Enhancing Blockchain Proof of Stake with Active Weighted Signatures: The ADAPT Framework

Jae Hyun Choi, Hobin Jang, Ik Rae Jeong, Changmin Lee

***Abstract*—Proof of Stake (PoS) blockchain systems require weighted threshold signatures where participants' voting powers reflect their stakes. As stakes change dynamically through deposits and withdrawals, efficient weight and threshold adjustments are essential for maintaining system security and availability without downtime. However, existing approaches face critical limitations: (1) virtualization-based schemes require $O(w)$ operations (signatures) per participant with weight w; (2) dynamic threshold / participants schemes do not support weighted participants; (3) schemes with both properties require trusted dealers or $O(n^2)$ re-setup, causing temporary unavailability.**

This paper introduces Active Weighted Signature (AWS), enabling dynamic adjustments without trusted dealers or re-setup. We propose Generalized Lagrange Interpolation (GLI), encoding weights as polynomial derivatives rather than virtualized participants, and instantiate AWS through ADAPT by applying GLI to the Schnorr-based threshold signature FROST. Our implementation shows that ADAPT achieves comparable efficiency to FROST for key generation, while weight and threshold adjustments complete in 4.1-22.3% of re-setup time. For uneven weight distributions, ADAPT achieves sub-linear scaling: 49× weight difference requires only 3.29× computation versus 49× in virtualization.

***Index Terms*—Blockchain, Proof of Stake (PoS), Threshold signature, Shamir secret sharing, Weighted threshold signature.**

I. Introduction

FROM its inception, blockchain technology has operated on a Proof of Work (PoW) basis. PoW ensures robust security and maintains network decentralization. However, its significant power consumption and escalating computational demands have led the blockchain community to consider transitioning to Proof of Stake (PoS). PoS allows participants to vote according to their stake, offering a more environmentally friendly and economically efficient alternative through reduced power consumption and more efficient resource management [2], [7], [16], [23].

PoS theoretically operates similarly to a shareholders' meeting where voting rights are proportional to the number of shares held. However, many PoS-based blockchains do not fully implement this concept. For instance, Ethereum grants equal voting rights to all miners during the mining process [1]. To clarify the PoS concept, miners' stakes should be differentiated, granting different voting rights based on their stakes. This requires transitioning from classic digital signatures to weighted threshold signatures.

Shareholders' meetings and blockchain PoS are not entirely identical, posing limitations on directly applying traditional weighted threshold signatures to blockchain [27]. In shareholders' meetings, the total amount of shares is fixed, whereas in blockchain systems, stakes can change dynamically. Miners can alter both their stakes and the total system stake at any time through deposits and withdrawals. These differences affect the total amount of stakes and the threshold.

For example, if more than 6 out of 10 stakes are required for a vote, and the total number of stakes reduces to 5 due to withdrawals, gathering 6 stakes becomes impossible, making a threshold of 3 stakes more reasonable.

To apply traditional weighted threshold signatures to blockchain, it is necessary to dynamically adjust participants, weights, and thresholds. However, conventional approaches require complete re-execution of key generation protocols whenever any modification occurs, incurring $O(n^2)$ communication complexity among all participants, causing temporary system unavailability, and introducing potential security risks such as synchronization issues in distributed environments. Recent studies on PoS blockchain security [15], [19], [28] emphasize the critical need for cryptographic primitives that can adapt to the dynamic nature of blockchain systems while maintaining both security and computational efficiency. Thus, new cryptographic techniques are required to enable dynamic weight and threshold adjustments without costly re-setup procedures.

A. Our Contributions

1. Active Weighted Signature (AWS) Framework: We introduce the first threshold signature framework simultaneously achieving fully dynamic threshold (both increase and decrease without re-setup), fully dynamic participants (adding and removing without trusted authority via distributed key generation), dynamic weighted participants (runtime weight adjustments without re-initialization), and constant-size weight

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korea Government (MSIT), Development of Digital Asset Transaction Tracking Technology to Prevent Malicious Financial Conduct in the Digital Asset Market under Grant RS-2024-00460321, and in part by the Korea University Grant (*Corresponding authors: Ik Rae Jeong; Changmin Lee*).

Jae Hyun Choi, Ik Rae Jeong, Changmin Lee are with the School of Cybersecurity, Korea University, Seoul 02841, South Korea (e-mail: 93jamie@korea.ac.kr; irjeong@korea.ac.kr; changminlee@korea.ac.kr). Hobin Jang is with Samsung SDS Co., Ltd., Seoul 05510, South Korea. (email: jhb0858@gmail.com)

This is the accepted version of the article accepted for publication in *IEEE Transactions on Dependable and Secure Computing*. © 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The final version of record will be available at IEEE Xplore (DOI to be assigned).

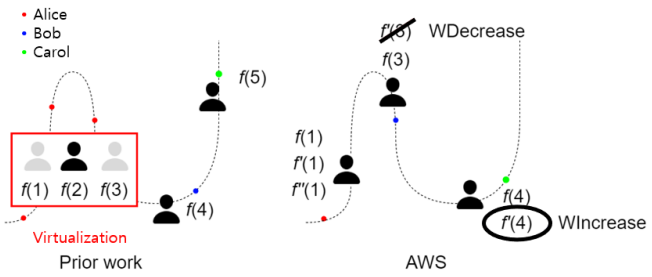

Fig. 1. Comparison of prior work and AWS. Red, blue, and green markers represent Alice, Bob, and Carol, respectively. In prior work, Alice's weight is represented via virtualization (red box). In AWS, WDecrease removes $f'(3)$ and WIncrease appends $f'(4)$ without re-setup.

representation ($O(1)$ public verification information). Table I demonstrates that our scheme is the only one achieving this complete combination of properties with decentralized setup.

2. Generalized Lagrange Interpolation (GLI): We propose GLI, a novel secret sharing technique representing participant i with weight w_i using derivatives $\{f^{(k)}(i)\}_{k=0}^{w_i-1}$ at a single evaluation point rather than virtualization's multiple points. This derivative-based encoding eliminates virtualization through sub-linear computation scaling (49× weight requires only 3.29× computation versus 49× in virtualization, Section VIII), enables efficient weight increase by computing additional derivative $f^{(w_i)}(i)$ (7.2% of key generation time), and supports efficient threshold adjustment via polynomial transformations (4.1-12.5% of key generation time). Crucially, GLI maintains $O(1)$ verification complexity regardless of participant weights, unlike virtualization where overhead grows with total weight.

3. ADAPT: Concrete AWS Instantiation: We construct ADAPT by integrating GLI with the Schnorr-based threshold signature scheme FROST [17]. ADAPT inherits FROST's efficiency and security properties while adding weighted and dynamic capabilities. We prove ADAPT's security under the discrete logarithm assumption via reduction to FROST's security (Theorem 2).

4. Implementation and Evaluation: Our implementation demonstrates ADAPT's practical efficiency with key generation and signing comparable to FROST (Figures 2-3), dynamic operations completing in 4.1-22.3% of re-setup time (Figure 4), verification achieving 184.1µs (1.5×-60× faster than existing weighted schemes, Table III), and sub-linear scaling maintaining efficiency even with extreme weight differences (Table IV).

B. Technical Approach

Our main strategy involves a new secret-sharing method based on Generalized Lagrange Interpolation (GLI). Many threshold schemes rely on Shamir secret sharing, which uses Lagrange interpolation to recover a secret. In such schemes, participants with pairs $\{(i, f(i))\}_{i=1}^{t}$ can uniquely recover the function $f(x)$ of degree $t-1$. The value of $f(0)$ represents the shared secret, and each pair $(i, f(i))$ corresponds to secret information for a participant i.

In many existing weighted threshold signature schemes, providing a weight of $w_0 \geq 1$ requires allocating w_0 pairs to a participant. This approach poses challenges when modifying individual weights or the total threshold, leading to changes in the $f(x)$ function.

To efficiently allocate weight, we employ GLI, which utilizes not only function values but also k-th derivative points, denoted as $(i, f^{(k)}(i))$ for $k \leq w_0$. This method still enables us to recover the function $f(x)$ when all weights exceed t.

This idea, though straightforward, is powerful in functionality. To increase an individual weight, a single additional derivative point suffices, eliminating the need for altering the initial polynomial $f(x)$. Moreover, when adjusting the total weight, we set the target polynomial to $f^{(1)}(x)$ for decrease or $F(x) = \int_0^x f(y)dy$ for increase, as applicable. This avoids resetting $f(x)$ when employing the novel secret-sharing method. Fig. 1 illustrates a comparison between prior works and our approach, using three participants: Alice (red), Bob (blue), and Carol (green). In prior works (left), Alice with weight 3 holds three independent shares $f(1)$, $f(2)$, $f(3)$ via virtualization (red box), treating her as three separate virtual participants. In our AWS approach (right), each participant holds derivative-based shares at a single evaluation point: *WDecrease* simply removes Carol's highest-order derivative $f'(3)$ (strikethrough), and *WIncrease* appends one additional share $f'(4)$ to Bob (circle), leaving all other participants unaffected.

Our ADAPT scheme integrates this secret-sharing method with FROST, offering advantages in signature size and verification time. This technique has broad applicability and can be employed in various threshold signature schemes.

C. Organization

In Section II, we analyze existing approaches to dynamic threshold and weighted signatures, comparing their capabilities and limitations. Section III introduces notation and fundamental cryptographic concepts. Section IV presents the definition of AWS along with its security model. Section V outlines the Generalized Lagrange Interpolation method. Section VI provides a detailed description of ADAPT, our concrete AWS instantiation. Section VII presents the security analysis. Section VIII evaluates our implementation, and Section IX concludes this paper.

II. Related Work

In dynamic weighted (t, n) threshold signature systems, several critical aspects must be considered: the ability to dynamically adjust participants, the threshold, and weights, as well as the setup mechanism employed. We analyze existing schemes according to these dimensions and their approaches to enabling adaptability in threshold cryptography. In Table I, we compare studies related to threshold schemes, verifying the fully dynamic attributes of threshold and participants, weight support (static or dynamic), and the initial setup mechanism.

A. Dynamic Threshold and Participant Management

Dynamic threshold schemes avoid re-running setup when adjusting parameters.

