FlexHi: A Flexible Hierarchical Threshold Signature Scheme

Muhammed Ali Bingol[1], Sermin Kocaman[2], Ali Doğan[3,4], and Sibel Kurt Toplu[4]

[1] De Montfort University, Cyber Technology Institute, Leicester, United Kingdom
muhammed.bingol@dmu.ac.uk
[2] Department of Cryptology, Institute of Applied Mathematics, METU, Turkey
sermin.cakin@metu.edu.tr
[3] Istanbul Technical University, Informatics Institute, Istanbul, Turkey
[4] Blockchain Technologies Unit, TUBITAK BILGEM, Kocaeli, Turkey
{doganali, sibel.toplu}@tubitak.gov.tr

Abstract. Threshold signature schemes have gained prominence in enhancing the security and flexibility of digital signatures, allowing a group of participants to collaboratively create signatures while maintaining a predefined threshold of participants for validity. However, conventional threshold signatures treat all participants equally, lacking the capability to accommodate hierarchical structures often seen in real-world applications. Hierarchical Threshold Signature Schemes (HTSS) naturally extend the concept of simple threshold signatures, offering a solution that aligns with hierarchical organizational structures. Our paper introduces a novel, efficient, and flexible HTSS that employs independent polynomials at each hierarchical level, removing limitations on threshold values. This adaptability enables us to tailor the scheme to diverse requirements, whether signing requires only top-level nodes or lower-level participants' involvement. Based on our analysis, our FlexHi integrated into the FROST scheme outperforms Tassa's hierarchical scheme on FROST and operates approximately 30% to 40% faster, depending on the number of participants and the chosen threshold values. This demonstrates that, in addition to flexibility, our scheme has practical benefits through improved performance.

Keywords: Hierarchical Threshold Signature, Secret-Sharing, Security, Privacy

1 Introduction

In the rapidly evolving landscape of secure digital communication and decentralized systems, cryptographic primitives play a pivotal role in ensuring data integrity, authenticity, and confidentiality. Digital signatures, a fundamental cryptographic tool, enable the verification of the origin and integrity of digital messages, thereby establishing trust in electronic transactions and communications. Threshold signature schemes have emerged as a prominent solution to enhance the security and flexibility of digital signatures, allowing a group of participants to collaboratively generate signatures while maintaining a predefined threshold of participants necessary to create a valid signature.

Threshold signatures have a wide array of applications, yet a notable limitation is that they traditionally assign equal weight to all participants. This equal weighting implies uniform rights among participants, which, while fair in some contexts, does not align with the hierarchical structures often required in real-world applications. An example of such a scenario is within financial systems, where the authority to approve high-value transactions may vary among roles, such as allowing a transaction to be approved by two directors, or a director and a president, but not by two presidents alone. This discrepancy highlights the inadequacy of traditional threshold signature schemes in accommodating hierarchical decision-making processes. Since the conventional threshold signature scheme falls short in addressing such scenarios, the Hierarchical Threshold Signature Scheme (HTSS) emerged as a natural extension of simple threshold signatures. Starting with the concept of hierarchical secret-sharing scheme, several constructions of these schemes for different authorized subsets of participants (access structure) have been proposed in the literature [22,23,11,25,26,3,13,14,27,17,10,31].

HTSS represents a noteworthy advancement in this domain, offering a structured approach to manage signatures within hierarchical organizational structures. These schemes facilitate the delegation of signing authority across different hierarchical levels, thereby providing a versatile framework for scenarios involving complex access control, distributed management, and efficient signature aggregation. While the concept of HTSS presents a promising avenue for various applications, their practical adoption and robustness demand rigorous exploration.

1.1 Related Work

In 1979, Shamir [22] pointed out a weighted threshold system as a version of a hierarchical threshold scheme in which participants take the number of shares, which are points on a polynomial, proportional to their level. In this scheme, the ratio of the size of the participant's share to the size of the secret equals the participant's assigned weight, which may be exponential in number of participants [2]. Thus, it can be seen that this construction is not ideal [27].

In 1988, Simmons [23] introduced the notion of a disjunctive multilevel threshold scheme and compartmented threshold scheme based on a geometric construction presented by Blakley [7]. As in the Blakley threshold scheme, the concept of intersecting hyperplanes is used. However, the proposed scheme is not efficient in the secret reconstruction since it requires the dealer to check the nonsingularity of the exponentially many matrices to prevent the unqualified set from finding the secret [25]. Further, the scheme is not ideal [14].

In 1989, Brickell [9] introduced an ideal multi-threshold secret-sharing scheme. Nonetheless, this scheme is inefficient, as it necessitates the dealer to compute an exponential number of matrices to guarantee the non-singularity of matrices. Afterward, in 1998, Ghodosi [11] presented a scheme designed for compartmented access structures by applying the Shamir secret-sharing scheme. Each level in this scheme has its own polynomial; however, the degrees of these polynomials are recursively defined. Since new participants cannot be added to any level except the last without resharing the secret, the scheme is not dynamic [30]. Also, the proposed scheme only works for a small number of shareholders [3].

In 2007, Tassa [25] proposed a conjunctive hierarchical scheme based on Birkhoff interpolation. In Tassa's scheme, polynomial derivatives are used to generate shares for participants of lower levels in the hierarchy. To generate the additional shares, the scheme uses the vector of coefficients of the polynomial, which is denoted by $a = (a_0, a_1, \dots, a_{t-1})$. The dealer then takes derivate of it to generate a new vector of coefficients for the distribution according to participant indexes, which is denoted by $a' = (a_1, 2.a_2, \dots, (t-1).a_{t-2})$. The derivative operator can be applied $t-1$ times to generate different hierarchy levels. The distributed key generation, which is based on Tassa's protocol and does not require a dealer, is also available [18]. Later, in 2009, Tassa and Dyn [26] proposed an ideal hierarchical secret-sharing scheme based on a bivariate interpolation technique. However, the proposed schemes ([25,26]) necessitate a large finite field with some limitation in assigned identities of the users [3]. In the rest of the paper, Tassa's scheme will refer to the first construction [25]. It is important to note that Tassa's technique, whose matrix needs to satisfy necessary Polya's condition, does not necessarily exist in all scenarios and requires exponential complexity due to the matrices' nonsingularity check [31]. After selecting the polynomial at the first level, it is not possible to increase the number of levels in sub-hierarchies, thus this technique might not be appropriate for all kinds of secrets or access hierarchies.

In 2008, Belenkiy [3] proposed disjunctive multi-level secret-sharing in which the users learn a point on a polynomial or in its derivative as in Tassa. But instead of classically choosing a constant number as the secret, the secret is chosen as a_{t-1} when the polynomial coefficients are $a = (a_0, a_1, \dots, a_{t-1})$. It is shown that this technique can be used directly to construct disjunctive secret-sharing without the need for an intermediary conjunctive scheme. However, the scheme is not suitable for all scenarios and is not very efficient.

In 2009, Käsper et al. [14] proposed strongly multiplicative hierarchical threshold secret-sharing in which players who have a share of the secret take the multiplication of these secret shares without knowing the original secrets, even if the active adversary is present. They gave an efficient linear secret-sharing construction for a hierarchical scheme contrary to the exponential construction in the general linear secret-sharing scheme. However, their scheme requires stronger conditions on the access structure [28].

In 2013, Tentu et al. [27] introduced a computationally perfect, conjunctive hierarchical scheme based on the maximum distance separable (MDS) codes. In the scheme, the dealer selects MDS codes, its codewords, and a distinct one-way function, then each participant takes exactly one share. The codewords are chosen in a way that they contain shared secret in its first component, next contain images of shares of relative level participants under distinct one-way functions, and the rest are chosen arbitrarily. This scheme does not require the ground field to be extremely large or any restrictions on the users' assigned identities. While they pointed out that the idea is also applicable to the disjunctive scheme, they just present a conjunctive scheme in their work.

