Verifiable Timed Proxy Signatures and Multi-signatures

Duygu Özden and Oğuz Yayla

Middle East Technical University, Ankara, Turkey

Abstract. Verifiable timed commitments serve as cryptographic tools that enable the binding of information to specific time intervals. By integrating these commitments into signature schemes, secure and tamper-evident digital signatures can be generated, ensuring the integrity of time-sensitive mechanisms. This article delves into the concept of verifiable timed commitments and explores their efficient applications in digital signature constructions. Specifically, it focuses on two important signature schemes: proxy signatures and multi-signatures. The idea of the timed proxy signature is to enable the delegation of signing rights for a specified period, allowing designated entities to sign messages on behalf of the original signer. On the other hand, multi-signatures allow multiple parties to collectively generate a single signature, ensuring enhanced security and accountability. The article presents an in-depth analysis of the underlying mechanisms, discussing their properties, strengths, and computational complexity. Through this exploration, the article aims to shed light on the potential of verifiable timed commitments and inspire further research in this evolving field of cryptography.

Keywords: Timed commitment · timed signature · multi-signature · proxy signature

1 Introduction

Cryptography plays a pivotal role in ensuring the security and integrity of digital communications and transactions. Over the years, researchers and practitioners have developed numerous cryptographic techniques to tackle various challenges and address evolving security requirements. One such area of exploration within cryptography is the concept of verifiable timed commitments. Verifiable timed commitments encompass a range of cryptographic constructs that enable time-bound signing and authentication capabilities, offering enhanced flexibility and accountability. This article delves into two important novel constructs in this domain: timed proxy signatures and multi-signatures.

Proxy signatures allow the delegation of signing rights, enabling designated entities to sign messages on behalf of the original signer. The creation of timed proxy signatures finds utility in scenarios where time-limited authorization is required, such as in cases where individuals or entities need to grant temporary

signing privileges or when time-sensitive decisions need to be made within a cryptographic framework.

Multi-signatures, on the other hand, provide a means for multiple parties to collectively generate a single signature. This construction enhances security by requiring the consent and participation of multiple signers, ensuring a higher degree of accountability and resistance against forgery or tampering. Multi-signatures have gained prominence in various domains, including blockchain technologies, where distributed consensus and verification are essential for maintaining the integrity of the system. Similar to timed proxy signatures, multi-signatures can be made as time-bounded to ensure the validity of the signed data is temporary.

Efficient construction of verifiable timed commitments within BLS [1], Schnorr [2] and ECDSA [3] signature algorithms is provided by Thyagarajan *et al.* [4]. Considering what other timed signature schemes might be useful in real-world applications, we chose proxy signatures and multi-signatures because of their widespread use. The concept of proxy signatures was initially proposed by Mambo *et al.* [5], which focuses on enabling the delegation of signing privileges while ensuring the security and integrity of the original signer's key. Since then, considerable research efforts have been dedicated to enhancing the efficiency, security, and functionality of proxy signature schemes. Notably, threshold proxy signature schemes [6] has emerged as a significant improvement. These schemes introduce a threshold mechanism that mandates the involvement of multiple proxies to generate a valid proxy signature, thereby bolstering security and mitigating the risk of single-point failure. Over time, various types of proxy signatures have been introduced, including accountable proxy signatures [7], identity-based proxy (multi-)signatures [8, 9], anonymous proxy signatures [10]), certificate-based proxy signatures [11, 12], and homomorphic proxy signatures [13]. In the context of proxy signatures, we believe that the use of verifiable timed signatures is a novel approach and enables the delegation of signing authority to a proxy signer within a predefined time frame. By incorporating timed commitments, it becomes possible to enforce accountability and ensure that the proxy signer's actions are constrained within the specified period. This feature adds an additional layer of security and trust to proxy signatures, as the timing of the delegated signatures can be verified, mitigating the risk of unauthorized or malicious use of the delegated authority. On the other hand, multi-signature protocols and specifically accountable subgroup multi-signature (ASM) schemes offer a distinct form of multi-signature scheme. ASM is an important one where a specific subgroup within a larger group can collectively sign a document, ensuring accountability while promoting efficiency. ASM schemes are particularly valuable in scenarios involving multiparty computation, where the need for accountability and efficiency is paramount. While the ASM is constructed in [14], a recent and highly efficient ASM scheme introduced by Boneh *et al.* [15] made a great impact, especially on the applications of blockchain. Notably, this scheme allows users to generate their individual signatures first and subsequently select the subgroup for signing, facilitating flexibility and practical implementation. Sim-

ilar to proxy signature schemes, in the accountable subgroup multi-signatures, the inclusion of verifiable timed commitments allows for the creation of multi-signatures that require the participation of specific subgroup members within a designated time window. This feature ensures the accountability and verifiability of subgroup members' contributions, making the resulting multi-signature more robust and tamper-resistant. In the end, considering all these situations and considering the suitability for the design of proxy multi-signatures, we propose a compact structure where we use ASM within the proxy multi-signature concept and make it timed. Overall, the integration of verifiable timed commitments in proxy signatures and accountable subgroup multi-signatures demonstrates their potential to enhance security, accountability, and verifiability in these important cryptographic protocols.

1.1 Our Motivation and Contribution

The primary objective of this study is to explore and assess the potential of verifiable timed commitments in the context of novel verifiable timed signature schemes. Specifically, our motivation is to:

1. Investigate the technical foundations and properties of verifiable timed commitments, emphasizing their relevance in addressing time-dependent cryptographic challenges.
2. Examine the integration of timed commitments into signature schemes, analyzing their impact on ensuring secure and tamper-evident digital signatures.
3. Provide insights into their potential applications and benefits, ultimately driving the adoption and further development of these techniques.

To the best of our knowledge, this study is the first proposal that constructs a verifiable timed version of proxy signatures and multi-signatures. The timed versions of MSP and ASM schemes are proposed by the authors [16]. In this paper, we extend it by including timed proxy signatures and proxy multi-signatures. In addition, this paper delves into the underlying mechanisms, analyzing their properties, strengths, and computational complexity, thus offering a holistic understanding of these constructs. In particular, we believe that creating a timed version of a proxy multi-signature structure by combining the accountable subgroup multi-signature concept can be a new approach in this field.

1.2 Organization

This study is structured into four sections. Section 1 provides an introduction to the topic, including the background, problem statement, objectives, and an overview of the article's structure. Section 2 presents the underlying cryptographic mechanisms and algorithms used in this work. Section 3 focuses on the integration of verifiable timed commitments into signature schemes, discussing their implications for generating secure and verifiable digital signatures. Finally, Section 4 concludes the article by summarizing the key findings, and outlining potential avenues for future research.

2 Preliminaries

This section serves as a comprehensive introduction to the fundamental concepts and background information necessary for understanding verifiable timed commitments and their usage in the signature schemes. By explaining the algorithms and security requirements used during the research, it is aimed to deal with the proposed systems from different perspectives.

2.1 RSA Encryption Scheme

Key Generation: Let p, q be distinct prime numbers. Calculate $N = p \cdot q$ and $\phi(N) = (p-1) \cdot (q-1)$. Select an integer e, where $gcd(\phi(N), e) = 1$ and $1 < e < \phi(N)$. Find d such that $e \cdot d \pmod{\phi(N)} = 1$. Parse public key as (e, N) and private key as (d, p, q).
Encryption: Let $M < N$ be a message to be encrypted. The encryption algorithm is $E(M) = M^e = c \pmod{N}$, where c is the ciphertext.

Decryption: Let c be a ciphertext to be decrypted. The decryption algorithm is $D(c) = c^d = M \pmod{N}$ where M is the plaintext.

2.2 RSA Signature Scheme

Key Generation: Let p, q be distinct prime numbers. Calculate N, ϕ, e, d, M as described in the RSA encryption.

RSA signature generation: Let $h = \text{Hash}(M)$. Compute the signature: $s = h^d \mod N$.

RSA signature verification: Calculate the hash of the message M as $h^{'} = \text{Hash}(M)$. Check if $h^{'} = h$. If it is equal, check if $h = s^e \mod N$. If it holds, the signature is verified.

2.3 Bilinear Pairing

A bilinear pairing is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ on groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ that satisfies bilinearity and non-degeneracy:

Bilinearity: For all integers b and d, for all elements a in $\mathbb{G}_1$ and c in $\mathbb{G}_2$, $e(a^b, c^d) = e(a, c)^{bd}$.

Non-degeneracy: The pairing $e(g_1, g_2)$ is not equal to 1 for all generators g_1 and g_2 of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.

Bilinear groups are fundamental to various cryptographic schemes, such as identity-based encryption, attribute-based encryption, and certain types of digital signatures. The properties of bilinear groups enable the construction of efficient and secure cryptographic protocols based on mathematical operations and pairings between different groups.

2.4 BLS Signature

Given groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ and generators (g_1, g_2) of the group pair $(\mathbb{G}_1, \mathbb{G}_2)$, consider an efficient, non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. A function $H : \{0,1\}^* \to \mathbb{G}_1$ is defined to map any arbitrary binary string onto the group $\mathbb{G}_1$. Let $H_0, H_2 : \{0,1\}^* \to \mathbb{G}_1$ and $H_1 : \{0,1\}^* \to \mathbb{Z}_q$ be the hash functions which are collision resistant.

BLS signature [1] has three phases: Key Generation (KeyGen), Signature Generation (SigGen), and Verification.

1. Key Generation (KeyGen): Randomly select a secret key sk from the set of integers modulo q, and use it to compute the corresponding public key $pk = g_2^{sk}$.

2. Signature Generation (SigGen): To generate a signature for a given message m, compute $\sigma = H(m)^{sk}$, where sk is the secret key.

3. Verification: Check if the following equation is satisfied:

$$e(H(m), pk) = e(\sigma, g_2) \tag{1}$$

2.5 Multi-signature Protocol (MSP)

Boneh *et al.* introduced the multi-signature protocol (MSP) in their work [15]. It is an efficient multi-signature scheme that builds on top of BLS and involves the steps given below:

Assume that the hash functions, groups, generators of the groups, and bilinear map are the same with BLS.

1. Key Generation: To generate a key pair in this scheme, a secret key sk_i is selected at random from $\mathbb{Z}_q$, and the corresponding public key pk_i is obtained by computing $g_2^{sk_i}$ for a user i.

2. Key Aggregation: Aggregated public key

$$apk = \prod_{i=1}^{n} pk_i^{a_i},$$

where

$$a_i = H_1(pk_i, \{pk_1, \ldots, pk_n\}).$$

3. Signature Generation: Calculate the multi-signature

$$\sigma_i = H_0(m)^{a_i sk_i} \tag{2}$$

where m is the message to be signed.

4. Signature Aggregation: Multi-signature is

$$\sigma = \prod_{j=1}^{n} \sigma_i.$$

5. Verification: Check if

$$e(H_0(m), apk) = e(\sigma, g_2). \tag{3}$$

2.6 Accountable Subgroup Multi-signatures (ASM)

An accountable subgroup multi-signature (ASM) scheme is a type of multi-signature in which a message m can be signed by any subgroup S of a group G, and the signatories from the subgroup S are responsible for the signature. ASM was defined first time in 2001 [14]. After the term construction with a generic protocol definition, one of the most important improvements in ASM schemes is the ASM scheme constructed by Boneh *et al.* [15]. The objective of developing this "short" ASM scheme was to reduce the size of the Bitcoin blockchain. In this algorithm, "short" means the signature size is $O(\lambda)$-bits, where λ is the security constant. This ASM scheme showed that it is very practical and applicable to blockchain cryptosystems.

2.6.1 ASM by Boneh *et al.* Boneh *et al.*'s ASM scheme [15] consists of 5 tuples: KeyGen, Group Setup, SignatureGen, Signature Aggregation (Signature Aggr), and Verification. These steps are explained below. PK is defined as the collection of public keys belonging to the members of group $\mathbb{G}$, denoted by $pk_1, \ldots, pk_n$.

- **KeyGen:** Every user $i \in \mathbb{G}$ gets a random secret key $sk_i \leftarrow \mathbb{Z}_q$ and calculates public key $pk_i \leftarrow g_2^{sk_i}$ where g_2 is the generator of the group $\mathbb{G}_2$
- **Group Setup:** Every user performs group setup using one round interactive protocol.
  - The user is responsible for computing the group's aggregated public key apk as: $apk = \prod_{i=1}^{n} pk_i^{a_i}$ where $a_i = H_1(pk_i, PK)$.
  - The user sends
  $$\mu_{ji} \coloneqq H_2(apk, j)^{a_i sk_i} \tag{4}$$
  to the j-th member for $j = 1, 2, \ldots, n$ and $j \neq i$.
  - After receiving μ_{ij}, the user calculates $\mu_{ii} = H_2(apk, i)^{a_i sk_i}$.
  - For a user i, the membership key is $mk_i = \prod_{j=1}^{n} \mu_{ij}$.
- **SignatureGen:** A signer calculates his/her individual signature as
$$s_i = H_0(apk, m)^{sk_i} \cdot mk_i$$
and delivers s_i to the combiner.
- **Signature Aggr:** The combiner first forms the set of signers $S \subseteq \mathbb{G}$ and computes the aggregated subgroup multi-signature $\sigma = (s, pk)$ where $s = \prod_{i \in S} s_i$ and $pk = \prod_{i \in S} pk_i$.
- **Verification:** The signature $\sigma = (s, pk)$ can be verified by anyone in possession of par, apk, S, m, σ by checking if:
$$e(s, g_2) = e(H_0(apk, m), pk).e(\prod_{j \in S} H_2(apk, j), apk)$$

2.7 Proxy Signatures

A proxy signature is a type of digital signature that allows one entity, known as the original signer/delegator, to delegate their signing authority to another entity, known as the proxy signer. The proxy signer can then generate signatures on behalf of the delegator. Proxy signatures are commonly used in scenarios where the delegator wants to delegate signing capabilities temporarily or to perform signatures without direct involvement. In general, the steps of a proxy signature scheme are as follows:

2.7.1 Key Generation: The original signer and the proxy signer generate their key pair consisting of a private key and a corresponding public key.

