

Timed Threshold Proxy Signature Scheme

Xiaohan Yue

xhyue@sut.edu.cn

Shenyang University of Technology

Zhanlei Zhao

Shenyang University of Technology

Shi Bai

Shenyang University of Technology

Fucai Zhou

Northeastern University

Research Article

Keywords: Threshold Signature, Proxy Signature, Time-Lock Puzzle, Non-Interactive Zero-Knowledge

Posted Date: May 25th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-9541497/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Timed Threshold Proxy Signature Scheme

Xiaohan Yue^{1*}, Zhanlei Zhao¹, Bai Shi¹, Fucui Zhou²

¹Shenyang University of Technology, Liaoning Province, China.

²Northeastern University, Liaoning Province, China.

*Corresponding author(s). E-mail(s): xhyue@sut.edu.cn;

Contributing authors: zzzminiuser@smail.sut.edu.cn; baishi@sut.edu.cn;
fczhou@mail.neu.edu.cn;

Abstract

Threshold proxy signatures distribute signing authority to prevent power abuse and single-point key compromise. However, existing schemes do not simultaneously support threshold delegation and cryptographically enforced delayed activation of delegated signing capability. To address this gap, we propose a Timed Threshold Proxy Signature (TTPS) scheme. The core idea of TTPS is to combine a time-lock mechanism with threshold proxy signing so that delegated proxy shares remain computationally unrecoverable before a designated release time under the sequential squaring assumption. In addition, we introduce a verifiable puzzle construction that enables efficient public checking of puzzle well-formedness. TTPS incorporates a pre-computation-based verifiable puzzle design, which eliminates the verifier’s linear dependence on the delay parameter T and reduces per-puzzle public verification to $O(1)$. We further formalize the TTPS model and analyze its unforgeability, robustness, delayed-share security before release, and delegation soundness.

Keywords: Threshold Signature, Proxy Signature, Time-Lock Puzzle, Non-Interactive Zero-Knowledge.

1 Introduction

Threshold proxy signatures [1] integrate proxy signature delegation [2] with threshold-signature robustness [3], enabling an original signer to delegate signing authority to a proxy group such that any subset of at least t proxy signers can jointly produce a valid signature. While this paradigm improves availability and fault tolerance, most existing constructions do not enforce cryptographic time constraints, which limits their suitability for time-sensitive authorization settings. We illustrate this gap through three representative scenarios.

Release of software updates: Large open-source ecosystems operate across organizations

and time zones [4]. Since project owners cannot always directly validate and sign every integration event, signing authority is often delegated to release teams [5]. Without an enforceable delay, an adversary can accumulate sufficient shares during the review and produce a valid signature before the intended release time. Hence, a mandatory cryptographic delay is required to preserve a genuine review window.

Fund release in crowdfunding systems:

In large crowdfunding deployments, requiring signatures from all investors is impractical; therefore, operational authority is delegated to a supervisory committee. However, traditional threshold-style mechanisms [6] remain vulnerable to premature share accumulation via adaptive corruption [7]: an

adversary may pre-collect enough shares and trigger premature fund release, violating predefined governance policies.

Corporate contract co-signing: In multinational M&A workflows, executives delegate authority to distributed managers under strict validity periods. Existing threshold proxy designs [8] generally lack cryptographically enforced expiration and public temporal verifiability, making it difficult for auditors to verify that signatures were generated strictly within authorized windows.

These scenarios motivate a unified primitive that simultaneously provides proxy delegation, threshold control, and enforceable temporal validity.

To the best of our knowledge, no existing scheme achieves all three properties simultaneously (proxy delegation, threshold control, and enforceable temporal validity). Existing partial solutions remain insufficient. Scheme [9] improves threshold-signature efficiency but supports neither proxy delegation nor time-lock enforcement. Schemes [10] [11] support delegation and distributed signing, yet lack cryptographic delay guarantees and are therefore exposed to early-share abuse. Scheme [12] advances verifiable timed signatures but does not address proxy delegation; moreover, its direct circuit embedding of sequential squaring incurs substantial proving overhead and does not support efficient pre-solve verifiability. Scheme [6] introduces time-related controls through warrant logic, but relies on manual revocation rather than cryptographically enforced delay. A structured comparison is given in Table 6.

