Enhancing Local Verification: Aggregate and Multi-Signature Schemes

Ahmet Ramazan Ağırtaş[1][0000−0002−4574−0067], Neslihan Yaman Gökce[2][0000−0002−5014−0816] *, and Oğuz Yayla[2][0000−0001−8945−2780]

[1] Nethermind, London, UK
[2] Institute of Applied Mathematics, Middle East Technical University, Ankara, Turkey

Abstract. An aggregate signature scheme is a digital signature protocol that enables the aggregation of multiple signatures. Given n signatures on n distinct messages from n different users, it is possible to combine all these signatures into a single, concise signature. This single signature, along with the n original messages, convinces the verifier that the n users indeed signed their respective n original messages. However, the verifier must have access to all the original messages to perform the verification, highlighting a potential limitation in terms of accessibility and efficiency. Goyal and Vaikuntanathan introduced the concept of local verification, which allows the verifier to determine if a specific message m is part of the aggregated signature by only accessing the message m. In this paper, we extend the single-signer locally verifiable aggregate signature scheme initially proposed by Goyal and Vaikuntanathan, adapting it to a multi-signer context. Our generalization allows the verifier to validate multiple signatures simultaneously using an auxiliary value generated by the LocalOpen algorithm, thereby enhancing verification efficiency. Furthermore, we integrate this approach into the multi-signature scheme proposed by Boneh, Drijvers, and Neven, demonstrating its broader applicability and potential benefits in complex cryptographic systems.

Keywords: aggregate signatures · locally verifiable signatures · multi-signatures.

1 Introduction

A digital signature is a cryptographic technique crucial for ensuring the integrity and authenticity of electronic data, making it indispensable for secure transactions over open networks. Digital signature schemes form the backbone of modern security infrastructure, facilitating trust and verification in various applications, including online banking, e-commerce, software distribution, and secure communications. As these technologies evolve, digital signature schemes continue to adapt, addressing emerging challenges with a focus on enhancing efficiency, scalability, and robustness in increasingly complex digital environments.

* Corresponding author

A multi-signature scheme [2,3,10,27] is a protocol that allows multiple signers to collaboratively produce a compact signature σ for a message m, ensuring that a verifier can be convinced that all participating parties have signed the same m. In particular, the verification process involves the input of the n public keys, the message m, and the resulting multi-signature σ. At the end of the verification, σ can be rejected or accepted. This collaborative approach significantly supports security by distributing trust across several parties, thus mitigating the risks associated with key compromise or unauthorized access [6]. Multi-signature protocols play a crucial role in numerous applications, such as blockchain transactions, joint bank accounts, and distributed ledger technologies, where collective approval is paramount [17].

The concept of aggregate signatures was introduced by Boneh et al. [11] and has since been extensively studied [1, 5, 7, 12, 19, 20, 26]. It enables multiple individual signatures to be combined into a single compact signature, regardless of whether the signatures are on the same or different messages and created by different signers. Aggregate signatures provide significant efficiency benefits in systems where multiple signatures need to be verified, such as in blockchain networks, Internet of Things (IoT) applications, and other distributed systems, since computational power, bandwidth, and storage are limited in these environments. Besides that, aggregate signatures not only streamline the verification process but also maintain the same security guarantees as traditional digital signatures, making them an essential tool in modern cryptographic protocols [6]. Recent research on aggregate signatures has focused on enhancing privacy, reducing computational overhead, and adapting to specific application needs such as blockchain and group authentication. These innovations underscore aggregate signatures' flexibility and increasing relevance in advanced cryptographic systems [3, 13, 24].

The verification process for aggregate signatures requires access to all n messages. Therefore, the cost of the verification process increases linearly with the number of messages. To address this issue, Goyal, Rishab, and Vaikuntanathan [18] proposed the concept of locally verifiable aggregate signatures, which enables efficient verification for only one message instead of the whole message set. They present two constructions for single-signer locally verifiable aggregate signatures: one relying on the RSA assumption [4,21,25] and the other on the bilinear Diffie-Hellman inversion assumption [8, 9, 22] within the random oracle model.

Locally verifiable signature schemes have not been studied extensively yet since the proposed idea is already new. However, this concept has swiftly inspired considerable research effort. In literature, two notable works based on the local verification concept based on [18] are the works of Duan et al. [16] and Zue et al. [29]. Both works adapt the foundational concepts to address specific challenges in health data verification and sharing.

Locally verifiable signatures allow efficient verification by enabling a verifier to check the presence of a particular message in an aggregate signature without accessing the entire set of messages. There are several important uses for this

concept. Certificate transparency logs can store compressed certificates as aggregate signatures. Users can verify the existence of a certificate by downloading a short hint instead of the entire log, reducing storage and computational costs while maintaining security. In blockchain applications, locally verifiable aggregate signatures can aggregate signatures of all transactions from a single payer. This allows users to efficiently prove the existence of a specific transaction with minimal communication. Additionally, it offers privacy benefits by enabling proof of a single transaction without revealing other transactions. On the other hand, in cryptocurrencies, multi-signature schemes allow multiple parties to jointly authorize a transaction. Locally verifiable multi-signatures can make the verification process more efficient by enabling the verification of individual signatures without accessing the entire transaction set.

In this paper, we first generalize the locally verifiable aggregate signature scheme proposed in [18]. It means that first, a hint is generated for any k out of ℓ messages using the local opening function. Then, a verifier can use the generated hint to verify selected k messages at once. Moreover, the multi-message local verification algorithm operates independently of the number of messages ℓ, which is a natural expectation from such an improvement. Secondly, we present a locally verifiable version of the multi-signature scheme proposed in [10]. This time, the localization is performed in the signers domain, instead of the messages domain. Similarly to that of multi-message local verification algorithm, locally verifiable multi-signature algorithm also operates independently of the total number of signers n.

The structure of the paper is as follows. In Section 2, we provide a concise overview, including definitions of bilinear pairings, computational problems such as Diffie-Hellman Inversion, Bilinear Diffie-Hellman Inversion, co-CDH problems, the generalized forking lemma, and locally verifiable aggregate signatures. In Section 3, we give the formal definition of our first contribution, namely Multi-Message Locally Verifiable Aggregate Signatures (MM-LVAS), and then give the details about the scheme. We also prove its security in the random oracle model. In Section 4, we first provide the formal definition of Locally Verifiable Multi-Signatures (LVMS) and then mention the details of the scheme. In the same section, we demonstrate its security in the random oracle model by employing the generalized forking lemma. Finally, the conclusion of our paper is given in Section 5.

Note that, throughout this paper, we consider three types of entities: signers, who execute key generation and signing operations; storage servers (or aggregators), who perform aggregation of the signatures and generate the hint; and verifiers, who carry out the verification processes.

2 Preliminaries

In this section, we define the concepts that we utilize in this paper. More precisely, we define bilinear pairings, underlying hard problems, generalized forking lemma, and locally verifiable aggregate signature schemes.

In general, we denote the set of all positive integers up to n as $[n] := \{1, \ldots, n\}$. Additionally, the set of all non-negative integers up to n, i.e., $[0, n] := 0 \cup [n]$ is denoted as $[0, n]$.

2.1 Bilinear Pairings

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic additive groups of prime order q and $\mathbb{G}_T$ be a cyclic multiplicative group with the same order. A pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ which satisfies the bilinearity and non-degeneracy properties:

- Bilinearity: $e(A^\alpha, B^\beta) = e(A, B)^{\alpha\beta}$ for all $\alpha, \beta \in \mathbb{Z}_q$, $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.
- Non-degeneracy: $e(A, B) \neq 1$ for all $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

Note that, in general, we can classify pairings into two classes, i.e., symmetric and asymmetric pairings. The above definition belongs to the latter. In this paper, we use both symmetric and asymmetric ones in our constructions. For the symmetric ones, we will not give another definition, yet one can simply consider $\mathbb{G}_1 = \mathbb{G}_2$.

2.2 Computational Problems

Definition 1 (Diffie-Hellman Inversion Problem [8]). *For groups $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T$ of order p, let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be an efficient and non-degenerate bilinear mapping. Let $x \in \mathbb{Z}_p^*$ be a randomly chosen secret and $q \in \mathbb{Z}$ be the hardness parameter of the problem, which is the length of the powers-in-exponent sequence given to the adversary. Given $g, g^x, g^{x^2}, \ldots, g^{x^q} \in \mathbb{G}$, the Diffie-Hellman Inversion (DHI) problem is to compute $g^{1/x} \in \mathbb{G}$.*

Definition 2 (Bilinear Diffie-Hellman Inversion Problem [8]). *For groups $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T$ of order p, let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be an efficient and non-degenerate bilinear mapping. Let $x \in \mathbb{Z}_p^*$ be a randomly chosen secret and $q \in \mathbb{Z}$ be the hardness parameter of the problem, which is the length of the powers-in-exponent sequence given to the adversary. Given $g, g^x, g^{x^2}, \ldots, g^{x^q} \in \mathbb{G}$, the Bilinear Diffie-Hellman Inversion (q-BDHI) problem is to compute $e(g, g)^{1/x} \in \mathbb{G}_T$.*

Definition 3 (Computational co-Diffie-Hellman Problem [10]). *For groups $\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle$ of prime order q, define $Adv^{co\text{-}CDH}_{\mathbb{G}_1, \mathbb{G}_2}$ of an adversary $\mathcal{A}$ as*

$$\Pr\left[y = g_1^{\alpha\beta} : (\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_q^2, y \leftarrow \mathcal{A}\left(g_1^\alpha, g_1^\beta, g_2^\beta\right)\right],$$

where the probability is taken over the random choices of $\mathcal{A}$ and the random selection of (α, β). $\mathcal{A}(\tau, \epsilon)$-breaks the co-CDH problem if it runs in time at most τ and has $Adv^{co\text{-}CDH}_{\mathbb{G}_1, \mathbb{G}_2} \geq \epsilon$. co-CDH is (τ, ϵ)-hard if no such adversary exists.