2.7.2 Setup: The original signer and the proxy signer establish a trusted relationship, typically through a secure communication channel or a trusted intermediary. Then, the original signer authorizes the proxy signer to act on their behalf.

2.7.3 Delegation: In the delegation step, the original signer creates a warrant w that contains relevant information about the delegation and signing process. This information generally contains the identity of the signers, the delegation period, and the details of the message.

2.7.4 Proxy Signature Generation: The proxy signer first checks if the delegation information is correct or not. Then, if it is correct, the proxy signer generates the proxy signature.

2.7.5 Proxy Signature Verification: A verifier receives the message, the proxy signature, and the public keys of the original and proxy signer and checks if the signature is valid or not.

2.8 Zero Knowledge Proof and Non-interactive Zero Knowledge Proof

The zero-knowledge proof (ZKP) cryptographic method enables the demonstration of a statement's truthfulness without revealing any additional information other than the statement's validity. Non-interactive zero-knowledge proof (NIZK) [17] is a type of it that allows proof of the statement by the prover without any interaction with the verifier. NIZK [17] contains 3-tuples: ZKSetup, ZKProve and ZKVerify:

- **ZKSetup phase(1^λ):** Note that 1^λ is a security constant .
  -The output will be the "common reference string" *(crs)*.

– **ZKProve phase(crs, x, w):** The presence of a "witness" (w) is crucial in proving the truthfulness of a statement (x).
-The statement's validity with the witness outputs the proof π.
– **ZKVerify phase(crs, x, π):** The verification will show if the statement and the corresponding proof are true.
-The output will be "yes/verified" or "no'/not verified", depending on whether the requirements for the verification of the protocol are met.

2.9 Timed Cryptography

Timed-release cryptography aims to achieve the idea that the message can be sent to the future and opened only after some predefined time T. As the first construction of timed cryptography, the concept of "timed commitments" was introduced for the first time in 2000 [18]. A timed commitment scheme for a message or any value provides the sender to commit the value. After some time T, the sender can prove this commitment or if the sender refuses this statement, the receiver can retrieve the committed message within time T by forcing it open. Therefore, a time commitment contains 3 phases:

– **Commitment generation:** To commit on a message.
– **Open:** Sender can open the commitment to show the message.
– **Forced Open:** If the sender chooses not to disclose the commitment, the receiver can use this algorithm to prove the commitment and unveil the committed message within a duration of T.

Timed cryptography contains several types of applications such as time-lock puzzle generation, timed release of signatures as well as timed commitments.

2.9.1 Timed Signatures Timed signatures are a type of time commitment in which the sender can commit to a signature of a message/document. More formally, a timed signature contains five tuples:

1. Setup phase: Any message signer generates a key pair $(publickey, privatekey)$ using a $KeyGen$ algorithm.

2. A signature generation algorithm: A valid signature contains 3 tuples, namely S, C, and Sig where C is a timed commitment when committing to a string S and Sig can be verified using *publickey* as a valid signature on a message M.

3. Commit phase: The signer of a message chooses a random private string S and creates a timed commitment on S named C.

4. Open phase: The sender (signer) opens the committed string S. The receiver (verifier) gets a valid signature tuple namely $\{S, C, Sig\}$.

5. Forced Open phase: Use forced open protocol to retrieve the committed value S by the related timed commitment phase.

One crucial requirement for timed signatures is verifiability, which is proof that the receiver gets the correct message with a valid signature. An efficient version of the term "Verifiable Timed Signature" was defined recently in an

article [4] by Thyagarajan *et al.* as a practical application of timed signature schemes.

2.9.2 Time Lock Puzzles (TLP) and Advantages of Homomorphism: Time lock puzzles [19] let a person encrypt a message to the future. According to the verifiable timed signature (VTS) and verifiable timed commitment (VTC) proposed by [4], usage of TLP provides efficiency to the current applications, and the linearly homomorphic time lock puzzle (LHTLP) version makes them even more efficient where ForceOpen algorithm needs to solve only one packed puzzle instead of many. We use the same notation of LHTLP given in [4] while constructing timed lock puzzles.

LHTLP [20] over the ring $(\mathbb{Z}_N, +)$ contains 4-tuples: PuzzleSetup, PuzzleGeneration, PuzzleSolve, and PuzzleEval and defined as follows:

- **PuzzleSetup phase($1^\lambda, T$):** Note that 1^λ is a security constant and T is the agreed time.
  -With prime numbers p' and q', sample $p = 2p' + 1$ and $q = 2q' + 1$ and set $N \coloneqq pq$.
  -Sample a uniform $\tilde{g} \xleftarrow{\$} \mathbb{Z}_N^*$ and set $g \coloneqq -\tilde{g}^2 \mod N$.
  -Calculate and set $h \coloneqq g^{2^T} \mod \phi(N)/2$.
  -In the end, the output will be public parameters $pp \coloneqq (T, N, g, h)$.
- **PuzzleGeneration phase(pp, s):** Note that s is the solution from the solution set S
  -Set $pp \coloneqq (T, N, g, h)$.
  -Sample an $r \xleftarrow{\$} \{1, \ldots, N^2\}$.
  -Calculate $u \coloneqq g^r \mod N$ and $v \coloneqq h^{r.N}.(1+N)^s \mod N^2$.
  -In the end, the output will be the puzzle $Z \coloneqq (u, v)$.
- **PuzzleSolve phase(pp, Z):**
  -Set $pp \coloneqq (T, N, g, h)$.
  -Set the puzzle $Z \coloneqq (u, v)$.
  -Calculate $w \coloneqq u^{2^T} \mod N$ by Repeated Squaring Method.
  -In the end, the output will be the solution $s \coloneqq \frac{v/(w)^N (mod N^2) - 1}{N}$.
- **PuzzleEval phase($\oplus, pp, Z_1, \ldots, Z_N$):** Note that $\oplus \coloneqq C \in C_\lambda$ is a circuit
  -Set $pp \coloneqq (T, N, g, h)$.
  -Set every $Z_i \coloneqq (u_i, v_i) \in \mathbb{J}_N \times Z_{N^2}^*$.
  -Calculate $\tilde{u} \coloneqq \prod_{i=1}^n u_i \mod N$ and $\tilde{v} \coloneqq \prod_{i=1}^n v_i \mod N^2$.
  -In the end, the output will be the puzzle $Z' = (\tilde{u}, \tilde{v})$.
  Note that, PuzzleSetup, PuzzleGeneration, and PuzzleEval phases are the probabilistic algorithms whereas the PuzzleSolve phase is a deterministic algorithm.

Here, it is important to explain NIZK for LHTLP [4]:

1. **ZKSetup phase:** Let N be an RSA modulus, pp are defined in LHTLP, intervals for the statements are L and B with $B < L$ and let k is the statistical security constant. Let the input here is the time lock puzzles $Z_1, \ldots, Z_l$.

2. **ZKProve phase:** Let $wit \coloneqq ((x_1, r_1), \ldots, (x_l, r_l))$ denote the witness, where $x_i \in [-B, B]$ for all i. If $Z_i \leftarrow HTLP.PuzzleGeneration(pp, x_i; r_i)$ for any i, then the ZKProve phase performs the following steps:
   (a) Let $y_1, \ldots, y_k \leftarrow [-L/4, L/4]$ and $r_1^{'}, \ldots, r_k^{'}$ be the values from the corresponding ring.
   (b) For $i = 1, \ldots, k$ compute $D_i \leftarrow HTLP.PGen(pp, y_i; r_i^{'})$.
   (c) Calculate $(t_1, \ldots, t_k) \leftarrow H(Z_1, \ldots, Z_l, D_1, \ldots, D_k)$, $\forall t_i \in \{0,1\}^l$.
   (d) For $i = 1, \ldots, k$, calculate $v_i \leftarrow y_i + \sum_{j=1}^{l} t_{i,j}.x_j$ and $w_i \leftarrow r_i^{'} + \sum_{j=1}^{l} t_{i,j}.r_j$.
   (e) Set $\pi \leftarrow (D_i, v_i, w_i)_{i \in [k]}$ and output π.
3. **ZKVerify phase:**
   (a) Calculate $(t_1, \ldots, t_k) \leftarrow H(Z_1, \ldots, Z_l, D_1, \ldots, D_k)$.
   (b) For $i = 1, \ldots, k$ calculate if $v_i \in [-L/2, L/2]$, then calculate $F_i \leftarrow D_i . \prod_{j=1}^{l} Z_j^{t_{i,j}}$ and calculate if $F_i = HTLP.PuzzleGeneration(pp, v_i; w_i)$.
   (c) If all the conditions are satisfied, output 1, else output 0.

2.10 Verifiable Timed Signatures (VTS)

Thyagarajan *et al.* recently proposed [4] efficient versions of VTS for BLS [1], Schnorr [2] and ECDSA [3]. VTS versions of these algorithms do not require any modification of signature algorithms, instead, they are chosen as committed signatures in a commitment scheme.

VTS has 4 phases, namely Commit, Vrfy, Open, and ForceOp:

- Commit : (C,π) $\leftarrow$ Commit(σ,T)
- Vrfy : 0 /1 $\leftarrow$ Vrfy(pk, m, C, σ)
- Open : (σ, r) $\leftarrow$ Open(C)
- ForceOp : $\sigma \leftarrow$ ForceOp(C)

We will include the VTS for the BLS (VT-BLS) because it is the only one that is pairing-based, and the pairing-based ones will be the focus of this study.

2.10.1 Verifiable Timed BLS Signature (VT-BLS): To show how VTS phases are used for BLS, the following VT-BLS [4] algorithm steps are explained. Note that, n will be used as a security parameter, threshold $t \coloneqq n/2 + 1$ and $|\sigma| = \lambda$ will be the (maximum) number of bits of the signature σ, $H' : \{0,1\}^* \rightarrow I \subset [n]$ with $|I| = t - 1$ is a random oracle.

- **Setup phase:** Run ZKSetup(1^λ) to generate crs_{range}. Generate public parameters

$$pp \leftarrow \text{LHTLP.PuzzleSetup}(1^\lambda, T)$$

  and output $crs \coloneqq (crs_{range}, pp)$.

– **Commit and prove phase:** For input (crs, wit), follow the steps.
-$wit := \sigma$, $crs := (crs_{range}, pp)$, pk is the public key generated in the BLS, m is the message.
-For all $i \in [t-1]$, sample $\alpha_i \leftarrow \mathbb{Z}_q$ and fix $\sigma_i = H(m)^{\alpha_i}$ and $h_i := g_2^{\alpha_i}$.
-For all $i \in \{t, \ldots, n\}$ compute:

$$\sigma_i = \left(\frac{\sigma}{\prod_{j \in [t-1]} \sigma_j^{l_j(0)}} \right)^{l_i(0)^{-1}} \tag{5}$$

and

$$h_i = \left(\frac{pk}{\prod_{j \in [t-1]} h_j^{l_j(0)}} \right)^{l_i(0)^{-1}} \tag{6}$$

where l_i is the i-th Lagrange Polynomial basis.
-For $i \in [n]$, generate puzzles and proofs:

$$r_i \leftarrow \{0,1\}^{\lambda}, Z_i \leftarrow \text{LHTLP.PuzzleGeneration}(pp, \sigma_i; r_i)$$

and

$$\pi_{range,i} \leftarrow \text{ZKProve}(crs_{range}, (Z_i, 0, 2^{\lambda}, T), (\sigma_i, r_i))$$

-Compute

$$I \leftarrow H'(pk, (h_1, Z_1, \pi_{range,1}), \ldots, (h_n, Z_n, \pi_{range,n})).$$

-Output the commitment $C := (Z_1, \ldots, Z_n, T)$ and corresponding range proof which is

$$\pi := (h_i, \pi_{range,i_{i \in [n]}}, I, \{\sigma_i, r_i\}_{i \in I}).$$

– **Verification phase:** (crs, pk, m, C, π) is the input values and the Vrfy algorithm works as follows:
-Parse $C := (Z_1, \ldots, Z_n, T)$, $\pi := (h_i, \pi_{range,i_{i \in [n]}}, I, \{\sigma_i, r_i\}_{i \in I})$ and $crs := (crs_{range}, pp)$.
-If any of the below conditions are correct, the Vrfy algorithm outputs 0:

1. There is $j \notin I$ satisfying

$$\prod_{i \in I} h_i^{l_i(0)} . h_j^{l_j(0)} \neq pk.$$

2. There is $i \in [n]$ satisfying

$$\text{ZKVerify}(crs_{range}, (Z_i, 0, 2^{\lambda}, T), \pi_{range,i}) \neq 1.$$

3. There is $i \in I$ satisfying

$$Z_i \neq \text{LHTLP.PuzzleGeneration}(pp, \sigma_i; r_i)$$

or

$$e(g_2, \sigma_i) \neq e(h_i, H(m)).$$

4. $I \neq H'(pk, (h_1, Z_1, \pi_{range,1}), \ldots, (h_n, Z_n, \pi_{range,n}))$.

- **Open phase:** The result of the open phase is opening the commitment and receiving $(\sigma, \{r_i\}_{i\in[n]})$. The committer is expected to open at least the puzzles for the challenge set I chosen by the verifier.
- **Force Open phase:** This phase takes input $C := (Z_1, \ldots, Z_n, T)$ and works as follows:
  -Performs $\sigma_i \leftarrow$ LHTLP.PuzzleSolve(pp, Z_i) for $i \in [n]$ and receives all the signature shares. It should be observed that, given the committer has revealed $t-1$ puzzles, the Force Open step will only involve solving $(n-t+1)$ puzzles.
  -Output $\sigma := \prod_{j\in[t]}(\sigma_j)^{l_j(0)}$ by considering the first t signatures shares are valid.