Based on this analysis, we address two core challenges:

(1) How can time-lock puzzles be integrated with threshold proxy signatures to prevent premature delegation abuse?

(2) How can puzzle solvability be verified efficiently before expensive sequential evaluation?

To resolve these challenges, we propose a Timed Threshold Proxy Signature (TTPS) scheme with two technical components.

For the challenge (1), we encapsulate each proxy key share inside a time-lock puzzle [13] (e.g., an RSA-based TLP driven by sequential squaring). This design enforces delayed activation of delegated proxy shares before the preset release

time, so that an adversary cannot recover a threshold number of usable shares without completing the required sequential computation.

For the challenge (2), we use zk-SNARK to provide public verifiability of the time-lock instance and proof-backed consistency of the delegated-share commitments. Directly encoding TLP evaluations into arithmetic circuits yields prohibitively large circuits and $O(T)$ proving overhead, while standard public puzzle verification natively suffers from $T_{\text{verify}} = O(T)$ due to the required T sequential squarings. Inspired by [14], we design a pre-computation strategy that converts the expensive modular exponentiation into a verifiable subset product check. Concretely, we exploit the trapdoor property of the RSA modulus in our NIZK setting to pre-compute a commitment to the quotient during setup. This completely removes the verifier’s linear dependence on the delay parameter T , yielding an $O(1)$ online verification cost per puzzle instance, following an offline setup phase that is linear in T .

To strengthen transparency, we further employ an electronic bulletin board (BBS) to anchor the warrant message m_w . This ensures public consistency and tamper evidence for delegation scope, threshold parameters, and validity intervals.

Our main contributions are summarized as follows.

(1) We introduce Timed Threshold Proxy Signature (TTPS) as a new primitive and formalize its system and security models. In particular, we define security notions for unforgeability (preventing fresh valid proxy-signature forgery), robustness (preventing the reconstruction of the delegated proxy secret from fewer than t valid shares), time-lock security (ensuring delegated shares remain unrecoverable before the release time and that signatures outside the authorized interval are invalid), and delegation soundness (ensuring that no malformed delegation tuple can pass public verification).

(2) We present a concrete TTPS construction: we combine threshold proxy signature with time-lock encapsulation, enforcing both threshold and temporal constraints on proxy-key usability. The proposed pre-computation mechanism reduces puzzle-verification complexity from $O(T)$ to $O(1)$.

(3) We prove security in the random oracle model under the discrete-logarithm assumption, the sequential squaring assumption, and the knowledge soundness and zero-knowledge of the underlying zk-SNARK.

2 Preliminaries

2.1 Notations

For clarity and consistency, Table 1 summarizes the notation used throughout this paper.

2.2 Complexity Assumptions

The security of the proposed TTPS construction is established under standard computational hardness assumptions. In particular, the proofs of unforgeability, robustness against collusion, and time-lock security are reduced to the intractability of the assumptions stated below. Throughout this paper, probabilistic polynomial-time algorithms are denoted by PPT, and $\text{negl}(\lambda)$ denotes a negligible function in the security parameter λ .

Discrete Logarithm Assumption [15]. Let $\mathcal{G}\text{Gen}(1^\lambda)$ be a PPT algorithm that outputs a cyclic group $\mathbb{G}$ of prime order p with generator g . The DL assumption holds in $\mathbb{G}$ if for any PPT adversary $\mathcal{A}$,

$$\Pr \left[\mathcal{A}(\mathbb{G}, p, g, g^x) = x : (\mathbb{G}, p, g) \leftarrow \mathcal{G}\text{Gen}(1^\lambda), \right. \\ \left. x \leftarrow \mathbb{Z}_p \right] \leq \text{negl}(\lambda). \quad (1)$$

Sequential Squaring Assumption [13].