In 2015, Nojoumian and Stinson [17] introduced sequential secret-sharing in which a group of players with varying levels share different yet related secrets with increasing thresholds. Multiple secrets are derived in this protocol by a linear combination of previous secrets. During the reconstruction phase, each subgroup of players can only recover secrets at their designated level. Consequently, the master secret can only be revealed if all the secrets in the higher levels are sequentially recovered. Considering FROST, since the secret needs to be disclosed for signature verification, it cannot be used for threshold signatures.

In 2016, Ersoy et al.[10] presented a new disjunctive and conjunctive multilevel secret-sharing scheme built around the anchor sequence, a unique primary sequence. The literature's CRT-based multilevel threshold secret-sharing scheme of Harn-Fuyou [12] is not applicable to all threshold configurations. Ersoy et al. presented their work and revealed that these new schemes can be integrated into function-sharing schemes. However, the scheme employs sequences of primes that could be widely spaced. Producing sequences of prime numbers that satisfy these conditions is resource-intensive. The scheme relies on hash functions that must be treated as random oracles for security purposes, and the information rate is notably high.

In 2022, Yuan et al. [31] presented a new hierarchical scheme based on Linear Homogeneous Recurrence (LHR) relations. In this scheme, linearly independent homogeneous recurrence relations are chosen by the distributor, and then pseudo shares of the participants from related levels are used to construct the related LHR relation. In the reconstruction, a qualified subset of participants solve relations to get the required values, and then obtain shared secrets. The complexity of hierarchical schemes is reduced from exponential to polynomial in this scheme. However, the participants are assumed to be semi-honest, and there is no verification check in their scheme.

Our Contribution. In this paper, we propose a flexible hierarchical threshold signature scheme, what we call *FlexHi*. Our scheme provides a simple and efficient hierarchical threshold system based on Shamir's secret-sharing [22]. As mentioned above, existing hierarchical threshold schemes often impose ordering and constraints on each level, limiting their flexibility. The core objective of this paper is to introduce an architecture that breaks free from these constraints and offers unparalleled flexibility. In our scheme, independent polynomials are utilized at each level, enabling us to accommodate any number of participants and define threshold values without limitation. This approach diverges from schemes like Tassa [25], which rely on Polya's condition that imposes restrictions. Our approach eliminates restrictions on the threshold values for independent polynomials at each level, making our scheme exceptionally adaptable. This flexibility allows the scheme to meet a wide range of application requirements, dependent on the chosen threshold values at each level. For instance, when the top-level hierarchical nodes suffice for signing, the lower-level threshold number must be less than or equal to the number of top-level hierarchical nodes. Conversely, if the involvement of individuals from lower-level sets is necessary for signing, the lower-level threshold number must surpass the number of top-level hierarchical nodes. Through our analysis, we demonstrate that our FlexHi integrated into the FROST scheme outperforms Tassa's hierarchical scheme on FROST, operating approximately 30% to 40% faster, depending on the number of participants and the chosen threshold values. This not only highlights the scheme's flexibility but also underscores its practical benefits through improved performance.

Paper Organization. The rest of the paper is organized as follows: Section 2 provides definitions of access structures as preliminaries. Section 3 describes the architecture of our proposed *FlexHi* scheme and its generalization, followed by its FROST application in Section 4. Section 5 provides a security and cost analysis of *FlexHi* scheme. Section 6 concludes the paper.

2 Preliminaries on Access Structure

The access structure is the family of authorized subsets. Hierarchical access structure consists of two classes: disjunctive and conjunctive access structures. In a hierarchical structure, participants are divided into separate levels based on their significance. These levels maintain a strict hierarchy, with parties in higher levels holding greater importance compared to those in lower levels. In a typical scenario involving a bank's workforce, the higher level might be composed of the board of directors. Simmons [23] introduced the initial hierarchical secret-sharing scheme known as disjunctive multilevel secret-sharing. Later, Tassa [25] modified this scheme into the conjunctive multilevel secret-sharing approach. In both of these schemes, a secret is distributed among participants occupying different authority tiers. The terms "disjunctive" and "conjunctive" were jointly introduced in [3]. To understand the distinction between these two types of access structures, we provide these definitions.

Definition 1 (Access structure [3]). *Let U be a set of users. An access structure $\Gamma \subseteq P(U)$ must satisfy these two conditions:*

- *monotonicity: if $A \in \Gamma$ and $A \subseteq B$ then $B \in \Gamma$*
- *non-triviality: if $A \in \Gamma$ then $|A| > 0$.*

If every set A is in Γ, A is authorized, and if every set B is not in Γ, B is unauthorized.

Definition 2 (Threshold access structure [3]). *We say that Γ is a threshold access structure corresponding to threshold t if $\Gamma = \{A \subseteq U : |A| \geq t\}$.*

Each level L has a threshold t_L such that $t_0 < t_1 < \ldots < t_n$. For the conjunctive multi-level access structure, if there exists a minimum of t_L users at levels $0, 1, \ldots, L$ *for every level* L, the secret can only be recovered. For a disjunctive multi-level access structure, if *for some level* L, there are at least t_L users at level $0, \ldots, L$, the secret can be recovered.

In a hierarchical secret-sharing scheme, a secret α is shared among the players with monotonically increasing thresholds $t_1 < t_2 < \ldots < t_n$. Let P be a set of n players and assume P is composed of l disjoint levels:

$$P = \bigcup_{i=1}^{l} P_i \ \text{ where } P_i \cap P_j = \emptyset \text{ for all } 1 \leq i < j \leq l \text{ and } |P_i| \geq t_i \text{ for all } i.$$

Definition 3 (Disjunctive hierarchical access structure [17]). *secret α can be recovered by a set of players A, i.e., an authorized subset, only if*

$$|A \cap (\bigcup_{i=1}^{j} P_i)| \geq t_j \quad \textit{for } \textbf{\textit{at least one}} \ j \ \textit{where } 1 \leq j \leq l,$$

i.e., at least one threshold must be satisfied at level 1 to j.

Definition 4 (Conjunctive hierarchical access structure [17]). *Secret α can be recovered by a set of players only if*

$$|A \cap (\bigcup_{i=1}^{j} P_i)| \geq t_j \quad \textit{for } \textbf{\textit{all}} \ j \ \textit{where } 1 \leq j \leq l,$$

3 A Flexible Hierarchical Threshold Signature Scheme

In this section, we present a novel hierarchical threshold signature scheme called a flexible hierarchical threshold signature (*FlexHi*) scheme. Our scheme utilizes independent polynomials at each level, offering the flexibility to allow any number of participants and set threshold values without limitations as opposed to the monotone threshold requirement in conjunctive or disjunctive hierarchical-based schemes. This flexibility ensures that the scheme can adapt to various hierarchical structures and security requirements, making it a versatile and robust solution for a hierarchical threshold signature. Now we define our flexible hierarchical secret-sharing as follows.

Definition 5 (Flexible hierarchical secret-sharing). *Let $\mathcal{L}$ denote a hierarchical structure comprising multiple levels such that*

$$\mathcal{L} = \bigcup_{j=1}^{n} \mathcal{L}_j \quad \textit{where } \mathcal{L}_j \cap \mathcal{L}_k = \mathcal{L}_j \ \textit{for all } 1 \leq j < k \leq n.$$

These levels are listed from the highest ($\mathcal{L}_1$) to the lowest ($\mathcal{L}_n$) hierarchy level, where each lower level includes all the nodes from the upper levels.Each $\mathcal{L}_j(t_j, m_j)$ level consists of $\bigcup_{s=1}^{j} \mathcal{N}_{s,i}$ nodes where $1 \leq i \leq m_j$ and t_j is the threshold value.

In *FlexHi* scheme, a private key share sk_j can be constructed when at least t_j nodes $\mathcal{N}_{j,i} \in \mathcal{L}_j(t_j, m_j)$ are engaged in a key construction algorithm (see Figure 2). Then, public key shares are generated from the corresponding private key shares depending on the underlying signature scheme. These independent the main public key. The hierarchical node has the advantage of extra secrets over lower-level nodes, and the scheme requires the use of this secret. To clarify the scheme, we begin with an example and then provide its definition and its generalized form.