2.11 Verifiable Timed Commitments (VTC)

Verifiable timed commitment (VTC) or verifiable timed dlog (VTD) [21] is used to generate a timed commitment for a secret value $x \in \mathbb{Z}_q^*$ satisfying $h = g^x$ where h is a publicly known value and g is a generator of G, which is a group of order q. This structure of VTC is similar to VTS in terms of steps followed and algorithms used.

- **Setup phase:** Run ZKSetup(1^λ) to generate crs_{range}, generate public parameters
$$pp \leftarrow \text{LHTLP.PuzzleSetup}(1^\lambda, T)$$
and output
$$crs := (crs_{range}, pp)$$
- **Commit and prove phase:** For a given (crs, wit), follow the steps:
  -$wit := x$, $crs := (crs_{range}, pp)$, $h := g^x$.
  -$\forall i \in [t-1]$, sample $x_i \leftarrow Z_q$ and fix $h_i = g^{x_i}$.
  -For all $i \in t, \ldots, n$ compute
$$x_i = \left(x - \sum_{j\in[t]} x_j.l_j 0\right).l_i(0)^{-1} \tag{7}$$
and
$$h_i = \left(\frac{h}{\prod_{j\in[t]} h_j^{l_j(0)}}\right)^{l_i(0)^{-1}} \tag{8}$$
where l_i is the i-th Lagrange polynomial basis.
  -For $i \in [n]$, generate puzzles of signature shares and related proofs as
$$r_i \leftarrow \{0,1\}^\lambda, Z_i \leftarrow \text{LHTLPuzzleGeneration}(pp, x_i; r_i)$$
and

$$\pi_{range,i} \leftarrow \text{ZKProve}(crs_{range}, (Z_i, a, b, T), (x_i, r_i)).$$

-Compute

$$I \leftarrow H'(pk, (h_1, Z_1, \pi_{range,1}), \ldots, (h_n, Z_n, \pi_{range,n})).$$

-Output the commitment $C := (Z_1, \ldots, Z_n, T)$ and corresponding range proof

$$\pi := (\{h_i, \pi_{range,i}\}_{i\in[n]}, I, \{x_i, r_i\}_{i\in I}).$$

-Final output is (h, C, π).

– **Verification phase:** Given (crs, h, C, π), the Vrfy algorithm works as follows.
-Let $C := (Z_1, \ldots, Z_n, T)$, $crs := (crs_{range}, pp)$ and

$$\pi := (\{h_i, \pi_{range,i}\}_{i\in[n]}, I, \{x_i, r_i\}_{i\in I}).$$

-If any of the below conditions are correct, the Vrfy algorithm outputs 0. Therefore, it is expected that these conditions are wrong.

1. There is $j \notin I$ satisfying

$$\prod_{i\in I} h_i^{l_i(0)}.h_j^{l_j(0)} \neq h.$$

2. There is $i \in [n]$ satisfying

$$\text{ZKVerify}(crs_{range}, (Z_i, a, b, T), \pi_{range,i}) \neq 1.$$

3. There is $i \in I$ satisfying $Z_i \neq$ LHTLP.PuzzleGeneration$(pp, x_i; r_i)$ or $h_i = g^{x_i}$.
4. $I \neq H'(pk, (h_1, Z_1, \pi_{range,1}), \ldots, (h_n, Z_n, \pi_{range,n}))$

– **Open phase:** The result of the open phase is opening the commitment and receiving $(x, \{r_i\}_{i\in[n]})$. The committer is expected to open at least the puzzles for the challenge set I chosen by the verifier.

– **Force Open phase:** This phase takes $C := (Z_1, \ldots, Z_n, T)$ and:
-Calculates $x_i \leftarrow$ LHTLP.PuzzleSolve(pp, Z_i) for $i \in [n]$ to retrieve all key shares. It should be observed that, given the committer has revealed $t-1$ puzzles, the Force Open step will only involve solving $(n-t+1)$ puzzles.
-Output $x := \sum_{j\in[t]}(x_j).l_j(0)$ by considering the first t signatures shares are valid.

2.12 Security Requirements

There are two main security requirements for VTS-VTC schemes: soundness and privacy. Soundness promises to the user that the ForceOpen algorithm will reveal the committed value after T time, under the given commitment C. All Parallel Random Access Machine (PRAM) algorithms [4] with run-time less than T can reveal the committed value using commitment and proof with only a negligible probability. The formal definitions for VTS are as follows. The formal definitions for VTC can be given similarly.

2.12.1 Security Requirements for VTS and VTC

Definition 1 (Soundness/Simulation-soundness). *A VTS scheme is sound if* $\forall A$ *which are the probabilistic polynomial time adversaries and* $\forall \lambda \in \mathbb{N}$*, there is a negligible function* $negl(\lambda)$ *that satisfies the following statement:*

$$\Pr\left[b_1 = 1 \wedge b_2 = 0 \ : \ \begin{array}{c} (pk, m, C, \pi, T) \leftarrow A(1^\lambda) \\ (\sigma, r) \leftarrow ForceOp(C) \\ b_1 \coloneqq Verification - forVTS(pk, m, C, \pi) \\ b_2 \coloneqq Verification - forSignature(pk, m, \sigma) \end{array} \right]$$

$\leq negl(\lambda)$.

The concept of simulation-soundness indicates that a prover cannot easily convince a verifier of an untrue statement, even if the prover can generate many simulated proofs for statements they desire.

Definition 2 (Privacy). *A VTS scheme can be considered as private if there exists a Probabilistic Polynomial Time(PPT) simulator* S*, negl as defined in Definition 1, and a polynomial* $\tilde{T}$ *satisfying* $\forall$ *polynomials* $T > \tilde{T}$*, the PRAM algorithms* A *whose running time is at most* t *which is smaller than* T*, all messages* $m \in \{0,1\}^*$ *and* $\forall \lambda \in \mathbb{N}$*, the following statement is satisfied:*

$$\left| \Pr\left[A(pk, m, C, \pi) = 1 \ : \ \begin{array}{c} (pk, sk) \leftarrow KeyGeneration(1^\lambda) \\ \sigma \leftarrow SignatureGeneration(sk, m) \\ (C, \pi) \leftarrow Commit(\sigma, T) \end{array} \right] - \Pr\left[A(pk, m, C, \pi) = 1 \ : \ \begin{array}{c} (pk, sk) \leftarrow KeyGeneration(1^\lambda) \\ (C, \pi, m) \leftarrow S(pk, T) \end{array} \right] \right|$$

$\leq negl(\lambda)$.

2.12.2 Security Requirements for Proxy Signatures In the proxy signatures with the delegation by the warrant, several security requirements play a crucial role [22]. Verifiability ensures that the validity of a proxy signature can be confirmed by any party, including the warrantor or a third party. Strong unforgeability guarantees that generating a valid proxy signature without the proxy signer's private key is computationally infeasible, preventing unauthorized parties from creating fraudulent signatures. Strong undeniability establishes evidence or proof to prevent the proxy signer from denying their actions. Strong identifiability enables the unique identification of the proxy signer. Finally, the prevention of misuse ensures that the delegated authority is used only within the defined limits and restrictions specified by the warrant, preventing any unauthorized or improper use of the signing rights.

2.12.3 Security Requirements for Multi-signatures As an expected security property, the unforgeability of multi-signatures is important because it enhances security by preventing unauthorized parties from creating valid signatures. It ensures that transactions require the consensus and cooperation of multiple parties, reducing the risk of fraud, enabling distributed trust, and providing accountability in digital transactions.

Definition 3 (Unforgeability). *We define an opponent A as a $(\tau, q_S, q_H, \epsilon)$-forger for a multi-signature if it can complete the following game within time τ. They can make q_S-many signing queries, q_H-many random oracle queries, and they win with a probability of ϵ as a bare minimum. We say that a multi-signature is $(\tau, q_S, q_H, \epsilon)$-unforgeable if there is no such individual capable of forging.*

3 Verifiable Timed Commitments within Signature Schemes

Considering the usage and the efficiency of VT-BLS, VT-ECDSA, and VT-Schnorr structures defined for VTS, which other signing algorithms can be used with them is discussed in this study. Adding accountability to the chosen VTS has been one of our main motivations. In this context, we chose accountable subgroup multi-signatures due to their default accountability feature and accountable proxy signatures that have many applications today. In this section, we explained our proposals for those signature schemes blended with the VTC concept. While creating these designs, we slightly modified the existing algorithm structures at some points and made them suitable for the timed signature/commitment structures.

3.1 Verifiable Accountable Timed Proxy Signatures (VAT-PS)

In this section, we introduce the "Verifiable Accountable Timed Proxy Signatures (VAT-PS)" which presents three practical construction scenarios for pairing-based proxy signature schemes. Our scheme focuses on incorporating verifiability, accountability, and timed cryptography to define a proxy signature providing all of these properties in a single scheme. Unlike the traditional proxy signatures, our design brings a novel approach by using timed commitments during the delegation process or signing process which allows a design of a proxy signature to involve only the original signer and the proxy signer. That means, there is no need to have a trusted or semi-trusted third party in such a proxy scheme. Its consideration of verifiability and accountability makes our proposal a desired digital signature design because it provides better transparency and traceability. Our proposed scenarios use the delegation by warrant type of proxy design. The warrant, generated by the original signer, plays a crucial role in establishing the accountability and trustworthiness of the delegation process in our scheme. In our schemes, the warrant consists of the membership keys of the users and the information about the message to be signed. Moreover, we consider our VAT-PS as a two-person multi-signature scheme consisting of the original signer and the proxy signer. In such schemes, both the membership key and the individual signature calculation part of the ASM structure proposed by Boneh *et al.* [15] are quite suitable. However, instead of using the individual signature generation of the ASM scheme directly, we slightly modify it to be compatible with the timed commitments proposed in [4]. The proposed schemes are given under the following three scenarios.

3.1.1 Scenario 1-VAT-PS by timed delegation In the first scenario, we aim to allow the original signer to delegate their signing rights to a proxy signature with a specific condition. The delegation is timed, meaning that after a specific time T, the proxy signature gains the authority to sign on behalf of the original signer. This scenario is a pretty suitable use case to apply timed signatures. To achieve this, it is intended that the original signer signs the warrant with the timed signature and sends it to the proxy signer. In this way, the authority can only be used in the future. Figure 1 shows the high-level construction of the timed delegation of the signature and the steps are explained below.

1. **Key Generation:** The original signer and the proxy signer generate their key pair, (sk_i, pk_i) as the same generation procedure as the BLS scheme. For simplicity, we call the original signer user 1 and the proxy signer as user 2. Therefore, their key pairs are (sk_1, pk_1) and (sk_2, pk_2), respectively.
2. **Setup:** The setup process uses the ASM group setup parameters and functions (H_0, H_1 and H_2 are defined such that H_0 and H_2 map binary strings to $\mathbb{G}_1$ and H_1 maps binary strings to $\mathbb{Z}_q$.) with an additional hash function defined as $H_3 : \mathbb{G}_1 \to \mathbb{Z}_q$. Therefore, considering one original signer and one proxy signer in the process, the group setup algorithm works as follows. Both users compute the aggregated public key

$$apk \leftarrow \prod_{i=1}^{2} pk_i^{H_1(pk_i, \{pk_1, pk_2\})}.$$

   Let $H_1(pk_i, \{pk_1, pk_2\})$ be parsed as a_i. Calculating the membership keys is similar to the ASM, both the original signer and the proxy signer have

$$mk_i \leftarrow \prod_{j=1}^{2} \mu_{ij}.$$

3. **(Timed) Delegation:** As a part of the delegation process, the original signer first produces a warrant value of w. Then, the original signer computes $sign_O \leftarrow H_0(apk, w)^{sk_1 \cdot H_3(mk_1)}$. Let k be the value of $sk_1 \cdot H_3(mk_1)$ in $\mathbb{Z}_q$. Then, k can be considered as the new secret key of the original signer. In this case, $g_2^k = pk'$ is the new public key of the original signer. Since the $sign_O$ is a BLS signature of apk, w with the new modified keys of the original signer, it is compatible with the VT-BLS structure.
   (a) **Setup phase:** Run ZKSetup(1^λ) to generate crs_{range}. Generate public parameters

$$pp \leftarrow \text{LHTLP.PuzzleSetup}(1^\lambda, T)$$

   and output $crs := (crs_{\text{range}}, pp)$.
   (b) **Commit and prove phase:** For input (crs, wit), follow the steps.
   -$wit := sign_O$, $crs := (crs_{\text{range}}, pp)$, pk' is the public key generated as in the BLS scheme, m is apk, w which is used as a message.
   -For all $i \in [t-1]$, sample $\alpha_i \leftarrow \mathbb{Z}_q$ and fix $\sigma_i = H(m)^{\alpha_i}$ and $h_i := g_2^{\alpha_i}$.

