_i, \vec{\pi}_i)) \in GD$, it checks $GID \stackrel{?}{=} GID'$ and performs the following checks:
   (a) It checks that $\vec{Y}_i \in \hat{\mathbb{G}}^w$ and $e(g, \langle \vec{Y}_i, \vec{r} \rangle) \stackrel{?}{=} e(\langle \vec{X}_i, \vec{r} \rangle, H_0(GID))$ by selecting a random vector $\vec{r} \in \mathbb{Z}_p^w$.
   (b) It checks that **NIZKPoK.Verify**$(CRS, \vec{X}_i[j], \vec{\pi}_i[j]) \stackrel{?}{=} 1$ for each $j \in [w]$.
   If one of these checks fail, it outputs $\perp$.
2. Let $\overline{w}_i = \sum_{k=1}^i w_k$ be the added weights of ith party. We define ψ_1 as a mapping from $j \in [n]$ to $i \in [m]$ such that $\psi_1(j) = i$ for all $j \in [\overline{w}_{i-1} + 1 : \overline{w}_{i-1} + w_i]$. We also define ψ_2 as a mapping from $j \in [n]$ to $i \in [w_{\psi_1(j)}]$ such that $\psi_2(j) = j - \overline{w}_{\psi_1(j)-1}$. That is, ψ_1 returns the index of a party and ψ_2 returns the relative index of a public key element in a party [2].
3. It defines a polynomial $f \in \mathbb{Z}_p[X]$ of degree $n-1$ such that $f(j) = \vec{s}_{\psi_1(j)}[\psi_2(j)]$ for each $j \in [1:n]$ where $\vec{s}_{\psi_1(j)}[\psi_2(j)] = \mathrm{dlog}_g(\vec{X}_{\psi_1(j)}[\psi_2(j)])$. That is, it defines $f(x) = \sum_{j \in [n]} f(j) L_j(x)$ by using the Lagrange interpolation where $f(j) = \vec{s}_{\psi_1(j)}[\psi_2(j)]$ and $L_j(x) = \prod_{k \in [n], k \neq j} \frac{x-k}{j-k}$ is the Lagrange basis.

[2] For instance, if $GD = (GID, \{(\ldots, P_1, w_1 = 3), (\ldots, P_2, w_2 = 1), (\ldots, P_3, w_3 = 2)\}, m = 3, n = 6, t = 4)$, then $\{\psi_1(1) = 1, \psi_1(2) = 1, \psi_1(3) = 1, \psi_1(4) = 2, \psi_1(5) = 3, \psi_1(6) = 3\}$ and $\{\psi_2(1) = 1, \psi_2(2) = 2, \psi_2(3) = 3, \psi_2(4) = 1, \psi_2(5) = 1, \psi_2(6) = 2\}$.

4. For each $k \in \{-(n-t), \ldots, -2, -1\}$, it derives the Lagrange coefficients $\{L_j(k)\}_{j\in[n]}$ and computes components as

$$V_k = g^{f(k)} = \prod_{j\in[n]} \vec{X}_{\psi_1(j)}[\psi_2(j)]^{L_j(k)},\ W_k = H_0(GID)^{f(k)} = \prod_{j\in[n]} \vec{Y}_{\psi_1(j)}[\psi_2(j)]^{L_j(k)}.$$

Next, it derives the Lagrange coefficients $\{L_j(0)\}_{j\in[n]}$ and computes a component as

$$V_0 = g^{f(0)} = \prod_{j\in[n]} \vec{X}_{\psi_1(j)}[\psi_2(j)]^{L_j(0)}.$$

5. It outputs a combine key $CK = \big(\{V_k\}_{k\in[-(n-t):-1]}, \{W_k\}_{k\in[-(n-t):-1]}\big)$ and a verification key $VK = \big(V_0, W = H_0(GID)\big)$.

NIDTS.Sign(M, SK, PP)**:** Let $M \in \{0,1\}^*$ and $SK = \vec{s}$. It outputs a partial signature $\vec{\sigma} = H_1(M)^{\vec{s}} \in \hat{\mathbb{G}}^w$.

NIDTS.VerifyPart$(\vec{\sigma}, M, PK, PP)$**:** Let $PK = \vec{X}$. It checks that $\vec{\sigma} \in \hat{\mathbb{G}}^w$ and $e(g, \langle\vec{\sigma}, \vec{r}\rangle) \stackrel{?}{=} e(\langle\vec{X}, \vec{r}\rangle, H_1(M))$ by selecting a random vector $\vec{r} \in \mathbb{Z}_p^w$. If the equation holds, it outputs 1. Otherwise, it outputs 0.

NIDTS.Combine(SS, M, GD, CK, PP)**:** Let $SS = \{\vec{\sigma}_i\}$, $GD = (GID, \{(PK_i, RK_i, P_i, w_i)\}_{i=1}^m, n, t)$ where $n = \sum_{i=1}^m w_i$, and $CK = (\{V_k\}, \{W_k\})$. Let P_σ be the set of parties in SS and P_G be the set of parties in GD.

1. It initializes $Q = \emptyset$ and $w_Q = 0$. For each $P_i \in P_\sigma \cap P_G$, it adds (P_i, w_i) to Q and updates $w_Q = w_Q + w_i$ if **VerifyPart**$(\vec{\sigma}_i, M, PK_i, PP) = 1$. If $w_Q < t$, it outputs $\perp$.
2. Let X_Q be the set of t evaluation points corresponding to weighted parties in Q since parties can be indexed by some canonical ordering [3]. It derives the Lagrange coefficients $\{L_i(0)\}_{i\in X_Q \cup [-(n-t):-1]}$ that satisfy $f(0) = \sum_{i\in X_Q\cup[-(n-t):-1]} f(i)L_i(0)$. We simply write $\{L_i(0)\}_{i\in X_Q}$ as $\{L_i^{par}\}$ and $\{L_i(0)\}_{i\in[-(n-t):-1]}$ as $\{L_i^{pub}\}$.
3. Next, it builds signature components

$$S_0 = \prod_{i\in X_Q} \sigma_i^{L_i^{par}},\ S_1 = \prod_{i\in[-(n-t):-1]} V_i^{L_i^{pub}},\ S_2 = \prod_{i\in[-(n-t):-1]} W_i^{L_i^{pub}}.$$

4. It outputs a threshold signature $\sigma_\tau = (S_0, S_1, S_2) \in \hat{\mathbb{G}} \times \mathbb{G} \times \hat{\mathbb{G}}$.

NIDTS.Verify(σ_τ, M, VK, PP)**:** Let $\sigma_\tau = (S_0, S_1, S_2)$ and $VK = (V_0, W)$. It checks that

$$e(g, S_0) \stackrel{?}{=} e(V_0/S_1, H_1(M)) \ \wedge\ e(S_1, W) \stackrel{?}{=} e(g, S_2).$$

If the equations hold, it outputs 1. Otherwise, it outputs 0.

[3] For instance, if $GD = (GID, \{(\ldots, P_1, w_1 = 3), (\ldots, P_2, w_2 = 1), (\ldots, P_3, w_3 = 2)\}, m = 3, n = 6, t = 4)$ and $Q = \{(P_1, w_1 = 3), (P_3, w_3 = 2)\}$, then $X_Q = \{1, 2, 3\} \cup \{5\}$.

3.4 Correctness

Theorem 3.1. *The NIDTS scheme is correct if the NIZKPoK system is correct.*

Proof. For the correctness of the NIDTS scheme, we should show that if the first two conditions of the correctness experiment are not satisfied, i.e., $(P_G \not\subseteq P_C) \wedge (\sum_{P_i \in S_H} w_i \geq t)$, then the third condition should be satisfied with all but negligible probability, i.e., $\mathbf{Verify}(\sigma_\tau^*, M^*, VK^*, PP) = 1$ where $\sigma_\tau^* = \mathbf{Combine}(SS^*, M^*, GD^*, CK^*, PP)$.