2.3 Generalized Forking Lemma

Pointcheval and Stern initially defined the forking lemma in [23], which has been widely used to establish the security of schemes based on Schnorr signatures within the random-oracle model [28]. Bellare and Neven later generalized this lemma in [6]. Boneh et al. used a further generalization of the forking lemma by Bagherzandi et al. [2] in their security proofs in [10]. In our work, we also utilize this latter generalized forking lemma in the security proofs of our second construction given in Section 4.

Let $\mathcal{A}$ be an algorithm interacting with a random oracle $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ and takes in as input. Let $f = (\rho, h_1, \ldots, h_{q_H})$ represent the randomness used during an execution of $\mathcal{A}$. Here, ρ denotes $\mathcal{A}$'s random tape, h_i is the response to $\mathcal{A}$'s i-th query to H, and q_H is the maximum number of random-oracle queries made by $\mathcal{A}$. Let Ω be the space of all randomness vectors like f and let $f|_i = (\rho, h_1, \ldots, h_{i-1})$ for any $i \leq q_H$. We consider an execution of $\mathcal{A}$ taking input in and randomness f, denoted $\mathcal{A}(in, f)$, as successful if it outputs a pair $\left(J, \{out_j\}_{j \in J}\right)$, where J is a multi-set that is a non-empty subset of $\{1, \ldots, q_H\}$ with $|J| = n$, and $(\{out_j\}_{j \in J})$ is a multi-set of side outputs. We say that $\mathcal{A}$ failed if it outputs $J = \emptyset$. Let ϵ be the probability that $\mathcal{A}(in, f)$ is successful for fresh randomness $f \xleftarrow{\$} \Omega$ and for input $in \xleftarrow{\$}$ IG generated by an input generator IG.

For a given input in, the generalized forking algorithm $\mathcal{GF}_{\mathcal{A}}$ is presented in Algorithm 1.

Lemma 1 (Generalized Forking Lemma [2]). *Let* IG *be a randomized algorithm generating in. Let $\mathcal{A}$ be a random algorithm running in time τ, making at most q_H random oracle queries that succeeds with probability ϵ. If $q > 8nq_H/\epsilon$, then $\mathcal{GF}_{\mathcal{A}}(in)$ runs in time at most $\tau \cdot 8n^2 q_H/\epsilon \cdot \ln(8n/\epsilon)$ and succeeds with probability at least $\epsilon/8$, where the probability is over the choice of in $\xleftarrow{\$}$* IG *and over the coins of $\mathcal{GF}_{\mathcal{A}}(in)$.*

2.4 Locally Verifiable Aggregate Signature Schemes

A locally verifiable aggregate signature scheme [18] is a tuple of seven algorithms, i.e., Setup, Sign, Verify, Aggregate, AggVerify, LocalOpen, LocalAggVerify, which can be described as follows.

$\textsc{Setup}(1^\lambda) \rightarrow (vk, sk)$. The setup algorithm takes the security parameter λ as input and outputs a pair of signing and verification keys (vk, sk).

$\textsc{Sign}(sk, m) \rightarrow \sigma$. The signing algorithm takes a signing key sk and a message m and outputs a signature σ.

$\textsc{Verify}(vk, m, \sigma) \rightarrow 0/1$. The verification algorithm takes a verification key vk, a message m , and a signature σ as input. It outputs 1 if the signature is valid or 0 otherwise.

Algorithm 1 Algorithm $\mathcal{GF}_{\mathcal{A}}$

Input: in
Output: $(J, \{\text{out}_j\}_{j\in J}, \{\text{out}'_j\}_{j\in J})$ **or** $fail$

$f = (\rho, h_1, \ldots, h_{q_H}) \leftarrow \Omega$
$(J, \{\text{out}_j\}_{j\in J}) \leftarrow \mathcal{A}(in, f)$
if $J = \emptyset$ **then**
  return $fail$
else
  Let $J = \{j_1, \ldots, j_n\}$ such that $j_1 \leq \ldots \leq j_n$
  for $i = 1, \ldots, n$ **do**
    $\text{succ}_i \leftarrow 0$
    $k_i \leftarrow 0$
    $k_{\max} \leftarrow \frac{8nq_{\text{H}}}{\epsilon} \cdot \ln\left(\frac{8n}{\epsilon}\right)$
    repeat
      $f'' \leftarrow \Omega$ such that $f''|_{j_i} = f|_{j_i}$
      Let $f'' = (\rho, h_1, \ldots, h_{j_i-1}, h''_{j_i}, \ldots, h''_{q_H})$
      $(J'', \{\text{out}''_j\}_{j\in J''}) \leftarrow \mathcal{A}(in, f'')$
      if $h''_{j_i} \neq h_{j_i}$ **and** $J'' \neq \emptyset$ **and** $j_i \in J''$ **then**
        $\text{out}'_{j_i} \leftarrow \text{out}''_{j_i}$
        $\text{succ}_i \leftarrow 1$
      end if
      $k_i \leftarrow k_i + 1$
    until $\text{succ}_i = 1$ **or** $k_i > k_{\max}$
  end for
  if $\text{succ}_i = 1$ for all $i = 1, \ldots, n$ **then**
    return $(J, \{\text{out}_j\}_{j\in J}, \{\text{out}'_j\}_{j\in J})$
  else
    return $fail$
  end if
end if

$\textsc{Aggregate}(vk, \{m_i, \sigma_i\}_i) \rightarrow \hat{\sigma}/\perp$. The signature aggregation algorithm takes a verification key vk, a sequence of tuples, each containing a message m_i and signature σ_i as input and it outputs either an aggregated signature $\hat{\sigma}$ or $\perp$.

$\textsc{AggVerify}(vk, \{m_i\}_i, \hat{\sigma}) \rightarrow 0/1$. The aggregate verify algorithm takes a verification key vk, a sequence of messages m_i, and an aggregated signature $\hat{\sigma}$ as input and outputs 1 if it is valid or 0 otherwise.

$\textsc{LocalOpen}(\hat{\sigma}, vk, \{m_i\}_{i\in[\ell]}, j \in [\ell]) \rightarrow aux_j$. The local opening algorithm takes an aggregated signature $\hat{\sigma}$, a verification key vk, a sequence of messages m_i for $i \in [\ell]$, and an index $j \in [\ell]$ as input. It outputs auxiliary information aux_j corresponding to the message m_j.

$\textsc{LocalAggVerify}(\hat{\sigma}, vk, m, aux) \rightarrow 0/1$. The local aggregate verification algorithm takes an aggregated signature $\hat{\sigma}$, a verification key vk, a message m,

and auxiliary information aux as input. It outputs 1 if the aggregate signature $\hat{\sigma}$ contains a signature for message m under verification key vk, 0 otherwise.

A locally verifiable aggregate signature scheme is considered correct and compact if for all $\lambda, \ell \in \mathbb{N}$, given every verification-signing key pair $(vk, sk) \leftarrow \textsc{Setup}(1^\lambda)$, messages m_i, and every signature $\sigma_i \leftarrow \textsc{Sign}(sk, m_i)$ for $i \in [\ell]$, the following conditions hold:

1. *Correctness of Local Opening.* For all $k \in [\ell]$,

$$\textsc{LocalAggVerify}\left(\hat{\sigma}, vk, m_k, \textsc{LocalOpen}\left(\hat{\sigma}, vk, \{m_i\}_i, k\right)\right) = 1$$

2. *Compactness of Opening.* $|aux| \leq \text{poly}(\lambda)$, i.e., the size of the auxiliary opening information is a fixed polynomial in the security parameter λ, independent of the number of aggregations ℓ.

Definition 4 (Aggregated Unforgeability [18]). *A single-signer aggregated signature scheme (Setup, Sign, Verify, Aggregate, AggVerify) is said to be a secure aggregate signature scheme if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\textsc{AggVerify}\left(vk, \{m_i^*\}_{i\in[\ell]}, \hat{\sigma}^*\right) = 1 : \begin{array}{l}(vk, sk) \leftarrow \textsc{Setup}\left(1^\lambda\right) \\ \left(\{m_i^*\}_{i\in[\ell]}, \hat{\sigma}^*\right) \leftarrow \mathcal{A}^{\textsc{Sign}(sk,\cdot)}\left(1^\lambda, vk\right)\end{array}\right] \leq negl(\lambda),$$

where $\mathcal{A}$ is admissible if there exists $i \in [\ell]$ such that m_i^* was not queried by $\mathcal{A}$ to the $\textsc{Sign}(sk, \cdot)$ oracle.

Definition 5 (Static Aggregated Unforgeability [18]). *Before an adversary $\mathcal{A}$ receives the verification key vk, if it is restricted to provide both the message queries $\{m_i\}_{i\in[q]}$ and the challenge messages $\{m_i^*\}_{i\in[\ell]}$ at the beginning of the game defined in Definition 4, then we say that this aggregated signature scheme is statically secure.*

Definition 6 (Agg. Unforgeability with Adversarial Opening [18]). *A locally verifiable aggregate signature scheme (Setup, Sign, Verify, Aggregate, AggVerify, LocalOpen, LocalAggVerify) is said to be a secure aggregate signature scheme against adversarial openings if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\textsc{LocalAggVerify}\left(\widehat{\sigma}^*, vk, m^*, aux^*\right) = 1 : \begin{array}{l}(vk, sk) \leftarrow \textsc{Setup}\left(1^\lambda\right) \\ \left(\widehat{\sigma}^*, aux^*, m^*\right) \leftarrow \mathcal{A}^{\textsc{Sign}(sk,\cdot)}\left(1^\lambda, vk\right)\end{array}\right] \leq negl(\lambda),$$

where $\mathcal{A}$ is admissible if m^* was not queried by $\mathcal{A}$ to the $\textsc{Sign}(sk, \cdot)$ oracle.