Let N be an RSA modulus, $z \leftarrow \mathbb{Z}_N^*$, and let $T(\cdot)$ be a time-hardness polynomial. For any polynomial-size adversary $\mathcal{A}$ whose parallel depth is bounded by $T^\epsilon(\lambda)$ for some constant $0 < \epsilon < 1$, it is hard to distinguish $z^{2^T} \bmod N$ from uniform. Formally,

$$\left| \Pr \left[\mathcal{A}(N, z, T, y) = 1 \mid y = z^{2^T} \bmod N \right] \right. \\ \left. - \Pr \left[\mathcal{A}(N, z, T, y) = 1 \mid y \xleftarrow{\$} \mathbb{Z}_N^* \right] \right| \leq \text{negl}(\lambda). \quad (2)$$

The decision formulation above directly implies the computational security of the RSA-based time-lock puzzle defined below: if $z^{2^T} \bmod N$ is indistinguishable from a uniform element of $\mathbb{Z}_N^*$, then in the random oracle model the derived

symmetric key $K = H_0(z^{2^T})$ is pseudorandom, and the ciphertext $C = \text{Enc}_K(s)$ computationally hides the secret s from any adversary whose parallel depth is less than T .

2.3 Time-Lock Puzzles

To delay the usability of proxy shares until time T_1 , we use an RSA-based time-lock puzzle. On input a secret s , the generation algorithm outputs a puzzle (u, N, T, C) ; the solving algorithm performs sequential squaring to recover u^{2^T} and then decrypts C .

Formally, a TLP scheme consists of two algorithms:

(1) $\text{TLP.Gen}(T, s, N) \rightarrow (\text{puz}, K)$: The generation algorithm takes a time parameter T , a secret s (or a message to be hidden), and a modulus N . It samples $u \leftarrow \mathbb{Z}_N^*$ and computes the locked value

$$Z = u^{2^T} \bmod N,$$

where $u \leftarrow \mathbb{Z}_N^*$.

To hide $s \in \{0, 1\}^\lambda$, the generator derives a symmetric key K from Z (e.g., via hashing) and outputs a ciphertext $C = \text{Enc}_K(s)$ together with the puzzle description $\text{puz} = (u, N, T, C)$. If the generator knows the trapdoor $\phi(N)$, it can compute Z efficiently by reducing the exponent modulo $\phi(N)$.

(2) $\text{TLP.Solve}(u, N, T) \rightarrow Z$: The solving algorithm takes (u, N, T) and performs T repeated squarings to recover Z :

$$u \xrightarrow{2} u^2 \xrightarrow{2} u^4 \dots \xrightarrow{2} u^{2^T} \pmod{N}$$

Since $\phi(N)$ is unknown to the solver, there is no known shortcut via exponent reduction. Therefore, the solver can sequentially execute the T squarings to obtain Z , reconstruct K , and decrypt C . This primitive is used in the timed delegation phase to encapsulate each delegated proxy share.

2.4 Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs)

The evaluation of Time-Lock Puzzles (TLP) inherently relies on sequential algebraic operations. Since zk-SNARKs [16] serve as an advanced

Table 1: Notations and Descriptions

Symbol	Meaning
1^λ	Security parameter
pp	Global public parameters
(pk_i, sk_i)	The public and private key pairs for the proxy signature group
n	The total number of signers
t	Threshold parameter
$\hat{\sigma}$	Proxy aggregate signature
ρ_i	randomness used to open the commitment
$comm_i$	Timed delegation tuple for signer PS_i
π_i	Proof tuple associated with $comm_i$
m	Message to be signed
m_w	Warrant message
$\mathbb{PS}$	The set of proxy signers
T_1	The predetermined release time for time-locked proxy key shares
T_2	The expiration time of the proxy signature
T_t	The calibrated TLP parameter
τ_s	The signing timestamp

verifiable computation paradigm specifically tailored for arithmetic circuits, they exhibit a natural structural compatibility with such sequential circuit logic [17]. However, as naively encoding deep sequential squarings would yield circuit sizes exceeding $O(T)$ constraints, we exploit this arithmetic compatibility in conjunction with our pre-computation strategy. This allows us to efficiently verify the structured proxy delegation relation without exposing the underlying key shares.