In the **GroupSetup** algorithm, we have that $e(g, \vec{Y}_i[j]) = e(\vec{X}_i[j], H_0(GID))$ and $\mathbf{NIZKPoK.Verify}(CRS, \vec{X}_i[j], \vec{\pi}_i[j]) = 1$ for all $i \in [m]$ and $j \in [w]$ from the property of batch verification [5, 13] and the correctness of NIZKPoK. Thus we have that the following equation for V_j and W_j in CK^* holds

$$\begin{aligned} e(V_k, W) &= e(\prod_{j \in [n]} \vec{X}_{\psi_1(j)}[\psi_2(j)]^{L_j(k)}, H_0(GID)) = \prod_{j \in [n]} e(\vec{X}_{\psi_1(j)}[\psi_2(j)], H_0(GID))^{L_j(k)} \\ &= \prod_{j \in [n]} e(g, \vec{Y}_{\psi_1(j)}[\psi_2(j)])^{L_j(k)} = e(g, \prod_{j \in [n]} \vec{Y}_{\psi_1(j)}[\psi_2(j)]^{L_j(k)}) = e(g, W_k). \end{aligned}$$

From the above equation, we have that the following second verification equation for S_1 and S_2 in σ_τ^* holds

$$\begin{aligned} e(S_1, W) &= e(\prod_{i \in [-(n-t):-1]} V_i^{L_i^{pub}}, W) = \prod_{i \in [-(n-t):-1]} e(V_i, W)^{L_i^{pub}} \\ &= \prod_{i \in [-(n-t):-1]} e(g, W_i)^{L_i^{pub}} = e(g, \prod_{i \in [-(n-t):-1]} W_i^{L_i^{pub}}) = e(g, S_2). \end{aligned}$$

For any $P_j \in S_H$, we have that $\vec{\sigma}_j = H_1(M^*)^{\vec{s}_j}$ and it passes **VerifyPart** since $\vec{\sigma}_j$ is generated by the oracle $\mathbf{OSign}(P_j, M^*)$ where $\vec{s}_j$ is the secret key of P_j. We also claim that for any $P_j \in Q \setminus S_H$, we have $\vec{\sigma}_j = H_1(M^*)^{\vec{s}_j}$ and it passes **VerifyPart** since if $\vec{\sigma}_j \neq H_1(M^*)^{\vec{s}_j}$, then $e(g, \langle \vec{\sigma}_j, \vec{r} \rangle) \neq e(\langle \vec{X}_j, \vec{r} \rangle, H_1(M^*))$ except with negligible probability by the property of batch verification. Thus we have $\sum_{P_i \in Q} w_i \geq t$ since $\sum_{P_i \in S_H} w_i \geq t$ and partial signatures in $S_H \cup (Q \setminus S_H)$ are valid. Since the degree of the polynomial $f(x)$ is at most $n-1$ and $|X_Q| = t$, it can be correctly reconstructed by using the Lagrange interpolation at points in $X_Q \cup \{-(n-t), \ldots, -1\}$. Thus, the following first verification equation for S_0 and S_1 in σ_τ^* holds

$$\begin{aligned} e(V_0/S_1, H_1(M^*)) &= e(\hat{g}^{f(0)} / \prod_{i \in [-(n-t):-1]} V_i^{L_i^{pub}}, H_1(M^*)) \\ &= e(\hat{g}^{\sum_{i \in X_Q} f(i) L_i^{par} + \sum_{i \in [-(n-t):-1]} f(i) L_i^{pub}} / \hat{g}^{\sum_{i \in [-(n-t):-1]} f(i) L_i^{pub}}, H_1(M^*)) \\ &= e(\hat{g}^{\sum_{i \in X_Q} f(i) L_i^{par}}, H_1(M^*)) = e(g, H_1(M^*)^{\sum_{i \in X_Q} f(i) L_i^{par}}) \\ &= e(g, \prod_{i \in X_Q} H_1(M^*)^{f(i) L_i^{par}}) = e(g, \prod_{i \in X_Q} \sigma_i^{L_i^{par}}) = e(g, S_0). \end{aligned}$$

Therefore, we have $\mathbf{Verify}(\sigma_\tau^*, M^*, VK^*, PP) = 1$. □

3.5 Extensions

In this section, we describe an interesting modification or extension of our NIDTS schemes.

Short Partial Signatures. The underlying BLS signature scheme has two variants: the short signature scheme or the short public key scheme, depending on how the group elements of public keys and signatures

are located [11]. Our NIDTS scheme supports short public keys because public keys are located in $\mathbb{G}$ and partial signatures are located in $\hat{\mathbb{G}}$. The signature scheme with short public keys is suitable for a blockchain environment where the public keys of parties must be recorded on-chain. If short public keys are preferred than short partial signatures in some environments, our NIDTS scheme can be modified to support short partial signatures by locating public keys in $\hat{\mathbb{G}}$, register keys in $\mathbb{G}$, and partial signatures in $\mathbb{G}$. In this case, the final threshold signature consists of three group elements in $\mathbb{G}^2 \times \hat{\mathbb{G}}$.

Group Specific Signatures. In our NIDTS scheme, the partial signature of a party has the form of $H_1(M)^s$. Thus, if a party creates a partial signature on a message M, the party creates a partial signature that can be applied to all groups to which the party belongs. That is, this partial signature is not related to one specific group, but is related to all groups. If a party wants to generate only a partial signature for one specific group to which the user belongs, the party can generate a partial signature in the form of $H_1(GID||M)^s$ where GID is a group identifier. In this case, if all other parties also generates partial signatures in the same way, a threshold signature for this specific group can be derived by combining these partial signatures of parties.

Batch Verification for Partial Signatures. The combining algorithm first verifies whether the partial signatures provided by signing parties are valid or not, and then combines these valid partial signatures to derive a threshold signature. If we analyze the performance of this algorithm, we can notice that the partial signature verification step takes up most of the algorithm running time rather than the partial signature combining step. To improve the performance of this algorithm, we can perform batch verification to verify all partial signatures at once [5, 13]. That is, the algorithm first collects a number of partial signatures to perform batch verification, and if the batch verification fails, then it verifies each partial signature one by one. In this case, if the partial signatures are mostly valid signatures, the partial signature verification can be reduced to 2 pairing operations and $2t$ multi-exponentiation operations, except for checking the membership of group elements.

4 Security Analysis

In this section, we prove the security (unforgeability) of our NIDTS schemes under complexity assumptions.

For the proof, we basically follow the security proofs of the BLS scheme and the MTS scheme [2, 11]. The basic idea of the security proof is to construct a reduction algorithm that can extract the solution of the co-CDH assumption from the forgery of an attacker. Initially, the challenge of the co-CDH assumption is given as $(g, g^\alpha, \hat{g}^\alpha, \hat{g}^\beta)$. Next the reduction algorithm embeds the value g^α in the public key and the value $\hat{g}^\alpha$ in the register key of some honest parties. In addition, the reduction algorithm embeds the value $\hat{g}^\beta$ in the H_1 hash query for some messages.

In this case, if the attacker requests a corrupt query for an honest party, then the reduction can answer this query if g^α is not embedded in the public key of the party. If the attacker requests a signature query for a message M, then the reduction can answer this query for all cases except the case where g^α is embedded in the public key and $\hat{g}^\beta$ is embedded in $H_1(M)$. Finally, if the attacker submits a forged threshold signature for the target message M^*, $\hat{g}^\beta$ is embedded in $H_1(M^*)$, and g^α is embedded in the one of the secret keys of group parties, then the reduction can derive the solution $\hat{g}^{\alpha\beta}$ from the forged threshold signature.

The main part of the reduction proof is to show that the reduction can extract the co-CDH solution from the forged threshold signature submitted by the attacker. In the BLS scheme, it is relatively easy to extract the co-CDH solution since it is in the single user setting. In contrast, in the NIDTS scheme, it is not easy to extract the co-CDH solution since the forged signature include not only the secret keys of honest parties selected by reduction, but also the secret keys of corrupted parties selected by the attacker. The reduction

uses a proof-of-knowledge extractor to obtain the secret keys of the corrupted parties chosen by the attacker, and a knowledge extractor of the exponent to ensure that the attacker creates a forged signature as a linear combination of public components which are generated from the public keys of all parties. The detailed security analysis is given in the following theorem.