Definition 7 (Static Agg. Unforgeability with Adversarial Opening [18]). *We say the locally verifiable aggregate signature scheme is statically secure against adversarial openings if the adversary in the previous game is confined to make all of its message queries $\{m_i\}_{i\in[q]}$ and declare the challenge message m^* at the beginning of the game before it receives the verification key vk.*

2.5 Pairing-based Locally Verifiable Aggregate Signatures

In this section, we discuss the primitives and functions utilized in the original scheme proposed by Goyal and Vaikuntanathan [18], and consequently, in our extended scheme. Additionally, we provide an overview of the structural framework of the original scheme that forms the basis of our work, which is an extension of the locally verifiable aggregate signature scheme proposed in [18].

Injective Message Hashing and Its Variants. An injective mapping from the message space $\mathcal{M}_\lambda = \{0,1\}^\lambda$ to the prime field $\mathbb{Z}_p$ for $p > 2^\lambda$ is used. To achieve static and fully adaptive security, we consider two straightforward mappings, $(HGen, H)$, namely identity mapping and random oracle mapping. The identity mapping employs a hash setup $HGen^{\mathcal{I}}$, which is a trivial algorithm that outputs $hk = \epsilon$. The corresponding hash function $H^{\mathcal{I}}(\epsilon, m)$ simply returns m, interpreting the output m as a field element of $\mathbb{Z}_p$. For the random oracle mapping, let $\mathcal{H} = \{\mathcal{H}_\lambda\}_\lambda$ denote a family of hash functions where each $h \in \mathcal{H}_\lambda$ processes λ bits of input and produces λ bits of output. The hash setup $HGen^{\mathcal{H}}$ randomly selects a hash function $h \in \mathcal{H}_\lambda$ and outputs $hk = h$. The hash function $H^{\mathcal{H}}(hk = h, m) = h(m)$ maps m to $h(m)$, with the output interpreted as a field element of $\mathbb{Z}_p$.

Aggregation Algorithm. This "key accumulation" algorithm was developed by Delerablée, Paillier, and Pointcheval [14,15]. The DPP algorithm takes an input sequence of group elements $\{g^{\frac{r}{\gamma+x_i}}, x_i\}_{i\in[\ell]}$ and produces the output $g^{\prod_{i\in[\ell]}\frac{r}{\gamma+x_i}}$. It is used to aggregate individual signatures in both their scheme and ours.

Gen Algorithm. Gen is a probabilistic polynomial-time (PPT) algorithm that, given a security parameter λ (in unary), generates a λ-bit prime p, an efficient description of groups $\mathbb{G}, \mathbb{G}_T$ of order p, and a generator $g \in \mathbb{G}$. Additionally, it outputs an efficient non-degenerate bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ such that $e(g,g) \neq 1_{\mathbb{G}_T}$ and for all $a, b \in \mathbb{Z}_p$, $e(g^a, g^b) = e(g,g)^{ab}$.

The construction of the locally verifiable single-signer aggregate signature scheme [18] is as follows. Notice that, in this construction, LocalOpen algorithm generates a hint for only one message.

$\textsc{Setup}(1^\lambda, 1^B) \to (vk^{(local)}, vk, sk)$. The setup algorithm takes the security parameter λ and the upper bound B for aggregations as input. It generates bilinear group parameters $\Pi = (p, \mathbb{G}, \mathbb{G}_T, g, e(\cdot,\cdot))$ using $Gen(1^\lambda)$, and samples a random exponent $\alpha \in \mathbb{Z}_p^*$. The public parameters for message hashing are set as $hk \leftarrow HGen(1^\lambda)$. The key pair is defined as $vk = (\Pi, hk, \{g^{\alpha^i}\}_{i\in[B]})$ and $sk = (\Pi, hk, \alpha)$. The local verification key is $vk^{(local)} = (\Pi, hk, g^\alpha)$.

$\textsc{Sign}(sk, m) \to \sigma$. It parses the secret key and hashes the message as $h_m = H(hk, m)$. The signature is then computed as $g^{(\alpha+h_m)^{-1}}$, leveraging the knowledge of α.

$\textsc{Verify}(vk, m, \sigma) \to 0/1$. It parses the verification key and hashes the message as $h_m = H(hk, m)$. The verification involves checking if $e(\sigma, g^{\alpha} g^{h_m}) = e(g, g)$, where g^{α} is obtained from vk. If the check is successful, it outputs 1; otherwise, it outputs 0.

$\textsc{Aggregate}(vk, \{m_i, \sigma_i\}_i) \to \hat{\sigma}/\perp$. The aggregation algorithm first verifies each input signature σ_i, and outputs $\perp$ if any verification fails. Otherwise, it computes the aggregated signature as

$$\hat{\sigma} = \mathrm{DPP}(\{\sigma_i, x_i\}_i)$$

where $x_i = H(hk, m_i)$.

$\textsc{AggVerify}(vk, \{m_i\}_i, \hat{\sigma}) \to 0/1$. The algorithm parses the verification key and computes the hash values $x_i = H(hk, m_i)$ for each $i \in [\ell]$. It then constructs the polynomial P to determine the coefficients $\{\beta_i \in \mathbb{Z}_p\}_{i \in [\ell]}$:

$$P_{\{x_i\}_{i \in [\ell]}}(y) = \prod_{i \in [\ell]} (y + x_i) = \sum_{i=0}^{\ell} \beta_i y^i \pmod{p}.$$

Next, it verifies that $\ell \leq B$ and checks, and it outputs 1 if it is valid or 0 otherwise.

$$e\left(\hat{\sigma}, \prod_{i=0}^{\ell} (g^{\alpha^i})^{\beta_i}\right) = e(g, g).$$

$\textsc{LocalOpen}(vk, \{m_i\}_{i \in [\ell]}, j \in [\ell]) \to aux_j$. The local opening algorithm parses vk and computes the hash sequence $x_i = H(hk, m_i)$ for each $i \in [\ell]$, and coefficients $\{\tilde{\beta}_i \in \mathbb{Z}_p\}_{i \in [\ell-1]}$ similar to $\textsc{AggVerify}$ except that now it removes $(y + x_j)$. Finally, it computes

$$P_{\{x_i\}_{i \in [\ell] \setminus \{j\}}}(y) = \prod_{i \in [\ell] \setminus \{j\}} (y + x_i) = \sum_{i=0}^{\ell-1} \tilde{\beta}_i y^i \pmod{p}.$$

It then generates the auxiliary opening information $aux_j = (aux_{j,1}, aux_{j,2})$, where

$$aux_{j,1} = \prod_{i=0}^{\ell-1} (g^{\alpha^i})^{\tilde{\beta}_i}, \quad aux_{j,2} = \prod_{i=0}^{\ell-1} (g^{\alpha^{i+1}})^{\tilde{\beta}_i},$$

with g^{α^i} obtained from the verification key vk.

$\textsc{LocalAggVerify}(\hat{\sigma}, vk^{local}, m, aux) \to 0/1$. The algorithm parses the local verification key vk^{local} and auxiliary value $aux = (aux_1, aux_2)$, then computes the hash of the message as $h_m = H(hk, m)$. It checks the following two conditions,

$$e(\hat{\sigma}, aux_1^{h_m} aux_2) = e(g, g)$$

$$e(g^{\alpha}, aux_1) = e(g, aux_2)$$

where g^{α} is obtained from the local verification key vk. If both conditions hold, it outputs 1; otherwise, it outputs 0.

In the following section, we formally define our first contribution, multi-message locally verifiable aggregate signature schemes, and then dive into the details of our scheme. We also provide its security proof.

3 Multi-Message Locally Verifiable Aggregate Signature Schemes

In [18], the authors proposed a pairing-based locally verifiable aggregate signature scheme, which is described in the previous section. In that scheme, the LocalOpen algorithm generates an auxiliary value aux for only one message. Consequently, the local verification algorithm verifies whether the aggregated signature $\hat{\sigma}$ contains a signature for that single message m corresponding to aux. In this section, we extend this scheme to a multi-message setting. Next, we give a formal definition and the security definitions of our proposed scheme.

A locally verifiable aggregate signature scheme with multi-message setting is a tuple of seven algorithms, i.e., Setup, Sign, Verify, Aggregate, AggVerify, LocalOpen, LocalAggVerify, which are defined as follows:

$\textsc{Setup}(1^{\lambda}, 1^{B}) \rightarrow (vk^{local}, vk, sk)$. The setup algorithm takes the security parameter λ as input and outputs vk^{local} and a pair of signing and verification keys (vk, sk).

$\textsc{Sign}(sk, m) \rightarrow \sigma$. The signing algorithm takes a signing key sk and a message m as input and outputs a signature σ.

$\textsc{Verify}(vk, m, \sigma) \rightarrow 0/1$. The verification algorithm takes a verification key vk , a message m, and a signature σ as input. It outputs 1 if the signature is valid or 0 otherwise.

$\textsc{Aggregate}(vk, \{m_i, \sigma_i\}_i) \rightarrow \hat{\sigma}/\perp$. The signature aggregation algorithm takes a verification key vk, a sequence of tuples, each containing a message m_i and signature σ_i, and it outputs either an aggregated signature $\hat{\sigma}$ or $\perp$.

$\textsc{AggVerify}(vk, \{m_i\}_i, \hat{\sigma}) \rightarrow 0/1$. The aggregate verify algorithm takes a verification key vk, a sequence of messages m_i, and an aggregated signature $\hat{\sigma}$ as input, and outputs 1 if it is valid or 0 otherwise.