Threshold Dynamics. Maxwell et al. [20] proposed the first dynamic scheme but limited to one-time adjustments. Early schemes [24], [25] based on NPIP and lattice constructions support only threshold increases. Nojoumian and Stinson [22] achieved full threshold dynamics using Zero addition (for increases) and public evaluation (for decreases), but this approach suffers from centralized operations by authorized parties and growing public data under frequent decreases. Garg et al. [14] recently introduced silent setup enabling fully dynamic thresholds with pairing-based hints, but participant dynamics are bounded by predefined maximums requiring re-initialization beyond these limits.

Participant Dynamics. Noack and Spitz [21] and Kubilay et al. [18] achieved fully dynamic participant management via Zero addition. Yuan and Li [29] proposed HLR-based techniques with pairing, while Wang et al. [26] developed Shamir-ECDSA integration but restricted modifications to predefined trustees (partially dynamic).

Post-Quantum Constructions. Recent lattice-based schemes [9], [11], [12] provide post-quantum security with Espitau et al. [11] achieving two-round efficiency. However, none support dynamic threshold, participant, or weight adjustments.

B. Weighted Threshold Signatures

Weighted schemes differentiate participants by authority levels. Dikshit and Singh [10] restrict participation to same-weight users. Garg et al. [13] maintain constant secret sizes per user. Baird et al. [4] employ virtualization where high-weight participants manage multiple virtual identities—this incurs computational overhead proportional to weights. Das et al. [8] use SNARK-based inner product arguments for weight verification. All existing weighted schemes assign static weights fixed at initialization.

Our scheme uniquely combines dynamic weight management with GLI, eliminating virtualization overhead. By encoding weights as polynomial derivatives rather than multiple shares, participants perform constant-complexity operations regardless of assigned weights. Furthermore, our approach supports fully dynamic participant, threshold, and weight adjustments simultaneously.

C. Setup Mechanisms

Most schemes rely on trusted authorities (TA) for initialization [4], [8], [9], [11]–[14], [20], [22], [24], [25], [29], unsuitable for decentralized applications. Distributed key generation (DKG) eliminates centralized trust [10], [17], [18], [21], [26]. Our scheme employs DKG while simultaneously supporting fully dynamic thresholds, participants, and weights—a combination not achieved by prior work as detailed in Table I.

Scheme	Threshold	Participants	Weight	Weight Info	Setup
[9], [11], [12]	✗	✗	✗	-	TA
[17]	✗	✗	✗	-	DKG
[8], [13]	✗	✗	Static	$O(1)$	TA
[10]	✗	✗	Dynamic	$O(n)$	DKG
[29]	✗	Partially	✗	-	TA
[26]	✗	Partially	✗	-	DKG
[4]	✗	Partially	Dynamic	$O(n)$	TA
[18]	✗	Fully	✗	-	TA
[21]	✗	Fully	✗	-	DKG
[20]	One-time	✗	✗	-	TA
[24], [25]	Partially	✗	✗	-	TA
[22]	Fully	✗	✗	-	TA
[14]	Fully	Partially	Static	$O(1)$	TA
Ours	**Fully**	**Fully**	**Dynamic**	$\mathbf{O(1)}$	**DKG**

TABLE I
COMPARISON OF DYNAMIC THRESHOLD SCHEMES. "TA" AND "DKG" DENOTE TRUSTED AUTHORITY AND DISTRIBUTED KEY GENERATION. ✗ INDICATES UNSUPPORTED FEATURES. FOR THRESHOLD AND PARTICIPANT DYNAMICS, "FULLY" DENOTES BIDIRECTIONAL ADJUSTMENT (BOTH INCREASES AND DECREASES); "PARTIALLY" INDICATES UNIDIRECTIONAL OR BOUNDED CHANGES; "ONE-TIME" PERMITS ONLY A SINGLE MODIFICATION. STATIC WEIGHTS ARE FIXED AT INITIALIZATION; DYNAMIC WEIGHTS ALLOW RUNTIME MODIFICATION. "WEIGHT INFO" REPRESENTS PUBLIC INFORMATION COMPLEXITY: $O(1)$ INDICATES CONSTANT SIZE; $O(n)$ INDICATES WEIGHT-PROPORTIONAL SIZE (TYPICALLY FROM VIRTUALIZATION).

III. BACKGROUND

Notation. We focus on a group denoted as G, characterized by a prime order p, with a known hard Decisional Diffie-Hellman problem [6]. Within this group, G, g signifies a generator. Additionally, we utilize H as a cryptographic hash function that maps its outputs to $\mathbb{Z}_p^*$. When we express x being drawn from the set S as $x \xleftarrow{\$} S$, it signifies that x is selected uniformly and randomly from the set S. Given a set S, we let $|S|$ denote its cardinality. For simplicity, we denote $\mathbf{Perm}(n, r) = \frac{n!}{(n-r)!}$. We also introduce the notation $\mathbb{Z}_p^d[x]$ to represent a collection of polynomials of a degree d. For a given function f, we use the notation $f^{(k)}(x)$ to denote the k-th derivative function of f. For a simple description, we regard $f(x)$ with $f^{(0)}(x)$.

A. Commitment Scheme

A commitment scheme allows a party to commit to a chosen value while keeping it hidden, with the ability to reveal it later. Formally, a commitment scheme consists of two algorithms:

- $\mathsf{Commit}(v; r)$: Given a value v and randomness $r \xleftarrow{\$} \mathbb{Z}_p$, outputs a commitment C.
- $\mathsf{Open}(C, v, r)$: Outputs 1 if C was correctly generated from v and r, and 0 otherwise.

A commitment scheme satisfies two properties:

- *Hiding*: The commitment C reveals no information about v.
- *Binding*: A committed party cannot open C to a different value $v' \neq v$.

In this work, we use Pedersen commitments of the form $C = g^v$, which provide computational binding under the discrete logarithm assumption. A *polynomial commitment*

$\mathbf{C} = \{g^{a_k}\}_{k=0}^{d}$ commits to each coefficient a_k of a polynomial $f(x) = \sum_{k=0}^{d} a_k x^k$, allowing any party to verify the correctness of an evaluated share $f(i)$ against the published commitments without learning the polynomial itself.

B. Digital Signature

A digital signature scheme, denoted by DS, consists of four polynomial-time algorithms: DS.Setup, DS.KeyGen, DS.Sign, and DS.Vrf.

- $\mathsf{DS.Setup}(1^\lambda)$: The setup algorithm takes the security parameter λ as input and outputs the public parameters pp.
- $\mathsf{DS.KeyGen}(pp)$: The key generation algorithm takes the public parameters pp as input and outputs a key pair (sk, pk), where sk is the signing key and pk is the verification key.
- $\mathsf{DS.Sign}(sk, m)$: The signing algorithm takes a signing key sk and a message m as input, and outputs a signature σ.
- $\mathsf{DS.Vrf}(\sigma, m, pk)$: The verification algorithm takes a signature σ, a message m, and a verification key pk as input, and outputs 1 if the signature is valid, or 0 otherwise.

A digital signature scheme should satisfy correctness, which means that for any message m, if $(sk, pk) \leftarrow \mathsf{DS.KeyGen}(pp)$ and $\sigma \leftarrow \mathsf{DS.Sign}(sk, m)$, then $\mathsf{DS.Vrf}(\sigma, m, pk) = 1$ with overwhelming probability.

C. Threshold Signature

Let $U = \{1, ..., n\}$ be a set of n parties and P be a subset of U such that $|P| \geq t$. A (t, n)-threshold signature scheme, denoted by (t, n)-TS, consists of six algorithms: TS.Setup, TS.KeyGen, TS.pSign, TS.pVrf, TS.Agg, and TS.Vrf.

- $\mathsf{TS.Setup}(1^\lambda)$: The setup algorithm takes the security parameter λ as input and outputs the public parameters pp.
- $\mathsf{TS.KeyGen}(pp, t, n)$: The algorithm outputs partial signing key sk_i, partial verification key pk_i for each user i, and verification key pk.
- $\mathsf{TS.pSign}(sk_i, m, P)$: The algorithm outputs a partial signature σ_i for a subset P of U such that $|P| \geq t$.
- $\mathsf{TS.pVrf}(\sigma_i, pk_i, m, P)$: The algorithm outputs 1 if σ_i is valid, or outputs 0 otherwise.
- $\mathsf{TS.Agg}(\{\sigma_i\}_{i \in P})$: The algorithm outputs the aggregated signature σ.
- $\mathsf{TS.Vrf}(\sigma, m, pk)$: The algorithm outputs 1 if σ is valid, or outputs 0 otherwise.

1) Shamir Secret Sharing-based Threshold Signature (SSTS): We assume that the output of the DS.Sign is operable within the group G. Shamir secret sharing uses Lagrange interpolation to reconstruct the secret $f(0)$ from a polynomial $f(x)$ of degree $t-1$, given a set of participants P such that $t \leq |P|$. A Shamir secret sharing-based threshold signature scheme (SSTS) applies this method to digital signatures:

- $\mathsf{SSTS.KeyGen}(pp, t, n)$: Sets polynomial $f \in \mathbb{Z}_p^t[x]$ and outputs $(sk_i, pk_i) \leftarrow \mathsf{DS.KeyGen}(pp)$ such that $sk_i = f(x_i)$ for distinct $x_i \neq 0$, and group key pk such that $(sk, pk) \leftarrow \mathsf{DS.KeyGen}(pp)$ where $sk = f(0)$.
- $\mathsf{SSTS.pSign}(sk_i, m, P)$: Outputs partial signature $\sigma_i \leftarrow \mathsf{DS.Sign}(sk_i', m)$ with $sk_i' = sk_i \cdot \lambda_i$, where λ_i is the i-th Lagrange coefficient to the set P.
- $\mathsf{SSTS.Agg}(\{\sigma_i\}_{i \in P})$: Outputs aggregated signature $\sigma = \sum_i \sigma_i$.

Unforgeability A (t, n)-threshold signature is unforgeable if no adversary controlling fewer than t participants can forge valid signatures on new messages, formalized through a standard security game with adaptive corruption.