Theorem 4.1. *The NIDTS scheme is unforgeable in the random oracle model if the co-CDH and n-AugKEA assumptions hold and the NIZKPoK system is zero-knowledge and simulation extractable.*

Proof. We first define a sequence of hybrids $\mathbf{Game}_0, \mathbf{Game}_1, \ldots, \mathbf{Game}_4$ where $\mathbf{Game}_0$ is the original game and $\mathbf{Game}_4$ is the final game that solves the co-CDH problem. For the proof of this theorem, we first describe the simulator of the final game and later we explain how the simulator of the original game can be modified to the final simulator through a sequence of hybrid games.

Suppose there exists an adversary $\mathcal{A}$ that forges the above NIDTS scheme with non-negligible advantage ε. Let $\mathbf{Sim}_{NIZKPoK}$ and $\mathbf{Ext}_{NIZKPoK}$ be the simulator and extractor of the NIZKPoK system. Let $\mathbf{Ext}_{n\text{-}AugKEA}$ be the knowledge extractor of the n-AugKEA. A simulator $\mathcal{B}$ in the final hybrid $\mathbf{Game}_4$ that breaks the co-CDH assumption is given as input a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}), g^{\alpha}, \hat{g}^{\alpha}, \hat{g}^{\beta})$. $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Description of the Simulator: $\mathcal{B}$ prepares CRS of NIZKPoK for DL on the cyclic group $\hat{\mathbb{G}}$. It sets $PP = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}), CRS, W)$ and gives PP to $\mathcal{A}$.
$\mathcal{A}$ can access $\mathbf{H}_0, \mathbf{H}_1$ hash oracles, **OGenKey**, **OGenRegKey**, **OCorrupt**, and **OSign** oracles. Let q_h be the number of $\mathbf{H}_1$ hash queries and q_k be the number of key generation queries. $\mathcal{B}$ simulates these oracles as follows:

- $\mathbf{H}_0(GID)$: If $GID \in H_0$-List, then it retrieves (GID, u, k') from H_0-List and responses with u. Otherwise, it selects a random $k' \in \mathbb{Z}_p$ and sets $u = \hat{g}^{k'}$. It adds (GID, u, k') to H_0-List and responds with u.

- $\mathbf{H}_1(M)$: If $M \in H_1$-List, then it retrieves (M, h, b, s') from H_1-List and responses with h. Otherwise, it generates a random coin $b \in \{0,1\}$ with $\Pr[b = 1] = 1/q_h$. It selects a random $s' \in \mathbb{Z}_p$ and sets $h = (\hat{g}^{\beta})^b \hat{g}^{s'}$. It adds (M, h, b, s') to H_1-List and responds with h.

- **OGenKey**(P_i): If $P_i \in P_H \cup P_C$, it responds with $\perp$. It proceeds the following steps:

  1. It generates a random coin $c \in \{0,1\}$ with $\Pr[c = 1] = 1/(q_k + 1)$ and selects a random vector $\vec{r} \in \mathbb{Z}_p^w$. If $c = 1$, it selects a random vector $\vec{a} \in \mathbb{Z}_p^{*W}$. Otherwise, it sets $\vec{a} = 0^w$. If $c = 0$, it sets $SK = \vec{r}$. Otherwise, it sets SK as empty. It sets $\vec{X} = (g^{\alpha})^{c\vec{a}} g^{\vec{r}}$.
  2. It updates $P_H = P_H \cup \{P_i\}$. It also updates $P_E = P_E \cup \{P_i\}$ if $c = 1$. It sets $PK = \vec{X}$, adds $(P_i, SK, PK, c, \vec{a}, \vec{r})$ to Key-List, and responds with PK.

- **OGenRegKey**(P_i, GID): If $P_i \in P_H \cup P_C$, it responds with $\perp$. It proceeds the following steps:

  1. It retrieves $(P_i, SK, PK, c, \vec{a}, \vec{r})$ from Key-List. It also queries $H_0(GID)$ and retrieves (GID, u, k') from H_0-List and sets $\vec{Y} = ((\hat{g}^{\alpha})^{c\vec{a}} \hat{g}^{\vec{r}})^{k'}$.
  2. If $c = 0$, then it generates $\vec{\pi}$ by running **NIZKPoK.Prove**$(CRS, \vec{X}[j], SK[j])$ for each $j \in [w]$. Otherwise, it generates $\vec{\pi}$ by using the simulator $\mathbf{Sim}_{NIZKPoK}(\vec{X}[j])$ for each $j \in [w]$.
  3. It sets $RK = (GID, \vec{Y}, \vec{\pi})$ and responds with RK.

- **OCorrupt**(P_i): It updates $P_C = P_C \cup \{P_i\}$. If $P_i \in P_H$, it updates $P_H = P_H \setminus \{P_i\}$. Otherwise ($P_i \notin P_H$), it responds with $\perp$. It proceeds the following steps:

  1. It retrieves $(P_i, SK, PK, c, \vec{a}, \vec{r})$ from Key-List. If $c = 0$, it responds with SK. Otherwise, it declares failure ($\mathsf{Bad}_1 = 1$) and aborts since it cannot handle this corrupt query.

- **OSign**(P_i, M): If $P_i \notin P_H$, it responds with $\perp$. It updates $P_{H,M} = P_{H,M} \cup \{P_i\}$. It proceeds the following steps:

  1. It retrieves (M, h, b, s') from H_1-List and retrieves $(P_i, SK, PK, c, \vec{a}, \vec{r})$ from Key-List.
  2. If $c = 0$, it obtains $\vec{\sigma}$ by running **Sign**$(M, SK[j], PP)$ for each j and responds with $\vec{\sigma}$.
  3. If $c = 1 \wedge b = 0$, it sets $\vec{\sigma} = ((\hat{g}^{\alpha})^{c\vec{a}} \hat{g}^{\vec{r}})^{s'}$ and responds with $\vec{\sigma}$.
  4. Otherwise ($c = 1 \wedge b = 1$), it declares failure ($\mathsf{Bad}_2 = 1$) and aborts since it cannot handle this sign query.

$\mathcal{A}$ outputs a group description $GD^* = (GID, \{(PK_i, RK_i, P_i, w_i)\}_{i=1}^m, n, t)$, a message M^*, and a threshold signature σ_τ^*. Let P_G be the set of parties $P_i \in GD^*$. $\mathcal{B}$ derives $CK^* = (\{V_k\}, \{W_k\})$ and $VK^* = (V_0, W)$ by running **GroupSetup**(GD^*, PP) where $f(x)$ is the polynomial of this ad-hoc group, and then proceeds the following steps:

1. If $P_E \cap P_G = \emptyset$, then it declares failure ($\mathsf{Bad}_3 = 1$) and aborts since g^{α} is not embedded in the target group.

2. It retrieves (M^*, h, b, s') from H_1-List. If $b = 0$, then it declares failure ($\mathsf{Bad}_4 = 1$) and aborts since $\hat{g}^{\beta}$ is not embedded in the target message. Otherwise, we have that $H_1(M^*) = (\hat{g}^{\beta})^b \hat{g}^{s'}$.

3. If $\sum_{P_i \in P_G \cap (P_C \cup P_{H,M^*})} w_i < t$ and **Verify**$(\sigma_\tau^*, M^*, VK^*, PP) = 1$, then it proceeds as follows:

   (a) For each $P_i \in P_C \cap P_G$, it extracts a secret key $\vec{r}_i$ of PK_i by using the extractor $\mathbf{Ext}_{NIZKPoK}(\mathcal{A}, \vec{\pi}_i[j], \vec{X}_i[j])$ for each j where $PK_i = \vec{X}_i$ and $RK_i = (GID, \vec{Y}_i, \vec{\pi}_i)$. If the extraction fails, then it declares failure ($\mathsf{Bad}_5 = 1$) and aborts.