$\textsc{LocalOpen}(vk, \{m_i\}_{i\in[\ell]}, \{j_1, j_2, \ldots, j_k\} \subset [\ell]) \rightarrow aux$. The local opening algorithm takes a verification key vk, a sequence of messages m_i for $i \in [\ell]$, and indexes $\{j_1, j_2, \ldots, j_k\}$ as input. It outputs auxiliary information aux corresponding to the set of messages $\{m_{j_1}, m_{j_2}, \ldots, m_{j_k}\}$.

$\textsc{LocalAggVerify}(\hat{\sigma}, vk^{local}, m_{j_1}, m_{j_2}, \ldots, m_{j_k}, aux) \rightarrow 0/1$. The local aggregate verification algorithm takes an aggregated signature $\hat{\sigma}$, a local verification key vk^{local}, a set of messages $m_{j_1}, m_{j_2}, \ldots, m_{j_k}$, and auxiliary information aux as input. It outputs 1 if the aggregated signature $\hat{\sigma}$ contains the signatures for messages $\{m_{j_1}, m_{j_2}, \ldots, m_{j_k}\}$ under verification key vk, or not.

Now, we define the idea of security for the setup of multi-message locally verifiable aggregate signature scheme.

Definition 8 (Aggregated Unforgeability with Adversarial Opening). *A multi-message locally verifiable aggregate signature scheme MM-LVAS =(*Setup, Sign, Verify, Aggregate, AggVerify, LocalOpen, LocalAggVerify*) is considered a secure aggregate signature scheme against adversarial openings if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\textsc{LocalAggVerify}\left(\hat{\sigma}^*, vk^{local}, \{m_i^*\}_{i\in[k]}, aux^*\right) = 1 : \begin{array}{l}(vk, sk) \leftarrow \textsc{Setup}\left(1^\lambda\right) \\ \left(\hat{\sigma}^*, aux^*, \{m_i^*\}_{i\in[k]}\right) \leftarrow \mathcal{A}^{\textsc{Sign}(sk,\cdot)}\left(1^\lambda, vk\right)\end{array}\right] \leq negl(\lambda)$$

where $\mathcal{A}$ is admissible if at least one of the challenge messages $\{m_i^*\}_{i\in[k]}$ was not queried by $\mathcal{A}$ to the $\textsc{Sign}(sk, \cdot)$ oracle.

Definition 9 (Static Agg. Unforgeability with Adversarial Opening). *Before an adversary $\mathcal{A}$ receives the verification key vk, if it is restricted to provide both the message queries $\{m_i\}_{i\in[q]}$ and the challenge messages $\{m_i^*\}_{i\in[k]}$ at the beginning of the game in Definition 8, then we say that MM-LVAS is statically secure against adversarial openings.*

3.1 Description of our scheme

This section presents a generalized version of the single-signer locally verifiable aggregate signatures proposed by Goyal et al. [18]. In this construction, which is based on [9], a verifier can verify k out of ℓ messages at once using a generated common hint. Note that the DPP algorithm used for signature aggregation and the *HGen* function employed for message hashing are utilized in the same manner as in the original design here.

$\textsc{Setup}(1^\lambda, 1^B) \rightarrow (vk^{local}, vk, sk)$. The setup process requires input parameters: the security parameter λ and the maximum allowed number of aggregations, denoted as B. It then generates vk^{local}, vk, and sk. It samples the bilinear group elements $\Pi = (p, \mathbb{G}, \mathbb{G}_T, g, e(\cdot,\cdot)) \leftarrow Gen(1^\lambda)$. It also samples a random element $\alpha \leftarrow {\mathbb{Z}_p}^*$. It additionally generates public parameters for message hashing through the process: $\text{hk} \leftarrow HGen(1^\lambda)$. The key pair is defined as $vk = (\Pi, hk, {g^{\alpha^i}}_{i\in[B]})$ and $sk = (\Pi, hk, \alpha)$. Furthermore, it establishes the local verification key as $vk^{local} = (\Pi, hk, g^\alpha)$.

$\textsc{Sign}(sk, m) \rightarrow \sigma$. It parses sk as (Π, hk, α) and computes the hash of the message as $x = H(hk, m)$. The signature is then computed as $g^{(\alpha+x)^{-1}}$. Notice that this computation is doable since α is known.

$\textsc{Verify}(vk, m, \sigma) \rightarrow 0/1$. It parses vk as $(\Pi, hk, {g^{\alpha^i}}_{i\in[B]})$, and computes the hash of the message as $x = H(hk, m)$. It verifies whether $e(\sigma, g^{\alpha} g^{x}) = e(g, g)$, where g^{α} is obtained from the verification key vk^{local}. It outputs 1 if the verification is valid or 0 otherwise.

$\textsc{Aggregate}(vk, \{(m_i, \sigma_i)\}_{i\in[\ell]}) \rightarrow \hat{\sigma}/\perp$. The aggregation algorithm verifies each input signature σ_i,and returns $\perp$ if any fail. If all verifications pass, the aggregated signature is calculated as

$$\hat{\sigma} = \mathrm{DPP}(\{\sigma_i, x_i\}_i) = g^{\prod_i (\alpha + x_i)^{-1}},$$

where $x_i = H(hk, m_i)$.

$\textsc{AggVerify}(vk, \{m_i\}_{i\in[\ell]}, \hat{\sigma}) \rightarrow 0/1$. The aggregated signature verification algorithm parses the verification key vk as $(\Pi, \mathrm{hk}, {g^{\alpha^i}}_{i\in[B]})$, and calculates the hash sequence as $x_i = H(hk, m_i)$ for all $i \in [\ell]$ where ℓ represents the number of aggregated messages. It symbolically computes the following polynomial P to determine the coefficients $\{\beta_i \in \mathbb{Z}_p\}_{i\in[\ell]\cup\{0\}}$,

$$P_{\{x_i\}_{i\in[\ell]}}(y) = \prod_{i\in[\ell]} (y + x_i) = \sum_{i=0}^{\ell} \beta_i y^i \pmod{p}. \tag{1}$$

Finally, it checks that $\ell \leq B$ and verifies

$$e\left(\hat{\sigma}, \prod_{i=0}^{\ell} (g^{\alpha^i})^{\beta_i}\right) = e(g, g),$$

where g^{α^i} are taken from the verification key vk. If this equation holds, it outputs 1; otherwise, it outputs 0.

$\textsc{LocalOpen}(vk, \{m_i\}_{i\in[\ell]}, \{j_1, j_2, \ldots, j_k\} \subset [\ell]) \rightarrow aux$. The local opening algorithm parses vk as $(\Pi, \mathrm{hk}, {g^{\alpha^i}}_{i\in[B]})$ and computes the sequence of hash messages as $x_i = H(hk, m_i)$ for all $i \in [\ell] \setminus \{j_1, j_2, \ldots, j_k\}$. It symbolically computes the following polynomial P to determine the coefficients $\{\tilde{\beta}_i \in \mathbb{Z}_p\}_{i\in[\ell-k]\cup\{0\}}$,

$$P_{\{x_i\}_{i\in[\ell]\setminus\{j_1,j_2,\ldots,j_k\}}}(y) = \prod_{i\in[\ell]\setminus\{j_1,j_2,\ldots,j_k\}} (y + x_i) = \sum_{i=0}^{\ell-k} \tilde{\beta}_i y^i \pmod{p}. \tag{2}$$

It then calculates the auxiliary opening information $aux = (aux_0, aux_1, \ldots, aux_k)$ are computed as

$$aux_0 = \prod_{i=0}^{\ell-k} (g^{\alpha^i})^{\tilde{\beta}_i}, \quad aux_1 = \prod_{i=0}^{\ell-k} (g^{\alpha^{i+1}})^{\tilde{\beta}_i}, \ldots, aux_k = \prod_{i=0}^{\ell-k} (g^{\alpha^{i+k}})^{\tilde{\beta}_i},$$

where g^{α^i} is obtained from the verification key vk. In general, aux_t for $t \in [k] \cup \{0\}$ is computed as

$$aux_t = \prod_{i=0}^{\ell-t} (g^{\alpha^{i+t}})^{\tilde{\beta}_i}.$$

LocalAggVerify($\hat{\sigma}$, vk^{local}, $m_{j_1}, m_{j_2}, \ldots, m_{j_k}, aux$) $\rightarrow$ 0/1. The local verification algorithm parses the local verification key vk^{local} as $(\Pi, \mathrm{hk}, g^{\alpha})$, as well as the auxiliary opening $aux = (aux_0, aux_1, \ldots, aux_k)$. It computes the message hashes as $x_i = H(hk, m_i)$ for $i \in \{j_1, j_2, \ldots, j_k\}$. It symbolically computes the following polynomial Q to determine the coefficients $\{A_i \in \mathbb{Z}_p\}_{i \in [k] \cup \{0\}}$,

$$Q_{\{x_{j_i}\}_{i \in [k]}}(y) = \prod_{i \in [k]} (y + x_{j_i}) = \sum_{i=0}^{k} A_i y^i \pmod{p}. \tag{3}$$

Finally, it verifies the following $k+1$ conditions:

$$\begin{aligned}
&e(\hat{\sigma}, {aux_0}^{A_0} \cdot {aux_1}^{A_1} \cdots {aux_k}^{A_k}) = e(g, g) \\
&e(g^{\alpha}, aux_0) = e(g, aux_1) \\
&e(g^{\alpha}, aux_1) = e(g, aux_2) \\
&e(g^{\alpha}, aux_2) = e(g, aux_3) \\
&\vdots \\
&e(g^{\alpha}, aux_{k-1}) = e(g, aux_k)
\end{aligned} \tag{4}$$

where g^{α} is taken from the local verification key vk^{local}. It outputs 1 if it is valid or 0 otherwise.