Formally, a zk-SNARK scheme consists of three algorithms (SNARK.Setup, SNARK.Prove, SNARK.Verify):

(1) $\text{SNARK.Setup}(1^\lambda, \mathcal{R}) \rightarrow (crs, pp)$: Takes the security parameter λ and a specific NP relation $\mathcal{R}$ (e.g., $\mathcal{R}_K$) as input. It generates the global public parameters pp and the common reference string crs used for generating and verifying proofs.

(2) $\text{SNARK.Prove}(crs, \mathcal{R}, \vec{x}, \vec{w}) \rightarrow \pi_{\mathcal{R}}$: Run by the prover. It takes a public statement $\vec{x}$ (e.g., pk_i, D_i, Y_i, C_i) and a private witness $\vec{w}$ (e.g., sk_i, s', ν, ρ_i) such that $(\vec{x}, \vec{w}) \in \mathcal{R}$, and outputs a succinct proof $\pi_{\mathcal{R}}$.

(3) $\text{SNARK.Verify}(crs, \mathcal{R}, \vec{x}, \pi_{\mathcal{R}}) \rightarrow \{0, 1\}$: Run by the verifier. It checks whether $\pi_{\mathcal{R}}$ is a valid proof for the statement $\vec{x}$ under relation $\mathcal{R}$, and outputs 1 if it accepts, or 0 otherwise.

2.5 Schnorr-Type Aggregate Proxy Signing Layer

Our TTPS instantiation uses a Schnorr-type Fiat-Shamir signing layer over a cyclic group $G = \langle g \rangle$ of prime order q . Each proxy signer PS_i holds a long-term secret key sk_i and a delegated proxy share $\hat{sk}_i$. For a participating signer set $\mathbb{PS}$, let w_i denote the Lagrange coefficient of signer i with respect to $\mathbb{PS}$.

The partial proxy response is computed as

$$\hat{\sigma}_i = k_i + (\hat{sk}_i w_i + sk_i) \cdot H(\tilde{r}, m, \mathbb{PS}, \tau_s),$$

where $k_i \xleftarrow{\$} \mathbb{Z}_q^*$, $r_i = g^{k_i}$, and $\tilde{r} = \prod_{i \in \mathbb{PS}} r_i$.

The aggregate proxy signature is $\hat{\sigma} = \sum_{i \in \mathbb{PS}} \hat{\sigma}_i$ and verification checks

$$g^{\hat{\sigma}} = \tilde{r} \cdot \left(K \cdot pk^{H(m_w, K)} \cdot \prod_{i \in \mathbb{PS}} pk_i \right)^{H(\tilde{r}, m, \mathbb{PS}, \tau_s)}.$$

Hence, the resulting signing equation is a Schnorr-type Fiat-Shamir relation under the effective aggregate public key

$$Y_{\mathbb{PS}} = K \cdot pk^{H(m_w, K)} \cdot \prod_{i \in \mathbb{PS}} pk_i.$$

3 Scheme Model and Security Model

3.1 System Model of the Timed Threshold Proxy Signature Scheme

The system model for the timed threshold proxy signature scheme comprises the following entities:

Electronic Bulletin Board (BBS): Serves as a transparent, publicly accessible platform for recording, ensuring the immutability of the authorization warrant m_w and supporting the verifiability of multi-party collaboration.

Original Signer (OS): The OS generates the warrant m_w , derives the proxy delegation secret, and creates one delegated share for each proxy signer. These shares are not delivered in the clear during delegation. Instead, for each PS_i , the OS publishes a time-lock puzzle. Thus, before the release time, only public delegation material is available, while the usable proxy share remains unrecoverable.

Proxy Signers (PS_i): Each PS_i utilizes its proxy signing private key $\hat{sk}_i$ to sign messages, producing partial proxy signatures $\hat{\sigma}_i$. A minimum of t proxy signers can cooperate to generate a valid proxy aggregate signature.

Verifier (V): The verifier V checks the validity of the proxy aggregate signature $\hat{\sigma}$, ensuring the signature is valid.