2) FROST (Flexible Round-Optimized Schnorr Threshold Signatures): FROST [17] is an efficient (t, n)-threshold signature scheme based on Schnorr signatures with two-round signing protocol. We provide a high-level overview of its key mechanisms:

Key Generation. FROST employs a distributed key generation (DKG) protocol based on Pedersen's scheme. Each participant i samples a random polynomial $f_i \in \mathbb{Z}_p^t[x]$ of degree $t-1$ and distributes shares $f_i(j)$ to other participants j. The participant's secret key is $sk_i = \sum_{j=1}^{n} f_j(i)$, and the group public key is $pk = g^{f(0)}$ where $f(x) = \sum_{i=1}^{n} f_i(x)$. Commitments $\mathbf{C}_i = \{C_{i,k} = g^{a_{i,k}}\}_{k=0}^{t-1}$ are broadcast for verification, where $f_i(x) = \sum_{k=0}^{t-1} a_{i,k} x^k$.

Signing Protocol. For an authorized set P with $|P| \geq t$, FROST executes a two-round signing process:

Round 1 (Preprocessing): Each participant i generates commitment pairs $(D_i, E_i) = (g^{d_i}, g^{e_i})$ for random $(d_i, e_i) \xleftarrow{\$} \mathbb{Z}_p \times \mathbb{Z}_p$ and publishes them. This round can be performed in advance, independent of the message.

Round 2 (Signing): Given message m and commitments from P, each participant i computes:

- Binding factor: $B = \{(i, D_i, E_i)\}_{i \in P}$ and $\rho_i = H_1(i, m, B)$
- Group commitment: $R = \prod_{i \in P} D_i \cdot E_i^{\rho_i}$
- Challenge: $c = H_2(R, m, pk)$
- Lagrange coefficient: $\lambda_i = \prod_{j \in P \setminus \{i\}} \frac{j}{j-i}$
- Partial signature: $\sigma_i = d_i + e_i \cdot \rho_i + \lambda_i \cdot sk_i \cdot c$

The aggregated signature is $\sigma = (R, \sum_{i \in P} \sigma_i)$, which can be verified using the standard Schnorr verification equation $g^{\sigma'} = R \cdot pk^c$.

Key Properties. FROST achieves: (1) Two-round online signing with preprocessing, (2) Signature compatibility with standard Schnorr signatures, (3) Security based on the discrete logarithm assumption and random oracle model [17]. However, FROST does not support weighted participants—each participant contributes equally regardless of their stake or authority.

D. Weighted Threshold Signature

In weighted threshold schemes, each user i has weight w_i, and a subset P is authorized if $\sum_{i \in P} w_i \geq t$. A (t, n)-weighted threshold signature scheme (WS) follows the same structure as threshold signatures but with weight-based authorization.

Virtualization Approach SSTS can be extended to a Shamir Secret sharing-based Weighted Signature scheme (SSWS), a

weighted variant of SSTS, using virtualization, where each user i with weight w_i controls w_i separate key pairs: $sk_i = (sk'_{i,0}, ..., sk'_{i,w_i-1})$ and $pk_i = (pk'_{i,0}, ..., pk'_{i,w_i-1})$, requiring w_i separate signing operations. This approach creates computational overhead proportional to assigned weights.

IV. Active Weighted Signature

We formally define an Active Weighted Signature (AWS) scheme, which extends standard weighted threshold signatures with dynamic weight and threshold adjustment capabilities to address the re-setup limitations discussed in Section I.

An AWS scheme requires: a set of participants $U = \{1, ..., n\}$ with public indices; weight assignment w_i for each participant i; threshold value t where $1 \leq t \leq \sum_{i=1}^{n} w_i$; and authorized participant sets $P \subset U$ where $\sum_{i \in P} w_i \geq t$.

An active weighted signature scheme (t, n)-AWS consists of the standard six algorithms (AWS.Setup, AWS.KeyGen, AWS.pSign, AWS.pVrf, AWS.Agg, AWS.Vrf) extending weighted threshold signatures, plus four dynamic functionality algorithms:

- AWS.WIncrease$(\{sk_i, pk_i\}_{i \in P}, j, \alpha)$: Increases user j's weight from w_j to $w_j + \alpha$ using authorized participants P.
- AWS.TIncrease$(\{sk_i, pk_i\}_{i \in P}, \alpha)$: Increases global threshold from t to $t+\alpha$ while maintaining user weights.
- AWS.WDecrease$(\{sk_i, pk_i\}_{i \in P}, j, \alpha)$: Decreases user j's weight from w_j to $w_j - \alpha$.
- AWS.TDecrease$(\{sk_i, pk_i\}_{i \in P}, \alpha)$: Decreases global threshold from t to $t-\alpha$ while maintaining user weights.

Correctness For any P such that $\sum_{i \in P} w_i \geq t$, an honestly generated aggregated signature (Resp. partial signature) should pass the verification (Resp. partial verification). Formally, it means that

$$\Pr[\mathsf{AWS.pVrf}(\mathsf{AWS.pSign}(sk_i, m, P), pk_i, m) = 1] = 1$$
$$\Pr[\mathsf{AWS.Vrf}(\mathsf{AWS.Agg}(\{\sigma_i\}_{i \in P}), m, pk) = 1] = 1$$

Unforgeability In our scheme, the unforgeability is considered under the condition that both weights and the threshold are fixed, aligning it with the unforgeability typically associated with general weighted signature schemes. This involves defining the unforgeability game against an adaptive and rushing adversary. 'Adaptive' implies that the adversary has the capability to corrupt signers during the protocol, and 'rushing' indicates that the adversary can choose its messages based on the messages from honest signers.

The unforgeability of a weighted signature scheme then consists of a three-stage game:

- Setup: The challenger $\mathcal{C}$ generates the public parameter $pp, \{w_i\}_{i \in U}$ and sends it to an adversary $\mathcal{A}$. Since we consider an adaptive adversary, we assume that the adversary initially corrupts a (strictly) subset $F \subset U$ of signers with weight $w_F := \sum_{i \in F} w_i < t$ when the challenger $\mathcal{C}$ conducts the $\mathsf{AWS.KeyGen}(pp, t, n, \{w_i\}_{i \in U})$ algorithm. Under the corruption, an adversary $\mathcal{A}$ gets a a challenge key pair $(\{sk_i\}_{i \in F}, \{pk_i\}_{i \in F})$.
- Queries: $\mathcal{A}$ is allowed to make arbitrarily many partial signatures on messages of its choice.
- Output: Finally, the adversary outputs a message signature forgery pair (σ, m^*). This forgery wins if $\mathsf{AWS.Vrf}(\sigma, m^*, pk) = 1$ and $\mathcal{A}$ has queried partial signatures of weight less than $t - w_F$ on the message m^*.

Definition IV.1. *We say $\mathcal{A}$ is a (τ, q, ϵ)-forger for (t, n)-weighted signature scheme if it runs in time τ, makes q queries, and wins the above game with probability at least ϵ with $t-1$ weight-corruption. (t, n)-AWS is (τ, q, ϵ)-unforgeable if no (τ, q, ϵ)-forger exists.*

Regarding the broader security aspects of this scheme, we can extend our considerations to include an active adversary. In this context, 'active' goes beyond merely making queries for partial signatures during the query phase. An adversary is considered active if it has the following capabilities:

- Active Queries: $\mathcal{A}$ is allowed to 1) make arbitrarily many partial signatures on messages of its choice and 2) performs the extra functionalities until an updated weight and threshold hold a constraint $t > w_F$.

Definition IV.2. *We say $\mathcal{A}$ is an active (τ, q, ϵ)-forger for (t, n)-weighted signature scheme if it runs in time τ, makes q active queries, and wins the above game with probability at least ϵ with $t-1$ weight-corruption. (t, n)-AWS is (τ, q, ϵ)-unforgeable if no active (τ, q, ϵ)-forger exists.*

Remark. While our general construction in Section VI only ensures Definition IV.1, we contend that developing a broader framework that secures Definition IV.2 represents a compelling and significant open challenge.

The difficulty stems from active adversaries being able to strategically manipulate weights and thresholds to approach security boundaries, potentially correlating information across multiple dynamic operations to extract partial polynomial structure, and the complexity of maintaining cryptographic consistency while preventing information leakage during concurrent operations. Traditional security reduction techniques relying on fixed corruption thresholds may not directly apply when adversaries can dynamically adjust the very parameters defining the security game.

V. Generalized Lagrange Interpolation

Lagrange interpolation is a mathematical technique used to construct a polynomial that passes through a given set of points. Assume that, there are given n points in the form $(x_1, y_1), \cdots, (x_n, y_n)$. The goal of Lagrange interpolation is to find a polynomial $F(x)$ of degree $n-1$ that passes through all the given points. The polynomial $F(x)$ can be constructed as:

$$F(x) = \sum_{i}^{n} \delta_i(x) F(x_i),$$

where $\delta_i(x)$ is the Lagrange basis polynomial corresponding to point (x_i, y_i), which is defined as $\delta_i(x) = \prod_{j \neq i}^{n} \frac{x - x_j}{x_i - x_j}$.

The Lagrange basis polynomials have the property that $\delta_i(x_i) = 1$ and $\delta_i(x_j) = 0$ for all $j \neq i$. This means that

when we evaluate the polynomial $F(x)$ at any of the given points (x_i, y_i), we get:

$$F(x_i) = \delta_1(x_i)y_1 + \cdots + \delta_n(x_i)y_n = y_i$$

which means that the polynomial $F(x)$ passes through all the given points.

While the original Lagrange interpolation demonstrates that an $(n-1)$-dimensional polynomial F can be recovered when given a data set $\{(x_i, F(x_i))\}_{i\in[n]}$ for n distinct inputs x_i, such a polynomial can be still recovered if a data set is given for a derivative function $(x_i, F^{(m)}(x_i))_{i\in[n]}$ (including at least one point of the form $(x_i, F(x_i))$). Remarkably, it does not require the constraint $x_i \neq x_j$. This implies that several data can be linked to one input x_i. We thus consider a data set $\mathcal{S} = \{(x_i, \{F^{(m)}(x_i)\}_{0\leq m<w_i})\}_{i\in[n]}$ for a simple description. In this section, we provide an algorithm to determine a polynomial F when $\mathcal{S}$ is given. Under this definition, the Lagrange interpolation algorithm can be regarded with the case $w_i = 1$ for all index i. We therefore refer to this algorithm as a Generalized Lagrange Interpolation (GLI).