Remark 1. Note that the verification process only needs g^{α} while checking the equation, meaning the local verification key vk^{local} would be sufficient.

A locally verifiable aggregate signature scheme with multi-message setting is considered correct and compact if for all $\lambda, \ell \in \mathbb{N}$, given every verification-signing key pair $(vk, sk) \leftarrow \text{Setup}(1^{\lambda})$, messages m_i, and every signature $\sigma_i \leftarrow \text{Sign}(sk, m_i)$ for $i \in [\ell]$, the following conditions hold:

1. *Correctness of signing.* This follows from the fact that $e(\sigma, g^{\alpha} g^{x}) = e(g, g)$ where $x = H(hk, m)$.

$$\begin{aligned}
e(\sigma, g^{\alpha} g^{x}) &= e(g^{(\alpha+x)^{-1}}, g^{\alpha} g^{x}) \\
&= e(g^{(\alpha+x)^{-1}}, g^{(\alpha+x)}) \\
&= e(g, g^{(\alpha+x)(\alpha+x)^{-1}}) \\
&= e(g, g)
\end{aligned}$$

Correctness of aggregation. Consider a sequence of messages $m_1, \ldots, m_t$ and their corresponding signatures $\sigma_i = g^{(\alpha+x_i)^{-1}}$ for $i \in [\ell]$, where $x_i = H(hk, m_i)$. It is known that these signatures are aggregated as $\hat{\sigma} = \mathrm{DPP}(\{\sigma_i, x_i\}_i)$. According to the correctness of the key accumulation algorithm from [14, 15], we have $\hat{\sigma} = g^{\prod_i (\alpha+x_i)^{-1}}$. The verification of the aggregated signature is as follows:

$$e\left(\hat{\sigma}, \prod_{i=0}^{\ell} (g^{\alpha^i})^{\beta_i}\right) = e(g, g),$$

where β_i values are such that $\sum_{i=0}^{\ell} \beta_i y^i = \prod_{i\in[\ell]}(y + x_i) \mod p$. Thus, we have

$$\prod_{i=0}^{\ell} (g^{\alpha^i})^{\beta_i} = g^{\sum_{i=0}^{\ell} \alpha^i \beta_i} = g^{\prod_{i\in[\ell]}(\alpha+x_i)}.$$

Therefore, for honestly computed and aggregated signatures, the above verification check succeeds, ensuring correctness.

2. *Correctness of local verification.* The correctness of all of the equalities in (4) except for the first one is trivial. Thus, we will only show the correctness of the first one.

$$\begin{aligned}
e(\hat{\sigma}, aux_0{}^{A_0} \ldots aux_k{}^{A_k}) &= e\left(\hat{\sigma}, \left(\prod_{i=0}^{\ell-k} (g^{\alpha^i})^{\tilde{\beta}_i}\right)^{A_0} \cdots \left(\prod_{i=0}^{\ell-k} (g^{\alpha^{i+k}})^{\tilde{\beta}_i}\right)^{A_k}\right) \\
&= e\left(\hat{\sigma}, \left(\prod_{i=0}^{\ell-k} (g^{\alpha^i})^{\tilde{\beta}_i}\right)^{A_0} \cdots \left(\prod_{i=0}^{\ell-k} (g^{\alpha^i})^{\tilde{\beta}_i \alpha^k}\right)^{A_k}\right) \\
&= e\left(\hat{\sigma}, \prod_{i=0}^{\ell-k} (g^{\alpha^i})^{A_0\tilde{\beta}_i} \cdots \prod_{i=0}^{\ell-k} (g^{\alpha^i})^{A_k\tilde{\beta}_i\alpha^k}\right) \\
&= e\left(\hat{\sigma}, \prod_{i=0}^{\ell-k} (g^{\alpha^i})^{\tilde{\beta}_i(A_0 + A_1\alpha + \cdots + A_k\alpha^k)}\right) \\
&= e\left(\hat{\sigma}, \prod_{i=0}^{\ell-k} (g^{\alpha^i})^{\tilde{\beta}_i \sum_{t=0}^{k} A_t\alpha^t}\right) \\
&= e\left(\hat{\sigma}, \prod_{i=0}^{\ell-k} g^{\alpha^i\tilde{\beta}_i \prod_{t\in[k]}(\alpha+x_{j_t})}\right) \\
&= e\left(\hat{\sigma}, g^{\sum_{i=0}^{\ell-k}(\alpha^i\tilde{\beta}_i \prod_{t\in[k]}(\alpha+x_{j_t}))}\right) \\
&= e\left(\hat{\sigma}, g^{\prod_{t\in[k]}(\alpha+x_{j_t})\cdot\sum_{i=0}^{\ell-k}\alpha^i\tilde{\beta}_i}\right) \\
&= e\left(\hat{\sigma}, g^{\prod_{t\in[k]}(\alpha+x_{j_t})\cdot\prod_{i\in[\ell]\setminus\{j_1,j_2,\ldots,j_k\}}(\alpha+x_i)}\right) \\
&= e\left(g^{\prod_{i=1}^{\ell}(\alpha+x_i)^{-1}}, g^{\prod_{i=1}^{\ell}(\alpha+x_i)}\right) \\
&= e(g, g)
\end{aligned}$$

3.2 Security Proof

Theorem 1 (Static Unforgeability). *Static unforgeability and static aggregated unforgeability are satisfied by MM-LVAS if the DHI assumption holds and* $(HGen, H)$ *is an identity hash.*

In addition, static aggregated unforgeability with adversarial openings is satisfied by MM-LVAS if the BDHI assumption holds and $(HGen, H)$ *is an identity hash.*

Proof. Note that aggregated unforgeability is more general than regular unforgeability, so here we consider the proof of aggregated unforgeability for our scheme. However, the aggregation technique of our signature scheme is the same as the original scheme in [18]. Since the entire proof is the same, we recommend readers look at that paper to get more detailed information about the proof of this part.

Here, we will show that our signature scheme, which relies on the hardness of BDHI, satisfies aggregated unforgeability with adversarial openings. Assume that there is a PPT adversary $\mathcal{A}$ that breaks aggregated unforgeability with adversarial openings with non-negligible probability ϵ. Based on this, we will construct a PPT adversary $\mathcal{B}$ that breaks the q-BDHI assumption in $\mathbb{G}_T$ with $\epsilon - negl$ probability for some negligible function $negl$.

Since we demonstrate only static security, the adversary $\mathcal{A}$ submits all of its signature queries $Q := \{m_i\}_{i\in[q_s]}$ together with the challenge messages $C := \{m_i^*\}_{i\in[k]}$ at the start of reduction algorithm $\mathcal{B}$. Since $\mathcal{A}$ is admissible, there exists an index $j^* \in [k]$ such that $m_{j^*}^* \in C \setminus Q$. We set $S := Q \cup C \setminus \{m_{j^*}^*\}$. Let B be the aggregation bound. The reduction algorithm $\mathcal{B}$ breaks the q-BDHI assumption in $\mathbb{G}_T$, where the hardness parameter q is such that $q \geq |S| + B$.

The BDHI challenger generates the bilinear group parameter set $\Pi = (p, \mathbb{G}, \mathbb{G}_T, h, e(\cdot,\cdot))$, and a sequence of group elements $\{h_i = h^{a^i}\}_{i=0}^{q}$ for a randomly selected element $a \leftarrow {\mathbb{Z}_p}^*$ and sends $(\Pi, h_1, \ldots, h_q)$ to $\mathcal{B}$.

Now, $\mathcal{B}$ will compute the signatures of the messages in Q and verification key $vk = \{g_i = g^{\alpha^i}\}_{i=1}^{B}$ in order to send back to adversary $\mathcal{A}$.

We will first focus on how the verification key is computed by $\mathcal{B}$. The reduction $\mathcal{B}$ sets the signing key $\alpha = a - m_{j^*}^*$ and the base group element $g = h_0^{\delta \prod_{m\in S}(\alpha+m)} = h_0^{\delta \prod_{m\in S}(a-m_{j^*}^*+m)}$ after sampling a random exponent $\delta \in \mathbb{Z}_p^*$. Then, for each $i \in [q]$, we have $g_i = h_0^{\delta(a-m_{j^*}^*)^i \prod_{m\in S}(a-m_{j^*}^*+m)}$. For ease of notation, we will use g, h as g_0, h_0, respectively. In order to compute each $g_j, j \in [B]$, $\mathcal{B}$ needs to define the following set of polynomials.

$$\forall j \in [0, B], \quad P_j(X) = \left(X - m_{j^*}^*\right)^j \prod_{m\in\mathcal{S}} \left(X - m_{j^*}^* + m\right) = \sum_{i=0}^{|\mathcal{S}|+j} \beta_j^{(i)} X^i \pmod p$$

Then, $\mathcal{B}$ computes each g_j as shown below:

$$g_j = h^{\delta P_j(a)} = h^{\delta \sum_{i=0}^{|\mathcal{S}|+j} \beta_j^{(i)} a^i} = \prod_{i=0}^{|\mathcal{S}|+j} h^{\delta\beta_j^{(i)} a^i} = \prod_{i=0}^{|\mathcal{S}|+j} h_i^{\delta\beta_j^{(i)}} \pmod p$$

It sets the verification key vk and vk^{local} as $vk = (p, \mathbb{G}, \mathbb{G}_T, g_0, e(\cdot,\cdot), hk = \epsilon, \{g_i\}_{i=1}^{B})$ and $\text{vk}^{local} = (p, \mathbb{G}, \mathbb{G}_T, g_0, e(\cdot,\cdot), hk = \epsilon, g_1)$. Note that g_0 is a generator, computed as previously mentioned, different from the one supplied by the BDHI challenger.