Example 1. Suppose the aim is to create a three-level flex hierarchical threshold signature scheme with thresholds[5] $t_1 = 2$, $t_2 = 2$, and $t_3 = 6$ where $\mathcal{L}_1(2, 3)$, $\mathcal{L}_2(2, 6)$, and $\mathcal{L}_3(6, 9)$. Let the set of nodes be:

$$\begin{aligned}
\mathcal{L}_1 &= \{\mathcal{N}_{1,1}, \mathcal{N}_{1,2}, \mathcal{N}_{1,3}\} \\
\mathcal{L}_2 &= \{\mathcal{N}_{1,1}, \mathcal{N}_{1,2}, \mathcal{N}_{1,3}, \mathcal{N}_{2,1}, \mathcal{N}_{2,2}, \mathcal{N}_{2,3}\} \\
\mathcal{L}_3 &= \{\mathcal{N}_{1,1}, \mathcal{N}_{1,2}, \mathcal{N}_{1,3}, \mathcal{N}_{2,1}, \mathcal{N}_{2,2}, \mathcal{N}_{2,3}, \mathcal{N}_{3,1}, \mathcal{N}_{3,2}, \mathcal{N}_{3,3}\}
\end{aligned}$$

`KeyGen:`

1. **($\mathcal{L}_1(2, 3)$).** Each node $\mathcal{N}_{1,i} \in L_1$, where $1 \leq i \leq 3$:
   (a) Uses secret sampling in Figure 2 to create his secret $s_{11,i}$ for $\mathcal{L}_1$.
   (b) Adds the related subshares obtained in $\mathcal{L}_1$ to construct his private key share $sk_{11,i}$ using key construction in Figure 2. In this construction, public key share $pk_{11,i}$ is also created from the underlying signature scheme, and it is made public.
   (c) Generates the first level public key pk_1 from all published public key shares.
2. **($\mathcal{L}_2(2, 6)$).** Each node $\mathcal{N}_{1,i}, \mathcal{N}_{2,i} \in \mathcal{L}_2$, where $1 \leq i \leq 3$:
   (a) Uses secret sampling in Figure 2 to create his secret $s_{21,i}$,$s_{22,i}$, respectively, for $\mathcal{L}_2$.
   (b) Adds the related subshares received in $\mathcal{L}_2$ to construct his private key share $sk_{21,i}, sk_{22,i}$, respectively, using key construction in Figure 2. In this construction, public key share $pk_{21,i}, pk_{22,i}$, respectively, is also created from the underlying signature scheme, and they are made public.
   (c) Generates the second level public key pk_2 from all published public signing key shares
3. **($\mathcal{L}_3(6, 9)$).** Each participant $\mathcal{N}_{1,i}, \mathcal{N}_{2,i}, \mathcal{N}_{3,i} \in \mathcal{L}_3$, where $1 \leq i \leq 3$:
   (a) Uses secret sampling in Figure 2 to create his secret $s_{31,i}, s_{32,i}, s_{33,i}$, respectively, for $\mathcal{L}_3$.
   (b) Adds the related subshares received in $\mathcal{L}_3$ to construct his private key share $sk_{31,i}$, $sk_{32,i}$, $sk_{33,i}$, respectively, using key construction in Figure 2. In this construction, public key share $pk_{31,i}, pk_{32,i}, pk_{33,i}$ respectively, is also created from the underlying signature scheme, and they are made public.

[5] In our scheme, there is no requirement for threshold values to be in a specific order, whether monotonous increasing or decreasing. Without loss of generality, these values can be chosen arbitrarily depending on the application e.g. $t_1 = 2$, $t_2 = 4$, and $t_3 = 3$.

(c) Generates the third level public key pk_3 from all published public signing key shares

<u>`KeyAgg`:</u> All level-based public keys pk_1, pk_2, pk_3 are combined to generate the main public key pk.
<u>`SignGen`:</u>

1. **($\mathcal{L}_1(2,3)$).** Any two of $\mathcal{N}_{1,i}$ use his private key share $sk_{11,i}$ to sign the message partially as $\sigma_{11,i}$, where $1 \leq i \leq 3$
2. **($\mathcal{L}_2(2,6)$).** Any two of $\mathcal{N}_{1,i}, \mathcal{N}_{2,i}$ use his private key share $sk_{21,i}, sk_{22,i}$, respectively, to sign the message partially as $\sigma_{21,i}, \sigma_{22,i}$, where $1 \leq i \leq 3$.
3. **($\mathcal{L}_3(6,9)$).** Any six of $\mathcal{N}_{1,i}, \mathcal{N}_{2,i}, \mathcal{N}_{3,i}$ use his private key share $sk_{31,i}, sk_{32,i}, sk_{33,i}$, respectively, to sign the message partially $\sigma_{31,i}, \sigma_{32,i}, \sigma_{33,i}$, where $1 \leq i \leq 3$.

<u>`SignAgg`:</u> The quorum partial signatures generate the main signature σ :

$$\{\sigma_{11,i} \in \mathcal{L}_1\}^{t_1=2} \wedge \{\sigma_{21,i} \in \mathcal{L}_2 \vee \sigma_{22,i} \in \mathcal{L}_2\}^{t_2=2}$$
$$\wedge \{\sigma_{31,i} \in \mathcal{L}_3 \vee \sigma_{32,i} \in \mathcal{L}_3 \vee \sigma_{33,i} \in \mathcal{L}_3\}^{t_3=6}$$

where $1 \leq i \leq 3$, $\wedge$ represents "and", $\vee$ represents "or" notation

3.1 Generalization of FlexHi Scheme

We now formalize the notion of our proposed *FlexHi* scheme that uses the $\mathcal{F}_{FlexHi}$ function. *FlexHi* scheme consists of a tuple of six polynomial time algorithms, $\mathcal{F}_{FlexHi}$ = (`Setup, KeyGen, KeyAgg, SignGen, SignAgg, Verif`) as introduced in Figure 1.

Flex Hierarchical Threshold Signature Scheme Functionality $\mathcal{F}_{FlexHi}$

`Setup`: On the input of the security parameter 1^λ, public parameters pp are generated according to the underlying signature scheme.

`KeyGen`: Taking public parameters pp, each node $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$, where $1 \leq s \leq n$:
- Runs the KeyGen as defined in Figure 2 to generate a public/private key shares $(pk_{s,i}, sk_{s,i})$.
- Stores $(pk_{s,i}, sk_{s,i})$ and send $pk_{s,i}$ to the other nodes in that level.
- Computes level's public key pk_s by combining all received public key shares.
- Sets an internal flag ready to 1 and ignore further calls.

`KeyAgg`: Taking different level's public keys pk_s from each level, do:
- Run the key aggregation algorithm in Figure 5, and construct the main public key pk.

`SignGen`: Taking (sid, m) from each node $\mathcal{N}_{s,l}$ in the same level $\mathcal{L}_s$, $sid = (\mathcal{N}_{s,l}, sid')$ for some sid', if ready $= 1$ and the session identifier sid has not been used previously, then each node $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$:
- Generates partial signature $\sigma_{s,i}$ on message m using the corresponding private key share $sk_{s,i}$, then sends it to each node $\mathcal{N}_{s,l}$ in the same level $\mathcal{L}_s$.
- Stores internally $(sid, \mathsf{delivered})$
- Computes level's signature σ_s after verifying the partial signatures at that level $\mathcal{L}_s$.

`SignAgg`: Taking each level's signatures σ_s, construct the main signature σ.

`Verif`: If $\mathsf{Verify}(\sigma, \mathsf{m}) = 1$, then the verification is done.

Fig. 1. Our Flex Hierarchical Threshold Signature Scheme Functionality $\mathcal{F}_{FlexHi}$ procedure

The KeyGen procedure in the Figure 2 algorithm consists of two rounds. In Round 1, nodes select secrets, create secret polynomials, generate knowledge proofs, and verify these proofs to ensure the validity of the secrets. After secret polynomials are constructed, in Round 2, nodes compute subshares and use them to calculate private key shares. They also generate public key shares using the underlying signature scheme. This process establishes the keys for the algorithm.