Remark on Notation. In the GLI algorithm, we use i, j as indices for the data points, while m consistently denotes the order of differentiation throughout this section.

The formal theorem is stated as:

Theorem 1. *Given a data set $\mathcal{S}$ of the form $\{(x_i, \{F^{(m)}(x_i)\}_{0\leq m<w_i})\}_{i\in[n]}$, where x_i are distinct real numbers, there exists a unique polynomial $G(x)$ of degree at most $\sum_{i=1}^{n} w_i - 1$ that passes through all these points. Even more, this polynomial $G(x)$ can be recovered in polynomial time in n, w.*

Proof. The key idea of this algorithm is to consider a polynomial $G(x)$ as a summation of the form:

$$G(x) = \sum_{i=1}^{n} \delta_i(x) \cdot g_i(x), \text{ where } \delta_i(x) = \prod_{j\neq i} \left(\frac{x - x_j}{x_i - x_j}\right)^{w_j}$$
$$g_i(x) = \sum_{k=0}^{w_i-1} c_{i,k} \cdot (x - x_i)^k$$

We then set the coefficient $c_{i,k}$ so that

$$(\delta_i \cdot g_i)^{(y)}(x) = \begin{cases} F^{(m)}(x_i) & x = x_i \wedge y = m \\ 0 & x \neq x_i \end{cases}. \quad (1)$$

This property immediately implies that $G^{(m)}(x_i) = F^{(m)}(x_i)$ when $m < w_i$.

We next describe how to compute the polynomial coefficients $c_{i,k}$ recursively. First, when $m = 0$, it holds that:

$$(\delta_i \cdot g_i)(x_i) = \prod_{j\neq i} \left(\frac{x_i - x_j}{x_i - x_j}\right)^{w_j} \cdot g_i(x_i) = c_{i,0}.$$

Thus, we set $c_{i,0}$ to $F(x_i)$. Next, suppose that $\{c_{i,k} = g_i^{(k)}(x_i)/k!\}_{k<y}$ for some y is obtained. By the multiplication formula of differentiation, it holds that:

$$(\delta_i \cdot g_i)^{(y)}(x_i) = \sum_{k=0}^{y} \binom{y}{k} \delta_i^{(y-k)}(x_i) \cdot g_i^{(k)}(x_i)$$
$$= g_i^{(y)}(x_i) + \sum_{k=0}^{y-1} \binom{y}{k} \delta_i^{(y-k)}(x_i) \cdot c_{i,k} \cdot k!.$$

According to this identity, $g_i^{(y)}(x_i) = y! \cdot c_{i,y}$ can be computed as $F^{(y)}(x_i) - \sum_{k=0}^{y-1} \binom{y}{k} \delta_i^{(y-k)}(x_i) \cdot c_{i,k} \cdot k!$. Hence, this computation gives a coefficient $c_{i,k}$ inductively for all $k < w_i$. In summary, one can recover the polynomial $(\delta_i \cdot g_i)(x)$ efficiently. This completes the proof. □

For a simple description in the next section, we define a function:

Definition 1. *For a polynomial $F(x)$ of degree $n-1$, let $P = \{x_1, \ldots, x_k\}$ be a set with a weight $\{w_i\}$ such that $\sum_{i\in P} w_i \geq n$. We define $GLI(P, x_i)$ as $\delta_i(0) \cdot g_i(0)$, where $\delta_i(x) = \prod_{j\neq i} \left(\frac{x-x_j}{x_i-x_j}\right)^{w_j}$, and $g_i(x) = \sum_{k=0}^{w_i-1} c_{i,k} \cdot (x - x_i)^k$ for $c_{i,k}$ computed in eq. (1).*

A Simple Example of GLI. To illustrate the GLI algorithm, consider reconstructing a polynomial from the following data:

$$F(1) = 4, \quad F'(1) = 4, \quad F(2) = 9$$

We have $n = 2$ distinct inputs with weights $w_1 = 2$ and $w_2 = 1$. For $x_1 = 1$:

$$\delta_1(x) = \left(\frac{x-2}{1-2}\right)^1 = -x + 2, \quad g_1(x) = c_{1,0} + c_{1,1}(x-1)$$

For $x_2 = 2$:

$$\delta_2(x) = \left(\frac{x-1}{2-1}\right)^2 = (x-1)^2, \quad g_2(x) = c_{2,0}$$

Computing the coefficients:

$$c_{1,0} = F(1) = 4$$
$$c_{1,1} = \frac{F'(1) - \delta_1'(1) \cdot c_{1,0}}{1!} = \frac{4 - (-1) \cdot 4}{1} = 8$$
$$c_{2,0} = F(2) = 9$$

The reconstructed polynomial is:

$$\begin{aligned} G(x) &= \delta_1(x) \cdot g_1(x) + \delta_2(x) \cdot g_2(x) \\ &= (-x+2) \cdot [4 + 8(x-1)] + (x-1)^2 \cdot 9 \\ &= x^2 + 2x + 1 \end{aligned}$$

Verification confirms $G(1) = 4$, $G'(1) = 4$, and $G(2) = 9$, demonstrating successful polynomial reconstruction via GLI.

Comparison to Hermite Interpolation. While Hermite interpolation recovers the same polynomial from identical input data, it fundamentally differs from GLI in computational structure. Hermite interpolation constructs the final polynomial by recursively combining intermediate results in a tree-like structure, requiring interactions among users to merge partial information. In contrast, GLI enables each user to independently compute $\delta_i(x) \cdot g_i(x)$ without interaction, as the target polynomial is simply $\sum_i \delta_i(x) \cdot g_i(x)$.

This independence property makes GLI far more suitable for threshold signature schemes in blockchain environments, where minimizing communication overhead and maximizing decentralization are crucial. Such interactions required by

Hermite interpolation impose significant communication costs and are often impractical in distributed settings. Therefore, while both methods are mathematically valid, GLI's non-interactive nature is essential for secret sharing applications, distinguishing it as the appropriate choice for our AWS construction.

VI. ADAPT: GLI-based Active Weighted Signature Scheme

A. Overview: From FROST to ADAPT

We propose ADAPT, a concrete instantiation of the AWS framework (Section IV) by applying Generalized Lagrange Interpolation to the FROST threshold signature scheme. FROST provides an efficient foundation with two-round signing, proven security under the discrete logarithm assumption, and Schnorr signature compatibility. However, FROST supports only equal-weight participants and requires complete re-setup for any system modification.

ADAPT extends FROST to support weighted participants with dynamic adjustments through three key modifications:

Derivative-based Share Representation. In FROST, each participant i stores a single share $sk_i = f(i)$ where $f(x)$ is the shared polynomial. ADAPT extends this to derivative shares: participant i with weight w_i stores $\{sk_i^{(k)} = f^{(k)}(i)\}_{k=0}^{w_i-1}$, representing the function value and its first (w_i-1) derivatives at point i. This eliminates virtualization—instead of creating w_i separate virtual participants, a single participant holds w_i derivative values at one evaluation point.

GLI-based Signing Coefficient. FROST computes partial signatures using Lagrange coefficients $\lambda_i = \prod_{j\in P\setminus\{i\}} \frac{j}{j-i}$ assuming equal weights. ADAPT replaces this with GLI coefficients $\text{GLI}(P,i) = \delta_i(0)\cdot g_i(0)$ where $\delta_i(x) = \prod_{j\in P\setminus\{i\}} \left(\frac{x-j}{i-j}\right)^{w_j}$ encodes other participants' weights, and $g_i(x)$ is constructed via GLI (Definition 1) to satisfy $(\delta_i \cdot g_i)^{(k)}(i) = sk_i^{(k)}$ for all $k < w_i$.

Dynamic Operations via Polynomial Manipulation. FROST requires complete re-setup for modifications. ADAPT enables:

- *Weight increase*: Compute additional derivative $f^{(w_i)}(i)$ without modifying $f(x)$ itself
- *Threshold increase*: Transform $f(x) \to f(x)\cdot h(x)$ where each participant updates shares using Leibniz rule
- *Weight/Threshold decrease*: Apply inverse transformations through differentiation or coefficient removal

These modifications maintain FROST's efficiency and security properties while adding weighted and dynamic capabilities. Table II shows that dynamic operations require significantly less computation than re-setup (Section VIII).

B. Formal Scheme Description

- $\mathsf{ADAPT.Setup}(1^\lambda)$: Select group G of order p with λ-bit security. Set $pp = \{p, g\}$. Each user in $U = \{1,\dots,n\}$ knows all participants' weights $\{w_i\}_{i\in U}$.
- $\mathsf{ADAPT.KeyGen}(pp, t, n)$:

  Round 1: Each user i picks random polynomial $f_i \xleftarrow{\$} \mathbb{Z}_p^t[x]$ of degree $t-1$, computes commitments $\mathbf{C}_i = \{C_{i,k} = g^{a_{i,k}}\}_{k=0}^{t-1}$ for $f_i(x) = \sum_{k=0}^{t-1} a_{i,k}x^k$, generates proof of knowledge π_i for $f_i(0)$, and broadcasts $(\pi_i, \mathbf{C}_i)$ with verification.

  Round 2: Each user i computes derivative shares $\mathbf{f}_i(j) = \{f_i^{(m)}(j)\}_{m=0}^{w_j-1}$ and securely sends to user j. After verification against commitments, each user computes partial secrets $sk_i^{(m)} = \sum_{k=1}^n f_k^{(m)}(i)$ for $m \in [0, w_i)$, and derives keys $pk_i = g^{sk_i^{(0)}}$ and $pk = \prod_{i=1}^n C_{i,0}$.