   (b) It computes (v_0, u_0) and $\{(v_j, u_j)\}_{j \in [-(n-t):-1]}$ such that $V_0 = (g^{\alpha})^{v_0} g^{u_0}$ and $V_j = (g^{\alpha})^{v_j} g^{u_j}$ for each $j \in [-(n-t) : -1]$ by using $\{\vec{r}_i\}_{P_i \in P_C \cap P_G}$ from extracted secret keys, $\{(c_i, \vec{a}_i, \vec{r}_i)\}_{P_i \in P_H \cap P_G}$ from Key-List, and Lagrange coefficients $\{L_i(k)\}_{i \in [n]}$ for all $k \in [-(n-t) : 0]$.
   This can be done as follows: Let L be the $n \times (n-t+1)$ matrix that maps $[f(1), \ldots, f(n)]$ to $[f(-(n-t)), \ldots, f(-1), f(0)]$ where the column of the matrix consists of $(L_1(k), \ldots, L_n(k))$ for $k \in [-(n-t) : 0]$ where $L_i(k)$ is the Lagrange coefficient. Then we have $(\vec{v}, v_0) = \vec{s}_a \cdot L$ and $(\vec{u}, u_0) = \vec{s}_r \cdot L$ where $\vec{s}_a = (\vec{a}_{\psi_1(1)}[\psi_2(1)] c_{\psi_1(1)}, \ldots, \vec{a}_{\psi_1(n)}[\psi_2(n)] c_{\psi_1(n)})$, $\vec{s}_r = (\vec{r}_{\psi_1(1)}[\psi_2(1)], \ldots, \vec{r}_{\psi_1(n)}[\psi_2(n)])$, $\vec{v} = (v_{-(n-t)}, \ldots, v_{-1})$, and $\vec{u} = (u_{-(n-t)}, \ldots, u_{-1})$ since $f(j) = \alpha \vec{a}_{\psi_1(j)}[\psi_2(j)] c_{\psi_1(j)} + \vec{r}_{\psi_1(j)}[\psi_2(j)]$ for $j \in [n]$ by setting $\vec{a}_{\psi_1(j)} = 0$ and $c_{\psi_1(j)} = 0$ if $P_{\psi_1(j)} \in P_C \cap P_G$.

   (c) It parses $\sigma_\tau^* = (S_0, S_1, S_2)$. It extracts coefficients $\{\delta_i\}_{i \in [-(n-t):-1]}$ by using the extractor $\mathbf{Ext}_{n\text{-}AugKEA}$ $(\mathcal{A}, CK^*)$ such that $S_1 = \prod_{i \in [-(n-t):-1]} V_i^{\delta_i}$ and $S_2 = \prod_{i \in [-(n-t):-1]} W_i^{\delta_i}$. If the extraction fails, then it declares failure ($\mathsf{Bad}_6 = 1$) and aborts. Next, it computes $v_1 = \langle \vec{v}, \vec{\delta} \rangle = \sum_{i \in [-(n-t):-1]} v_i \delta_i$ and $u_1 = \langle \vec{u}, \vec{\delta} \rangle = \sum_{i \in [-(n-t):-1]} u_i \delta_i$ such that $S_1 = (g^{\alpha})^{v_1} g^{u_1}$.

(d) If $v_0 - v_1 = 0$, then it declares failure ($\mathsf{Bad}_7 = 1$) and aborts. Otherwise, it outputs the co-CDH solution as

$$\hat{g}^{\alpha\beta} = \left(S_0 \cdot H_1(M^*)^{-(u_0-u_1)}\right)^{1/(v_0-v_1)} \cdot (\hat{g}^{\alpha})^{-s'}.$$

This completes the description of the simulator in the final game.

Analysis of the Hybrid Games. Let Forge_i be the event that the adversary $\mathcal{A}$ outputs a valid forgery that satisfies two conditions in a hybrid game $\mathbf{Game}_i$.

$\mathbf{Game}_0$: This game is the original experiment in Definition 3.4 with $\mathbf{H}_0, \mathbf{H}_1$ hash oracles. A simulator $\mathcal{B}$ in this game simply handles these hash oracle queries by selecting a random value and keeps this value in H_0-List and H_1-List hash tables respectively. The adversary $\mathcal{A}$ in this experiment will produce a valid forgery that satisfies two conditions with non-negligible probability. That is,

$$\Pr[\mathsf{Forge}_0] \geq \varepsilon. \tag{4.1}$$

$\mathbf{Game}_1$: In this game, we modify the register key generation oracle **OGenRegKey** of $\mathcal{B}$ to generate π of some public keys by using the simulator $\mathbf{Sim}_{NIZKPoK}$ of the NIZKPoK system although it has the corresponding secret keys. The modified **OGenKey** and **OGenRegKey** is described as follows:

- **OGenKey**(P_i): If $P_i \in P_H \cup P_C$, it responds with $\perp$. It proceeds the following steps:
  1. It generates a random coin $c \in \{0,1\}$ with $\Pr[c = 1] = 1/(q_k + 1)$ and selects a random vector $\vec{r} \in \mathbb{Z}_p^w$. It sets $SK = \vec{r}$. It sets $\vec{X} = g^{\vec{r}}$.
  2. It updates $P_H = P_H \cup \{P_i\}$. It sets $PK = \vec{X}$, adds $(P_i, SK, PK, c, -, \vec{r})$ to Key-List, and responds with PK.
- **OGenRegKey**(P_i, GID): If $P_i \notin P_H$, it responds with $\perp$. It proceeds the following steps:
  1. It retrieves $(P_i, SK, PK, c, -, \vec{r})$ from Key-List. It queries $H_0(GID)$ and retrieves (GID, u, k') from H_0-List and sets $\vec{Y} = (g^{\vec{r}})^{k'}$.
  2. If $c = 0$, then it generates $\vec{\pi}$ by running **NIZKPoK.Prove**$(CRS, \vec{X}[j], SK[j])$ for each j. Otherwise, it generates $\vec{\pi}$ by using the simulator $\mathbf{Sim}_{NIZKPoK}(\vec{X}[j])$ for each j.
  3. It sets $RK = (GID, \vec{Y}, \vec{\pi})$ and responds with RK.

Because of the zero-knowledge property of the NIZKPoK system, we have that

$$\Pr[\mathsf{Forge}_1] = \Pr[\mathsf{Forge}_0]. \tag{4.2}$$

$\mathbf{Game}_2$: In this game, we modify the key generation oracle **OGenKey** to embed the challenge g^{α} to some public keys and the hash oracle $\mathbf{H}_1$ to embed the challenge $\hat{g}^{\beta}$ to some messages. The modified oracles **OGenKey** and $\mathbf{H}_1$ are described in the above simulator. Because of this modification, the simulator $\mathcal{B}$ declares failure by setting bad events and aborts at the following places:

1) It declares failure ($\mathsf{Bad}_1 = 1$) and aborts in **OCorrupt** if it cannot answer a secret key without the knowledge of α.

2) It declares failure ($\mathsf{Bad}_2 = 1$) and aborts in **OSign** if it cannot answer a partial signature without the knowledge of α and β.

3) After the output of $\mathcal{A}$, it declares failure ($\mathsf{Bad}_3 = 1$) and aborts if g^{α} is not embedded to some public keys in the target group GD^*.

4) After the output of $\mathcal{A}$, it declares failure ($\mathsf{Bad}_4 = 1$) and aborts if $\hat{g}^{\beta}$ is not embedded to the target message M^*.

By using the conditional probability and the claims 4.2, 4.3, and 4.4, we have the following probability

$$\begin{aligned}\Pr[\wedge_{i=1}^{4}\neg\mathsf{Bad}_i \wedge \mathsf{Forge}_2] &= \Pr[\neg\mathsf{Bad}_1 \wedge \neg\mathsf{Bad}_2] \cdot \Pr[\mathsf{Forge}_2|\neg\mathsf{Bad}_1 \wedge \neg\mathsf{Bad}_2] \cdot \\ &\qquad \Pr[\mathsf{Forge}_2 \wedge \neg\mathsf{Bad}_3 \wedge \neg\mathsf{Bad}_4|\neg\mathsf{Bad}_1 \wedge \neg\mathsf{Bad}_2 \wedge \mathsf{Forge}_2] \\ &\geq 1/(e^4 q_h) \cdot \Pr[\mathsf{Forge}_1] \end{aligned} \tag{4.3}$$

where e is the base of natural logarithms.