Formally, a timed threshold proxy signature scheme TTPS is defined as a tuple of algorithms (Setup, Kgen, PKeyGen, TimedDel, PPSign, TExSign, PVerify):

Setup($1^\lambda, T_t, n, t$): Executed by the trusted third party (e.g., Certificate Authority) to initialize the scheme. Outputs public parameters and setup information, including a common reference string crs (and, if necessary, auxiliary public parameters pp).

Kgen(pp): Executed by the OS and proxy signers to generate key pairs for all parties and the authorization warrant. On input pp , outputs $(\{(pk_i, sk_i)\}_{i=1}^n, (pk, sk), m_w)$.

PKeyGen(sk, m_w): Executed by the OS to derive the proxy share for signer i . On input the OS's secret key sk , and the warrant m_w , outputs the proxy signing key share $\hat{sk}_i$ and the corresponding proxy public key $\hat{pk}_i$.

TimedDel($crs, \hat{sk}_i, \hat{pk}_i, T_t$): Executed by OS to produce time-related commitment material. Outputs a commitment $comm_i$ (e.g., a time-lock instance) and a proof π_i attesting to the well-formedness of the commitment.

PPSign($crs, \hat{pk}_i, comm_i, \pi_i$): For each participating signer PS_i , the signer first verifies the delegated commitment material ($comm_i, \pi_i$), then PS_i runs TExSign($comm_i$) to recover the proxy share and produce a partial signature $\hat{\sigma}_i$. The partial signatures are then collected and aggregated as described in PPSign.

TExSign($comm_i$): Executed by PS_i to extract the proxy signing key share after the pre-determined time and generate a partial proxy signature $\hat{\sigma}_i$.

PVerify($m_w, m, \Sigma, \mathbb{PS}, \tau_s$): Executed by the verifier to check validity. On input the warrant m_w , message m , a proxy aggregate Σ , and the set of participating signers $\mathbb{PS}$, outputs a bit in $\{0, 1\}$ indicating acceptance or rejection.

3.2 Security Model of the Timed Threshold Proxy Signature Scheme

To formalize these adversarial capabilities in security experiments, we model the probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ as interacting with a challenger via a set of stateful oracles (i.e., query interfaces). During the security game, $\mathcal{A}$ may adaptively issue oracle queries, and the challenger responds while maintaining internal state. Three sets are initialized and maintained by the challenger: $\mathcal{PC}$ (the set of corrupted signers), $\mathcal{PH}$ (the set of honest signers), and $\mathcal{PQ}$ (the set of signing queries, i.e., signed messages and corresponding signers). The oracles are defined as follows:

Corruption Oracle ($\mathcal{O}_c$): On input a proxy signer index i , the challenger checks if $i \notin \mathcal{PH}$ or if $|\mathcal{PC}| \geq t - 1$. If either condition holds, it returns $\perp$ (failure). Otherwise, it updates the sets by moving i from $\mathcal{PH}$ to $\mathcal{PC}$ and returns the private key sk_i to $\mathcal{A}$. By permitting up to $t - 1$ queries to the corruption oracle $\mathcal{O}_c$, our model captures premature share accumulation via collusion (previously motivated as corruption attacks), guaranteeing temporal security even against early node compromise.

Proxy Signing Oracle ($\mathcal{O}_p$): On input a proxy signer index i and a message m , the challenger checks if $i \notin \mathcal{PH}$. If i is corrupted, it returns $\perp$. Otherwise, it updates the query record $\mathcal{PQ}[m] \leftarrow \mathcal{PQ}[m] \cup \{i\}$ and internally simulates the two-round signing protocol for signer i (acting on behalf of all other honest signers as needed), producing a partial signature $\hat{\sigma}_i$. It returns $\hat{\sigma}_i$ to $\mathcal{A}$ (or $\perp$ if verification of $(comm_i, \pi_i)$ fails).