Now, $\mathcal{B}$ needs to calculate the signatures of all query messages $m_1, \ldots, m_{q_s}$ as

$$\forall i \in [q_s], \quad \sigma_i = g^{(a-m_{j^*}^*+m_i)^{-1}} = h^{\delta \frac{P_0(a)}{a-m_{j^*}^*+m_i}} = h^{\delta \prod_{m\in S\setminus\{m_i\}}(a-m_{j^*}^*+m)}.$$

Let Q_{-i} be the following polynomial with coefficients $\{\gamma_i^{(j)} \in {\mathbb{Z}_p}^*\}_{j=0}^{|S|-1}$

$$\forall i \in [q_s], \quad Q_{-i}(X) = \prod_{m\in\mathcal{S}\setminus\{m_i\}} \left(X - m_{j^*}^* + m\right) = \sum_{j=0}^{|\mathcal{S}|-1} \gamma_i^{(j)} X^j \pmod p.$$

Then,

$$\sigma_i = h^{\delta Q_{-i}(a)} = h^{\delta \sum_{j=0}^{|\mathcal{S}|-1} \gamma_i^{(j)} a^j} = \prod_{j=0}^{|\mathcal{S}|-1} h_j^{\delta\gamma_i^{(j)}}.$$

Now, the reduction algorithm $\mathcal{B}$ sends vk and $\{\sigma_i\}_{i\in[q_s]}$ to adversary $\mathcal{A}$. $\mathcal{A}$ then sends its forged signature $\hat{\sigma}^*$ and $aux^* = (aux_0, aux_1, \ldots, aux_k)$. $\mathcal{B}$ then checks if $\hat{\sigma}^*$ is a valid signature by executing the local verification algorithm. It aborts if it is an invalid signature. Otherwise, the next step for $\mathcal{B}$ is to compute $e(h,h)^{\frac{1}{a}}$.

Note that for all $i \in [k], aux_i = aux_0^{\alpha^i}$ from the equation (4). Then, also using the equation (3), we get

$$\prod_{i=0}^{k} aux_i^{A_i} = \prod_{i=0}^{k} aux_0^{A_i\alpha^i} = aux_0^{\sum_{i=0}^{k} A_i\alpha^i} = aux_0^{\prod_{i\in[k]}(\alpha+m_i^*)}.$$

Using the equation above and the first equation of (4), we get

$$e(g,g) = e\left(\widehat{\sigma}^*, aux_0^{\prod_{i\in[k]}(a-m_{j^*}^*+m_i^*)}\right) = e\left(\widehat{\sigma}^*, aux_0^{\prod_{m\in C}(a-m_{j^*}^*+m)}\right).$$

Moreover, we have

$$e(g,g) = e\left(h^{\delta P_0(a)}, h^{\delta P_0(a)}\right) = e\left(h,h\right)^{\delta^2 P_0(a))^2}.$$

Therefore, by merging the previous two equations, we get

$$e(\widehat{\sigma}^*, aux_0) = e(h,h)^{\delta^2 \frac{P_0(a)^2}{\prod_{m\in C}(a-m_{j^*}^*+m)}}.$$

Then, $\mathcal{B}$ computes the polynomial

$$P^*(X) = \frac{P_0(X).P_0(X)}{\prod_{m\in C}(X-m_{j^*}^*+m)} = \frac{\beta_{-1}^*}{X} + \sum_{i=0}^{|S\setminus C|+|\mathcal{S}|-1} \beta_i^* X^i \pmod p$$

so that $e(\widehat{\sigma}^*, aux_0) = e(h,h)^{\delta^2 P^*(a)}$. As we continue from the previous equation,

$$\begin{aligned}
e(\widehat{\sigma}^*, aux_0) &= e(h,h)^{\delta^2\left(\frac{\beta^*_{-1}}{a} + \sum_{i=0}^{|S\setminus C|+|\mathcal{S}|-1} \beta_i^* a^i\right)} \\
&= e(h,h)^{\delta^2 \frac{(\beta^*_{-1})}{a}} \cdot e(h,h)^{\delta^2 \sum_{i=0}^{|S\setminus C|+|\mathcal{S}|-1} \beta_i^* a^i} \\
&= e(h,h)^{\delta^2(\frac{\beta^*_{-1}}{a})} \cdot \prod_{i=0}^{|S\setminus C|+|\mathcal{S}|-1} e(h,h)^{\delta^2 \beta_i^* a_i} \\
&= e(h,h)^{\delta^2(\frac{\beta^*_{-1}}{a})} \cdot \prod_{i=0}^{|S\setminus C|+|\mathcal{S}|-1} e(h^{a^{\lceil i/2\rceil}}, h^{a^{\lfloor i/2\rfloor}})^{\delta^2 \beta_i^*} \\
&= e(h,h)^{\delta^2(\frac{\beta^*_{-1}}{a})} \cdot \prod_{i=0}^{|S\setminus C|+|\mathcal{S}|-1} e(h_{\lceil i/2\rceil}, h_{\lfloor i/2\rfloor})^{\delta^2 \beta_i^*}.
\end{aligned}$$

Thus, the reduction algorithm $\mathcal{B}$ outputs BDHI solution as

$$e(h,h)^{1/a} = \left(e(\widehat{\sigma}^*, aux_0)^{1/\delta^2} \cdot \prod_{i=0}^{|S\setminus C|+|\mathcal{S}|-1} e\left(h_{\lceil i/2\rceil}, h_{\lfloor i/2\rfloor}\right)^{-\beta_i^*} \right)^{1/\beta^*_{-1}}.$$

As a result, whenever there exists an admissible adversary $\mathcal{A}$, the algorithm $\mathcal{B}$ can break BDHI assumption with $\epsilon - negl$ probability for some negligible function $negl$ for $q \geq |S| + B$.

□

Next, we demonstrate how MM-LVAS provides full unforgeability if the message hashing is instantiated in the ROM.

Theorem 2 (Full Unforgeability). *MM-LVAS meets full unforgeability and aggregated unforgeability if the DHI assumption holds and $(HGen, H)$ is instantiated in the ROM.*

Also, MM-LVAS meets full aggregated unforgeability with adversarial openings if the BDHI assumption holds and $(HGen, H)$ is instantiated in the ROM.

As in the previous proof, it will be sufficient to prove the part for adversarial openings in this section.

Proof. We can reduce full aggregated unforgeability with adversarial openings to BDHI in the ROM with some modifications as in the previous proof.

Assume that there is a PPT adversary $\mathcal{A}$ that breaks full unforgeability with adversarial openings with non-negligible probability ϵ. Then, we will construct a PPT adversary $\mathcal{B}$ that breaks q-BDHI assumption in $\mathbb{G}_T$ with $\epsilon/Q^{RO} - negl$ probability for some negligible function $negl$. Here, Q^{RO} denotes the number of queries $\mathcal{A}$ makes to the random oracle. The reduction algorithm $\mathcal{B}$ is explained below. It is constructed similar to that in the proof of Theorem 1, except that it now exploits programmability.

Remember from the previous proof that the set of signature queries and challenge messages are $Q := \{m_i\}_{i\in[q_s]}$ and $C := \{m_i^*\}_{i\in[k]}$, respectively. At the

beginning of the game, the adversary $\mathcal{A}$ queries the random oracle for all of the messages in the set $U := Q \cup C$. At the same time, the reduction algorithm $\mathcal{B}$ programs the random oracle, meaning samples the hash values associated with the messages. Now, $\mathcal{B}$ chooses an element $m^*_{j^*}$ such that $j^* \in [k]$ expecting that $m^*_{j^*} \notin Q$ and sets $\alpha = a - h_{m^*_{j^*}}$ implicitly.

The q-BDHI challenger samples the bilinear parameters $\Pi = (p, \mathbb{G}, \mathbb{G}_T, h, e(\cdot,\cdot))$, and a sequence of group elements $\{h_i = h^{a^i}\}_{i=0}^{q}$ for a randomly selected element $a \leftarrow {\mathbb{Z}_p}^*$ and sends $(\Pi, h_1, \ldots, h_q)$ to $\mathcal{B}$.

The reduction algorithm $\mathcal{B}$ computes the verification key vk, computed in the same way as in the previous proof, and then sends it to the adversary $\mathcal{A}$. After that, $\mathcal{A}$ sends the hash values of the signature query set Q, which is named H_Q, to the reduction algorithm $\mathcal{B}$. The reduction algorithm aborts if $h_{m^*_{j^*}} \in H_Q$. Otherwise, $\mathcal{B}$ sends the signatures of queried messages to the adversary $\mathcal{A}$. Similar to the case of vk, $\mathcal{B}$ computes the signatures in an analogous way to the previous proof. The remaining operations are the same as the proof of the Theorem 1. As a result, the probability that $\mathcal{B}$ breaks q-BDHI assumption turns out to be $\epsilon/Q^{RO} - negl$ for some negligible function $negl$. Note that we have $1/Q^{RO}$ factor emerges from the probability of $\mathcal{B}$ guessing $m^*_{j^*}$ correctly. □

In the next section, we first provide the formal definition of locally verifiable multi-signatures. Subsequently, we provide detailed information about the scheme and its security proof.

4 Locally Verifiable Multi-Signatures

In this section, we integrate the notion of local verification into multi-signature schemes. The multi-signature scheme that we choose to modify is the one given in [10]. Given a locally verifiable multi-signature scheme with n signers, the verifier can verify that the signatures of k out of n signers are included in the multi-signature using an auxiliary value aux.