  Preprocessing: Each user generates commitment pairs $(D_{i,j}, E_{i,j}) = (g^{d_{i,j}}, g^{e_{i,j}})$ for random nonces and publishes them.
- $\mathsf{ADAPT.pSign}(sk_i, m, P)$: User i constructs binding factor $B = \{(j, D_j, E_j)\}_{j\in P}$, computes $\rho_i = H_1(i, m, B)$, group commitment $R = \prod_{j\in P} D_j \cdot E_j^{\rho_j}$, and challenge $c = H_2(R, m, pk)$. Construct GLI polynomial $F_i(x) = \sum_{k=0}^{w_i-1} b_{i,k}x^k$ via GLI algorithm such that:

$$(\delta_i \cdot F_i)^{(k)}(x) = \begin{cases} sk_i^{(k)} & \text{if } x = i \\ 0 & \text{if } x \in P \setminus \{i\} \end{cases}$$

  Compute partial signature $\sigma_i = d_i + e_i\cdot\rho_i + (\delta_i\cdot F_i)(0)\cdot c$ and send with commitment $\mathbf{C}_{i,sig} = \{g^{b_{i,k}}\}$.
- $\mathsf{ADAPT.pVrf}(\sigma_i, \mathbf{C}_{i,sig}, pk_i, m, P)$: Verify $pk_i \stackrel{?}{=} \prod_{k=0}^{w_i-1} C'^{\,i^k}_{i,k}$ and $g^{\sigma_i} \stackrel{?}{=} R_i \cdot C'^{\,c\cdot\delta_i(0)}_{i,0}$.
- $\mathsf{ADAPT.Agg}(\{\sigma_i\}_{i\in P})$: Output $\sigma = (R, \sum_{i\in P}\sigma_i)$.
- $\mathsf{ADAPT.Vrf}(\sigma, m, pk)$: Validate $g^{\sigma'} \stackrel{?}{=} R\cdot pk^c$ where $\sigma = (R, \sigma')$ and $c = H_2(R, m, pk)$.

Correctness. GLI ensures $(\delta_i\cdot F_i)(i) = sk_i^{(0)}$, giving $pk_i = g^{sk_i^{(0)}}$. Partial verification succeeds by the Schnorr structure. For aggregation, GLI guarantees $f(x) = \sum_{i\in P}(\delta_i\cdot F_i)(x)$ where $f(x) = \sum_{i=1}^n f_i(x)$, so $pk = g^{f(0)} = g^{\sum_{i\in P}(\delta_i\cdot F_i)(0)}$, yielding $g^{\sigma'} = R\cdot pk^c$.

C. Active Functionality

We describe the four dynamic operations of ADAPT. All operations require authorized participants P with $\sum_{i\in P} w_i \geq t$ and use preprocessing commitments (D_i, E_i) with additional hash functions H_3, H_4.

- $\mathsf{ADAPT.WIncrease}(\{sk_i, pk_i\}_{i\in P}, j, \alpha)$: To increase user j's weight by α, each $i \in P$ constructs GLI polynomial $F_i(x)$ encoding their secrets, computes additional derivatives $\mu_{i,k} = (\delta_i\cdot F_i)^{(w_j+k)}(j)$ for $k \in [0,\alpha)$, generates proof $\hat{s}_i = \hat{c}_i\cdot sk_i^{(0)} + \hat{r}_i$ with $\hat{c}_i = H_3(\mathbf{C}_{i,Win}, \{\mu_{i,k}\}, \hat{R}_i)$, and sends to j. After verification, j updates $sk_j^{(w_j+k)} = \sum_{i\in P}\mu_{i,k}$.
- $\mathsf{ADAPT.TIncrease}(\{sk_i, pk_i\}_{i\in P}, \alpha)$: To increase threshold by α, each $i \in P$ chooses polynomial $h_i(x) \in \mathbb{Z}_p^{\alpha+1}[x]$, computes transformation $\mu_{i,\ell} = (\delta_i\cdot F_i)(\ell)\cdot h_i(\ell) + d_i + e_i\cdot\rho_i$, and distributes to all users $\ell \in U$. Each user updates keys using Leibniz rule: $sk_\ell^{(0)} = \sum_{i\in P}\mu_{i,\ell}$ and $sk_\ell^{(k)} = \sum_{t=0}^k \binom{k}{t} sk_\ell^{(t)} h^{(k-t)}(\ell)$. Global key becomes $pk = R$.
- $\mathsf{ADAPT.WDecrease}(\{sk_i, pk_i\}_{i\in P}, j, \alpha)$: To decrease user j's weight by α, each $i \in P$ chooses masking polynomial $h_i(x)$ with $h_i(0) = 0$, distributes appropriate

Algorithm	ADAPT Complexity	FROST Complexity
KeyGen	$\mathcal{O}(n^2)\cdot \mathsf{T}_e + \mathcal{O}(n^2)\cdot \mathsf{T}_d$	$\mathcal{O}(n^2)\cdot \mathsf{T}_e$
Sign	$\mathcal{O}(m^2)\cdot \mathsf{T}_e + \mathcal{O}(t)\cdot \mathsf{T}_d$	$\mathcal{O}(t^2)\cdot \mathsf{T}_e$
WIncrease	$\mathcal{O}(m^2)\cdot \mathsf{T}_e + \mathcal{O}(t)\cdot \mathsf{T}_d$	$\mathcal{O}(m^2)\cdot \mathsf{T}_e$ (virtualization) $\mathcal{O}(n^2)\cdot \mathsf{T}_e$ (re-setup)
TIncrease	$\mathcal{O}(n)\cdot \mathsf{T}_e + \mathcal{O}(t)\cdot \mathsf{T}_d$	$\mathcal{O}(n^2)\cdot \mathsf{T}_e$
WDecrease	$\mathcal{O}(t\cdot n)\cdot \mathsf{T}_e + \mathcal{O}(n)\cdot \mathsf{T}_d$	$\mathcal{O}(n^2)\cdot \mathsf{T}_e$
TDecrease	$\mathcal{O}(n)\cdot \mathsf{T}_e + \mathcal{O}(n)\cdot \mathsf{T}_p + \mathcal{O}(t)\cdot \mathsf{T}_d$	$\mathcal{O}(n^2)\cdot \mathsf{T}_e$

TABLE II
COMPLEXITY COMPARISON OF ADAPT AND FROST. n DENOTES THE NUMBER OF PARTICIPANTS, m DENOTES THE CARDINALITY OF AUTHORIZED PARTY P, t DENOTES THE THRESHOLD, T_e DENOTES A TIME TO COMPUTE AN EXPONENTIAL OPERATION, T_p DENOTES A TIME TO COMPUTE A PERMUTATION OPERATION, AND T_d DENOTES A TIME TO COMPUTE A DIFFERENTIAL OPERATION. FOR FROST, OPERATIONS OTHER THAN Sign AND WIncrease (VIRTUALIZATION) REQUIRE FULL RE-SETUP EQUIVALENT TO KeyGen. $m \le t \le n$; WHEN WEIGHTS ARE UNIFORM $m \approx t$, AND WHEN WEIGHTS ARE SKEWED $m \ll t$.

derivative information (reduced for j, full for others), and all users update: $sk_\ell^{(k)} := sk_\ell^{(k)} + \sum_{i\in P} h_i^{(k)}(\ell)$ with appropriate bounds.

- ADAPT.TDecrease$(\{sk_i, pk_i\}_{i\in P}, \alpha)$: To decrease threshold by α, each $i \in P$ computes high-order derivatives $\mu_{i,k} = (\delta_i \cdot F_i)^{(t-k)}(0)$ for $k \in [1,\alpha]$, sends to aggregator who computes coefficients $f_{t-k} = \frac{\sum_{i\in P}\mu_{i,k}}{(t-k)!}$ and broadcasts. All users update: $sk_i^{(j)} := sk_i^{(j)} - \sum_{k=1}^{\alpha} f_{t-k}\cdot(\mathbf{Perm}(t,j)\cdot i^{(t-k-j)})$.

We then assess the time complexity of KeyGen and the functionalities within ADAPT concerning the number of participants n, and the threshold t. The results are summarized in Table II. Let m be a cardinality of authorized party P, T_e be a time to compute exponential operation, t_p be a time to compute permutation operation, and t_d be a time to compute differential operation.

We have previously established that t, m, and n satisfy the following conditions: $m \le t \le n$.

KeyGen is executed by all nodes participating in the system. Thus it exhibits a time complexity proportional to n^2. On the other hand, all other functionality algorithms require less then n^2 operations, resulting in less complexity compared to KeyGen. In summary, our functionality executes significantly faster in comparison to KeyGen. The complexity of Sign is measured based on the full signature operation. In other words, it is an integrated algorithm that includes pSign, pVrf, and Agg.

VII. SECURITY ANALYSIS

We first establish that the GLI-based secret sharing maintains equivalent security to traditional approaches. Then, We present a security analysis for ADAPT.

Lemma 1 (GLI Structure Security)**.** *The GLI-based secret sharing used in ADAPT provides equivalent security to traditional Shamir secret sharing, with the additional property that derivative information does not leak the original secret.*

Proof. For a participant i with weight w_i, the secret shares $\{f^{(k)}(i)\}_{k=0}^{w_i-1}$ are related by $f^{(k)}(i) = \sum_{j=k}^{t-1}\binom{j}{k}\cdot a_j \cdot i^{j-k}$. The key observation is that higher-order derivatives provide independent information about the polynomial coefficients. Specifically, the matrix formed by function values and their derivatives has full rank t:

$$\operatorname{rank}\begin{pmatrix} 1 & i & i^2 & \cdots & i^{t-1} \\ 0 & 1 & 2i & \cdots & (t-1)i^{t-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & (t-1)! \end{pmatrix} = t \tag{2}$$

This ensures that t total derivative evaluations (counting multiplicities) are needed to reconstruct the secret, maintaining the threshold property. The GLI-based approach is therefore equivalent to traditional Shamir secret sharing in terms of information-theoretic security, threshold properties, and reconstruction capabilities, while providing more efficient weight representation without compromising security. □

Theorem 2 (Unforgeability of ADAPT)**.** *If the discrete logarithm problem in $\mathbb{G}$ is (t',ϵ')-hard, then the ADAPT signature scheme over $\mathbb{G}$ with n signing participants, a threshold of t, and maximum weight $w_{\max}$ is $(t, n_h, n_p, n_s, \epsilon)$-secure whenever*

$$\epsilon' \le \frac{\epsilon^2}{2n_h + (w_{\max}+1)n_p + 1}$$

and

$$t' = 4t + (30w_{\max}n_p + (4t-2)n_s + (n+t-1)t + 6)\cdot t_{exp} + O(w_{\max}n_p + n_s + n_h + 1)$$

where t_{exp} denotes the computational time required to perform a single exponentiation operation in the group $\mathbb{G}$. The parameter n_h represents the total number of hash queries that the adversary is permitted to make during the security game. The value n_p corresponds to the number of preprocessing queries allowed, while n_s indicates the number of signing queries that can be issued. Furthermore, the security analysis requires that the total weight controlled by the adversary remains strictly below the threshold value t.