ForceOpen Oracle ($\mathcal{O}_{fo}$): On input a proxy signer index i , this oracle simulates the adversary's ability to break the time-lock puzzle before maturity. The challenger checks if the current time $t_c > T_1$. If so, it returns $\perp$. Otherwise, it forceopens the time-lock material in $comm_i$ and reconstructs the proxy share $\hat{sk}_i$ following the ForceOpen steps in TExSign, then returns $\hat{sk}_i$ to $\mathcal{A}$.

Based on the above oracle model, we now formally define the security properties of the TTPS scheme.

Definition 1 (Unforgeability under chosen-message attacks (UF-CMA)) If for any probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that

$$\text{Adv}_{\mathcal{A}, T}^{\text{UF-CMA}}(\lambda) = \Pr[\text{UF-CMA}_T^{\mathcal{A}}(\lambda) \Rightarrow 1] \leq \text{negl}(\lambda), \quad (3)$$

then the timed threshold proxy signature scheme satisfies UF-CMA security.

Exp1 UF-CMA $_T^{\mathcal{A}}(\lambda)$

- 1) $PC \leftarrow \emptyset, PH \leftarrow [n], PQ \leftarrow \emptyset;$
 - 2) $(crs, pp) \leftarrow \text{Setup}(1^\lambda, T_t, n, t);$
 - 3) $(\{(pk_i, sk_i)\}_{i \in [n]}, m_w, (pk, sk)) \leftarrow \text{Kgen}(pp);$
 - 4) $(K, \hat{pk}, \{(\hat{sk}_i, pk_i)\}_{i \in [n]}) \leftarrow \text{PKeyGen}(sk, m_w);$
 - 5) for each $i \in [n]$:
 $(comm_i, \pi_i) \leftarrow \text{TimedDel}(crs, \hat{sk}_i, \hat{pk}_i, T_t);$
 - 6) $(m^*, \Sigma^*, \mathbb{PS}^*) \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_p}(pp, crs, pk, m_w, \{pk_i\}_{i \in [n]}, \{(comm_i, \pi_i)\}_{i \in [n]});$
where $\Sigma^* = (K^*, \hat{r}^*, \hat{\sigma}^*)$.
 - 7) If $|PC| \leq t - 1$ and
 $((\mathbb{PS}^* \cap PH) \cap PQ[m^*] = \emptyset)$
and $\text{PVerify}(m_w, m^*, \Sigma^*, \mathbb{PS}^*, \tau_s^*) = 1$,
return 1;
 - 8) else return 0.
-

Definition 2 (Robustness under chosen-message attacks in TTPS (RB-CMA)) If for any probabilistic

polynomial-time adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that

$$\text{Adv}_{\mathcal{A}, T}^{\text{RB-CMA}}(\lambda) = \Pr[\text{RB-CMA}_T^{\mathcal{A}}(\lambda) \Rightarrow 1] \leq \text{negl}(\lambda),$$

then the timed threshold proxy signature scheme satisfies RB-CMA security.

Exp2 RB-CMA $_T^{\mathcal{A}}(\lambda)$

- 1) $PC \leftarrow \emptyset, PH \leftarrow [n], PQ \leftarrow \emptyset;$
 - 2) $(crs, pp) \leftarrow \text{Setup}(1^\lambda, T_t, n, t);$
 - 3) $(\{(pk_i, sk_i)\}_{i \in [n]}, m_w, (pk, sk)) \leftarrow \text{Kgen}(pp);$
 - 4) $(K, \hat{pk}, \{(\hat{sk}_i, pk_i)\}_{i \in [n]}) \leftarrow \text{PKeyGen}(sk, m_w);$
 - 5) for each $i \in [n]$:
 $(comm_i, \pi_i) \leftarrow \text{TimedDel}(crs, \hat{sk}_i, \hat{pk}_i, T_t);$
 - 6) $\hat{sk}^* \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_p}(pp, crs, pk, m_w, \{pk_i\}_{i \in [n]}, \{\hat{pk}_i\}_{i \in [n]}, \{(comm_i, \pi_i)\}_{i \in [n]});$
 - 7) If $|PC| \leq t - 1$ and $g^{\hat{sk}^*} = K \cdot pk^{H(m_w, K)}$
return 1;
 - 8) else return 0.
-

Definition 3 (Time-lock security (TL-SC)) The TTPS scheme satisfies time-lock security if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that

$$\text{Adv}_{\mathcal{A}, T}^{\text{TL-SC}}(\lambda) = \Pr[\text{TL-SC}_T^{\mathcal{A}}(\lambda) \Rightarrow 1] \leq \text{negl}(\lambda).$$

Intuitively, before T_1 , $\mathcal{A}$ cannot recover any honest delegated share sk_i from the public delegation transcript with non-negligible probability.