A locally verifiable multi-signature scheme LVMS is a tuple of seven algorithms, i.e., Setup, Sign, Verify, Aggregate, AggVerify, LocalOpen, LocalAggVerify, which are defined as follows:

$\textsc{Setup}(1^\lambda) \rightarrow (vk, sk)$. The setup algorithm takes the security parameter λ and generates the bilinear group elements $\Pi = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. It outputs (sk, vk) for every signer $i \in [n]$.

$\textsc{Sign}(sk, m, VK) \rightarrow \sigma$. The signing algorithm takes a secret key sk, a message m, and the set of verification keys $VK = \{vk_1, vk_2, \ldots, vk_n\}$ as input, and it computes the signature σ.

$\textsc{Verify}(\sigma, m, VK) \rightarrow 0/1$. The verification algorithm takes a signature σ, a message m, and the set of verification keys $VK = \{vk_i\}_{i=1}^{n}$ and outputs 1 if the signature is valid or 0 otherwise.

$\textsc{Aggregate}(\{\sigma_i\}_{i=1}^n, VK, m) \rightarrow \hat{\sigma}/\perp$. The aggregation algorithm takes the individual signatures $\{\sigma_i\}_{i=1}^n$, the set of verification keys VK, and the message m as input. It outputs either a multi-signature $\hat{\sigma}$ or $\perp$.

$\textsc{AggVerify}(\hat{\sigma}, \{vk_i\}_{i=1}^n, m) \rightarrow 0/1$. The aggregation verification algorithm takes the multi-signature $\hat{\sigma}$, the set of verification keys $VK = \{vk_i\}_{i=1}^n$ and the message m as input. It outputs 1 if it is valid or 0 otherwise.

$\textsc{LocalOpen}(\{vk_i\}_{i=1}^n, j_1, j_2, \cdots, j_k, m) \rightarrow aux$. The local opening algorithm takes all verification keys $\{vk_i\}_{i=1}^n$, indices $j_1, j_2, \ldots, j_k$ and the message m as input. It outputs an auxiliary value aux corresponding to the signers.

$\textsc{LocalAggVerify}(aux, \{vk_{j_i}\}_{i=1}^k, m, \hat{\sigma}, VK) \rightarrow 0/1$. The local verification algorithm takes an auxiliary information aux, verification keys $\{vk_{j_i}\}_{i=1}^k$, the message m, a multi-signature $\hat{\sigma}$ and the entire set of verification keys VK as input. It outputs 1 if it is valid or 0 otherwise.

Definition 10 (Unforgeability of Multisignature with Adversarial Opening). *An adversary $\mathcal{A}$ is called a $(\tau, q_S, q_H, \epsilon)$-forger for locally verifiable multi-signature scheme LVMS = (*$\textsc{Setup}$*,* $\textsc{Sign}$*,* $\textsc{Verify}$*,* $\textsc{Aggregate}$*,* $\textsc{AggVerify}$*,* $\textsc{LocalOpen}$*,* $\textsc{LocalAggVerify}$*) if it runs in time τ, makes q_S signing queries, makes q_H random oracle queries and satisfies*

$$\Pr\left[\textsc{LocalAggVerify}(aux^*, \{vk_i\}_{i\in[k]}, m^*, \hat{\sigma}^*, VK^*) = 1 : \begin{array}{l}(vk, sk) \leftarrow \textsc{Setup}(1^\lambda) \\ (\hat{\sigma}^*, \{vk_i\}_{i\in[k]}, m^*, VK^*, aux^*) \leftarrow \mathcal{A}^{\textsc{Sign}(sk,\cdot)}(1^\lambda, vk)\end{array}\right] \geq \epsilon$$

where $vk \in \{vk_i\}_{i=1}^k \subset VK^*$. If there exist no such an adversary, we say that LVMS is $(\tau, q_S, q_H, \epsilon)$-unforgeable. Note that $\mathcal{A}$ is admissible if m^* was not queried by $\mathcal{A}$ to the $\textsc{Sign}(sk, \cdot)$ oracle.

4.1 Description of Our Scheme

In this section, we propose a multi-signature scheme with local verification property, which we refer to as LVMS. Our proposed scheme is derived from the MSP scheme proposed by Boneh, Drijvers, and Neven in [10]. Note that since this scheme is a multi-signature scheme, the number of signed messages is only one. We provide the following scheme for the local verification of k out of n signatures. Assume the hash functions $H_0 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q$. Remember that we refer the set of all verification keys $\{vk_1, vk_2, \ldots, vk_n\}$ as VK.

$\textsc{Setup}(1^\lambda) \rightarrow (vk_i, sk_i)$. The setup algorithm samples the bilinear group elements $\Pi = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. It also generates secret keys randomly such that $sk_i \xleftarrow{\$} \mathbb{Z}_q$, computes $vk_i \leftarrow {g_2}^{sk_i}$ and outputs (sk_i, vk_i) for each signer i.

$\textsc{Sign}(sk_i, m, VK) \rightarrow \sigma_i$. The signing algorithm computes $\sigma_i \leftarrow H_0(m)^{a_i \cdot sk_i}$, where $a_i \leftarrow H_1(vk_i, VK)$.

$\textsc{Verify}(\sigma_i, m, vk_i, VK) \rightarrow 0/1$. The verification algorithm verifies whether the following equation holds or not and outputs 1 if it is valid or 0 otherwise.

$$e(\sigma_i, g_2) \stackrel{?}{=} e(H_0(m), {vk_i}^{a_i})$$

$\textsc{Aggregate}(\{\sigma_i\}_{i=1}^n, VK, m) \rightarrow \hat{\sigma}/\perp$. The aggregation algorithm firstly verifies all the input signatures σ_i, and it returns $\perp$ if any fail. Otherwise, it computes the multi-signature as follows:

$$\hat{\sigma} \leftarrow \prod_{i=1}^{n} \sigma_i$$

$\textsc{AggVerify}(\hat{\sigma}, \{vk_i\}_{i=1}^n, m) \rightarrow 0/1$. The aggregate verification algorithm checks the following equality and outputs 1 if it is valid or 0 otherwise.

$$e(\hat{\sigma}, g_2) \stackrel{?}{=} e\left(H_0(m), \prod_{i=1}^{n} vk_i^{a_i}\right)$$

$\textsc{LocalOpen}(\{vk_i\}_{i=1}^n, j_1, j_2, \cdots, j_k, m) \rightarrow aux$. The local opening algorithm takes all verification keys, indices $\{j_i\}_{i=1}^k \subset [n]$ and the message m . It outputs an auxiliary value aux corresponding to the signers in the set $\{j_i\}_{i=1}^k$.

$$aux = e\left(H_0(m), \prod_{\substack{i=1, \\ i \neq j_1, j_2, \ldots, j_k}}^{n} vk_i^{a_i}\right)$$

$\textsc{LocalAggVerify}(aux, \{vk_{j_i}\}_{i=1}^k, m, \hat{\sigma}, VK) \rightarrow 0/1$. The local verification function takes an auxiliary information aux, verification keys $\{vk_{j_i}\}_{i=1}^k$, the message m and a multi-signature $\hat{\sigma}$ as input. It then checks the following equation and outputs 1 if it is valid or 0 otherwise.

$$e(\hat{\sigma}, g_2) \stackrel{?}{=} aux \cdot e\left(H_0(m), \prod_{i=1}^{k} vk_{j_i}^{a_{j_i}}\right)$$

4.2 Security Proof

Theorem 3. *LVMS is an unforgeable multi-signature scheme under the computational co-Diffie-Hellman problem in the random-oracle model. More precisely, LVMS is $(\tau, q_S, q_H, \epsilon)$-unforgeable in the random-oracle model if $q > 8q_H/\epsilon$ and if co-CDH is $((\tau + q_H \cdot \tau_{\exp_1} + q_S \cdot \tau_{exp_1}) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon), \epsilon/(8q_H))$-hard, where l is the maximum number of signers involved in a single multi-signature, $\tau_{\exp_1}$ and $\tau_{\exp_2}$ denote the time required to compute exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $\tau_{\exp_1^i}$ and $\tau_{\exp_2^i}$ denote the time required to compute i-multi-exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.*

Proof. Suppose we have a $(\tau, q_S, q_H, \epsilon)$-forger $\mathcal{F}$ against the LVMS scheme. Consider an input generator IG that generates random tuples $(A, B_1, B_2) = (g_1^{\alpha}, g_1^{\beta}, g_2^{\beta})$, where $\alpha, \beta \in \mathbb{Z}_q$ are chosen uniformly at random. Finally, consider an algorithm $\mathcal{A}$ that, given the input (A, B_1, B_2) and randomness $f = (\rho, h_1, \ldots, h_{q_S})$, proceeds as follows.

Algorithm $\mathcal{A}$ selects an index $k \in \{1, \ldots, q_H\}$ randomly and executes the forger $\mathcal{F}$ using the input $vk^* \leftarrow B_2$ with random tape ρ. It addresses the i-th H_0 query from $\mathcal{F}$ by randomly sampling $r_i \in \mathbb{Z}_q$ and returning $g_1^{r_i}$ if $i \neq k$. The k-th H_0 query is answered by returning A. Without loss of generality, it is assumed that $\mathcal{F}$ does not repeat any H_0 queries. $\mathcal{A}$ responds to $\mathcal{F}$'s H_1 queries by discriminating them as follows:

1. If this is a query on (vk, VK) with $vk \in VK$ and $vk^* \in VK$, and this is the first such query with VK,
   (a) $\mathcal{A}$ selects a random value for $H_1(vk_j, VK)$ for each $vk_j \neq vk^*$
   (b) $\mathcal{A}$ sets $H_1(vk^*, VK)$ to h_i, where the query is the i-th query of this type
   (c) $\mathcal{A}$ returns $H_1(vk, VK)$.
2. If this is a query on (vk, VK) with $vk \in VK$ and $vk^* \in VK$, and a query with VK has already been made before, $\mathcal{A}$ returns the previously assigned value.
3. For any other kind of queries, $\mathcal{A}$ simply returns a random value in $\mathbb{Z}_q$.