KeyGen of $\mathcal{F}_{FlexHi}$

Round 1. (**Secret Sampling**) On input security parameter 1^λ, each $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$ does the followings ($1 \le i \le m_s$):

- Takes a secret $s_{s,i} \in F_q$
- Creates a polynomial $f_{s,i}(x) = s_{s,i} + \sum_{i=1}^{t-1} a_i.x^i$ where $a_i \in F_q$ is the randomly chosen coefficients of the polynomial
- Runs ProofGen($1^\lambda, id_{s,i}, g^{s_{s,i}}$) algorithm in Figure 3 to prove knowledge of secret by outputting $PoK(s_{s,i})$, then broadcast it to all $\mathcal{N}_{s,l}$ node in the same level $\mathcal{L}_s$ ($1 \le l \le m_s$, $l \ne i$)
- Runs ProofVerify($1^\lambda, id_{s,i}, PoK(s_{s,i})$) algorithm in Figure 4 to verify the knowledge of secret $s_{s,i}$

Round 2. (**Key Construction**)
After constructing secret polynomial $f_{s,i}(x)$ from Round 1, each $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$ does the followings ($1 \le i \le m_s$):

- Computes subshares $ss_{i,l}$ for $\mathcal{N}_{s,l}$ by evaluating a point on polynomial $f_{s,i}(id_{s,l})$, then sends these computed values to $\mathcal{N}_{s,l}$ ($1 \le l \le m_s$, $i \ne l$).
- When receiving ($m_s - 1$) subshares, computes its private key share $sk_{s,i}$ by adding related subshares $ss_{l,i}$ where $1 \le l \le m_s$. Public key share $pk_{s,i}$ is also generated from $sk_{s,i}$ using the underlying signature scheme.

Fig. 2. Our KeyGen of $\mathcal{F}_{FlexHi}$ procedure

The ProofGen procedure in Figure 3 algorithm is responsible for generating a proof of knowledge, denoted as $PoK(s_{s,i})$, for a specific secret $s_{s,i} \in F_q$. This process is carried out by each node $\mathcal{N}_{s,i}$ within the same level $\mathcal{L}_s$. The resulting proof, denoted as $PoK(s_{s,i})$, ensures that the node possesses the knowledge of the secret $s_{s,i}$ without revealing the secret. This proof is verified by the ProofVerify procedure in Figure 4. The KeyAgg procedure in Figure 5 combines level-based public keys to generate the main public key.

ProofGen of $\mathcal{F}_{FlexHi}$

On input security parameter 1^λ, secret in commitment $g^{s_{s,i}}$, and id $id_{s,i}$, each $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$ ($1 \le i \le m_s$) calculates a proof of knowledge $PoK(s_{s,i})$ to corresponding secret $s_{s,i} \in F_q$

Fig. 3. Our ProofGen of $\mathcal{F}_{FlexHi}$ procedure

ProofVerify of $\mathcal{F}_{FlexHi}$

On input security parameter 1^λ, a proof of knowledge $PoK(s_{s,i})$, and id $id_{s,i}$, each $\mathcal{N}_{s,i}$ in the same level $\mathcal{L}_s$ ($1 \le i \le m_s$) verifies a proof of knowledge of secret

Fig. 4. Our ProofVerify of $\mathcal{F}_{FlexHi}$ procedure

`KeyAgg` of $\mathcal{F}_{FlexHi}$

The level-based public keys pk_s $(1 \leq s \leq n)$ are combined to generate the main public key $pk = pk_1 * pk_2 \cdots pk_n$ where $*$ denotes the combination operator.

Fig. 5. Our `KeyAgg` of $\mathcal{F}_{FlexHi}$ procedure

4 FlexHi FROST Scheme Application

In this section, we apply our FlexHi scheme to the FROST (Flexible Round-Optimized Schnorr Threshold) signature scheme. FROST [16] is a round-optimized threshold signature scheme based on the Schnorr signature [21], which enhances robustness by allowing a quorum of honest participants to identify instances of misbehavior. FROST includes a semi-trusted role in the final stage of the signature, which is referred to as the signature aggregator $\mathcal{SA}$. The purpose of $\mathcal{SA}$ is to minimize communication between participants. Thus, it can also be implemented without the need for a $\mathcal{SA}$. We adhere to the algorithm in FROST and apply our hierarchical model. Our scenario consists of two levels $\mathcal{L}_1(1, 1)$ and $\mathcal{L}_2(t, m)$. The first level is hierarchically strong and has one hierarchical node, the other is the level with m nodes and t threshold value.

Key Generation Stage. The key generation phase of our FROST application is given in Figure 6. The key generation of FROST is based on Pedersen's Distributed Key Generation [19]. In addition to Pedersen DKG, participants must prove the constant term of the polynomials (secret) they produce with zero-knowledge proofs to prevent rogue key attacks [4] in which attackers are allowed to arbitrarily choose their public keys. Since $\mathcal{L}_1(1, 1)$ consists of a single node $\mathcal{N}_{1,1}$ in our scenario, it does not need key generation rounds at his level. It will be enough for him to generate the private key himself and publish its public key. However, as stated in the previous section, $\mathcal{N}_{1,1}$ needs to join the key generation of $\mathcal{L}_2(t, m)$. In this case, $\mathcal{N}_{1,1}$ has an extra secret. On the other hand, the nodes in $\mathcal{L}_2(t, m)$ should run the protocol as is, adhering to the `KeyGen` phase in FROST. Each node $\mathcal{N}_{2,i}$ randomly selects a polynomial of degree t and creates a Schnorr Proof for its constant term by running `ProofGen`. They then broadcast the proof values and the commitment of the coefficients of the polynomial. Nodes verify proof values from other nodes by running `ProofVerify`. If there is no error during verification, each node sends subshare of secret values to other nodes. After that, each node $\mathcal{N}_{2,i}$ creates its private key share using incoming subshares. After these steps, public key share, and level-based public key are calculated.

Key Aggregation Stage. In the key aggregation algorithm of our FROST application that is given in Figure 9, the main public key `pk` is generated with a multiplication operator taken as a combination operator.

Preprocessing Stage. The preprocessing stage of our FROST application in Figure 10 is performed before the signing operation. In this stage, every node at every level generates a set of π random number pairs, which are one-time use

`KeyGen`

For $\mathcal{L}_1(1,1)$:

1. $\mathcal{N}_{1,1}$ samples a random private key $sk_1 \xleftarrow{\$} \mathbb{Z}_q$
2. $\mathcal{N}_{1,1}$ computes level public key $pk_1 = g^{sk_1}$

For $\mathcal{L}_2(t,m)$:

Round 1 For $1 \le i \le m$,

1. Every node $\mathcal{N}_{2,i}$ chooses t random values $\left(a_{2i,0}, \ldots, a_{2i,(t-1))}\right) \xleftarrow{\$} \mathbb{Z}_q$, and uses them as coefficients to define a degree $t-1$ polynomial $f_{2i}(x) = a_{2i,0} + \sum_{j=1}^{t-1} a_{2i,j}x^j$ where $a_{2i,0}$ is chosen secret.
2. Node $\mathcal{N}_{2,i}$ runs $\texttt{ProofGen}(1^\lambda, id_{2i}, \Phi, g^{a_{2i,0}})$ in Figure 7 for $\mathcal{L}_2(t,m)$ to demonstrate $PoK(a_{2i,0})$ by outputting κ_{2i} and commitment $\vec{C}_{2i}$.
3. Upon receiving $\vec{C}_{2\ell}, \kappa_{2\ell}$ from other nodes $1 \le \ell \le m$, $\ell \ne i$, Node $\mathcal{N}_{2,i}$ runs $\texttt{ProofVerify}(1^\lambda, id_{2\ell}, \Phi, \kappa_{2\ell}, \vec{C}_{2\ell})$ in Figure 8 for $L_2(t,m)$ to verify proof of their corresponding secret.