Proof. We prove the theorem by contradiction using the generalized forking lemma. Assume there exists a forger $\mathcal{F}$ that $(t, n_h, n_p, n_s, \epsilon)$-breaks ADAPT. We construct an algorithm $\mathcal{C}$ that uses $\mathcal{F}$ to solve the discrete logarithm problem in $\mathbb{G}$.

Setup Phase. Given a DLP challenge $\omega \in \mathbb{G}$, algorithm $\mathcal{C}$ embeds ω into the ADAPT key generation. Let the adversary $\mathcal{F}$ control a set of participants $F \subset U$ with total weight $w_F = \sum_{i\in F} w_i < t$. Without loss of generality, we assume the honest participant $P_t \notin F$ with weight w_t.

To simulate the key generation, $\mathcal{C}$ sets $\phi_{t,0} = \omega$ as the base commitment for P_t. For the derivative shares, $\mathcal{C}$ samples random values $\bar{x}_{t,i}^{(k)} \xleftarrow{\$} \mathbb{Z}_p$ for each $i \in \{1,\dots,t-1\}$ and $k \in \{0,1,\dots,w_t-1\}$. The polynomial commitments are computed as:

$$\phi_{t,j} = \omega^{\lambda_{j,0}} \cdot g^{\sum_{i=1}^{t-1}\sum_{k=0}^{w_i-1}\lambda_{j,i}^{(k)}\cdot\bar{x}_{t,i}^{(k)}}$$

where $\lambda_{j,i}^{(k)}$ denotes the GLI coefficient for the k-th derivative at point i when interpolating over the polynomial of degree $t-1$. The public key share for P_t is computed as

$Y_t = \prod_{j=1}^{n} \prod_{k=0}^{t-1} \phi_{j,k}^{t^k \bmod q}$, and the group public key is $Y = \prod_{j=1}^{n} \phi_{j,0}$.

Simulation of Preprocessing and Signing. Algorithm $\mathcal{C}$ constructs a simulator $\mathcal{A}$ that responds to $\mathcal{F}$'s queries. For the preprocessing phase, when generating the j-th commitment pair for P_t, the simulator selects the next unused random oracle output $\bar{c}_j = h_{ctr}$ and samples $\bar{z}_{t,j} \xleftarrow{\$} \mathbb{Z}_p$. It then computes $D_{t,j} = g^{\bar{z}_{t,j}} \cdot Y_t^{-\bar{c}_j}$ and generates $E_{t,j} = g^{e_{t,j}}$ with $e_{t,j} \xleftarrow{\$} \mathbb{Z}_p^*$.

For a signing query on message m with authorized participant set P satisfying $\sum_{i \in P} w_i \geq t$, the simulator computes the binding values $\rho_i = H_1(i, m, B)$ for all $i \in P$, where $B = \{(i, D_i, E_i)\}_{i \in P}$ is the ordered list of commitments and $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$ is the hash function specified in ADAPT. The group commitment is computed as $R = \prod_{i \in P} D_i \cdot E_i^{\rho_i}$.

Since $\mathcal{A}$ knows all binding values before $\mathcal{F}$, it can compute R in advance and program the random oracle $H_2 : \{0,1\}^* \to \mathbb{Z}_p^*$ such that $H_2(R, Y, m) = \bar{c}_j$. For the honest participant P_t, the simulator constructs a GLI polynomial $F_t(x) = \sum_{k=0}^{w_t - 1} b_{t,k} x^k$ satisfying the constraint:

$$(\delta_t \cdot F_t)^{(k)}(t) = sk_t^{(k)}, \quad \forall k \in \{0, 1, \ldots, w_t - 1\}$$

where $\delta_t(x) = \prod_{j \in P \setminus \{t\}} \left(\frac{x-j}{t-j}\right)^{w_j}$ is the GLI basis polynomial. The partial signature is then $z_t = \bar{z}_{t,j} + e_{t,j} \cdot \rho_t$.

Application of the Generalized Forking Algorithm. The coordinator $\mathcal{C}$ applies the generalized forking algorithm (GFA) to obtain two related forgeries. In the first execution, GFA runs $\mathcal{A}$ with random tape β and random oracle outputs $\{h_1, \ldots, h_{n_r}\}$ where $n_r = 2n_h + (w_{\max} + 1)n_p + 1$. When $\mathcal{F}$ outputs a forgery $\sigma = (R, z)$ on message m, let J be the index such that $h_J = H_2(R, m, Y)$.

GFA then reruns $\mathcal{A}$ with the same random tape β but with fresh random oracle outputs $\{h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_{n_r}\}$ where $h'_i \xleftarrow{\$} \mathbb{Z}_p^*$ for $i \geq J$. If the second execution produces a forgery $\sigma' = (R, z')$ with the same R but $h'_J \neq h_J$, then GFA outputs both forgeries.

The key insight is that due to the same random tape up to the fork point, both executions produce the same commitment R. The signatures satisfy:

$$z = \sum_{i \in P} (d_i + e_i \cdot \rho_i + \text{GLI}(P, sk_i) \cdot c)$$
$$z' = \sum_{i \in P} (d_i + e_i \cdot \rho_i + \text{GLI}(P, sk_i) \cdot c')$$

where $\text{GLI}(P, sk_i) = \sum_{k=0}^{w_i - 1} \text{GLI}_k(P, i) \cdot sk_i^{(k)}$ encapsulates all derivative information for participant i.

Extraction of the Discrete Logarithm. From the verification equations $g^z = R \cdot Y^c$ and $g^{z'} = R \cdot Y^{c'}$, we derive:

$$g^{z - z'} = Y^{c - c'}$$

which yields the discrete logarithm of the group public key:

$$s = \text{dlog}(Y) = \frac{z' - z}{c' - c} \bmod p$$

Since $Y = \prod_{i=1}^{n} \phi_{i,0} = \omega \cdot \prod_{i \neq t} g^{a_{i,0}}$, and $\mathcal{C}$ extracted the values $\{a_{i,0}\}_{i \neq t}$ from the zero-knowledge proofs during key generation, the discrete logarithm of the challenge is:

$$\text{dlog}(\omega) = s - \sum_{i \neq t} a_{i,0} \bmod p$$

Analysis of Success Probability and Running Time. By the generalized forking lemma, if $\mathcal{F}$ succeeds with probability ϵ, then GFA produces two valid forgeries with probability at least ϵ^2 / n_r. The factor $(w_{\max} + 1)$ in n_r accounts for the maximum number of derivatives each participant can contribute during preprocessing.

The running time t' includes four executions of the forger (due to the double forking), GLI computations requiring $O(w_{\max} \cdot t)$ exponentiations per signing operation, and additional overhead from preprocessing and key generation simulation. This completes the proof. □

Lemma 2 (Dynamic Operation Security)**.** *Assume the discrete logarithm problem is hard with negligible advantage ϵ_{DLP}. Then for any ADAPT instance $\mathcal{I}$ with threshold t and adversary controlling weight $w_F < t$, any dynamic operation $\mathcal{O} \in \{\text{WIncrease}, \text{TIncrease}, \text{WDecrease}, \text{TDecrease}\}$ performed by authorized set P with $\sum_{i \in P} w_i \geq t$ produces an instance $\mathcal{I}'$ that is secure with advantage bound $\epsilon' = \mathcal{O}(\sqrt{\epsilon_{DLP} \cdot n'_r})$, where $n'_r = 2n_h + (w_{\max} + 1)(n_p + 1) + 1$. Here, n_h, $n_p + 1$, and n_s denote the number of hash queries, preprocessing queries, and signing queries respectively made by an adversary attacking $\mathcal{I}'$.*

Proof. Let $\mathcal{I}$ and $\mathcal{I}'$ denote the pre- and post-operation instances with thresholds t and t' respectively, where the adversary controls weights $w_F < t$ and $w'_F < t'$. We prove security of $\mathcal{I}'$ via reduction to the discrete logarithm problem. Suppose for contradiction that there exists a forger $\mathcal{F}'$ that breaks $\mathcal{I}'$ with success probability ϵ' in time τ' making n_h hash queries, $n_p + 1$ preprocessing queries, and n_s signing queries. We construct an algorithm $\mathcal{C}$ that uses $\mathcal{F}'$ to solve a DLP challenge $\omega \in \mathbb{G}$.

Challenge Embedding and Operation Participation. The constraint $w_F < t$ in the pre-operation instance guarantees the existence of at least one honest participant P_t with weight w_t. Following the embedding strategy of Theorem 2, algorithm $\mathcal{C}$ sets $\phi_{t,0} = \omega$ as P_t's base commitment and samples random values $\{\bar{x}_{t,i}^{(k)}\}$ for the remaining polynomial coefficients. This embedding allows $\mathcal{C}$ to simulate all of $\mathcal{F}'$'s queries before operation $\mathcal{O}$ is invoked.

When $\mathcal{O}$ is executed by authorized set P satisfying $\sum_{i \in P} w_i \geq t$, the weight constraint ensures P contains at least one honest participant. The simulator participates as P_t, which enables it to control P_t's polynomial contribution to the transformation and record transformation parameters for extraction. Specifically, in threshold increase, the simulator chooses $h_t(x)$ and records both $h_t(0)$ and the aggregate $h(0) = \sum_{i \in P} h_i(0)$ from protocol messages. In weight decrease, it observes the masking values $\{h_j(0)\}_{j \in P \setminus \{t\}}$.