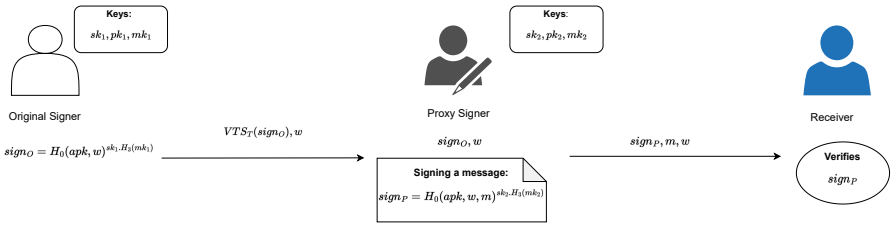

Fig. 1: VAT-PS by Timed Delegation

-For all $i \in \{t, \ldots, n\}$ compute:

$$\sigma_i = \left(\frac{\sigma}{\prod_{j \in [t-1]} \sigma_j^{l_j(0)}} \right)^{l_i(0)^{-1}} \tag{9}$$

and

$$h_i = \left(\frac{pk_{'}}{\prod_{j \in [t-1]} h_j^{l_j(0)}} \right)^{l_i(0)^{-1}}, \tag{10}$$

where l_i is the i-th Lagrange Polynomial basis.
-For $i \in [n]$, generate puzzles and proofs

$$r_i \leftarrow \{0,1\}^{\lambda}, Z_i \leftarrow \text{LHTLP.PuzzleGeneration}(pp, \sigma_i; r_i)$$

and

$$\pi_{\text{range},i} \leftarrow \text{ZKProve}(crs_{\text{range}}, (Z_i, 0, 2^{\lambda}, T), (\sigma_i, r_i)),$$

respectively.
-Compute

$$I \leftarrow H'(pk^{'}, (h_1, Z_1, \pi_{\text{range},1}), \ldots, (h_n, Z_n, \pi_{\text{range},n})).$$

-Output the commitment $C \coloneqq (Z_1, \ldots, Z_n, T)$ and corresponding range proof which is

$$\pi \coloneqq (h_i, \pi_{\text{range},i\, i \in [n]}, I, \{\sigma_i, r_i\}_{i \in I}).$$

Therefore, the original signer can compute the VT-BLS of $sign_O$ to send the signature to the proxy signer as a verifiable timed signature on the warrant. In the end, we can consider w as a valid warrant value whose signature can only be opened after some pre-defined time T and can be verified.

4. **Proxy Signature Generation:**
Note that, since the warrant is sent in a verifiable manner, the signature of the warrant and the committed shares of its signature can be verified during the verification phase before opening the commitments.

(a) **Verification phase:** $(crs, pk^{'}, m, C, \pi)$ is the input values and the Vrfy algorithm works as follows:
-Parse $C \coloneqq (Z_1, \ldots, Z_n, T)$, $\pi \coloneqq (h_i, \pi_{\text{range},i_{i\in[n]}}, I, \{\sigma_i, r_i\}_{i\in I})$ and $crs \coloneqq (crs_{\text{range}}, pp)$.
-If any of the below conditions are correct, the Vrfy algorithm outputs 0:

i. There is $j \notin I$ satisfying

$$\prod_{i\in I} h_i^{l_i(0)}.h_j^{l_j(0)} \neq pk^{'}.$$

ii. There is $i \in [n]$ satisfying

$$\text{ZKVerify}(crs_{\text{range}}, (Z_i, 0, 2^{\lambda}, T), \pi_{\text{range},i}) \neq 1.$$

iii. There is $i \in I$ satisfying

$$Z_i \neq \text{LHTLP.PuzzleGeneration}(pp, \sigma_i; r_i)$$

or

$$e(g_2, \sigma_i) \neq e(h_i, H(m)).$$

iv. $I \neq H'(pk^{'}, (h_1, Z_1, \pi_{\text{range},1}), \ldots, (h_n, Z_n, \pi_{\text{range},n}))$.

After time T passed, the proxy signer can force open the VTS and the proxy key $(sign_O, w)$ can be obtained. There is also another option that the original signer chooses to provide a mechanism that opens all the committed values (Open phase) after time T automatically. This is completely optional and depends on the original signer's choice.

i. **Open phase:** The result of the open phase is opening the commitment and receiving $(sign_O, \{r_i\}_{i\in[n]})$. In this case, the committer is expected to open at least the puzzles for the challenge set I chosen by the verifier.

ii. **Force Open phase:** This phase takes input $C \coloneqq (Z_1, \ldots, Z_n, T)$ and works as follows:
-Performs $\sigma_i \leftarrow \text{LHTLP.PuzzleSolve}(pp, Z_i)$ for $i \in [n]$ and receives all the signature shares. It should be observed that, given the committer has revealed $t-1$ puzzles, the Force Open step will only involve solving $(n-t+1)$ puzzles. In [4], the possibility of solving one puzzle instead of $n-t+1$ puzzles is explained in detail with the help of LHTLP.
-Output $sign_O \coloneqq \prod_{j\in[t]}(\sigma_j)^{l_j(0)}$ by considering the first t signatures shares are valid.

After all the verification of the timed signature is completed, the proxy signer is ready to sign the messages.
Let m be a message to be signed by the proxy signature on behalf of the original signer. The proxy signer first calculates $g_2^{sk_2\cdot H_3(mk_2)}$, names it as p and makes it public. Then, the proxy signer signs m by computing $sign_P \leftarrow H_0(apk, w, m)^{sk_2\cdot H_3(mk_2)}$.

(b) **Proxy Signature Verification:**
Given the values $(sign_P, apk, w, m, p)$, the verifier accepts the proxy signature $sign_P$ if the equation $e(sign_P, g_2) = e(H_0(apk, w, m), p)$ holds.

3.1.2 Scenario 2-VAT-PS by time-bounded delegation In the second scenario, we aim to allow the original signer to delegate their signing rights to a proxy signature for a strictly limited duration in a verifiable and accountable way. Upon expiration of the temporary authorization, the warrant is added to a public revocation list and should no longer be used. This method can be thought of as similar to the concept of certificate transparency in digital certificates. Since the temporary authorization will be public when it is added to the revocation list, it is assumed that the information it contains is not confidential. In addition, since the warrant does not have to contain the duration period for the signing rights, we prevent fraudulent activities over the warrant or the need for a trusted third party to check if the warrant contains the intended time limit. The steps are explained as follows:

1. **Key Generation:** Same as scenario 1.
2. **Setup:** Same as scenario 1.
3. **(Time-bounded) Delegation:** As a part of the time-bounded delegation process, the original signer first produces a warrant value of w. Then, the original signer calculates g_2^w to make it compatible with the VTC. In the delegation process, the original signer computes the VTC of the w and publishes it publicly, and sends the original w to the proxy signer, which is similar to scenario 1 but using VTC protocol instead of VTS. After the committed time T passed, the warrant value is written in a public revocation list. In this way, the proxy signer will only be eligible to sign messages on behalf of the original signer before the revocation.
4. **Proxy Signature Generation:**
Note that, since the warrant is published in a verifiable way, the proxy signer can easily verify the committed value with the verification step of VTC. Let m be the message to be signed by the proxy signature on behalf of the original signer. The proxy signer first calculates $p := g_2^{sk_2 \cdot H_3(mk_2)}$, and makes it public. Then, the proxy signer signs m by computing $sign_P \leftarrow H_0(apk, w, m)^{sk_2 \cdot H_3(mk_2)}$.
5. **Proxy Signature Verification:**
Given the values $(sign_P, apk, w, m, p)$, the verifier accepts the proxy signature $sign_P$ if w is not in the revocation list and the equation $e(sign_P, g_2) = e(H_0(apk, w, m), p)$ holds.

Figure 2 shows the high-level construction of the time-bounded (i.e. temporal) delegation of the signature.

3.1.3 Scenario 3-VAT-PS as a timed signature In the third scenario, we aim to allow the proxy signer to sign documents on behalf of the original signer

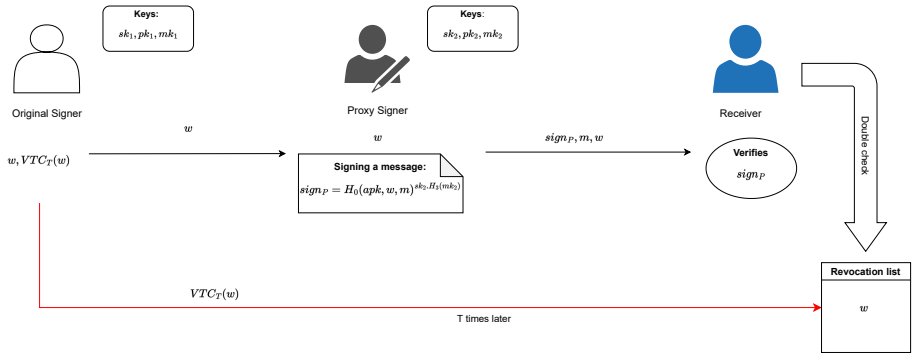

Fig. 2: VAT-PS by Time-bounded Delegation

but also endow them with the ability to apply timed signatures. As a delegation rule, the original signer authorizes the proxy signer, but preconditions that the signature of the message can only be opened after time T. This expectation can be found in the warrant. The steps are explained as follows:

1. **Key Generation:** Same as scenario 1.
2. **Setup:** Same as scenario 1.
3. **Delegation:** As a part of the delegation process, the original signer first produces a warrant value of w. Then, the original signer computes $sign_O \leftarrow H_0(apk, w)^{sk_1 \cdot H_3(mk_1)}$. Let $k := sk_1 \cdot H_3(mk_1) \mod q$. Then, k can be considered as the new secret key of the original signer. In this case, $r = g_2^k$ is the new public key of the original signer. The proxy keys to be used during the proxy signing process are the signature of the warrant and the warrant itself.
4. **(Timed) Proxy Signature Generation:**
   Let m be the message to be signed by the proxy signature on behalf of the original signer. The proxy signer needs to first verify that $e(sign_O, g_2) = e(H_0(apk, w), r)$ holds. The proxy signer then calculates $p := g_2^{sk_2 \cdot H_3(mk_2)}$, and makes it public. Then, the proxy signer signs m by computing $sign_P \leftarrow H_0(apk, w, m)^{sk_2 \cdot H_3(mk_2)}$. By using VT-BLS structure for the $sign_P$ under the new secret key $sk_2 \cdot H_3(mk_2)$ and the new public key p, the proxy signer calculates the timed version of its signature by using the VT-BLS process and calls it VAT-PS and sends it to the receiver.
5. **Proxy Signature Verification:**
   After time T of the VAT-PS passed, the signature can be obtained by the force open. Then, given the values $(sign_P, apk, w, m, p)$, the verifier accepts the proxy signature $sign_P$ if the equation $e(sign_P, g_2) = e(H_0(apk, w, m), p)$ holds. Also, since the commitments are designed as verifiable, the whole signature scheme and its committed shares are verifiable.

Figure 3 shows the high-level construction of the timed proxy signature construction.

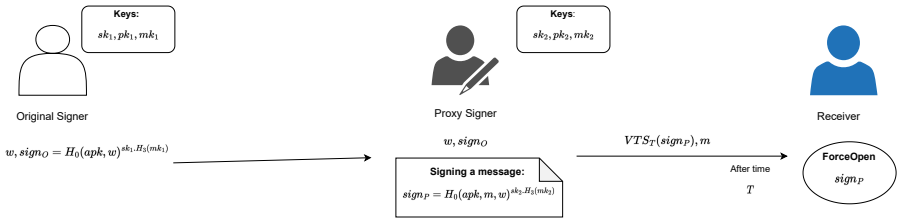

Fig. 3: VAT-PS as a Timed Signature

3.1.4 Security Analysis of VAT-PS Schemes In this section, we discuss the security requirements for the proposed constructions in terms of several considerations used for proxy signature schemes and verifiable timed commitments/signatures.

3.1.4.1 VTS/VTC Security: The security analysis of VTS/VTC is explained here:
Scenario 1:

- **Theorem 1 (Soundness).** It can be inferred that if LHTLP is a time-lock puzzle with perfect correctness, then scenario 1 outlined in Figure 1 meets the soundness requirements outlined in Definition 1, assuming the random oracle model.
- **Theorem 2 (Privacy).** It is asserted that if LHTLP is a secure time-lock puzzle, then scenario 1 outlined in Figure 1 meets the privacy requirements outlined in Definition 2 within the random oracle model.

The proofs can be found in Appendix A and B, respectively.

Scenario 2:

- **Theorem 3 (Soundness).** It can be inferred that if LHTLP is a time-lock puzzle with perfect correctness, then scenario 2 outlined in Figure 2 meets the soundness requirements outlined in Definition 1, assuming the random oracle model.
- **Theorem 4 (Privacy).** It is asserted that if LHTLP is a secure time-lock puzzle, then scenario 2 outlined in Figure 2 meets the privacy requirements outlined in Definition 2 within the random oracle model.

The proofs can be found in Appendix C and D, respectively.

Scenario 3:

- **Theorem 5 (Soundness).** It can be inferred that if LHTLP is a time-lock puzzle with perfect correctness, then scenario 3 outlined in Figure 3 meets the soundness requirements outlined in Definition 1, assuming the random oracle model.

– **Theorem 6 (Privacy).** It is asserted that if LHTLP is a secure time-lock puzzle, then scenario 3 outlined in Figure 3 meets the privacy requirements outlined in Definition 2 within the random oracle model.

The proofs are very similar to ones given in Appendix A and B, respectively.

The theorems play a crucial role in verifying the security of the timed commitments that form the foundation of the proposed schemes. Alternatively, when assessing the proposed schemes based on broader considerations, we can make the following considerations:

3.1.4.2 Verifiability: Verifiability is one of the most important requirements while designing digital signature schemes. Observe that, since we use verifiable timed commitments in each proposal, the committed values are already verifiable by default verifiability property of the VTS/VTC. We prove the verifiability of the proxy signature itself as follows:

Scenario 1: Given $(sign_P, apk, w, m, p)$, $sign_P$ should hold the equation $e(sign_P, g_2) = e(H_0(apk, w, m), p)$.
We can show its correctness as follows:

$$\begin{aligned} e(H_0(apk, w, m), p) &= e(H_0(apk, w, m), g_2^{sk_2 \cdot H_3(mk_2)}) \\ &= e(H_0(apk, w, m)^{sk_2 \cdot H_3(mk_2)}, g_2) \\ &= (sign_P, g_2). \end{aligned}$$

Note that, since scenarios 2 and 3 use a similar signing algorithm, their verification can be shown similarly.