Round 2 For $1 \le i \le m$,

1. Each node $\mathcal{N}_{2,i}$ securely sends to each other level 2 node $\mathcal{N}_{2\ell}$ a secret share (subshare) $(id_{2\ell}, f_{2i}(id_{2\ell}))$, deleting f_{2i} and each share afterward except for $(id_{2i}, f_{2i}(id_{2i}))$ which they keep for themselves.
2. Each node $\mathcal{N}_{2,i}$ verifies their shares by calculating: $g^{f_{2\ell}(id_{2i})} \stackrel{?}{=} \prod_{k=0}^{t-1} \phi_{2\ell,k}^{{id_{2i}}^k \bmod q}$, aborting if the check fails.
3. Each node $\mathcal{N}_{2,i}$ calculates their long-lived private key share by computing $sk_{2i} = \sum_{\ell=1}^{m} f_{2\ell}(id_{2i})$, stores sk_{2i} securely, and deletes each $f_{2\ell}(id_{2i})$.
4. Each node $\mathcal{N}_{2,i}$ calculates their public verification share $pk_{2i} = g^{sk_{2i}}$, and the level's public key $pk_2 = \prod_{j=1}^{m} \phi_{2j,0}$. Any node can compute the public verification share of any other node by calculating

$$pk_{2i} = \prod_{j=1}^{m} \prod_{k=0}^{t-1} \phi_{2j,k}^{id_{2i}^k \bmod q}.$$

Fig. 6. Our FlexHi `KeyGen` procedure for FROST scheme

`ProofGen`

For $\mathcal{L}_1(1,1)$:

1. $\mathcal{N}_{1,1}$ calculates a proof of knowledge to the corresponding private key sk_1 by calculating $\kappa_1 = (R_1, \mu_1)$, such that $r_1 \xleftarrow{\$} \mathbb{Z}_q$, $R_1 = g^{r_1}$, $c_1 = H\left(id_{1,1}, \Phi, g^{sk_1}, R_1\right)$, $\mu_1 = r_1 + sk_1 \cdot c_1$ with Φ being a context string to prevent replay attacks. Note that $\mathcal{N}_{1,1}$ has already computed commitment g^{sk_1} as Pk_1 value.

For $\mathcal{L}_2(t,m)$:

1. Every node $\mathcal{N}_{2,i}$ calculates a proof of knowledge to the corresponding secret $a_{2i,0}$ by calculating $\kappa_{2i} = (R_{2i}, \mu_{2i})$, such that $r \xleftarrow{\$} \mathbb{Z}_q$, $R_{2i} = g^r$, $c_{2i} = H\left(id_{2i}, \Phi, g^{a_{2i,0}}, R_{2i}\right)$, $\mu_{2i} = r + a_{2i,0} \cdot c_{2i}$ with Φ being a context string to prevent replay attacks.
2. Also, every node $\mathcal{N}_{2,i}$ calculates a commitment $\vec{C}_{2i} = \langle \phi_{2i,0}, \ldots, \phi_{2i,(t-1)} \rangle$ such that $\phi_{2i,j} = g^{a_{2i,j}}$, where $0 \le j \le t-1$.

Fig. 7. Our FlexHi `ProofGen` procedure for FROST scheme

ProofVerify

For $\mathcal{L}_1(1,1)$**:**

1. Upon receiving pk_1, κ_1 from $\mathcal{N}_{1,1}$, each level 2 nodes $1 \leq i \leq m$, $\mathcal{N}_{2,i}$ verifies $\kappa_1 = (R_1, \mu_1)$, aborting on failure, by checking $R_1 \overset{?}{=} g^{\mu_1} \cdot pk_1^{-c_1}$, where $c_1 = H\left(id_{1,1}, \Phi, g^{sk_1}, R_1\right)$.

For $\mathcal{L}_2(t,m)$**:**

1. Upon receiving $\vec{C}_{2\ell}, \kappa_{2\ell}$ from level 2 nodes $1 \leq \ell \leq m, \ell \neq i$, node $\mathcal{N}_{2,i}$ verifies $\kappa_{2\ell} = (R_{2\ell}, \mu_{2\ell})$, aborting on failure, by checking $R_{2\ell} \overset{?}{=} g^{\mu_{2\ell}} \cdot \phi_{2\ell,0}^{-c_{2\ell}}$, where $c_{2\ell} = H\left(id_{2\ell}, \Phi, \phi_{2\ell,0}, R_{2\ell}\right)$.

Fig. 8. Our FlexHi `ProofVerify` procedure for FROST scheme

KeyAgg

1. Node $\mathcal{N}_{1,1}$ runs `ProofGen`$(1^\lambda, id_{1,1}, \Phi, g^{sk_1})$ for $\mathcal{L}_1(1,1)$ that outputs κ_1 to demonstrate zero knowledge proof of corresponding private key sk_1, then broadcasts κ_1 and pk_1 to all other level 2 nodes.
2. Node $\mathcal{N}_{2,i}$ runs `ProofVerify`$(1^\lambda, id_{1,1}, \Phi, \kappa_1, pk_1)$ for $L_1(1,1)$ to verify zero knowledge proof of corresponding private key sk_1.
3. After the proof verification, each level 2 node $1 \leq i \leq m$, $\mathcal{N}_{2,i}$ sends pk_{2i} to $\mathcal{N}_{1,1}$.
4. For $1 \leq i \leq m$, each level 2 nodes $\mathcal{N}_{2,i}$ and $\mathcal{N}_{1,1}$ compute main public key

$$pk = pk_1 * pk_2$$

Fig. 9. Our FlexHi `KeyAgg` procedure for FROST scheme

Preprocessing

For $\mathcal{L}_1(1,1)$**:**

1. $\mathcal{N}_{1,1}$ creates an empty list $\mathcal{L}_{11}$. Then, for $1 \leq j \leq \pi$ where j be a counter for a specific commitment share pair, and π be the number of pairs generated at a time perform the following:
   (a) Sample single-use nonces $(d_{11,j}, e_{11,j}) \overset{\$}{\leftarrow} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$
   (b) Derive commitment shares $(D_{11,j}, E_{11,j}) = \left(g^{d_{11,j}}, g^{e_{11,j}}\right)$.
   (c) Append $(D_{11,j}, E_{11,j})$ to L_{11}.
   (d) Store $((d_{11,j}, D_{11,j}), (e_{11,j}, E_{11,j}))$ for later use in signing operations.
2. Publish $(id_{1,1}, L_{11})$ to a predetermined location.

For $\mathcal{L}_2(t,m)$**:**

1. For $1 \leq i \leq m$, every node $\mathcal{N}_{2,i}$ creates an empty list $\mathcal{L}_{2i}$. Then, for $1 \leq j \leq \pi$ where j be a counter for a specific commitment share pair, and π be the number of pairs generated at a time perform the following:
   (a) Sample single-use nonces $(d_{2i,j}, e_{2i,j}) \overset{\$}{\leftarrow} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$
   (b) Derive commitment shares $(D_{2i,j}, E_{2i,j}) = \left(g^{d_{2i,j}}, g^{e_{2i,j}}\right)$.
   (c) Append $(D_{2i,j}, E_{2i,j})$ to L_{2i}.
   (d) Store $((d_{2i,j}, D_{2i,j}), (e_{2i,j}, E_{2i,j}))$ for later use in signing operations.
2. Publish $(id_{2i}, \mathcal{L}_{2i})$ to a predetermined location.

Fig. 10. Our FlexHi `Preprocessing` procedure for FROST scheme

private nonces and their corresponding commitment shares. Each node calculates its own commitment shares using these number pairs and shares them with the other participants. The value of π determines how many signing operations can be performed before the next preprocess stage.

Signing Stage. In the signing phase of our FROST application, $\mathcal{N}_{1,1}$ and each level 2 node $\mathcal{N}_{2,i}$ perform similar operations as defined in Figure 11. Each node creates the same group commitment by using the nonces they choose during the preprocessing phase and creates a partial signature by processing them with their private keys. Finally, they send these signatures to $\mathcal{SA}$.