Claim 4.2. $\Pr[\neg \mathit{Bad}_1 \wedge \neg \mathit{Bad}_2] \geq 1/e^2$.

Proof. By using the conditional probability and the fact that two events are independent since signing queries are requested for non-corrupted parties, we obtain the following probability

$$\Pr[\neg\mathsf{Bad}_1 \wedge \neg\mathsf{Bad}_2] = \Pr[\neg\mathsf{Bad}_1] \cdot \Pr[\neg\mathsf{Bad}_2].$$

Let q_c be the number of corrupt queries for honest parties and q_s be the number of signing queries. Since $q_c \leq q_k$ and $q_s \leq q_h q_k$, we can obtain the following lower bounds as

$$\begin{aligned}\Pr[\neg\mathsf{Bad}_1] &= (1 - 1/(q_k+1))^{q_c} \geq (1 - 1/(q_k+1))^{q_k} \geq 1/e, \\ \Pr[\neg\mathsf{Bad}_2] &= (1 - 1/(q_h(q_k+1)))^{q_s} \geq (1 - 1/(q_h q_k+1))^{q_h q_k} \geq 1/e.\end{aligned}$$

Thus, we have $\Pr[\neg\mathsf{Bad}_1 \wedge \neg\mathsf{Bad}_2] \geq 1/e^2$. □

Claim 4.3. $\Pr[\mathit{Forge}_2|\neg \mathit{Bad}_1 \wedge \neg \mathit{Bad}_2] = \Pr[\mathit{Forge}_1]$.

Proof. In the game $\mathbf{Game}_2$, the responds of hash queries, key generation queries, corrupt queries, and signing queries are the same as the game $\mathbf{Game}_1$ since public keys, corrupted secret keys, and signatures are all valid if two events Bad_1 and Bad_2 are not occurred. Thus, $\mathcal{A}$ will produce a valid forgery with probability $\Pr[\mathsf{Forge}_1]$. □

Claim 4.4. $\Pr[\mathit{Forge}_2 \wedge \neg \mathit{Bad}_3 \wedge \neg \mathit{Bad}_4|\neg \mathit{Bad}_1 \wedge \neg \mathit{Bad}_2 \wedge \mathit{Forge}_2] \geq 1/(e^2 q_h)$.

Proof. By using the independence of bad events and the conditional probability, we can derive the following probability

$$\begin{aligned}&\Pr[\mathsf{Forge}_2 \wedge \neg\mathsf{Bad}_3 \wedge \neg\mathsf{Bad}_4|\neg\mathsf{Bad}_1 \wedge \neg\mathsf{Bad}_2 \wedge \mathsf{Forge}_2] \\ &= \Pr[\neg\mathsf{Bad}_3|\neg\mathsf{Bad}_1 \wedge \neg\mathsf{Bad}_2 \wedge \mathsf{Forge}_2] \cdot \Pr[\neg\mathsf{Bad}_4|\neg\mathsf{Bad}_1 \wedge \neg\mathsf{Bad}_2 \wedge \mathsf{Forge}_2] \\ &= \Pr[\neg\mathsf{Bad}_3] \cdot \Pr[\neg\mathsf{Bad}_4|\neg\mathsf{Bad}_2] \\ &= \Pr[\neg\mathsf{Bad}_3] \cdot \frac{\Pr[\neg\mathsf{Bad}_4]}{\Pr[\neg\mathsf{Bad}_2]} \cdot \Pr[\neg\mathsf{Bad}_2|\neg\mathsf{Bad}_4] \\ &\geq \Pr[\neg\mathsf{Bad}_3] \cdot \Pr[\neg\mathsf{Bad}_4] \cdot \Pr[\neg\mathsf{Bad}_2|\neg\mathsf{Bad}_4].\end{aligned}$$

Since $P_E \subseteq P_H$, $|P_H| \leq q_k$, and $t-1 \leq q_k$ from the definition of unforgeability, we obtain the following probability bounds

$$\begin{aligned}
&\Pr[\neg\mathsf{Bad}_3] = (1-1/(q_k+1))^{|P_E \cap P_G|} \geq (1-1/(q_k+1))^{q_k} \geq 1/e,\\
&\Pr[\neg\mathsf{Bad}_4] = 1/q_h,\\
&\Pr[\neg\mathsf{Bad}_2|\neg\mathsf{Bad}_4] = (1-1/(q_k+1))^{t-1} \geq (1-1/(q_k+1))^{q_k} \geq 1/e.
\end{aligned}$$

This completes the claim. □

Game$_3$: In this game, we modify the simulator to extracts all secret keys of corrupted parties in the target group GD^* by using the knowledge extractor of the NIZKPoK system, and to extract all Lagrange coefficients by using the knowledge extractor of the n-AugKEA. To invoke this knowledge assumption, we set the group elements $h_1,\ldots,h_{n-t},\hat{h}_1^a,\ldots,\hat{h}_{n-t}^a$ to public components $g^{f(-1)},\ldots,g^{f(-(n-t))}$, $H_0(GID)^{f(-1)},\ldots,$ $H_0(GID)^{f(-(n-t))}$ by programming $H_0(GID)=\hat{g}^a$ where $GID \in GD^*$ and set auxiliary input z as co-CDH instance $(g^\alpha,\hat{g}^\alpha,\hat{g}^\beta)$. Next, we fix all randomness of the simulator and treat $\mathcal{A}$ as a circuit that plays the security game. The simulator can handle all queries of the adversary by using the auxiliary input which is the co-CDH instance and extract the all coefficients by using the knowledge extractor since the n-AugKEA holds for all $h_1,\ldots,h_n,\hat{h}_1,\ldots,\hat{h}_n$. Because of this modification, the simulator declares failure ($\mathsf{Bad}_5=1$) and aborts if the extraction of these secret keys fails, and declares failure ($\mathsf{Bad}_6=1$) and aborts if the extraction of these coefficients fails. Since the probability of the knowledge extractors fail is negligible, we obtain the following equation

$$\begin{aligned}
\Pr[\wedge_{i=1}^6\neg\mathsf{Bad}_i \wedge \mathsf{Forge}_3] &= \Pr[\wedge_{i=5}^6\neg\mathsf{Bad}_i]\cdot\Pr[\wedge_{i=1}^4\neg\mathsf{Bad}_i\wedge\mathsf{Forge}_3|\wedge_{i=5}^6\neg\mathsf{Bad}_i]\\
&= (1-\Pr[\vee_{i=5}^6\mathsf{Bad}_i])\cdot\Pr[\wedge_{i=1}^4\neg\mathsf{Bad}_i\wedge\mathsf{Forge}_2]\\
&\geq \Pr[\wedge_{i=1}^4\neg\mathsf{Bad}_i\wedge\mathsf{Forge}_2]-\Pr[\mathsf{Bad}_5\vee\mathsf{Bad}_6]\\
&\geq \Pr[\wedge_{i=1}^4\neg\mathsf{Bad}_i\wedge\mathsf{Forge}_2]-\mathrm{negl}(\lambda). \qquad (4.4)
\end{aligned}$$

Game$_4$: In this game, we modify the simulator to check that $v_0-v_1\neq 0$ after computing v_0 and v_1, and then to solve the solution of the co-CDH assumption. Because of this modification, the simulator declares failure ($\mathsf{Bad}_7=1$) and aborts if $v_0-v_1=0$. From the claims 4.5 and 4.6, we obtain the following equation

$$\begin{aligned}
\mathbf{Adv}^{co\text{-}CDH} &= \Pr[\wedge_{i=1}^7\neg\mathsf{Bad}_i\wedge\mathsf{Forge}_4]\\
&= \Pr[\neg\mathsf{Bad}_7]\cdot\Pr[\wedge_{i=1}^6\neg\mathsf{Bad}_i\wedge\mathsf{Forge}_4|\neg\mathsf{Bad}_7]\\
&\geq (1/(q_k+1)-1/p)\cdot\Pr[\wedge_{i=1}^6\neg\mathsf{Bad}_i\wedge\mathsf{Forge}_3]. \qquad (4.5)
\end{aligned}$$

Claim 4.5. $\Pr[\neg\mathsf{Bad}_7]\geq 1/(q_k+1)-1/p$.