3.1.4.3 Strong Unforgeability: In all of the proposed scenarios, we use a group setup process that is used to create membership keys for the original signer and proxy signer. Since the secret key of the original signer is used when generating the proxy signer's membership key, and the private key of the proxy signer is used in the signing process, no one other than the proxy signer can generate this signature. Even the original signer cannot create the proxy signature since it requires the private key of the proxy signer.

3.1.4.4 Strong Undeniability: In all of the proposed scenarios, the warrant w contains the membership keys and determines the involvement of the original and proxy signers. Also, since the aggregated public key apk, which is produced by the public keys of both users, is used to verify the signature, our schemes provide the undeniability property.

3.1.4.5 Strong Identifiability: All verification processes need apk and the warrant values. Also, the warrant contains the membership key of the proxy signer which provides identifiability.

3.1.4.6 Prevention of Misuse: Since the warrant contains the information about the message to be signed and the responsibility of the proxy signer, the misuse of the proxy signer is prevented in our schemes. The warrant is used in the verification process, therefore anyone can check if the information is valid or not.

3.1.5 Computational Complexity of VAT-PS Schemes In this section, we calculate the computational complexity of the proposed VAT-PS algorithms and two recently proposed pairing-based proxy signature schemes [12, 13]. Considering that there exists no proxy signature scheme constructed using verifiable timed commitments with the time lock puzzles, the complexity of the commitment creation process counts as an extra cost. The comparison of those algorithms shows that our proposed schemes bring more complexity in terms of computation because of the timed commitment calculation complexity. However, our schemes provide additional transparency and traceability which make them more suitable for some use cases in decentralized infrastructures such as blockchain. In addition, VT-BLS [4] offers the advantages of homomorphism while creating time lock puzzles. This opportunity lets the receiver of the puzzles solve one single puzzle instead of more during the force open phase. Therefore, we believe that our proposal is quite suitable for specific environments. The following notations are used in Table 1:

- We calculate the computational complexity of the delegation, delegation verification, proxy signature generation, and proxy signature verification steps to compare our results with [12,13]. Note that, we decide to separate the delegation verification step since it brings additional cost when the delegation values themselves need to be checked.
- We also mention the revocation mechanism used in the proposed schemes and the articles to highlight where the delegation and signing period is given.
- T_H is the number of hash queries for H_0, H_1, H_2, H_3.
- T_{SM} is the time complexity for scalar multiplication.
- T_{MM} is the time complexity for modular multiplication.
- T_{pair} is the time complexity for pairing operation for e.
- T_{Exp} is the time complexity for modular exponentiation.
- T_I is the time complexity for the modular inverse.
- T_{VTS_C} is the time complexity for the VT-BLS commit and prove phase.
- T_{VTS_V} is the time complexity for the VT-BLS verification phase.
- T_{VTC_C} is the time complexity for the VTC commit and prove phase.
- T_{VTC_V} is the time complexity for the VTC verification phase.

3.2 Verifiable Timed Multi-signatures

In this section, we aim to design and analyze verifiable timed multi-signatures and compare them with different types of multi-signature in a bilinear pairing setup. In particular, we give an extended version of verifiable timed proxy

Table 1: Computational complexity of the proposed VAT-PS schemes and recent proposals for pairing-based proxy signature schemes

Proxy Signature Schemes	[13]	[12]	Our Scenario 1	Our Scenario 2	Our Scenario 3
Delegation	$1T_H$	$1T_{SM} + 1T_H$	$1T_{SM} + 2T_H + 1T_{VTS_C}$	$1T_{Exp} + 1T_{VTC_C}$	$1T_{SM} + 2T_H + 1T_{Exp}$
Delegation Verification	$1T_H$	$2T_{Exp} + 1T_H$	$1T_{VTS_V}$	$1T_{VTC_V}$	$2T_{Exp} + 1T_H$
Proxy Signature Generation	$3T_{Exp} + 1T_{MM} + 1T_H + N.T_{SM}$	$1T_{SM} + 1T_I$	$1T_{SM} + 2T_H + 1T_{Exp}$	$1T_{SM} + 2T_H + 1T_{Exp}$	$2T_{SM} + 2T_{Exp} + 3T_H + 1T_{VTS_C}$
Proxy Signature Verification	$5T_{Exp} + 2T_H + N.T_{SM}$	$2T_{Exp} + 1T_{SM} + 1T_H$	$2T_{Exp} + 1T_H$	$2T_{Exp} + 1T_H$	$2T_{Exp} + T_{VTS_V}$
Revocation	By the warrant	By the warrant	By the VTS	By the VTC	By the VTS

signatures, namely verifiable timed proxy multi-signatures. Then, we compute their computational complexity and compare them with the other signature algorithms. To achieve this, we propose timed versions of multi-signature schemes within alternative scenarios. MSP is selected as the initial foundation for developing a time-based variation of a multi-signature. Nonetheless, the primary emphasis of this research lies in timed adaptations of ASM schemes. Initially, we introduce a modified form of MSP that necessitates users to generate VTS for their signatures. Subsequently, we proceed to construct a revised version of the ASM scheme, referred to as mASM, by making slight adjustments to Boneh *et al.*'s original ASM scheme. After that, we use mASM in the construction of timed versions of ASM and proxy multi-signatures.

3.2.1 Verifiable Timed Multi-signature Protocol (VT-MSP-v1) The concept revolves around creating a multi-signature on a message m by using VTS. To construct such a scheme, users create their signatures by MSP, see (2) and put them in a VTS protocol to make sure that the combiner can only receive the valid signatures after some predefined time T. Note that, since the signature generation for individual users is similar to BLS, VT-MSP-v1 is very similar to VT-BLS except that the initial $[t-1]$ signature parts are assigned as follows:

-$\forall i \in [t-1]$, $\alpha_i \xleftarrow{\$} \mathbb{Z}_q$ and fix $\sigma_i = H(m)^{a_i \cdot \alpha_i}$ and $h_i := g_2^{\alpha_i}$ where $a_i = H_1(pk_i, \{pk_1, \ldots, pk_n\})$. This selection does not change the calculation of the rest of the shares, therefore the rest of the shares are defined as (12) and (13).

Each user except one can define his/her own time $t \leq T$ to lock the signatures. However, if the requirement is to produce a multi-signature after some predefined time T, at least one user should lock his/her signature with time T. Once the time T has passed, the combiner can obtain all the individual signatures and

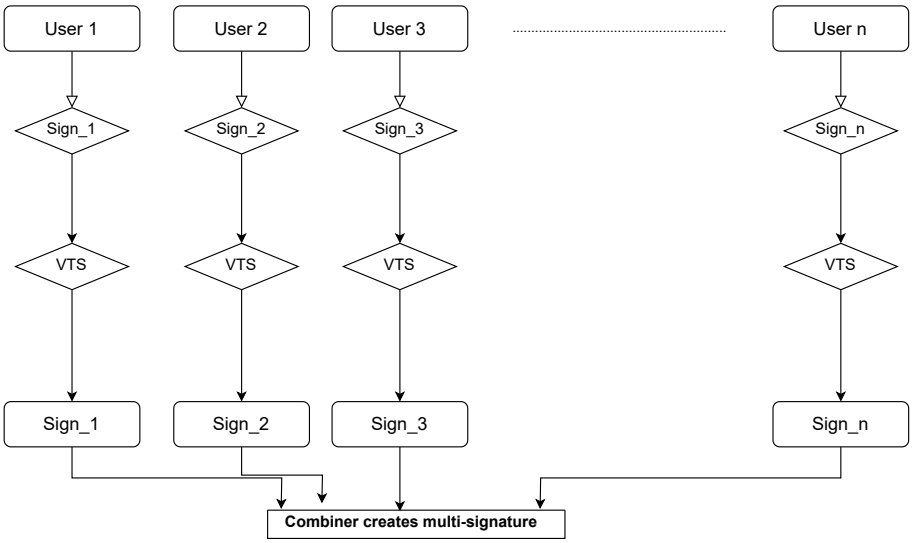

Fig. 4: VT-MSP-v1

calculates the multi-signature. In other words, the combiner calculates

$$\sigma = \prod_{j=1}^{n} \sigma_i.$$

If anyone wants to verify the created multi-signature, they can use the below verification:

$$e(H_0(m), apk) = e(\sigma, g_2).$$

In addition, the individual signatures generated can be checked by the VTS verification phase. Thus, it is possible to verify both the individual signatures and the multi-signature created by the combiner using the same public keys. Figure 4 shows the high-level construction of VT-MSP-v1.

3.2.1.1 Security of VT-MSP-v1: The security of VT-MSP-v1 depends on the LHTLP and the unforgeability of the multi-signature as given in the theorems below.

- **Theorem 7 (Soundness).** It can be inferred that if LHTLP is a time-lock puzzle that guarantees perfect correctness, then VT-MSP-v1 outlined in Figure 4 meets the soundness requirements outlined in Definition 1.
- **Theorem 8 (Privacy).** It is asserted that if LHTLP is a secure time-lock puzzle, then VT-MSP-v1 outlined in Figure 4 meets the privacy requirements outlined in Definition 2.

– **Theorem 9 (Unforgeability).** In the random-oracle model, the VT-MSP-v1 scheme generates an unforgeable multi-signature scheme based on the computational co-Diffie-Hellman problem.

The proofs of Theorem 7 and 8 are similar to the ones in Appendix A, B, and the proof of Theorem 9 can be found in Appendix E, respectively.

3.2.2 Verifiable Timed Multi-signature Protocol - VT-MSP-v2 Version 2 of VT-MSP is similar to version 1. However, in this version, we would like to use 1 VTS instead of n by making the combiner responsible for the timed signature. Let us consider a payment signed by a company's board of directors that is requested to reach the recipient after time T passes. In such cases, sending the produced multi-signature with the help of timed commitments is beneficial in terms of both verifiability and efficiency.

In this scheme, first, users create their individual signatures using the MSP signature generation scheme. After that, the combiner creates multi-signature by calculating

$$\sigma = \prod_{j=1}^{n} \sigma_i = \prod_{j=1}^{n} H_0(m)^{a_j sk_j} = H_0(m)^{a_1 sk_1 + \ldots + a_n sk_n}. \tag{11}$$

Since σ needs to be created as a timed signature, the combiner takes

$$a_1 sk_1 + \ldots + a_n sk_n$$

and calls it *sec*. After that, the combiner creates VTS of the multi-signature as follows:

– **Setup phase:** Run ZKSetup(1^λ) to generate crs_{range}. Generate public parameters

$$pp \leftarrow \text{LHTLP.PuzzleSetup}(1^\lambda, T)$$

and output $crs \coloneqq (crs_{range}, pp)$.

– **Commit:** For input (crs, wit), follow the steps.
-$wit \coloneqq \sigma$, $crs \coloneqq (crs_{range}, pp)$, pk is the public key value g_2^{sec} created and published by the combiner, m is the message.
-$\forall i \in [t-1]$, $sec_i \xleftarrow{\$} \mathbb{Z}_q$ and fix $\sigma_i = H(m)^{sec_i}$ and $h_i := g_2^{sec_i}$.
-$\forall i \in \{t, \ldots, n\}$ compute:

$$\sigma_i = \left(\frac{\sigma}{\prod_{j \in [t-1]} \sigma_j^{l_j(0)}} \right)^{l_i(0)^{-1}} \tag{12}$$

and

$$h_i = \left(\frac{pk}{\prod_{j \in [t-1]} h_j^{l_j(0)}} \right)^{l_i(0)^{-1}} \tag{13}$$

-$\forall i \in [n]$, calculate puzzles and corresponding proofs:

$$r_i \leftarrow \{0,1\}^{\lambda}, Z_i \leftarrow \text{LHTLP.PuzzleGeneration}(pp, \sigma_i; r_i)$$

and

$$\pi_{range,i} \leftarrow \text{ZKProve}(crs_{range}, (Z_i, 0, 2^{\lambda}, T), (\sigma_i, r_i))$$

-Calculate

$$I \leftarrow H'(pk, (h_1, Z_1, \pi_{range,1}), \ldots, (h_n, Z_n, \pi_{range,n})).$$

-Publish the commitment and corresponding range proof which is

$$\pi \coloneqq (h_i, \pi_{range,i_{i\in[n]}}, I, \{\sigma_i, r_i\}_{i\in I}).$$

– **Vrfy:** (crs, pk, m, C, π) is the input values and the Vrfy algorithm works as follows:
-Assign $C \coloneqq (Z_1, \ldots, Z_n, T)$, $\pi \coloneqq (h_i, \pi_{range,i_{i\in[n]}}, I, \{\sigma_i, r_i\}_{i\in I})$ and $crs \coloneqq (crs_{range}, pp)$.
-If any of the below conditions are correct, the Vrfy algorithm outputs 0:
1. There is $j \notin I$ satisfying

$$\prod_{i\in I} h_i^{l_i(0)}.h_j^{l_j(0)} \neq pk.$$

2. There is $i \in [n]$ satisfying

$$\text{ZKVerify}(crs_{range}, (Z_i, 0, 2^{\lambda}, T), \pi_{range,i}) \neq 1.$$

3. There exists $i \in I$ satisfying

$$Z_i \neq \text{LHTLP.PuzzleGeneration}(pp, \sigma_i; r_i)$$

or

$$e(g_2, \sigma_i) \neq e(h_i, H(m)).$$

4. $I \neq H'(pk, (h_1, Z_1, \pi_{range,1}), \ldots, (h_n, Z_n, \pi_{range,n}))$.