Extraction via Forking. We extend the generalized forking lemma [5] to the dynamic operation setting by adjusting the

query parameter to $n'_r = 2n_h + (w_{\max}+1)(n_p+1)+1$, where the additional $(w_{\max}+1)$ term accounts for preprocessing queries during operation $\mathcal{O}$. Applying this to random oracle H_2 yields two valid forgeries $\sigma = (R, z)$ and $\sigma' = (R, z')$ with probability at least $(\epsilon')^2/n'_r$ where ϵ' is $\mathcal{F}'$'s success probability. These forgeries satisfy the verification equations $g^z = R \cdot (pk')^c$ and $g^{z'} = R \cdot (pk')^{c'}$ with distinct challenges $c \neq c'$, yielding:

$$\text{dlog}(pk') = \frac{z' - z}{c' - c}$$

where $pk' = g^{f'(0)}$ is the post-operation public key and $f'(x)$ is the post-operation shared polynomial.

We extract $\text{dlog}(\omega)$ by inverting the polynomial transformation. Let $f(x)$ denote the pre-operation polynomial with $pk = g^{f(0)}$. Define extraction function $E_{\mathcal{O}} : \mathbb{Z}_p^4 \to \mathbb{Z}_p$ based on operation type:

Weight increase / Threshold decrease: The polynomial is invariant under these operations: $\tilde{f}(x) = f(x)$, thus $\tilde{f}(0) = f(0)$. The extraction function is:

$$E_{\mathsf{WI}}(z, z', c, c') = E_{\mathsf{TD}}(z, z', c, c') = \frac{z' - z}{c' - c} - \sum_{i \neq t} a_{i,0}$$

Threshold increase: The transformation is $\tilde{f}(x) = f(x) \cdot h(x) + r$ where $h(x) = \sum_{i \in P} h_i(x)$ is the aggregate polynomial and $r = \sum_{i \in P}(d_i + e_i \cdot \rho_i)$ is the preprocessing sum. Evaluating at zero yields $\tilde{f}(0) = f(0) \cdot h(0) + r$. Since the simulator knows $h(0)$ (from protocol participation as P_t and observation of other contributions) and r (from the preprocessing phase), extraction proceeds via:

$$E_{\mathsf{TI}}(z, z', c, c') = \frac{1}{h(0)} \left(\frac{z' - z}{c' - c} - r \right) - \sum_{i \neq t} a_{i,0}$$

This requires $h(0) \neq 0$. The simulator ensures this by setting $h_t(0) = 1$ when acting as P_t. Should $h(0) = 0$ occur, this would require all other participants' contributions to sum to -1. The probability of this event is at most $1/p$ over their random choices, and the simulator aborts in this negligible-probability event.

Weight decrease: The transformation is $\tilde{f}(x) = f(x) + \sum_{i \in P} h_i(x)$ where the protocol specification requires $h_t(0) = 0$ for the target participant's weight reduction. Thus $\tilde{f}(0) = f(0) + \sum_{j \in P \setminus \{t\}} h_j(0)$. The simulator observes all non-zero values $\{h_j(0)\}_{j \in P \setminus \{t\}}$ from protocol messages, enabling extraction:

$$E_{\mathsf{WD}}(z, z', c, c') = \frac{z' - z}{c' - c} - \sum_{j \in P \setminus \{t\}} h_j(0) - \sum_{i \neq t} a_{i,0}$$

For each operation type, the extraction function $E_{\mathcal{O}}$ correctly computes $f(0)$ by inverting the polynomial relationship. Since the initial embedding establishes $f(0) = \text{dlog}(\omega) \cdot \lambda + \sum_{i \neq t} a_{i,0}$ for known constant λ and coefficients $\{a_{i,0}\}_{i \neq t}$ (obtained from zero-knowledge proofs during key generation), we extract:

$$\text{dlog}(\omega) = \frac{E_{\mathcal{O}}(z, z', c, c')}{\lambda}$$

Security Preservation. The post-operation weight constraint $w'_F < t'$ ensures the adversary controls strictly fewer than t' shares in instance $\mathcal{I}'$. By Lemma 1, the GLI-based secret sharing maintains the rank-t' property: any coalition with total weight $< t'$ possesses insufficient information to reconstruct the secret, even information-theoretically. Therefore, the adversary's corrupted shares alone cannot determine $\tilde{f}(0)$, and forking extraction via $\mathcal{F}'$ remains the unique method for algorithm $\mathcal{C}$ to obtain $\text{dlog}(\omega)$.

By the generalized forking lemma, algorithm $\mathcal{C}$ solves DLP with advantage at least $(\epsilon')^2/n'_r$ whenever forking succeeds. Since DLP has advantage bound ϵ_{DLP} by assumption, we have $(\epsilon')^2/n'_r \leq \epsilon_{DLP}$, which gives $\epsilon' \leq \sqrt{\epsilon_{DLP} \cdot n'_r}$. Therefore, no polynomial-time forger with success probability greater than $\mathcal{O}(\sqrt{\epsilon_{DLP} \cdot n'_r})$ can exist. Thus $\mathcal{I}'$ is secure with the stated advantage bound under the DLP assumption. □

Corollary VII.1 (Security Under Operation Sequences)**.** *Let $\mathcal{I}$ be an* ADAPT *instance with threshold t_0 where the adversary controls weight $w_F < t_0$. After any sequence of authorized dynamic operations producing thresholds $\{t_1, \ldots, t_k\}$, if the adversary's final weight $w'_F < t_k$, then $\mathcal{I}$ remains $(t_k, n_h, n_p + k, n_s, \epsilon')$-secure for appropriate ϵ'.*

Proof. By induction on Lemma 2, each operation preserves security if the weight constraint holds. The simulator applies inverse transformations in reverse order to extract ω from final forgeries. □

Remark (Active Adversary and Open Problem). While Corollary VII.1 ensures security when $w'_F < t'$ holds throughout, Definition IV.2 presents a fundamental challenge not specific to ADAPT but inherent to any dynamic threshold cryptographic system. Consider: adversary with $w_F = 99 < t = 100$ participates in TIncrease($t \to 150$), receives WIncrease($w_F \to 149$), then benefits from TDecrease($t \to 100$), yielding $w'_F = 149 > t'' = 100$. Each step satisfies $w_F < t$ individually, yet the composition violates security.

This demonstrates why Definition IV.2 remains open: preventing such strategic manipulation requires either (1) operation ordering restrictions, (2) predictive adversarial weight tracking, or (3) novel cryptographic techniques maintaining security when $w_F \geq t$ temporarily. Current DLP-based reductions fail as the simulator loses extraction capability once adversaries exceed threshold. Our scheme provides provable security guaranteeing: *if $w'_F < t'$ after every operation, the system remains secure*—a practical and deployable guarantee for real-world applications enforced through access control policies.

VIII. Implementation and Performance Analysis

We evaluate ADAPT through three complementary approaches since direct AWS comparisons are not possible as this paper introduces the AWS concept for the first time.

A. Comparison to the virtualization method

We extend FROST to a weighted threshold scheme using virtualization for comparison with ADAPT. We implement

both schemes using Go 1.21 over Edwards 25519, running on Ubuntu 22.04 with Intel i3-6100U 2.30GHz CPU and 8GB RAM [3].

We assume total weights $N \in \{20, 100, 500\}$ and threshold percentages $\{50\%, 70\%, 100\%\}$ with discrete uniform weight distribution from 1 to 4. Hash functions used are SHA512(H_0, H_1) and SHA3-512(H_2, H_3).

As depicted in Fig. 2, ADAPT consistently outperforms FROST in KeyGen across all settings of N and t. The efficiency gain stems from GLI's derivative-based encoding, which reduces the number of actual participants compared to FROST's virtualization approach where each weighted participant is expanded into multiple virtual participants. While both schemes share $\mathcal{O}(n^2)$ complexity, the performance gap widens as N grows: at $N = 500$ and $t = 250$, ADAPT completes KeyGen approximately 10% faster than FROST, demonstrating that the reduction in virtual participant count becomes increasingly significant at scale.

Fig. 3 demonstrates that ADAPT's signing advantage over FROST grows substantially as N and t increase. At $N = 20$ and $t = 10$, both schemes perform comparably. However, at $N = 500$ and $t = 500$, ADAPT achieves approximately 3.8× speedup over FROST (0.38s vs. 1.45s). This dramatic improvement arises because FROST's Sign complexity scales as $\mathcal{O}(t^2)$, requiring exactly t participants, whereas ADAPT's $\mathcal{O}(m^2)$ complexity requires only m participants whose combined weights satisfy the threshold.

Fig. 4 compares the execution times of the four dynamic operations against re-setup (Setup). Across all settings, Setup dominates in cost, confirming that ADAPT's dynamic operations provide substantial efficiency gains over re-initialization. Among the four operations, WIncrease incurs the highest cost due to its $\mathcal{O}(m^2)$ complexity from GLI polynomial construction, growing noticeably with both N and t. TIncrease and WDecrease exhibit similar moderate costs ($\mathcal{O}(n)$ and $\mathcal{O}(t \cdot n)$ respectively), as they distribute updates across all participants but avoid heavy polynomial construction. TDecrease consistently achieves the lowest cost among all four operations owing to its $\mathcal{O}(n)$ complexity, requiring only coefficient extraction and broadcast without per-participant GLI construction. At $N = 500$ and $t = 500$, all four operations complete within 4.1–22.3% of the Setup time, confirming ADAPT's practical efficiency.

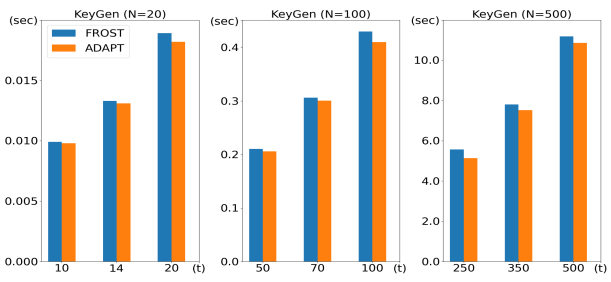

Fig. 2. FROST and ADAPT's KeyGen algorithm average execution time according to N and t of each node

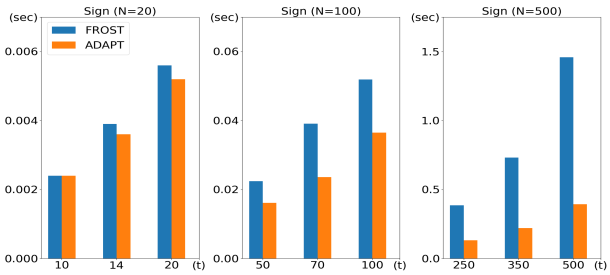

Fig. 3. FROST and ADAPT's Sign algorithm average execution time according to N and t of each node

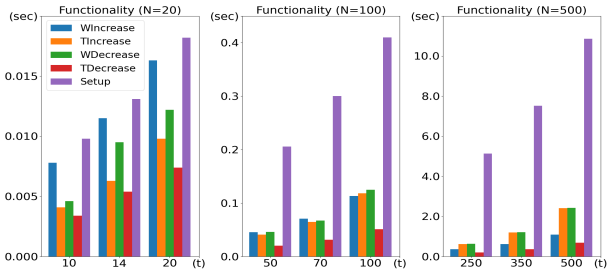

Fig. 4. ADAPT Functionality algorithm average execution time according to N and t of each node

B. Comparison to other WS

We compare ADAPT with existing weighted signature schemes across Schnorr-based (Edwards25519) and BLS-based (BN254) approaches. Implementation uses Go 1.21 and open-source libraries, utilizing AWS EC2 c4.xlarge instances. BN254 pairing operations are approximately 241.6 times slower than addition operations on Edwards25519.