Proof. Let $\vec{\delta}=(\delta_{-(n-t)},\ldots,\delta_{-1})$ be the vector of extracted coefficients from the adversary. To argue this claim, we have that

$$v_0-v_1=-(v_1-v_0)=-\langle(\vec{v},v_0),(\vec{\delta},-1)\rangle=-\vec{s}_a\cdot L\cdot(\vec{\delta},-1)^T.$$

Let $\vec{\delta}_T=(\vec{\delta},-1)\cdot L^T$. Since L^T is full row rank and $(\vec{\delta},-1)\neq 0^{n-t+1}$, $\vec{\delta}_T$ contains less than $n-t+1$ zero entries. Since the total weight of $\mathcal{A}$ is at most $t-1$, there must exist some index i such that 1) $\vec{\delta}_T[i]\neq 0$ by pigeonhole principle and 2) $\vec{s}_a[i]$ is randomly sampled with probability $1/(q_k+1)$. Let $\vec{s}_a^*$ and $\vec{\delta}_T^*$ be the

truncated vectors that only consists of these indexes. Then we have $\Pr[v_1 - v_0 = 0] = \Pr_{\vec{s}_a}[\langle \vec{s}_a^*, \vec{\delta}_T^* \rangle = 0] \leq 1/p$. Thus, we have

$$\Pr[v_1 - v_0 = 0] \leq 1 - 1/(q_k + 1) + 1/p.$$

This completes the claim. □

Claim 4.6. $\Pr[\wedge_{i=1}^{7} \neg \mathsf{Bad}_i \wedge \mathsf{Forge}_4] = \boldsymbol{Adv}^{co\text{-}CDH}$.

Proof. Since $V_0 = (g^{\alpha})^{v_0} g^{u_0}$ and $S_1 = (g^{\alpha})^{v_1} g^{u_1}$, we have that $S_0 = H_1(M^*)^{\alpha v_0 + u_0} \cdot H_1(M^*)^{-(\alpha v_1 + u_1)} = H_1(M^*)^{\alpha(v_0 - v_1)} H_1(M^*)^{(u_0 - u_1)}$. By the setting $H_1(M^*) = \hat{g}^{\beta} \hat{g}^{s'}$ and the condition $v_0 - v_1 \neq 0$, the final step of the simulator can correctly derive the co-CDH solution as

$$\left(S_0 \cdot H_1(M^*)^{-(u_0 - u_1)}\right)^{1/(v_0 - v_1)} \cdot (\hat{g}^{\alpha})^{-s'} = (\hat{g}^{\beta} \hat{g}^{s'})^{\alpha} \cdot (\hat{g}^{\alpha})^{-s'} = \hat{g}^{\alpha\beta}.$$

This completes the claim. □

Therefore, by combining the above equations (4.1), (4.2), (4.3), (4.4), and (4.5) of hybrid games, we obtains the following equation

$$\mathbf{Adv}^{co\text{-}CDH} \geq \left(\frac{1}{q_k + 1} - \frac{1}{p}\right)\left(\frac{1}{e^4 q_h} \cdot \varepsilon - \mathrm{negl}(\lambda)\right) \approx \frac{1}{e^4 q_h (q_k + 1)} \cdot \varepsilon \tag{4.6}$$

where ε is the advantage of the adversary. This completes our proof. □

5 Implementation and Performance Analysis

In this section, we implement our NIDTS scheme and compare the performance of our NIDTS scheme with the MTS scheme.

We implement our NIDTS scheme by using Rust and release it on GitHub[4]. To do this, we implement our NIDTS scheme by modifying the implementation of the MTS scheme. To measure the performance, we use a laptop with Intel i7-1185G7 3.0GHz CPU and 16GB RAM specifications, and implement a single-threaded version of the algorithm.

5.1 Implementation

Since our NIDTS scheme uses pairing groups, we choose the BLS12-381 curve as the base pairing-based curve. The BLS12-381 curve which belongs to the BLS family of Barreto, Lynn, and Scott [3], is a pairing-based elliptic curve with 128-bit security and was proposed by Sean Bowe [12]. In BLS12-381, although the embedding degree is 12, the size of the $\mathbb{G}$ group is 48 bytes and the size of the $\hat{\mathbb{G}}$ group is 96 bytes, and an efficient map-to-hash algorithm exists. Because of these reasons, BLS12-381 is widely used in recent blockchains. The benchmarks of basic operations in the BLS12-381 curve is given in Table 2.

In our NIDTS scheme, the GroupSetup algorithm is the most time consuming one. This algorithm consists of a part that verifies the public keys and register keys of parties, a part that calculates the Lagrange coefficients, and a part that derives the combining key and verification key by using the Lagrange interpolation method. If this algorithm is implemented in a simple way, the part of calculating the Lagrange

[4]Our NIDTS implementation is available at `https://github.com/guspinlee/nidts`

Table 2: Benchmarks of operations in bilinear groups

Curve	$\mathbb{G}$	$\hat{\mathbb{G}}$	$\mathsf{M}_{\mathbb{Z}_p}$	$\mathsf{H}_{\mathbb{G}}$	$\mathsf{H}_{\hat{\mathbb{G}}}$	$\mathsf{E}_{\mathbb{G}}$	$\mathsf{E}_{\hat{\mathbb{G}}}$	P
BLS12-381	381 bits	762 bits	0.0002	0.130	0.729	0.452	1.493	1.801

All benchmarks are measured in milliseconds. We use symbols $\mathsf{M}_{\mathbb{Z}_p}$ for multiplication in $\mathbb{Z}_p$, H_G for map-to-hash in G, E_G for exponentiation in G, and P for pairing.

Table 3: Key and signature size analysis of MTS and NIDTS schemes

Scheme	PK	OM or RK	CK	VK	PS	TS
MTS [2]	$w\mathbb{G}$	$(n-t+w)\mathbb{G}+w\mathbb{Z}_p$	$2(n-t)\mathbb{G}$	$\mathbb{G}+\hat{\mathbb{G}}$	$w\hat{\mathbb{G}}$	$2\mathbb{G}+\hat{\mathbb{G}}$
NIDTS	$w\mathbb{G}$	$w\mathbb{G}+w\hat{\mathbb{G}}+w\mathbb{Z}_p$	$(n-t)\mathbb{G}+(n-t)\hat{\mathbb{G}}$	$\mathbb{G}+\hat{\mathbb{G}}$	$w\hat{\mathbb{G}}$	$\mathbb{G}+2\hat{\mathbb{G}}$

Let n be the total number of weights, t be a threshold, and w be a weight bound. We use PK for public key, OM for online message in MTS, RK for register key in NIDTS, CK for combine key, VK for verification key, PS for partial signature, and TS for threshold signature.

coefficients takes up most of the algorithm computation as the number of parties increases. The reason is that this algorithm requires $(n-t)n$ number of Lagrange coefficients, and one Lagrange coefficient requires n scalar multiplications, so it takes approximately $O(n^3)$ scalar multiplications to calculate all Lagrange coefficients. This problem of computing Lagrange coefficients is the same problem that occurs not only in our NIDTS scheme but also in the MTS scheme.