`SignGen`

Let S_2 be set of $\alpha : t \leq \alpha \leq n$ nodes that are selected for signing in the second level threshold scheme. Let $B = \langle(id_{2i}, id_{1,1}, D_{2i}, E_{2i}, D_{11}, E_{11})\rangle_{i \in S_2}$ denote the ordered list of node indices corresponding to each node $\mathcal{N}_{s,i}$. Let H_1, H_2 be hash functions whose outputs are in $\mathbb{Z}_q^*$

– For each $i \in S$, $\mathcal{SA}$ sends $\mathcal{N}_{2,i}$ and $\mathcal{N}_{1,1}$ the tuple (m, B)

For $\mathcal{L}_1(1,1)$**:**

1. After receiving (m, B), $\mathcal{N}_{1,1}$ first validates the message m, and then checks $D_{11}, E_{11} \in \mathbb{G}^*$ in B, aborting if either check fails.
2. $\mathcal{N}_{1,1}$ computes the set of binding values $\rho_{2\ell} = H_1(id_{2\ell}, m, B)$, and $\rho_{11} = H_1(id_{1,1}, m, B)$, $\ell \in S_2$. $\mathcal{N}_{1,1}$ then derives the group commitment $R = \prod_{\ell \in S_2} D_{2\ell} \cdot D_{11} \cdot (E_{2\ell})^{\rho_{2\ell}} \cdot (E_{11})^{\rho_{11}}$, and the challenge $c = H_2(R, pk, m)$.
3. Using the first level private key sk_1, $\mathcal{N}_{1,1}$ computes $z_1 = d_{11} + (e_{11} \cdot \rho_{11}) + sk_1 \cdot c$,
4. $\mathcal{N}_{1,1}$ securely deletes $((d_{11}, D_{11}), (e_{11}, E_{11}))$ from the local storage, and returns z_1 to $\mathcal{SA}$.

For $\mathcal{L}_2(t, m)$**:**

1. After receiving (m, B), each $\mathcal{N}_{2,i}$ first validates the message m, and then checks $D_{2\ell}, E_{2\ell} \in \mathbb{G}^*$ for each commitment in B, aborting if either check fails.
2. Each $\mathcal{N}_{2,i}$ then computes the set of binding values $\rho_{2\ell} = H_1(id_{2\ell}, m, B)$, and $\rho_{11} = H_1(id_{1,1}, m, B)$, $\ell \in S_2$. Each $\mathcal{N}_{2,i}$ then derives the group commitment $R = \prod_{\ell \in S_2} D_{2\ell} \cdot D_{1,1} \cdot (E_{2\ell})^{\rho_{2\ell}} \cdot (E_{11})^{\rho_{11}}$ and the challenge $c = H_2(R, pk, m)$.
3. Each $\mathcal{N}_{2,i}$ computes their response using their long-lived private key share sk_{2i} by computing $z_{2i} = d_{2i} + (e_{2i} \cdot \rho_{2i}) + \lambda_{2i} \cdot sk_{2i} \cdot c$, using S_2 to determine the i^{th} Lagrange coefficient λ_{2i}.
4. Each $\mathcal{N}_{2,i}$ securely deletes $((d_{2i}, D_{2i}), (e_{2i}, E_{2i}))$ from their local storage, and then returns z_{2i} to $\mathcal{SA}$.

Fig. 11. Our FlexHi `SignGen` procedure for FROST scheme

Signature Aggregation Stage. In the signature aggregation phase defined in Figure 12, $\mathcal{SA}$ verifies each signature and creates the signature of the level. Finally,$\mathcal{SA}$ creates the signature by combining these two signatures.

Signature Verification Stage. As in the standard Schnorr's verification [21] operation, the main signature $\sigma = (R, z)$ on message m is verifiable with main public key pk.

SignAgg

1. $\mathcal{SA}$ derives $\rho_{2i} = H_1(id_{2i}, m, B)$, $\rho_{11} = H_1(id_{1,1}, m, B)$, $R_{2i} = D_{2i,j} \cdot (E_{2i,j})^{\rho_{2i}}$, and $R_{11} = D_{11,j} \cdot (E_{11,j})^{\rho_{11}}$ for $i \in S$. Subsequently, $\mathcal{SA}$ derives $R = \prod_{i \in S} R_{2i} \cdot R_{11}$ and $c = H_2(R, pk, m)$.
2. $\mathcal{SA}$ verifies the validity of each response by checking $g^{z_{2i}} \overset{?}{=} R_{2i} \cdot pk_{2i}^{c \cdot \lambda_{2i}}$ for each signing share $z_{2i}, i \in S$. Also, $g^{z_1} \overset{?}{=} R_{11} \cdot pk_{11}^{c}$ is checked. If the equality does not hold, identify and report the misbehaving node and then abort. Otherwise, continue.
3. $\mathcal{SA}$ computes the level's response $z_2 = \sum z_{2i}$
4. $\mathcal{SA}$ computes $z = z_1 + z_2$, then broadcasts the main signature as $\sigma = (R, z)$ along with m.

Fig. 12. Our FlexHi `SignAgg` procedure for FROST scheme

5 Our Analysis

We now proceed to our analysis section, which comprises two essential components: security and cost analysis. In the security analysis, we first examine indistinguishability and unforgeability properties using game-based security then address ideality, perfectness, and collusion attack resistance. In the cost analysis, we assess the efficiency of our scheme in comparison to the FROST protocol and Tassa's hierarchical threshold signature scheme integrated with the FROST scheme.

5.1 Security Analysis

Definition 6 (Lagrange interpolation [1]). *For every field $\mathbb{F}$, and a given t different points (x_i, y_i), $1 \le i \le t$, there exits a unique polynomial P of degree at most $t-1$ over $\mathbb{F}$ such that $P(x_i) = y_i$, $1 \le i \le t$.*

The Lagrange interpolating polynomial is calculated as follows:

$$P(x) = \sum_{i=1}^{t} P_i(x) \ , \ P_i(x) = y_i. \prod_{j=1, j \ne i}^{t} \frac{x - x_j}{x_i - x_j}.$$

Game-based Security. In this subsection, we show the indistinguishability and unforgeability properties of *FlexHi* scheme using game-based security where the adversary is modeled as a polynomial time algorithm. We denote the security parameter by λ, and the negligible success probability of an adversary by $\mathsf{negl}(\lambda)$, which means it is so small as to be zero [8].

Based on the game-based security proofs for secret-sharing schemes in [29], we can prove the secret indistinguishability of the proposed *FlexHi* scheme by defining the following experiment $\mathsf{Exp}_{\mathcal{A}}^{SecInd,b}(\lambda)$ between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$:

– The challenger $\mathcal{C}$ generates the field $\mathbb{F}$ as public parameter.

- Adversary $\mathcal{A}$ chooses two distinct but the same length secret $\mathsf{s_0}$ and $\mathsf{s_1}$ in that field, then sends them to the challenger $\mathcal{C}$.
- The challenger $\mathcal{C}$ selects uniformly random bit $\mathsf{b} \xleftarrow{\$} \{0,1\}$, then sends challenge subshare $(\mathsf{ss}^{\mathsf{b}}_{\mathsf{i,j}})_{j\in A'} \leftarrow \mathsf{Share}(\mathsf{s_b})$ to $\mathcal{A}$ using oracle (A' represents an unauthorized subset)
- Adversary $\mathcal{A}$ outputs a guess bit $\mathsf{b}' \in \{0,1\}$

The output of the experiment is defined to be 1 if $\mathsf{b}' = \mathsf{b}$

Theorem 1. FlexHi *scheme satisfies secret indistinguishability property given that*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{SecInd},b}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

Proof. *FlexHi* scheme uses the key generation of Shamir's secret-sharing scheme without a trusted dealer version [19]. Based on this, in our scheme (see Figure 2) each node chooses uniformly random secret s_i, then creates their private key share sk_i by taking all related subshares ss_i. Thus, each node contributes one piece of that private key share. Since the secrets are chosen uniformly randomly, the distribution of subshares is indistinguishable from uniform distribution. □

Unforgeability is another important security property in signature schemes, demonstrating an adversary's inability to create a valid signature for a message that hasn't been previously signed. In the context of threshold unforgeability, the assumption is that the adversary has compromised $t-1$ servers. This notion is captured by the following game $\mathsf{Game}^{Unf}_{\mathcal{A}}(\lambda)$:

- The challenger $\mathcal{C}$ generates the public key of the scheme, and provides access to signing oracle $\mathcal{O}_{\mathsf{Sign}}$.
- When adversary $\mathcal{A}$ requests a signature σ for some messages m, the challenger $\mathcal{C}$ outputs their signatures.
- Eventually $\mathcal{A}$ generates a new signature σ^* for message m^* that is not asked to $\mathcal{C}$

The adversary succeeds in the game if the verification of the signature σ^* for message m^* is correct, but m^* is not the same as any of the queried inputs to the oracle.