When $\mathcal{F}$ makes a signing query on a message m with signers VK, $\mathcal{A}$ looks up $H_0(m)$. If this is A, then $\mathcal{A}$ aborts with output $(0, \perp)$. Otherwise, it must be of form g_1^r and $\mathcal{A}$ can simulate the honest signer by computing $\sigma \leftarrow B_1^r$.

When $\mathcal{F}$ fails to output a successful forgery, then $\mathcal{A}$ outputs $(0, \perp)$. If $\mathcal{F}$ successfully outputs a forgery for a message m so that $H_0(m) \neq A$, then $\mathcal{A}$ again outputs $(0, \perp)$. Otherwise, $\mathcal{F}$ has output a forgery $(aux, \{vk_i\}_{i=1}^k, m, \hat{\sigma}, VK)$ such that

$$e(\hat{\sigma}, g_2) = aux \cdot e\left(A, \prod_{i=1}^{k} vk_i^{a_i}\right)$$

where $vk^* \in \{vk_i\}_{i=1}^k \subset VK$ and $a_i = H_1(vk_i, VK)$.

Let j_f be the index such that $H_1(vk^*, VK) = h_{j_f}$. Then, $\mathcal{A}$ outputs $(J = \{j_f\}, \{(\hat{\sigma}, VK, aux, h_{j_f})\})$.

The running time of algorithm $\mathcal{A}$ is the running time of algorithm $\mathcal{F}$ plus the additional computations made by $\mathcal{A}$. Let q_H denote the total number of hash queries made by $\mathcal{F}$, including queries to both H_0 and H_1. $\mathcal{A}$ requires a single exponentiation in $\mathbb{G}_1$ to respond to H_0 queries; hence, it requires a maximum of $q_H \cdot \tau_{\exp_1}$ to answer the hash queries. For signing queries with a VK of size at most l, $\mathcal{A}$ computes one exponentiation in $\mathbb{G}_1$, which costs τ_{exp_1}. Therefore, the total signing queries cost is $q_S \cdot \tau_{exp_1}$. As a result, the total runtime of $\mathcal{A}$ is $\tau + q_H \cdot \tau_{\exp_1} + q_S \cdot \tau_{exp_1}$. $\mathcal{A}$'s overall success probability is $\epsilon_{\mathcal{A}} = \epsilon / q_H$, which is the probability that $\mathcal{F}$ succeeds and that $\mathcal{A}$ correctly guessed the hash index of $\mathcal{F}$'s forgery, which occurs with probability at least $1/q_H$.

We prove the theorem by constructing an algorithm $\mathcal{B}$ that solves the co-CDH problem in ($\mathbb{G}_1$,$\mathbb{G}_2$) given an input of a co-CDH instance $(A, B_1, B_2) \in$

$\mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2$ and a forger $\mathcal{F}$. Specifically, $\mathcal{B}$ uses the previously mentioned algorithm $\mathcal{A}$ to execute the generalized forking algorithm $\mathcal{GF}_{\mathcal{A}}$ from Lemma 1 on input (A, B_1, B_2). Observe that the co-CDH-instance is distributed identically to the output of IG. If $\mathcal{GF}_{\mathcal{A}}$ outputs $(0, \perp)$, then $\mathcal{B}$ aborts. If $\mathcal{GF}_{\mathcal{A}}$ outputs $(\{j_f\}, \{out\}, \{out'\})$, then $\mathcal{B}$ proceeds as follows. $\mathcal{B}$ parses out as $(\hat{\sigma}, VK, aux, h_{j_f})$ and out' as $(\hat{\sigma}', VK', aux', h'_{j_f})$. Note that out and out' were obtained from two executions of $\mathcal{A}$ with randomness f and f' such that $f|_{j_f} = f'|_{j_f}$ for some integer $j_f \leq q_S$. This means that these executions are identical up to the j_f-th H_1-query of the type (1) that has not been queried before, but they differ at j_f. To be more specific, this indicates that $VK = VK'$ since VK is an argument to this query. Moreover, $aux = aux'$ because all of the values used to compute aux are calculated before the forking point. As a consequence, we get $e(\hat{\sigma}, g_2) = aux \cdot e(A, \prod_{i=1}^{k} vk_i^{a_i})$ and $e(\hat{\sigma}', g_2) = aux \cdot e(A, \prod_{i=1}^{k} vk_i^{a'_i})$. Without loss of generality, we assume that $vk^* = vk_1$. Then, the previous two equations can be written as $e(\hat{\sigma}, g_2) = aux \cdot e(A, (vk^*)^{h_{j_f}} \prod_{i=2}^{k} vk_i^{a_i})$ and $e(\hat{\sigma}', g_2) = aux \cdot e(A, (vk^*)^{h'_{j_f}} \prod_{i=2}^{k} vk_i^{a_i})$, respectively. By getting the ratio of these equations, we obtain $e(\hat{\sigma}/\hat{\sigma}', g_2) = e(A, {B_2}^{h_{j_f} - h'_{j_f}})$. From this, we get $\hat{\sigma}/\hat{\sigma}' = A^{\beta \cdot (h_{j_f} - h'_{j_f})}$. As a result, $\mathcal{B}$ can compute a solution to the co-CDH instance as $(\hat{\sigma}/\hat{\sigma}')^{1/(h_{j_f} - h'_{j_f})} = g_1^{\alpha\beta}$.

Using Lemma 1, we know that if $q > 8q_H/\epsilon$, then $\mathcal{B}$ runs in time at most $(\tau + q_H \cdot \tau_{\exp_1} + q_S \cdot \tau_{exp_1}) \cdot 8q_H^2/\epsilon \cdot \ln(8q_H/\epsilon)$ and succeeds with probability $\epsilon' \geq \epsilon/(8q_H)$. □

5 Conclusion

In this work, we first propose a multi-message locally verifiable aggregate signature scheme, which builds upon and generalizes the scheme proposed in [18]. With the locality property, the local opening function generates an auxiliary value corresponding to a subset of messages. This allows a verifier to determine if the final aggregated signature includes the signatures of these messages using a single auxiliary value. This eliminates the need for the verifier to access to all messages during the verification phase, thereby significantly enhancing the efficiency of the verification process by introducing only the auxiliary value. Additionally, we present a locally verifiable multi-signature scheme, which is based on the work in [10]. In this context, the local opening function produces an auxiliary value for specific signers. This allows a verifier to efficiently verify which signer's/signers' signatures are included in the multi-signature. Note that as the number of signatures to be verified increases, the verification phase operates more efficiently in both schemes.

Future research could extend the concept of local verification to various other signature schemes, potentially discovering new methods to further improve the efficiency of the verification process.

Acknowledgments. The second author is partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) 2211-A graduate scholarship programs.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 473–484, 2010.
2. Ali Bagherzandi, Jung-Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 449–458, 2008.
3. Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Francois Garillot, Jonas Lindstrom, Ben Riva, Arnab Roy, Mahdi Sedaghat, Alberto Sonnino, Pun Waiwitlikhit, and Joy Wang. Subset-optimized bls multi-signature with key aggregation. *Cryptology ePrint Archive*, 2023.
4. Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*, pages 480–494. Springer, 1997.
5. Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *Automata, Languages and Programming: 34th International Colloquium, ICALP 2007, Wrocław, Poland, July 9-13, 2007. Proceedings 34*, pages 411–422. Springer, 2007.
6. Mihir Bellare and Gregory 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.
7. Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 276–285, 2007.
8. Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *Annual International Cryptology Conference*, pages 443–459. Springer, 2004.
9. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 56–73. Springer, 2004.
10. Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer, 2018.
11. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 416–432. Springer, 2003.

12. Dan Boneh, Craig Gentry, Ben Lynn, Hovav Shacham, et al. A survey of two signature aggregation techniques, 2003.
13. Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 168–195. Springer, 2024.
14. Yevgeniy Dodis, Birgit Pfitzmann, and Bertram Poettering. Information-theoretic key recycling for message authentication. *Advances in Cryptology – CRYPTO 2007*, pages 541–560, 2007.
15. Yevgeniy Dodis and Bertram Poettering. New notions and feasibility results for authentication amplification. In *Advances in Cryptology – ASIACRYPT 2008*, pages 45–63, Berlin, Heidelberg, 2008. Springer.
16. Ruolan Duan, Yun Song, and Xinli Gan. Locally verifiable aggregate signature scheme for health monitoring systems. In *International Conference on Computer Engineering and Networks*, pages 1–10. Springer, 2023.
17. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*, pages 156–174. Springer, 2016.
18. Rishab Goyal and Vinod Vaikuntanathan. Locally verifiable signature and key aggregation. In *Annual International Cryptology Conference*, pages 761–791. Springer, 2022.
19. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*, pages 465–485. Springer, 2006.
20. Di Ma and Gene Tsudik. Forward-secure sequential aggregate authentication. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 86–91. IEEE, 2007.
21. Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.
22. Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 85(2):481–484, 2002.
23. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13:361–396, 2000.
24. Tian Qiu and Qiang Tang. Predicate aggregate signatures and applications. Cryptology ePrint Archive, Paper 2023/1694, 2023. https://eprint.iacr.org/2023/1694.
25. Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
26. Markus Rückert and Dominique Schröder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In *International Conference on Information Security and Assurance*, pages 750–759. Springer, 2009.
27. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
28. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4:161–174, 1991.

29. Liehuang Zhu, Yumeng Xie, Yuao Zhou, Qing Fan, Chuan Zhang, and Ximeng Liu. Enabling efficient and secure health data sharing for healthcare iot systems. *Future Generation Computer Systems*, 149:304–316, 2023.