To overcome this problem, we show that it is possible to compute all Lagrange coefficients with $O(n^2)$ scalar multiplications instead of $O(n^3)$ scalar multiplications by using the barycentric form of the Lagrange basis polynomial [7]. We can derive the barycentric form of the Lagrange basis polynomial as follows:

$$L_i(x) = \prod_{k\in[n]\setminus\{i\}} \frac{(x-k)}{(i-k)} = \prod_{k\in[n]} (x-k) \cdot \prod_{k\in[n]\setminus\{i\}} (i-k)^{-1} \cdot \frac{1}{(x-i)} = \mathrm{comp}(x) \cdot \mathrm{baryw}_i \cdot \frac{1}{(x-i)}.$$

In the above formula, the first part $\mathrm{comp}(x)$ is a common polynomial to every basis polynomial since it depends on a variable x but is independent of an index i, the second part baryw_i is a constant that is independent of x and only depends on i, and the last part $1/(x-i)$ is a value dependent on both x and i. Thus, if an algorithm pre-computes all values $\{\mathrm{comp}(x)\}_{x\in[-(n-t):-1]}$ and $\{\mathrm{baryw}_i\}_{i\in[n]}$, then it can compute all Lagrange coefficients with approximately $(n-t)*n+n*n+n*n = O(n^2)$ scalar multiplications. We applied this efficient Lagrange coefficient computation to our NIDTS scheme as well as the MTS scheme for fair comparison. For instance, when we set $m = 500, w = 1$, and $t = 250$, the $\mathrm{UnivGen}_{off}$ algorithm of the MTS scheme takes 983 seconds in the simple method, but it only takes 226 seconds in the modified method, which is 4.3 times faster. In the same setting, the GroupSetup algorithm of our NIDTS scheme takes 765 seconds in the simple method, but it only takes 42 seconds in the modified method, which is approximately 18.2 times faster.

5.2 Size and Algorithm Analysis

We compare the size of the public key, combine key, verification key, partial signature, and threshold signature of the NIDTS and MTS schemes. The detailed comparison for these keys and signatures are shown in

Table 4: Algorithm analysis of MTS and NIDTS schemes

Scheme	GenKey	Key Setup	Sign	Combine	Verify
MTS [2]	$w\mathsf{E}_{\mathbb{G}}$	$n^2\mathsf{P} + n^2\mathsf{E}_{\mathbb{G}} + 2n^2\mathsf{M}_{\mathbb{Z}_p}$	$w\mathsf{E}_{\hat{\mathbb{G}}} + \mathsf{H}_{\hat{\mathbb{G}}}$	$n\mathsf{P} + \frac{1}{2}n(w+2)\mathsf{E}_{\mathbb{G}} + \frac{1}{2}n(w+1)\mathsf{E}_{\hat{\mathbb{G}}}$	$4\mathsf{P} + \mathsf{H}_{\hat{\mathbb{G}}}$
NIDTS	$w\mathsf{E}_{\mathbb{G}}$	$2n\mathsf{P} + \frac{1}{2}n^2\mathsf{E}_{\mathbb{G}} + \frac{1}{2}n^2\mathsf{E}_{\hat{\mathbb{G}}} + 2n^2\mathsf{M}_{\mathbb{Z}_p}$	$w\mathsf{E}_{\hat{\mathbb{G}}} + \mathsf{H}_{\hat{\mathbb{G}}}$	$n\mathsf{P} + \frac{1}{2}n(w+1)\mathsf{E}_{\mathbb{G}} + \frac{1}{2}n(w+2)\mathsf{E}_{\hat{\mathbb{G}}}$	$4\mathsf{P} + \mathsf{H}_{\hat{\mathbb{G}}}$

Let m be the number of parties, w be a weight bound, and n be the total weights such that $n = mw$. We set a threshold $t = n/2$. We use symbols H_G for map-to-hash in G, E_G for exponentiation in G, $\mathsf{M}_{\mathbb{Z}_p}$ for multiplication in $\mathbb{Z}_p$, and P for pairing.

Table 3. In both the MTS and NIDTS schemes, the public key, the verification key, and the partial signature are consist of $w\mathbb{G}, \mathbb{G} + \hat{\mathbb{G}}$, and $w\hat{\mathbb{G}}$, respectively, and are identical to both schemes. That is, in the case of $w = 1$, the size of the public key is 48 bytes, the size of the verification key is 143 bytes, and the size of the partial signature is 96 bytes. The combine key of the MTS scheme consists of $2(n-t)\mathbb{G}$ elements, whereas the combine key of the NIDTS scheme consists of $(n-t)\mathbb{G}$ elements and $(n-t)\hat{\mathbb{G}}$ elements. For instance, in the case of setting $m = 1000, w = 1$, and $t = 500$, the combine key of the MTS scheme is 47 kilobytes and the combine key of the NIDTS scheme is 71 kilobytes. The threshold signature has a fixed length size for both schemes, but the MTS scheme consists of $2\mathbb{G} + \hat{\mathbb{G}}$ which is 191 bytes, and the NIDTS scheme consists of $\mathbb{G} + 2\hat{\mathbb{G}}$ groups which is 239 bytes. Thus, the threshold signature size of our NIDTS scheme is 1.25 times longer than that of the MTS scheme.

We analyze the inbound bandwidth of transmitted messages from the point of view of a combiner. In the key setup process of the MTS scheme, at least t individual parties should generates and delivers online messages to the combiner. Thus, it requires that approximately $t(n-t+w)\mathbb{G} + tw\mathbb{Z}_p$ elements must be delivered. In the key setup process of the NIDTS scheme, m parties generates and delivers register keys to the combiner. Thus, it requires that approximately $mw\mathbb{G} + mw\hat{\mathbb{G}} + mw\mathbb{Z}_p$ elements should be delivered. For instance, in the case of setting $m = 1000, w = 1$, and $t = 500$, the bandwidth of the MTS scheme is approximately 11.946 MB and the bandwidth of the NIDTS scheme is approximately 0.174 MB. Thus, the bandwidth of the NIDTS scheme is about 68 times more efficient than that of the MTS scheme.

Next, we analyze the key generation, key setup, signing, combining, and verification algorithms of the MTS and NIDTS schemes. Most of the algorithms of two schemes are almost similar in performance except the key setup algorithm that generates the combine key and verification key. The asymptotic analysis of these algorithms is given in Table 4. In two schemes, the key generation, signing, and verification algorithms have a fixed amount of computation if the weight bound is fixed. In the combining algorithms of both schemes, the most of the time is spent for verifying partial signatures generated by external parties, and there is a slight difference in the amount of computation due to the different group elements are used in the combine key.

We analyze the performance of the key setup algorithms of both schemes. In the case of the MTS scheme, the key setup process consists of the UnivSetup_{on} algorithm performed by the parties and the UnivSetup_{off} algorithm performed by the combiner. Since each party can perform this algorithm in parallel, we consider the sum of the execution of one UnivSetup_{on} algorithm and the execution of the UnivSetup_{off} algorithm as the amount of computation of the key setup process. Similarly, since the NIDTS scheme consists of the GenRegKey algorithm performed by the parties and the GroupSetup algorithm performed

Table 5: Performance analysis of our NIDTS scheme

(m,w,n)	GenKey	GenRegKey	GroupSetup	Sign	Combine	Verify
$(100,1,100)$	0.0005	0.002	2.279	0.002	0.289	0.007
$(200,1,200)$	0.0005	0.002	8.834	0.002	0.548	0.007
$(300,1,300)$	0.0005	0.002	17.867	0.002	0.773	0.007
$(100,5,500)$	0.002	0.010	50.657	0.008	0.905	0.008
$(200,5,1000)$	0.002	0.011	164.704	0.007	1.645	0.007
$(300,5,1500)$	0.002	0.010	352.857	0.007	2.503	0.007
$(100,10,1000)$	0.005	0.019	161.081	0.015	1.106	0.010
$(200,10,2000)$	0.005	0.019	619.387	0.015	2.354	0.007
$(300,10,3000)$	0.005	0.019	1399.526	0.014	3.744	0.007

All benchmarks are measured in seconds. Let m be the number of parties, w be a weight bound, and n be the total weights. We set a threshold $t = n/2$.

the combiner, we consider the sum of the GenRegKey execution and the GroupSetup execution as the key setup process. As pointed out above, we can reduce the computation of Lagrange coefficients from $O(n^3)$ multiplication to $O(n^2)$ multiplication by using the barycentric form. In this case, the MTS scheme needs to perform approximately $O(n^2)$ pairings, $O(n^2)$ exponentiations, and $O(n^2)$ scalar multiplications. In the NIDTS scheme, the key setup requires $O(n)$ pairings, $O(n^2)$ exponentiations, and $O(n^2)$ scalar multiplications.