– **Open:** The combiner is expected to open at least the puzzles for the challenge set I chosen by the verifier. Otherwise, the commitments can be opened within the force open phase similar to VT-BLS.

As it is seen, the combiner can create a timed multi-signature when it is authorized both to create a multi-signature and to time this signature. What should be noted here is that it can be easily verified that the integrity of the whole process is preserved and that the signature has not been changed, even though it seems that the combiner has been given too much authority. Figure 5 shows our high-level construction of MSP with VTS as a version 2.

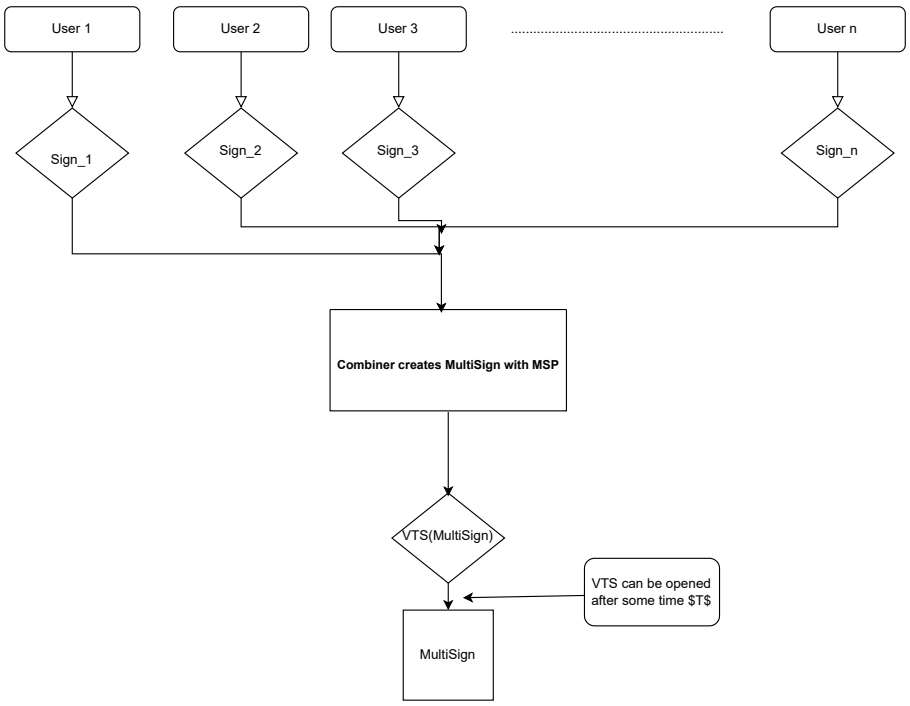

Fig. 5: VT-MSP-v2

3.2.2.1 Security of VT-MSP-v2: The security of VT-MSP-v2 depends on the LHTLP and the unforgeability of the multi-signature as given in the theorems below.

- **Theorem 10 (Soundness).** Assuming the random oracle model, if LHTLP is a time-lock puzzle with perfect correctness, then the VT-MSP-v2 scheme depicted in Figure 5 satisfies the soundness criteria outlined in Definition 1.
- **Theorem 11 (Privacy).** If LHTLP is a secure time-lock puzzle, then VT-MSP-v2 outlined in Figure 5 meets the privacy requirements outlined in Definition 2.
- **Theorem 12 (Unforgeability).** In the random-oracle model, the VT-MSP-v2 scheme creates an unforgeable multi-signature scheme based on the computational co-Diffie-Hellman problem.

The proofs of theorem 10, 11, and 12 are similar to A, B, and E respectively.

3.2.3 Verifiable Timed Accountable Subgroup Multi-signatures

3.2.3.1 Modified Accountable Subgroup Multi-signatures: Boneh *et al.* showed that their ASM construction is efficient and applicable in blockchain cryptosystems. In this section, two modified versions of ASM are introduced. The modified

versions of the ASM scheme, namely mASM-v1 and mASM-v2, are similar to the ASM scheme until individual signature generation.

mASM-v1:

Individual signatures in the mASM-v1 scheme are calculated as follows:

- SignatureGen: Let H_3 be defined as an additional hash function that maps elements from $\mathbb{G}_1$ to $\mathbb{Z}_q$. A signer i calculates his signature as:

$$s_i = H_0(apk, m)^{sk_i \cdot H_3(mk_i)} \tag{14}$$

  and delivers s_i to the combiner. Note that, the aim of this modification is creating a similar structure with VT-BLS to make it easily adaptable to the timed version.
- Signature Aggregation: The individual signatures let the combiner construct the set of signers $S \subseteq G$. After that, the combiner calculates the ASM $\sigma = (s, pk)$ where $s = \prod_{i \in S} s_i$ and $pk = \prod_{i \in S} pk_i$.
- Verification: With having (par, apk, S, m, σ), anyone can check if:

$$e(H_0(apk, m), pk) = e(s, g_2)$$

mASM-v2:

Another modified version of ASM is introduced with authentication of the subgroup. In this case, we want the subgroup to be known from the beginning by creating subgroup-specific membership keys instead of membership keys. To make the scheme simpler, individual signatures can be calculated as follows:

- SignatureGen: $i \in S$ calculates his own signature as:

$$s_i = H_0(apk, m)^{sk_i \cdot H_3(smk_i)} \tag{15}$$

  where smk_i is defined as the subgroup-specific membership key and equal to $\prod_{j \in S} \mu_{ij}$, where μ_{ij} is defined in (4). Also, H_3 is a hash function as defined in mASM-v1. Then, the user sends s_i to the combiner.
- Signature Aggregation: By utilizing the individual signatures, the combiner can compute the subgroup multi-signature $\sigma = (s, pk)$, where $s = \prod_{i \in S} s_i$ and $spk = \prod_{i \in S} pk_i$.
- Verification: With having $par, apk, spk, S, m, \sigma$, anyone can check if

$$e(H_0(apk, m), spk) = e(s, g_2).$$

3.2.3.2 VTC with mASM-v1 (VT-mASM-v1): Although both mASM modifications proposed are suitable to be timed, we explained the timed mASM-v1 here as an idea, due to the similarities of the schemes. VTC usage in mASM-v1 allows a user to send their membership key to the combiner so that the combiner can only receive individual membership keys after some predefined time T. In this way, membership keys will not be public at the beginning of the protocol but a combiner will be able to use them to construct a multi-signature on behalf of a group of n people. The following steps show how a user sends their

mk with VTC. For simplicity, assume that a user has membership key mk and membership key shares are defined as mk_i.

- **Setup phase:** Same as VTC Setup phase.
- **Commit:** For input (crs, wit):
  -Parse $wit \coloneqq sk \cdot H_3(mk)$, $crs \coloneqq (crs_{range}, pp)$, $h \coloneqq g^{sk \cdot H_3(mk)}$.
  -$\forall i \in [t-1]$, $sk'_i \xleftarrow{\$} Z_q$ of the form $sk \cdot H_3(mk)$ and set $h'_i = g^{sk'_i}$.
  -$\forall i \in t, \ldots, n$, compute

$$sk'_i = \left(sk \cdot H_3(mk) - \sum_{j \in [t]} sk'_j . l_j 0 \right) . l_i(0)^{-1}$$

  and

$$h'_i = \left(\frac{h}{\prod_{j \in [t]} h_j^{\prime l_j(0)}} \right)^{l_i(0)^{-1}}$$

  where l_i is the i-th Lagrange polynomial basis.
  -For $i \in [n]$, calculate puzzles and corresponding range proofs:

$$r_i \leftarrow \{0,1\}^{\lambda}, Z_i \leftarrow \text{LHTLP.PuzzleGeneration}(pp, sk'_i; r_i)$$

  and

$$\pi_{range,i} \leftarrow \text{ZKProve}(crs_{range}, (Z_i, a, b, T), (sk'_i, r_i)).$$

  - Set

$$I \leftarrow H'(pk, (h'_1, Z_1, \pi_{range,1}), \ldots, (h'_n, Z_n, \pi_{range,n})).$$

  -Output the commitment $C \coloneqq (Z_1, \ldots, Z_n, T)$ and corresponding range proof which is

$$\pi := (\{h'_i, \pi_{range,i}\}_{i \in [n]}, I, \{sk'_i, r_i\}_{i \in I}).$$

  -Final output is (h, C, π).
- **Vrfy:** By using (crs, h, C, π), the Vrfy algorithm works as follows:
  -Let $C \coloneqq (Z_1, \ldots, Z_n, T)$,
  $crs \coloneqq (crs_{range}, pp)$ and $\pi \coloneqq (\{h'_i, \pi_{range,i}\}_{i \in [n]}, I, \{sk'_i, r_i\}_{i \in I})$.
  -If any of the below conditions are correct, the Vrfy algorithm outputs 0. Therefore, the expectation is that these conditions are wrong.
  1. There is $j \notin I$ satisfying

$$\prod_{i \in I} h_i^{\prime l_i(0)} . h_j^{\prime l_j(0)} \neq h.$$

  2. There is $i \in [n]$ satisfying

$$\text{ZKVerify}(crs_{range}, (Z_i, a, b, T), \pi_{range,i}) \neq 1.$$

  3. There is $i \in I$ satisfying $Z_i \neq \text{LHTLP.PuzzleGeneration}(pp, sk'_i; r_i)$ or $h_i = g^{sk'_i}$.
  4. $I \neq H'(pk, (h'_1, Z_1, \pi_{range,1}), \ldots, (h'_n, Z_n, \pi_{range,n}))$.

– **Open:** The result of the open phase is opening the commitment and receiving $(sk \cdot H_3(mk), \{r_i\}_{i\in[n]})$. The combiner is expected to open at least the puzzles for the challenge set I chosen by the verifier.
– **ForceOp:** This phase takes $C := (Z_1, \ldots, Z_n, T)$ and then performs the following steps.
-Calculates $sk'_i \leftarrow$ LHTLP.PuzzleSolve(pp, Z_i) for $i \in [n]$ to retrieve all membership key shares. It should be observed that, given the committer has revealed $t-1$ puzzles, the Force Open step will only involve solving $(n-t+1)$ puzzles.
-Publish $sk \cdot H_3(mk) \coloneqq \sum_{j\in[t]}(sk'_j).l_j(0)$ by considering the initial t shares are correct.

In this scenario, a combiner cannot change the membership keys as they are received within a verifiable commitment and are related to the user's secret keys. Also, a combiner cannot create an invalid multi-signature on behalf of a group of n people, as verification of the multi-signature requires aggregated public keys of users which can be verified by any person. Note that, since the combiner needs to choose subgroup S to create mASM, they do not have to calculate all individual signatures of users. In that case, the combiner can only calculate the signatures of a subgroup which decreases computational load. Also, since multi-signature is defined as the multiplication of individual signatures in the subgroup, the combiner can first calculate the sum of membership keys of the chosen subgroup and uses it to calculate multi-signature. This method can be considered a delegated multi-signature scheme. VT-mASM-v1 can also be reconstructed if the subgroup is known from the beginning of the protocol. Figure 6 shows the high-level construction of VT-mASM-v1.

3.2.3.3 Security of VT-mASM-v1:

– **Theorem 13 (Soundness).** If LHTLP is a time-lock puzzle with perfect correctness, then VT-mASM-v1 illustrated in Figure 6 meets the soundness requirements outlined in Definition 1, assuming the random oracle model.
– **Theorem 14 (Privacy).** If LHTLP is a secure time-lock puzzle, then VT-mASM-v1 outlined in Figure 6 meets the privacy requirements outlined in Definition 2.
– **Theorem 15 (Unforgeability).** In the random-oracle model, the VT-mASM-v1 scheme generates an unforgeable accountable subgroup multi-signature scheme based on the computational co-Diffie-Hellman problem.

The proofs of theorem 13, 14, and 15 are similar to C, D, and E respectively.

3.2.4 Verifiable Accountable Timed Proxy Multi-signatures (VAT-PMS): Proxy multi-signature is a cryptographic mechanism that allows a proxy signer to sign messages or transactions on behalf of multiple original signers. Instead of each original signer individually signing a message, the proxy entity

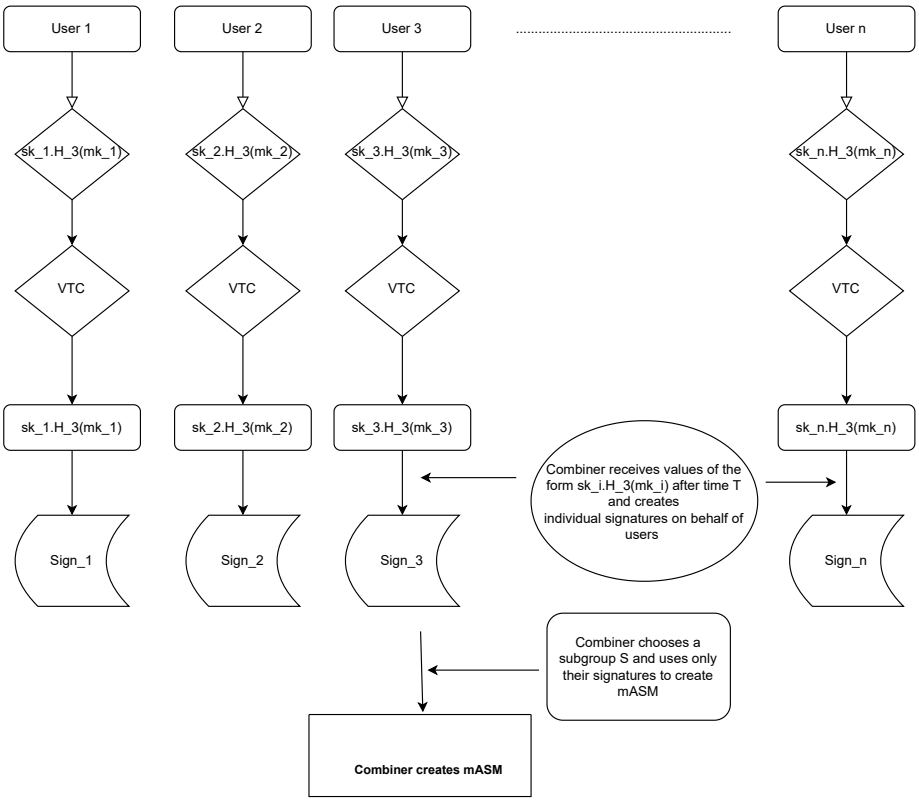

Fig. 6: VT-mASM-v1

holds the necessary private keys and generates a collective signature. In other words, it is similar to proxy signatures, but instead of one original signer, we have more than one original signer. It is particularly useful in scenarios where multiple signers need to delegate their signing power to a trusted intermediary.