Scheme	Signing cost per participant	Signature size (Bytes)	Verification cost / time(μs)
Dikshit et al. [10]	$(\frac{t(t+1)N^2}{2} + 2N) \cdot \mathsf{C_e}$	G + Z (96B)	$2\mathsf{C_e}$ (187.5 μs)
Wang et al. [26]	$(2t+2)\mathsf{C_e}$	G + Z (96B)	$2\mathsf{C_e}$ (187.4 μs)
Garg et al. [13]	$15t\mathsf{C_e} + \epsilon$	2Z (64B)	$(t+2)\mathsf{C_e}$ ($90t$+187.4 μs)
Baird et al. [4]	$t\mathsf{C_e} + 2t\mathsf{C_p}$	Z (32B)	$4\mathsf{C_p}$ (3874.8 μs)
Das et al. [8]	$(m+7)\mathsf{C_e} + 2t\mathsf{C_p}$	8G + Z (542B)	$11\mathsf{C_p} + 6\mathsf{C_e}$ (10955.1 μs)
Ours	$5m\mathsf{C_e}$	G + Z (96B)	$2\mathsf{C_e}$ (184.1 μs)

TABLE III
COMPUTATION COST COMPARISON OF THRESHOLD SIGNATURE SCHEMES

Table III reveals significant performance advantages of ADAPT. Wang et al. [26] and ADAPT exhibit similar signature size (96B) and verification time (187µs), but [26] requires operations proportional to threshold t while our scheme scales with authorized party size m, where typically $m < t$.

Compared to other weighted schemes, ADAPT achieves the lowest signing cost ($5m\mathsf{C_e}$) and highest verification efficiency. Specifically, while Garg et al. [13] offers smaller signatures (64B vs 96B), their verification cost grows linearly with threshold ($90t$+187.4µs vs 184.1µs), making ADAPT at least 1.5x faster for practical thresholds. BLS-based schemes like Baird et al. [4] and Das et al. [8] suffer from expensive pairing operations, resulting in 21x and 60x slower verification respectively. Notably, ADAPT is the only scheme providing

Scenario	Sign	WIncrease	TIncrease	WDecrease	TDecrease
25/50 (Alice:24, Bob:1)	2.44	1.70	2.42	1.95	2.32
37/50 (Alice:36, Bob:1)	2.92	1.90	2.86	2.06	2.67
50/50 (Alice:49, Bob:1)	3.29	2.30	3.24	2.31	3.23

TABLE IV
EXECUTION TIME RATIOS (ALICE/BOB) WITH DIFFERENT WEIGHT DISTRIBUTIONS

comprehensive dynamic functionality without trusted authorities or system reinitialization.

C. Weight Distribution Impact Analysis

We conducted experiments with extreme weight allocations using two participants (Alice, Bob) with significantly different weights under various threshold conditions (50%, 75%, 100%). Table IV shows ADAPT's efficiency with uneven weight distributions. The ratios represent execution time comparisons between Alice (high-weight participant) and Bob (low-weight participant). In the most extreme case (50/50 scenario), Alice holds 49x more weight than Bob but requires only 3.29x more computation time for signing. This sub-linear scaling is significant: traditional virtualization approaches would require Alice to perform 49x more operations than Bob. Performance varies by operation type: WIncrease and WDecrease show better relative efficiency (1.7-2.3x ratios) due to focused derivative calculations, while Sign and TIncrease show higher ratios (2.4-3.3x) due to full polynomial evaluations. As threshold requirements increase from 50% to 100%, the computational gap widens but remains substantially below the weight differential. ADAPT's GLI-based approach provides substantial efficiency benefits in blockchain scenarios with highly uneven stake distributions, addressing a critical scalability limitation of existing approaches.

IX. CONCLUSION

We introduce Active Weighted Signature (AWS), the first threshold signature framework enabling dynamic weight and threshold adjustments without re-setup or trusted dealers. Our ADAPT instantiation leverages Generalized Lagrange Interpolation (GLI) to achieve: (1) sub-linear computation scaling—49× weight difference requires only 3.29× computation versus 49× in virtualization approaches, (2) efficient dynamic operations completing in 4.1-22.3% of KeyGen time for $N = 500$, and (3) constant-size verification (184.1µs) outperforming existing weighted schemes by 1.5-60×.

For Proof of Stake blockchains, AWS enables immediate stake adjustments without system downtime, ensuring consensus security and network availability during frequent stake changes. This addresses a critical limitation in current PoS implementations requiring disruptive re-initialization procedures.

REFERENCES

[1] Ethereum consensus-sepcs. Online Available. https://github.com/ethereum/consensus-specs, 2024.

[2] Ethereum proof-of-stake(pos). Online Available. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/, 2024.

[3] ADAPT: Active weighted signature implementation. Online Available. https://github.com/maskingidentity/ADAPT_imple, 2025.

[4] L. Baird, S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. Threshold signatures in the multiverse. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2057–2073. IEEE Computer Society, 2022.

[5] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399, 2006.

[6] D. Boneh. The decision diffie-hellman problem. In *International algorithmic number theory symposium*, pages 48–63. Springer, 1998.

[7] V. Buterin. Ethereum: platform review. *Opportunities and Challenges for Private and Consortium Blockchains*, 45, 2016.

[8] S. Das, P. Camacho, Z. Xiang, J. Nieto, B. Bünz, and L. Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 356–370, 2023.

[9] R. Del Pino, S. Katsumata, M. Maller, F. Mouhartem, T. Prest, and M.-J. Saarinen. Threshold raccoon: practical threshold signatures from standard lattice assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 219–248. Springer, 2024.

[10] P. Dikshit and K. Singh. Efficient weighted threshold ecdsa for securing bitcoin wallet. In *2017 ISEA Asia Security and Privacy (ISEASP)*, pages 1–9. IEEE, 2017.

[11] T. Espitau, S. Katsumata, and K. Takemure. Two-round threshold signature from algebraic one-more learning with errors. *Journal of Cryptology*, 38(4):31, 2025.

[12] T. Espitau, G. Niot, and T. Prest. Flood and submerse: Distributed key generation and robust threshold signature from lattices. In *Annual International Cryptology Conference*, pages 425–458. Springer, 2024.

[13] S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. Cryptography with weights: Mpc, encryption and signatures. In *Annual International Cryptology Conference*, pages 295–327. Springer, 2023.

[14] S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. hints: Threshold signatures with silent setup. In *2024 IEEE symposium on security and privacy (SP)*, pages 3034–3052. IEEE, 2024.

[15] N. Gupta. A deep dive into security and privacy issues of blockchain technologies. In *Handbook of research on blockchain technology*, pages 95–112. Elsevier, 2020.

[16] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017.

[17] C. Komlo and I. Goldberg. Frost: flexible round-optimized schnorr threshold signatures. In *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*, pages 34–65. Springer, 2021.

[18] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar. Korgan: An efficient pki architecture based on pbft through dynamic threshold signatures. *The Computer Journal*, 64(1):564–574, 2019.

[19] J. Leng, M. Zhou, J. L. Zhao, Y. Huang, and Y. Bian. Blockchain security: A survey of techniques and research directions. *IEEE Transactions on Services Computing*, 15(4):2490–2510, 2020.

[20] K. M. Martin, J. Pieprzyk, R. Safavi-Naini, and H. Wang. Changing thresholds in the absence of secure channels. In *Information Security and Privacy: 4th Australasian Conference, ACISP'99 Wollongong, NSW, Australia, April 7–9, 1999 Proceedings 4*, pages 177–191. Springer, 1999.

[21] A. Noack and S. Spitz. Dynamic threshold cryptosystem without group manager. *Cryptology ePrint Archive*, 2008.

[22] M. Nojoumian and D. R. Stinson. On dealer-free dynamic threshold schemes. *Adv. Math. Commun.*, 7(1):39–56, 2013.

[23] A. I. Sanka and R. C. Cheung. A systematic review of blockchain scalability: Issues, solutions, analysis and future research. *Journal of Network and Computer Applications*, 195:103232, 2021.

[24] R. Steinfeld, J. Pieprzyk, and H. Wang. Lattice-based threshold changeability for standard shamir secret-sharing schemes. *IEEE transactions on information theory*, 53(7):2542–2559, 2007.

[25] C. Tartary and H. Wang. Dynamic threshold and cheater resistance for shamir secret sharing scheme. In *Information Security and Cryptology: Second SKLOIS Conference, Inscrypt 2006, Beijing, China, November 29-December 1, 2006. Proceedings 2*, pages 103–117. Springer, 2006.

[26] H. Wang, W. Ma, F. Deng, H. Zheng, and Q. Wu. Dynamic threshold ecdsa signature and application to asset custody in blockchain. *Journal of Information Security and Applications*, 61:102805, 2021.

[27] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020.

[28] X. Yi, D. Wu, L. Jiang, Y. Fang, K. Zhang, and W. Zhang. An empirical study of blockchain system vulnerabilities: Modules, types, and patterns. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 709–721, 2022.

[29] J. Yuan and L. Li. A fully dynamic secret sharing scheme. *Information Sciences*, 496:42–52, 2019.