Theorem 2. FlexHi *scheme satisfies unforgeability property if the adversary's advantage is negligible*

$$\left|\Pr[\mathsf{Game}^{\mathsf{Unf}}_{\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. In *FlexHi* scheme, the message is signed partially by the eligible node's private key share. These shares are constructed from the subshares of the chosen secret in Shamir. Thus, the security of *FlexHi* scheme is based on Lagrange's interpolation as in Definition 6. According to this, the private key share can be generated by only an authorized subset of participants who hold t points (subshare) of the polynomial P. On the other hand, an authorized subset $t-1$ of participants who hold $t-1$ points (subshare) of the polynomial P cannot generate the secret polynomial, and thus cannot forge private key share. □

Ideality and Perfectness. The most important parameter for the secret-sharing system is the perfectness of the system that is required to protect the secret. Perfect schemes are referred to as unconditionally secure schemes [24]. The other desirable property is the ideality of the scheme. However, it should be noted that for every access structure, we cannot find an ideal scheme [5].

Definition 7 (Perfectness [24]). *A secret-sharing scheme is a perfect realization of access structure Γ if*

- *An authorized subset of participants $\mathsf{A} \in \mathsf{\Gamma}$ can always reconstruct the secret,*
- *An unauthorized subset $t-1$ of participants $\mathsf{A}' \in \mathsf{\Gamma}$ cannot obtain any information about the secret.*

Definition 8 (Ideality [24]). *A secret-sharing scheme is ideal if the length of the share of all participants is less than or equal to the size of the secret.*

Lemma 1. *The proposed scheme is not ideal, but it is perfect.*

Proof. In our approach, the secret is divided into shares for each level, and the shares in each level generate the polynomial of that level. Each level's polynomial is based on Shamir secret-sharing which is perfect since its coefficient matrix is a square Vandermonde matrix and it is always nonsingular [30]. This makes our scheme is also perfect. On the other hand, our scheme uses a general flexible access structure that allows us to choose which subset of participants can reconstruct the secret. However, since each participant takes a certain number of shares in proportion to their levels, the length of the shares at a higher level becomes larger than the secret. Thus, we cannot expect our general flexible access structure to be ideal. $\square$

Collusion attack resistance. In our scheme, the shares of secret signing keys are maintained by level-based polynomials. However, there is no correlation between each of the level-based polynomials. Considering a collusion attack, the attackers need to get all level polynomials carrying the secret key shares, but this is not practical in a real-world scenario. Thus, our scheme provides resistance against collusion attacks.

5.2 Cost Analysis

To illustrate the overhead of our hierarchical threshold signature scheme, we conduct a comparative analysis with two key reference points. We compare the efficiency of our scheme to the FROST scheme [16], examining the overhead introduced by the hierarchical structure in contrast to a basic threshold signature scheme. Second, we evaluate our system against Tassa's hierarchical threshold signature scheme [25] applied to the FROST scheme. Tassa's pioneering work is rooted in the construction of a polynomial based on an unstructured set of point and derivative values, offering a novel approach. By making this comparison, we provide a practical benchmark and highlight the strengths of our system.

Theoretical Complexity Analysis. We first analyze the theoretical complexity in Table 1. The symbols E, M, and I in the analysis stand for modular exponentiation, modular multiplication, and modular inverse operation, respectively.

Table 1. Cost Analysis of the Algorithms in *FlexHi* scheme Application to FROST

Scheme	**KeyGen**	**KeyAgg**	**SignGen**	**SignAgg**
Plain FROST without hierarchy $(t+1, m+1)$	$(2(m+1)(t+1)-2(m+1)+2)(m+1)$ E $(2(m+1)(t+1)-(t+1))(m+1)$ M $m(m+1)$ I	—	$(t+1)^2+3(t+1)$ E $(2(t+1)+2)(t+1)+4(t+1)-1$ M	—
Tassa's HTSS on FROST $(\mathcal{L}_1(1,1)+\mathcal{L}_2(t,m))$	$(2mt+2t-m+2)(m+1)$ E $(3mt+2t-m-1)(m+1)$ M	—	$(t+1)^2+3(t+1)$ E $(2(t+1)+2)(t+1)+4(t+1)-1$ M	—
Our FHTSS on FROST $\mathcal{F}_{FlexHi}$ $(\mathcal{L}_1(1,1)+\mathcal{L}_2(t,m))$	$(2mt-2m+2)m+1$ E $(2mt-t)m$ M $(m-1)m$ I	$(m+1)$ E $(2m+2)$ M m I	$(t+1)^2$ E $(2(t+1)+2)(t+1)$ M	$3t+3$ E $3t+2$ M

Since modular exponentiation and modular inverse operations can be written in terms of modular multiplication, the computation cost of these schemes can finally be examined in terms of modular multiplications. As stated in [15], modular exponentiation E takes 240 times longer than modular multiplication M, and modular inverse I requires 3 modular multiplication M. Using this information, Figure 13 shows the total modular multiplication cost for key generation and aggregation algorithms. According to the Figure 13, the first bar chart shows our *FlexHi* FROST application runs 29% faster than Tassa's FROST application in the $(t, m) = (6, 7)$ case, and 36% faster in the $(t, m) = (3, 7)$ case. Also, the second bar chart shows that our *FlexHi* FROST application runs 28% faster than Tassa's FROST application in the $(t, m) = (4, 10)$ case, and 38% faster in the $(t, m) = (4, 5)$ case. It is clear from this that our scheme works faster than Tassa's scheme.

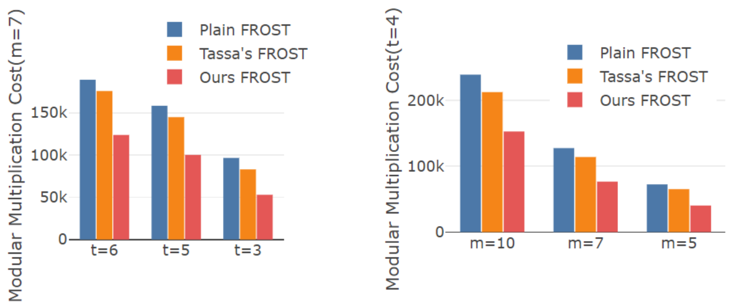

Fig. 13. Modular Multiplication Cost of Key Generation and Key Aggregation Phases of FROST and its variants where the first bar chart represents $m = 7$ and the second bar chart represents $t = 4$

Implementation. To substantiate our argument concerning theoretical efficiency, we have implemented our *FlexHi* scheme on the FROST as an open-source code [6]. Specifically, our efforts centered around a comprehensive comparative analysis of three distinct variants of FROST: Plain FROST with threshold signing scheme (Plain TSS), Tassa's HTSS on FROST, and our *FlexHi* scheme on FROST. The primary objective was to evaluate their computational efficiency and suitability across a spectrum of threshold values and participant counts in alignment with the theoretical framework. Table 2 displays the calculated times for different parameters. The results of the tests are performed on a computer with i7-1165g7 @ 2.80 GHz and 16 GB RAM. The test environment incorporated two distinct elliptic curves, P256 [20] and ed25519 [6]. In Table 2, only results with the ed25519 curve are available, tests with the P256 curve are available in our GitHub codes. Throughout our experiments, we maintained a consistent threshold value while varying the number of participants. This empirical evidence underscores the efficiency of our proposed scheme, particularly in contexts that prioritize computational efficiency and resource optimization.