In this section, we explain our proposal as a timed version of a proxy multi-signature scheme. While constructing this design, we use our mASM-v1 scheme suggested above as a multi-signature. In addition, we use scenario 3, which we suggest in the proxy signature design above, during the timed signature construction phase of the proxy multi-signature. In this way, we create a unified proxy multi-signature using the combination of the designs we suggested.

3.2.4.1 VAT-PMS as a timed signature: In this scheme, we aim to allow the proxy signer to sign documents on behalf of a group of original signers but also endow them with the ability to apply timed signatures. Before the delegation, a warrant is assumed to be created and it needs to be signed by a group of original signers. In this one-time multi-signature creation of the warrant, we

choose mASM-v1. While creating mASM over the warrant, wlog, we assume that original signer 1 is the combiner of the mASM who interacts with the proxy signer. As a delegation rule, the original signers authorize the proxy signer, but preconditions that the signature of the message can only be opened after time T. This expectation can be found in the warrant. The steps are explained as follows:

1. **Key Generation:** Let $o_1, \ldots, o_n$ be the list of n-original signers, and let P be the $n+1$-th signer, who is a proxy signer. The original signers generate their key pair (sk_i, pk_i) and the proxy signer generates his key pair sk_{n+1}, pk_{n+1} as the same key generation procedure as the BLS scheme. For simplicity, we delegate "the combiner rights" to o_1. Therefore, the key pairs of o_1 and P are (sk_1, pk_1) and (sk_{n+1}, pk_{n+1}), respectively.
2. **Setup:** The setup process uses the ASM group setup parameters and functions (H_0, H_1 and H_2 are defined such that H_0 and H_2 map binary strings to $\mathbb{G}_1$ and H_1 maps binary strings to $\mathbb{Z}_q$.) with an additional hash function defined as $H_3 : \mathbb{G}_1 \to \mathbb{Z}_q$. Therefore, considering n original signers and one proxy signer in the process, the group setup algorithm works as follows. All users compute the aggregated public key

$$apk \leftarrow \prod_{i=1}^{n+1} pk_i^{H_1(pk_i, \{pk_1, \ldots, pk_{n+1}\})}.$$

   Let $H_1(pk_i, \{pk_1, \ldots, pk_{n+1}\})$ be parsed as a_i. Calculating the membership keys is similar to the ASM, both the original signers and the proxy signer have

$$mk_i \leftarrow \prod_{j=1}^{n+1} \mu_{ij}.$$

3. **Delegation:** As a part of the delegation process, the original signers first produce and agree on a common warrant value of w. After that, n original signers create their own signature $\sigma_i = H_0(apk, w)^{sk_i \cdot H_3(mk_i)}$ and sends them to the combiner, which is o_1 in our case. o_1 first forms the set of signers $S \subseteq \mathbb{G}$. Then, the combiner computes the aggregated subgroup multi-signature $\sigma = (mASM_O, pk)$ where $mASM_O = \prod_{i \in S} \sigma_i$ and $pk = \prod_{i \in S} pk_i$. Here, $mASM_O$ is the signature of the warrant and it is sent by the o_1 to the proxy signer.
4. **(Verifiable Accountable Timed) Proxy Multi-Signature Generation:**
   Let m be the message to be signed by the proxy signature on behalf of the group of original signers. The proxy signer needs to first verify that the warrant and its signature are indeed valid, i.e. $e(mASM_O, g_2) = e(H_0(apk, w), pk)$ holds. The proxy signer then calculates $p \coloneqq g_2^{sk_{n+1} \cdot H_3(mk_{n+1})}$, and makes it public. Then, the proxy signer signs m by computing

$$sign_P \leftarrow H_0(apk, w, m)^{sk_{n+1} \cdot H_3(mk_{n+1})}$$

By using VT-BLS structure for the $sign_P$ under the new secret key $sk_{n+1} \cdot H_3(mk_{n+1})$ and the new public key(p) as $g_2^{sk_{n+1} \cdot H_3(mk_{n+1})}$, the proxy signer calculates the timed version of its signature by using VT-BLS process and calls it VAT-PMS and sends it to the receiver. It can be noticed here that the signing process is accountable, delegated to the proxy signer by the original signers, and the proxy multi-signature is created.

5. **(Verifiable Accountable Timed) Proxy Multi-Signature Verification:**
   After time T of the VAT-PMS passed, the signature can be obtained by the open or force open. Then, given the values $(sign_P, apk, w, m, p)$, the verifier accepts the proxy signature $sign_P$ if the equation $e(sign_P, g_2) = e(H_0(apk, w, m), p)$ holds. Also, since the commitments are designed as verifiable, the whole signature scheme and its committed shares are verifiable.

Figure 7 shows the high-level construction of the timed proxy signature construction.

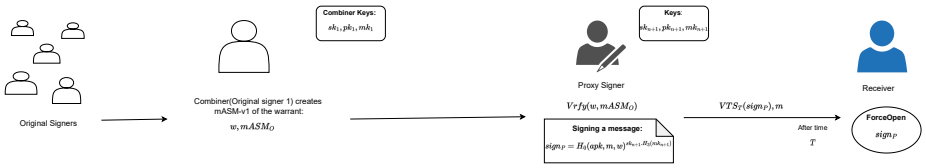

Fig. 7: VAT-PMS

As seen in VAT-PMS, the mASM creation process is delegated to one of the original signers. Here, it is aimed to make a more efficient design, and the structure in scenario 3 is tried to be preserved by determining the person in charge to hand the delegation process to the proxy signer. It is possible to construct different schemes where mASM is created by a different combiner(someone not from the original signer group) or other multi-signature protocols can be used except mASM.

3.2.4.2 Security of VAT-PMS:

- **Theorem 16 (Soundness).** If LHTLP is a time-lock puzzle with perfect correctness, then VT-mASM-v1 illustrated in Figure 7 meets the soundness requirements outlined in Definition 1, assuming the random oracle model.
- **Theorem 17 (Privacy).** If LHTLP is a secure time-lock puzzle, then VT-mASM-v1 outlined in Figure 7 meets the privacy requirements outlined in Definition 2.
- **Theorem 18 (Unforgeability).** In the random-oracle model, the VT-mASM-v1 scheme generates an unforgeable accountable subgroup multi-signature scheme based on the computational co-Diffie-Hellman problem.

The proofs of theorem 16, 17, and 18 are similar to A, B, and E respectively.

3.2.5 Performance Evaluation of the Proposed Multi-signatures: In this section, we calculate the time complexity of the VT-BLS algorithm [4] and our proposed VT-based multi-signature schemes. The complexity of VT-BLS was calculated with our notation to understand how much more calculation would be needed to create a verifiable timed multi-signature using the method in a pairing-based verifiable timed signature algorithm. This comparison makes sense since all of our proposed VT-based schemes are also pairing-based and considering pairing is an expensive operation compared to other digital signature schemes (Schnorr, ECDSA, etc.) decreasing the pairing amount is an important improvement. Some operations are assumed to have relatively low computational complexity and are therefore ignored.

Table 2: Computational complexity of VT-BLS and proposed VT-based multi-signature schemes

VT-BLS [4]	$T_H.T_{Exp_1} + T_{pair} + T_{VTS_C} + T_{VTS_V}$
VT-MSP-v1	$T_H.T_{Exp_1} + T_S.(T_{MExp_2} + T_{Exp_1}) + T_{MExp_2} + T_{pair} + n.(T_{VTS_C} + T_{VTS_V})$
VT-MSP-v2	$T_H.T_{Exp_1} + T_S.(T_{MExp_2} + T_{Exp_1}) + T_{MExp_2} + T_{pair} + (T_{VTS_C} + T_{VTS_V})$
VT-mASM-v1	$T_H.max(T_{MExp_2} + 2T_{Exp_1}) + T_G.(n-1)T_{Exp_1} + T_S.(T_{MExp_2} + T_{Exp_1}) + T_{pair} + 3T_{MExp_1} + n.(T_{VTC_C} + T_{VTC_V})$
VAT-PMS	$T_H.max(T_{MExp_2} + 3T_{Exp_1}) + T_G.(n-1)T_{Exp_1} + T_S.(T_{MExp_2} + T_{Exp_1}) + T_{pair} + 3T_{MExp_1} + (T_{VTS_C} + T_{VTS_V})$

Note that, n is the number of users to join multi-signature and s is the subgroup size for ASM. T_H is the total hash queries for H_0, H_1, H_2, H_3. T_G is the time complexity for group set-up queries. T_S is the time complexity for signing queries. T_{pair} is the time complexity for pairing operation for e. T_{Exp_1} is the time complexity for exponentiation in G_1. T_{Exp_2} is the time complexity for exponentiation in G_2. T_{MExp_1} is the time complexity for multi-exponentiation in G_1. T_{MExp_2} is the time complexity for multi-exponentiation in G_2. T_{VTS_C} is the time complexity for VTS Commit and Prove phase. T_{VTS_V} is the time complexity for the VTS Verification phase. T_{VTC_C} is the time complexity for VTC Commit and Prove phase. T_{VTC_V} is the time complexity for the VTC Verification phase.

Table 2 illustrates the extent to which our constructions introduce additional costs to covert VT-based signatures into a VT-based multi-signature. Considering the amount of puzzle generation, the computationally feasible proposed multi-signature scheme is VT-MSP-v2 since it only uses 1 VTS. VT-MSP-v2 only brings $T_S.(T_{MExp_2} + T_{Exp_1}) + T_{MExp_2} + T_{pair}$ as an additional cost to convert VT-signature(VT-BLS) into VT-multi-signature. VT-MSP-v1 and VT-mASM algorithms are computationally expensive, but they would also be useful in such scenarios where each user wants their signature to be opened after some time t.

4 Conclusion and Future Work

This study explores the potential and significance of integrating verifiable timed commitments into signature schemes. The utilization of verifiable timed commitments enables the generation of secure and tamper-evident digital signatures, bolstering the integrity of time-sensitive mechanisms. Thus, these structures can be used effectively in applications that require signing where time dependency can be used as an advantage. In that sense, we think it would be appropriate to timed proxy signatures and designs that require multiple signatures, which are found in many current applications. The timed signing designs we recommend address these mechanisms in terms of both security and complexity. As future work, further research is needed to enhance the efficiency and scalability of the proposed timed signature schemes. Optimizing the computational costs and communication overheads associated with these protocols can significantly improve their practical feasibility. Additionally, exploring the applicability of verifiable timed commitments in other time-sensitive scenarios beyond signature schemes could expand their utility and impact. For instance, investigating their potential in areas such as secure time-stamping, contract enforcement, or decentralized finance can provide valuable insights.

References

1. D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7*. Springer, 2001, pp. 514–532.
2. C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology—CRYPTO'89 Proceedings 9*. Springer, 1990, pp. 239–252.
3. D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, pp. 36–63, 2001.
4. S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder, "Verifiable timed signatures made practical," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1733–1750.
5. M. Mambo, K. Usuda, and E. Okamoto, "Proxy signatures: Delegation of the power to sign messages," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 79, no. 9, pp. 1338–1354, 1996.
6. K. Zhang, "Threshold proxy signature schemes," in *Information Security: First International Workshop, ISW'97 Tatsunokuchi, Ishikawa, Japan September 17–19, 1997 Proceedings 1*. Springer, 1998, pp. 282–290.
7. H. Du and J. Wang, "An anonymous but accountable proxy multi-signature scheme." *J. Softw.*, vol. 8, no. 8, pp. 1867–1874, 2013.
8. M. R. Asaar, M. Salmasizadeh, and W. Susilo, "A short id-based proxy signature scheme," *International Journal of Communication Systems*, vol. 29, no. 5, pp. 859–873, 2016.

9. R. A. Sahu and S. Padhye, "Identity-based multi-proxy multi-signature scheme provably secure in random oracle model," *Transactions on Emerging Telecommunications Technologies*, vol. 26, no. 4, pp. 547–558, 2015.
10. J.-S. Chou, "A novel anonymous proxy signature scheme," *Advances in Multimedia*, vol. 2012, pp. 13–13, 2012.
11. J. Li, L. Xu, and Y. Zhang, "Provably secure certificate-based proxy signature schemes." *J. Comput.*, vol. 4, no. 6, pp. 444–452, 2009.
12. G. K. Verma and B. Singh, "Short certificate-based proxy signature scheme from pairings," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 12, p. e3214, 2017.
13. Q. Lin, J. Li, Z. Huang, W. Chen, and J. Shen, "A short linearly homomorphic proxy signature scheme," *IEEE Access*, vol. 6, pp. 12 966–12 972, 2018.
14. S. Micali, K. Ohta, and L. Reyzin, "Accountable-subgroup multisignatures," in *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2001, pp. 245–254.
15. D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part II*. Springer, 2018, pp. 435–464.
16. D. Ozden and O. Yayla, "Verifiable timed accountable subgroup multi-signatures," 2023, manuscript submitted for publication.
17. A. De Santis, S. Micali, and G. Persiano, "Non-interactive zero-knowledge proof systems," in *Advances in Cryptology—CRYPTO'87: Proceedings 7*. Springer, 1988, pp. 52–72.
18. D. Boneh and M. Naor, "Timed commitments," in *Annual international cryptology conference*. Springer, 2000, pp. 236–254.
19. R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.
20. G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I*. Springer, 2019, pp. 620–649.
21. S. A. Thyagarajan, G. Malavolta, and P. Moreno-Sanchez, "Universal atomic swaps: Secure exchange of coins across all blockchains," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1299–1316.
22. B. Lee, H. Kim, and K. Kim, "Strong proxy signature and its applications," in *Proceedings of SCIS*, vol. 2001, 2001, pp. 603–608.
23. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology—CRYPTO'86: Proceedings 6*. Springer, 1987, pp. 186–194.
24. D. Unruh, "Random oracles and auxiliary input," in *Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27*. Springer, 2007, pp. 205–223.