5.3 Performance Analysis

The detailed performance of our NIDTS algorithms according to the number of parties and the weight bound is shown in Table 5. As analyzed above, the key generation, register key generation, and signing algorithms increase execution time linearly as the weight bound increases, but it can be seen that they are very efficient in an environment where the weight bound is small. The verification algorithm is very efficient regardless of the number of parties and the weight bound. The group setup algorithm requires $O(n^2)$ exponentiations where $n = mw$, so the execution time increases quickly as the number of parties increases and the weight bound increases. However, the group setup algorithm is performed by a strong external server (or combiner), and it is sufficient for the server to run this algorithm only once during the initial group setup. In addition, parallelization can be used to further improve performance. The combining algorithm is also performed by an external combiner, not by individual parties, and it can be seen that it is quite efficient even when n increases, and additional improvement in performance is possible by using parallelization.

The performance comparison of our NIDTS algorithms and MTS algorithms is shown in Table 6. For a fair comparison, we modify the implementation of the MTS scheme to verify all online messages and externally given partial signatures, and to use fast Lagrange coefficients computation using the barycentric form. For comparison, we set different values for the number of parties, m, and set the weight bound and the threshold to $w = 1$ and $t = n/2$, respectively. As revealed in the algorithm analysis, the performance of the key generation, signing, combining, and verification algorithms have little difference between the two schemes, and it can be confirmed that they operate very efficiently.

The biggest difference between the two schemes is the key setup process. In the MTS scheme, the key

Table 6: Performance comparison of MTS and NIDTS schemes

Scheme	Parties m	100	200	500	1000	2000
MTS [2]	GenKey	0.0005	0.0005	0.0005	0.0005	0.0005
	UnivGen_{on}	0.618	2.484	15.532	66.139	255.704
	UnivGen_{off}	9.178	36.170	233.242	902.874	3887.207
	Sign	0.002	0.002	0.002	0.002	0.002
	Combine	0.259	0.502	1.248	2.512	5.186
	Verify	0.008	0.008	0.009	0.008	0.008
NIDTS	GenKey	0.0005	0.0005	0.0005	0.0005	0.0005
	GenRegKey	0.002	0.002	0.002	0.002	0.002
	GroupSetup	2.279	8.834	42.783	164.804	623.025
	Sign	0.002	0.002	0.002	0.002	0.002
	Combine	0.289	0.548	1.292	2.589	5.321
	Verify	0.007	0.007	0.007	0.007	0.007

All benchmarks are measured in seconds. Let m be the number of parties. We set a weight bound $w = 1$ and a threshold $t = n/2$.

setup process consists of the UnivGen_{on} algorithm that is performed by an individual party to generate an online message and the UnivGen_{off} algorithm that is performed by the combiner to generate a combine key and verification key for the universe. At this time, to ensure the security of the MTS scheme, at least t parties must perform the UnivGen_{on} algorithm and these online messages must be given to the UnivGen_{off} algorithm. However, if the number of parties is $m = 1000$, it is not easy for a single party to run this algorithm because the execution time of the UnivGen_{on} algorithm exceeds 66 seconds. To complete key setup, individual parties must generate online messages and the combiner must run UnivGen_{off} algorithm. For instance, it takes at least 968 seconds for $m = 1000$ and it takes at least 4142 seconds for $m = 2000$. Unlike this, the key setup process of the NIDTS scheme consists of the GenRegKey algorithm and the GroupSetup algorithm. For instance, it takes about 164 seconds for $m = 1000$ and about 623 seconds for $m = 2000$. Thus, comparing the two schemes, the key setup process of the NIDTS scheme is 5.9 times faster for $m = 1000$ and 6.6 times faster for $m = 2000$ compared to the MTS scheme. In addition, the key setup process of the NIDTS scheme supports the non-interactive setup since individual parties can directly generate register keys before all parties who will participate in the group are known since the group identifier string *GID* is enough to run the register key generation algorithm.

6 Conclusion

We proposed an NIDTS scheme that supports the non-interactive and transparent key setup. The key setup process of our NIDTS scheme does not require message exchange between parties, only requires for each party to transfer a register key without waiting for a combiner to fix group parties, and is transparent since the generation of a verification key is deterministic. In addition, individual algorithms such as key generation, register key generation, and signing performed by individual parties are all very efficient because the computation and communication of these algorithm is regardless of the total number of parties. The evalua-

tion results of our implementation show that the key setup of our NIDTS scheme is very efficient compared to the MTS scheme.

One drawback of our NIDTS scheme supporting weights is that the size of public and register keys increases according to the weights of a party. The cause of this drawback is that a virtualization method that increases the secret shares corresponding to the weights is used. Thus, it is an interesting problem to modify our NIDTS scheme to support weights more efficiently without using the virtualization method. There are some interesting approaches to efficiently support weights [20,21], but it seems difficult to remove interactions in the key setup process.

Acknowledgements

This work was supported by Institute of Information & communications Technology Planning & evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00518, Blockchain privacy preserving techniques based on data encryption).

References

[1] Renas Bacho and Julian Loss. On the adaptive security of the threshold BLS signature scheme. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM SIGSAC Conference on Computer and Communications Security, CCS 2022*, pages 193–207. ACM, 2022.

[2] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Threshold signatures in the multiverse. In *IEEE Symposium on Security and Privacy, SP 2023*, pages 1454–1470. IEEE Computer Society, 2023.

[3] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks - SCN 2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2002.

[4] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022*, volume 13510 of *Lecture Notes in Computer Science*, pages 517–550. Springer, 2022.

[5] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 1998.

[6] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2004.

[7] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange interpolation. *SIAM Rev.*, 46(3):501–517, 2004.

[8] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017.

[9] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *Public-Key Cryptography - PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[10] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464. Springer, 2018.

[11] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

[12] Sean Bowe. BLS12-381: New zk-SNARK elliptic curve construction. `https://electriccoin.co/blog/new-snark-curve/`, 2017. Accessed: 2023-07-11.

[13] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. *J. Cryptology*, 25(4):723–747, 2012.

[14] Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Paper 2021/1375, 2021. `https://eprint.iacr.org/2021/1375`.

[15] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 1991.

[16] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127. Springer, 1987.

[17] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1989.

[18] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[19] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2005.

[20] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Cryptography with weights: MPC, encryption and signatures. Cryptology ePrint Archive, Paper 2022/1632, 2022. `https://eprint.iacr.org/2022/1632`.

[21] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. Cryptology ePrint Archive, Paper 2023/567, 2023. `http://eprint.iacr.org/2023/567`.

[22] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in*

Cryptology - EUROCRYPT 2013, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[23] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve A. Schneider, editors, *Applied Cryptography and Network Security - ACNS 2016*, volume 9696 of *Lecture Notes in Computer Science*, pages 156–174. Springer, 2016.

[24] Rosario Gennaro, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Threshold RSA for dynamic and ad-hoc groups. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 88–107. Springer, 2008.

[25] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer, 1996.

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

[27] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[28] Jens Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Paper 2021/339, 2021. `https://eprint.iacr.org/2021/339`.

[29] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021*, volume 12696 of *Lecture Notes in Computer Science*, pages 147–176. Springer, 2021.

[30] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM SIGSAC Conference on Computer and Communications Security, CCS 2020*, pages 1751–1767. ACM, 2020.

[31] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65. Springer, 2020.

[32] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM Conference on Computer and Communications Security, CCS 2001*, pages 245–254. ACM, 2001.

[33] NIST. Multi-party threshold cryptography. `https://csrc.nist.gov/Projects/threshold-cryptography`, 2023. Accessed: 2023-07-11.

[34] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.

[35] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: robust asynchronous Schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM SIGSAC Conference on Computer and Communications Security, CCS 2022*, pages 2551–2564. ACM, 2022.

[36] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, 1991.

[37] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy, SP 2017*, pages 444–460. IEEE Computer Society, 2017.

[38] The DFINITY Team. The internet computer for geeks. Cryptology ePrint Archive, Paper 2022/087, 2022. `https://eprint.iacr.org/2022/087`.

[39] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *IEEE Symposium on Security and Privacy, SP 2020*, pages 877–893. IEEE, 2020.