Table 2. The calculated times for different threshold values and the number of participants in different algorithms where Plain TSS (threshold signature scheme) corresponds to the plain FROST [16] Scheme, Tassa's HTSS refers to its application on FROST scheme, and Our *FlexHi* scheme refers to its application on FROST scheme

Scheme (Curve)	Threshold (t)	Time (ms)	Threshold (t)	Time (ms)
Plain TSS (ed25519)	$\mathcal{L}(4,5)$	10.104	$\mathcal{L}(3,7)$	13.453
Tassa's HTSS (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,4)$	8.996	$\mathcal{L}_1(1,1)+\mathcal{L}_2(2,6)$	12.213
Our *FlexHi* (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,4)$	6.498	$\mathcal{L}_1(1,1)+\mathcal{L}_2(2,6)$	10.781
Plain TSS (ed25519)	$\mathcal{L}(4,7)$	17.139	$\mathcal{L}(5,7)$	19.147
Tassa's HTSS (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,6)$	15.952	$\mathcal{L}_1(1,1)+\mathcal{L}_2(4,6)$	17.829
Our *FlexHi* (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,6)$	13.280	$\mathcal{L}_1(1,1)+\mathcal{L}_2(4,6)$	15.127
Plain TSS (ed25519)	$\mathcal{L}(4,10)$	32.196	$\mathcal{L}(6,7)$	22.288
Tassa's HTSS (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,9)$	29.006	$\mathcal{L}_1(1,1)+\mathcal{L}_2(5,6)$	20.624
Our *FlexHi* (ed25519)	$\mathcal{L}_1(1,1)+\mathcal{L}_2(3,9)$	25.768	$\mathcal{L}_1(1,1)+\mathcal{L}_2(5,6)$	17.584

6 Conclusion

Traditional threshold signature schemes, while essential, have limitations when applied to hierarchical structures, where varying levels of authority and access control are required. Our proposed *FlexHi* scheme, built upon Shamir's construction, enhanced with independent polynomials at each hierarchical level, offers a novel and adaptable solution. It eliminates the rigid constraints on threshold values, enabling the scheme to conform to a variety of real-world scenarios. Therefore, our scheme is capable of seamlessly adapting to a wide range of real-world scenarios and organizational structures. Our hierarchical threshold scheme is adaptable and can be applied to most existing threshold schemes.

[6] https://github.com/midmotor/hierarchical-threshold-signature

We demonstrated its applicability by implementing it on the state-of-the-art, round-optimized FROST scheme. In our comparison, our FROST-based *FlexHi* application significantly outperforms Tassa's FROST application. Based on our analysis, our scheme runs about 30% - 40% faster, depending on the number of participants and the threshold values. Our *FlexHi* scheme not only introduces a more adaptable approach to hierarchical threshold signature but also demonstrates its practical advantages through faster execution.

Future research in this field could concentrate on key areas, such as integrating *FlexHi* into various threshold schemes and identifying the optimal schemes that align best with hierarchical settings. Additionally, there is potential for improvement by adding dynamic functionality to *FlexHi*, enabling the adjustment of the threshold without the need for regenerating the master public key. However, these aspects are left as subjects for future research.

References

1. Beimel, A.: Secret-sharing schemes: A survey. In: International conference on coding and cryptology. pp. 11–46. Springer (2011)
2. Beimel, A., Tassa, T., Weinreb, E.: Characterizing ideal weighted threshold secret sharing. In: Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2. pp. 600–619. Springer (2005)
3. Belenkiy, M.: Disjunctive multi-level secret sharing. Cryptology ePrint Archive (2008)
4. Bellare, M., Boldyreva, A., Staddon, J.: Randomness re-use in multi-recipient encryption schemeas. In: Public Key Cryptography—PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6. pp. 85–99. Springer (2002)
5. Benaloh, J., Leichter, J.: Generalized secret sharing and monotone functions. Springer (1990)
6. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of cryptographic engineering 2(2), 77–89 (2012)
7. Blakley, G.R.: Safeguarding cryptographic keys. In: Managing Requirements Knowledge, International Workshop on. pp. 313–313. IEEE Computer Society (1979)
8. Boneh, D., Shoup, V.: A graduate course in applied cryptography. Draft 0.5 (2020)
9. Brickell, E.F.: Some ideal secret sharing schemes. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 468–475. Springer (1989)
10. Ersoy, O., Kaya, K., Kaskaloglu, K.: Multilevel threshold secret and function sharing based on the chinese remainder theorem. Preprint arXiv:1605.07988 (2016)
11. Ghodosi, H., Pieprzyk, J., Safavi-Naini, R.: Secret sharing in multilevel and compartmented groups. In: Information Security and Privacy: ACISP'98 Brisbane, Australia. pp. 367–378. Springer (1998)
12. Harn, L., Fuyou, M.: Multilevel threshold secret sharing based on the chinese remainder theorem. Information processing letters 114(9), 504–509 (2014)
13. Karaoglan Altop, D., Bingol, M.A., Levi, A., Savas, E.: Dkem: Secure and efficient distributed key establishment protocol for wireless mesh networks. Ad Hoc Networks 54, 53–68 (2017)

14. Käsper, E., Nikov, V., Nikova, S.: Strongly multiplicative hierarchical threshold secret sharing. In: Information Theoretic Security: Second International Conference, ICITS 2007, Madrid, Spain, May 25-29, 2007. pp. 148–168. Springer (2009)
15. Koblitz, N., Menezes, A., Vanstone, S.: The state of elliptic curve cryptography. Designs, codes and cryptography 19, 173–193 (2000)
16. Komlo, C., Goldberg, I.: FROST: flexible round-optimized schnorr threshold signatures. In: Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada, October 21-23, 2020. pp. 34–65. Springer (2021)
17. Nojoumian, M., Stinson, D.R.: Sequential secret sharing as a new hierarchical access structure. Cryptology ePrint Archive (2015)
18. Pakniat, N., Noroozi, M., Eslami, Z.: Distributed key generation protocol with hierarchical threshold access structure. IET Information Security 9(4), 248–255 (2015)
19. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10. pp. 522–526. Springer (1991)
20. Qu, M.: Sec 2: Recommended elliptic curve domain parameters. Certicom Res., Mississauga, ON, Canada, Tech. Rep. SEC2-Ver-0.6 (1999)
21. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 239–252. Springer New York, New York, NY (1990)
22. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (nov 1979), https://doi.org/10.1145/359168.359176
23. Simmons, G.J.: How to (really) share a secret. In: Conference on the Theory and Application of Cryptography. pp. 390–448. Springer (1988)
24. Stinson, D.R.: An explication of secret sharing schemes. Designs, Codes and Cryptography 2(4), 357–390 (1992)
25. Tassa, T.: Hierarchical threshold secret sharing. In: Theory of Cryptography Conference. pp. 473–490. Springer (2004)
26. Tassa, T., Dyn, N.: Multipartite secret sharing by bivariate interpolation. Journal of Cryptology 22, 227–258 (2009)
27. Tentu, A.N., Paul, P., Venkaiah, V.C.: Ideal and perfect hierarchical secret sharing schemes based on mds codes. Cryptology ePrint Archive (2013)
28. Traverso, G., Demirel, D., Buchmann, J.: Performing computations on hierarchically shared secrets. In: Progress in Cryptology–AFRICACRYPT 2018: 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7–9, 2018, Proceedings 10. pp. 141–161. Springer (2018)
29. Xia, Z., Yang, Z., Xiong, S., Hsu, C.F.: Game-based security proofs for secret sharing schemes. In: Second International Conference on Security with Intelligent Computing and Big Data Services (SICBS-2018). pp. 650–660. Springer (2020)
30. Yılmaz, R.: Some ideal secret sharing schemes. Ph.D. thesis, Bilkent Universitesi (Turkey) (2010)
31. Yuan, J., Yang, J., Wang, C., Jia, X., Fu, F.W., Xu, G.: A new efficient hierarchical multi-secret sharing scheme based on linear homogeneous recurrence relations. Information Sciences 592, 36–49 (2022)