A Proof of Theorem 1

Proof. In this context, we are focusing on the interactive version of our protocol, and it is worth noting that the soundness of the non-interactive protocol can

be inferred from [23] in the case of constant-round protocols. Let A be an adversary who breaks the sound by producing the commitment $C = (Z_1, \ldots, Z_n)$ such that $\forall Z_i \notin I$, satisfies LHTLP.PuzzleSolve$(pp, Z_i) = \tilde{\sigma}_i$ where $e(g_2, \tilde{\sigma}_i) \neq e(h_i, H(m))$.

Let us suppose the opposite is true. In that case, it would be possible to recover a legitimate signature on the message m by interpolating $\tilde{\sigma}_i$ with $\sigma_{ii \in I}$, which meets the aforementioned inequality condition. Also, let Z_is be well-formed. In this case, with given Z_i values, through the process of solving the puzzles and confirming which signature shares meet the inequality relation, we can obtain a polynomial-time solution for the set $I^{'}$. When $I^{'} = I$, the verifier can accept the statement provided by the prover, indicating that the prover accurately guessed a randomly chosen n-bit string with $n/2$-many 0's. This situation could happen with a probability $\frac{((n/2)!)^2}{n!}$. In the non-interactive variant of our protocol, the above statement remains true in the face of any number of simulated proofs, so long as the NIZK has the property of simulation-soundness. Therefore, starting with a simulation-sound NIZK makes our scheme simulation-sound as well.

B Proof of Theorem 2

Proof. Considering all VTS operations are identical for any time value t, it is enough to prove only one of them. Assume that A is an adversary of depth bounded by T^{ϵ}, where $0 \leq \epsilon < 1$ and T is the predefined time for the puzzles. We construct a series of hybrids to be used in S which is similar to VT-BLS privacy proof [4].

- Hybrid $\mathcal{H}_0$: Original execution.
- Hybrid $\mathcal{H}_1$: In this case, the random oracle is created using lazy sampling, which distinguishes it from $\mathcal{H}_0$ [24]. Moreover, the set I^*, comprising $t-1$ elements, is selected beforehand and can be used in the cut-and-choose step.
- Hybrid $\mathcal{H}_2$: Sample a simulated crs_{range}. As it is used in the Zero-Knowledge setup, changing crs_{range} is indistinguishable.
- Hybrid $\mathcal{H}_3 \ldots \mathcal{H}_{3+n}$: $\forall i \in [n]$, the NIZK proof $\pi_{range,i}$ is calculated by the simulator in the hybrid $\mathcal{H}_{3+i}$. As it is used in the Zero-Knowledge setup, the distance between all the hybrids is negligibly small.
- Hybrid $\mathcal{H}_{3+n} \ldots \mathcal{H}_{3+2n-t+1}$: $\forall i \in [n-(t-1)]$, the puzzle of the i-th value of the complement of the set I^* is calculated by

$$\text{LHTLP.PuzzleGeneration}(pp, 0^{\lambda}; r_i).$$

  The A is depth-bounded and therefore indistinguishability is based on the security of the puzzle.
- Hybrid $\mathcal{H}_{3+2n-t+2}$: The prover computes cut-and-choose protocol and corresponding puzzles as in the VT-MSP-v1 by choosing the first $t-1$ share $i \in I^*$.

Here, S is the same as the last hybrid. There is no information processed about the witness. Therefore, the algorithm is private against the adversary A.

C Proof of Theorem 3

Proof. In this context, we are focusing on the interactive version of our protocol, and it is worth noting that the soundness of the non-interactive protocol can be inferred from [23] in the case of constant-round protocols. Let A be an adversary who breaks the sound by producing the commitment $C = (Z_1, \ldots, Z_n)$ such that $\forall Z_i \notin I$, satisfies LHTLP.PuzzleSolve$(pp, Z_i) = \tilde{mk}_i$ where $h_i \neq g^{\tilde{mk}_i}$.

Let us suppose the opposite is true. In that case, it would be possible to recover a legitimate membership key of a user by interpolating $\tilde{mk}_i$ with $\{mk_i\}_{i \in I}$, which meets the aforementioned inequality condition. Also, let Z_is be well-formed. In this case, with given Z_i values, through the process of solving the puzzles and confirming which signature shares meet the inequality relation, we can obtain a polynomial-time solution for the set $I^{'}$. When $I^{'} = I$, the verifier can accept the statement provided by the prover, indicating that the prover accurately guessed a randomly chosen n-bit string with $n/2$-many 0's. This situation could happen with a probability $\frac{((n/2)!)^2}{n!}$. In the non-interactive variant of our protocol, the above statement remains true in the face of any number of simulated proofs, so long as the NIZK has the property of simulation-soundness. Therefore, starting with a simulation-sound NIZK makes our scheme simulation-sound as well, similar to Theorem 1.

D Proof of Theorem 4

Proof. Considering all VTC operations are identical for any time value t, it is enough to prove only one of them. Assume that A is an adversary of depth bounded by T^{ϵ}, where $0 \leq \epsilon < 1$ and T is the predefined time for the puzzles. We construct a series of hybrids to be used in S which is similar to VT-BLS privacy proof [4].

- Hybrid $\mathcal{H}_0$: Original execution.
- Hybrid $\mathcal{H}_1$: In this case, the random oracle is created using lazy sampling, which distinguishes it from $\mathcal{H}_0$. [24]. Moreover, the set I^*, comprising $t-1$ elements, is selected beforehand and can be used in the cut-and-choose step.
- Hybrid $\mathcal{H}_2$: Sample a simulated crs_{range}. As it is used in the Zero-Knowledge setup, changing crs_{range} is indistinguishable.
- Hybrid $\mathcal{H}_3 \ldots \mathcal{H}_{3+n}$: $\forall i \in [n]$, the NIZK proof $\pi_{range,i}$ is calculated by the simulator in the hybrid $\mathcal{H}_{3+i}$. As it is used in the Zero-Knowledge setup, the distance between all the hybrids is negligibly small.
- Hybrid $\mathcal{H}_{3+n} \ldots \mathcal{H}_{3+2n-t+1}$: $\forall i \in [n-(t-1)]$, the puzzle of the i-th value of the complement of the set I^* is calculated by the

$$\text{LHTLP.PuzzleGeneration}(pp, 0^{\lambda}; r_i)$$

  . The A is depth-bounded and therefore indistinguishability is based on the security of the puzzle.

– Hybrid $\mathcal{H}_{3+2n-t+2}$: The prover computes cut-and-choose protocol and corresponding puzzles as in the VT-mASM-v1 by choosing the first $t-1$ share $i \in I^*$.

Here, S is the same as the last hybrid. There is no information processed about the witness. Therefore, the algorithm is private against the adversary A.

E Proof of Theorem 9

Proof. Following the proof of [15] for MSP protocol where the message to be signed is m, we can put anything on a message set. The reason is that the signing algorithm uses H_0 which takes binary strings as an input. This means any message that is defined as a binary string can be used in a multi-signature creation. Therefore, the proof is given as follows:

Assume that we have a $(\tau, q_S, q_H, \epsilon)$-forger Adv where the multi-signature scheme runs in time τ, creates q_S signing and q_H random oracle queries and Adv can forge with probability at least ϵ. Let IG be an input generator which creates $(a, b_1, b_2) = (g_1^{\alpha}, g_1^{\beta}, g_2^{\beta})$ for $(\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_{\shortparallel}$. Assume that there is an algorithm Alg on the input (a, b_1, b_2) and a randomness function $f = (\rho, h_1, \ldots, h_{q_S})$.

First, Alg randomly selects an index k from the set $1, ..., q_H$ and executes the Adv using input pk^* from B_2 and a random tape ρ. When replying i-th H_0 query of the Adv, it chooses r_i randomly from $\mathbb{Z}_q$ and returns $g_1^{r_i}$ if i is not equal to k. The $k-th$ H_0 query is answered by returning Alg. Here, consider that Adv does not make any repeated H_0 queries. Alg handles Adv's H_1 queries in the following manner. We categorize the H_1 queries into three types:

1. A query on the (pk, PK) where PK is the list of public keys of the multi-signature creators. Here, we assume that $pk, pk^* \in PK$ and it is the first such query.
2. A query on the (pk, PK) with $pk, pk^* \in PK$. Here, we assume that a prior query has been created.
3. Any other queries.

The algorithm Alg works the i-th type 1 query by choosing any random value for $H_1(pk_i, PK)$ for every $pk_i \neq pk^* \in PK$. In this case, h_i is fixed to $H_1(pk^*, PK)$ and it gives $H_1(pk, PK)$. Then, Alg works the type by giving the values chosen when type 1 of PK was created. The algorithm also works a type 3 by giving any random value in $\mathbb{Z}_q$.

When forger Adv sends a signing query for a message m, using signers PK, algorithm Alg calculates apk by applying a key aggregation step and retrieves $H_0(m)$. If the retrieved value is equal to a, algorithm Alg terminates. Otherwise, it assumes the value is in the form g_1^r, allowing algorithm Alg to simulate an honest signer by computing σ_i as b_1^r. If forger Adv fails to produce a successful

forgery, Alg terminates. If forger Adv successfully generates a forgery for a message m where $H_0(m)$ is not equal to a, algorithm A also terminates. However, if forger Adv outputs a forgery σ, PK, m satisfying the condition

$$e(\sigma, g_2) = e(a, apk)$$

.

Let j_f be an index of $H_1(pk^*, PK) = h_{j_f}$ and let a_j be $H_1(pk_j, PK)$. Then, the algorithm Alg produces $(J = j_f, (\sigma, PK, apk, a_1, \ldots, a_n))$.

The run-time of algorithm Alg consists of the computation time of forger Adv and the additional computations performed by Alg. The total number of hash queries made by Adv denoted as q_H, includes both H_0 and H_1 queries. A requires one exponentiation in $\mathbb{G}_1$ to respond to H_0 queries, resulting in a time complexity of at most $q_H \cdot \tau_{exp_1}$ for answering the hash queries.

For signing queries with a public key (PK) of size l, Alg performs one multi-exponentiation that takes $\tau_{exp_2^l}$ time, along with one exponentiation in $\mathbb{G}_1$ costing τ_{exp_1}. This contributes to a total time complexity of $q_S \cdot (\tau_{exp_2^l} + \tau_{exp_1})$.

Additionally, Alg computes the output values, incurring an additional $\tau_{exp_2^l}$ time to calculate apk. Therefore, Alg's run-time can be expressed as $\tau + q_H\tau_{exp_1} + q_S(\tau_{exp_2^l} + \tau_{exp_1}) + \tau_{exp_2^l}$. The probability of success for Alg is determined by both the probability of Adv succeeding and the probability of Alg correctly guessing the hash index of Adv's forgery. The latter happens with a probability of at least $1/q_H$, making Alg's overall success probability equal $\epsilon_{Alg} = \frac{\epsilon}{q_H}$.

To prove the theorem, assume that Alg' is an algorithm that can solve the computational co-Diffie-Hellman problem in $(\mathbb{G}_{\not\perp}, \mathbb{G}_{\not\perp})$. Let GF_{Alg} be the generalized forking lemma as defined in [15]. If GF_{Alg} terminates, $Alg^{'}$ also terminates. If GF_{Alg} outputs $(j_f, out, out^{'})$, then the algorithm $Alg^{'}$ works as follows: $Alg^{'}$ puts out as $(\sigma, PK, apk, a_1, \ldots, a_n)$ and $out^{'}$ as $(\sigma^{'}, PK^{'}, apk^{'}, a_1^{'}, \ldots, a_n^{'})$. These executions are identical until the j_f-th H_1 query of type 1 by following the GF_{Alg}. This basically means that $PK = PK^{'}$ and $n = n^{'}$. We already know that $apk = \prod_{j=1}^{n} pk_j^{a_j}$ and $apk^{'} = \prod_{j=1}^{n} pk_j^{a_j^{'}}$. Since Alg uses $H_1(pk_j, PK)$ as $a_j \forall j \neq i$ before the forking point, then $a_j = a_j^{'}$ for $j \neq i$ and $\frac{apk}{apk^{'}} = pk^{*a_i - a_i^{'}}$. Knowing the fact that those values of Alg satisfy the bilinear mapping used in the verification, this shows that $(\frac{\sigma}{\sigma^{'}})^{\frac{1}{(a_i - a_i^{'})}}$ is a solution for computational co-Diffie Hellman. By considering GF_{Alg}, $Alg^{'}$'s run-time is at most $(\tau + q_H\tau_{exp_1} + q_S(\tau_{exp_2^l} + \tau_{exp_1}) + \tau_{exp_2^l}) \cdot 8q_H^2/\epsilon \cdot ln(8q_H/\epsilon)$ and this is successful with the probability $\epsilon^{'} \geq \epsilon/(8q_H)$.