Exp3 TL-SC $_T^{\mathcal{A}}(\lambda)$

- 1) $PC \leftarrow \emptyset, PH \leftarrow [n], PQ \leftarrow \emptyset;$
 - 2) $(crs, pp) \leftarrow \text{Setup}(1^\lambda, T_t, n, t);$
 - 3) $(\{(pk_i, sk_i)\}_{i \in [n]}, m_w, (pk, sk)) \leftarrow \text{Kgen}(pp);$
 - 4) $(K, \hat{pk}, \{(\hat{sk}_i, pk_i)\}_{i \in [n]}) \leftarrow \text{PKeyGen}(sk, m_w);$
 - 5) for each $i \in [n]$:
 $(comm_i, \pi_i) \leftarrow \text{TimedDel}(crs, \hat{sk}_i, \hat{pk}_i, T_t);$
 - 6) $(i^*, \hat{sk}_{i^*}^*, t_c) \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_p, \mathcal{O}_{fo}}(pp, crs, pk, m_w, \{pk_i\}_{i \in [n]}, \{\hat{pk}_i\}_{i \in [n]}, \{(comm_i, \pi_i)\}_{i \in [n]});$
 - 7) If $t_c < T_1$ and $i^* \in PH$ and $g^{\hat{sk}_{i^*}^*} = \hat{pk}_{i^*}$,
return 1;
 - 8) else return 0.
-

Definition 4 (Delegation soundness) Let R_K be the delegation relation defined in Section 4.2. The TTPS

scheme satisfies delegation soundness if, for any PPT adversary $\mathcal{A}$,

$$\text{Adv}_{\mathcal{A}}^{\text{DPS}}(\lambda) := \Pr \left[\text{SNARK.Verify}(crs_{vt}, R_K, \tilde{x}^*, \pi^*) = 1 \wedge \tilde{x}^* \notin R_K \right] \leq \text{negl}(\lambda). \quad (4)$$

Where $(crs, pp) \leftarrow \text{Setup}(1^\lambda, T_t, n, t)$ with $crs = (crs_{vt}, N, \tau_t, n, t)$, and $(\tilde{x}^*, \pi^*) \leftarrow \mathcal{A}(crs_{vt}, pp)$.

4 Timed Threshold Proxy Signature Scheme

To optimize efficiency in time-sensitive scenarios, we adopt a pre-computation strategy that shifts computationally intensive, message-independent operations to an offline phase, reducing online latency for proxy signature generation and verification. Driven by this underlying architectural choice, the concrete construction of the Timed Threshold Proxy Signature (TTPS) scheme is detailed in Section 4.2.

4.1 Pre-computation Strategy

A critical bottleneck in combining Time-Lock Puzzles (TLPs) with threshold proxy signatures is the cost of publicly validating the time-lock instance. In a standard RSA-based TLP, checking that a published target value

$$Y = z^{2^T} \pmod{N}, \quad z \in \mathbb{Z}_N^*$$

is well-formed would require the verifier to re-execute the 2^T -power relation, which is prohibitively expensive when the time parameter T is large. To avoid this linear-time verification, we instantiate a pre-computation subprotocol inspired by the “check-before-you-solve” paradigm.

In our construction, the TLP layer and the share-consistency layer are separated:

- Y_i denotes the public target value of the time-lock puzzle associated with signer PS_i ;
- D_i denotes the commitment to the encoded proxy share s'_i ;
- the TLP layer is verified by PreCom.Verify ;
- Independently, a SNARK proof certifies that the same encoded share (s'_i, ρ_i) is: (i) committed in D_i , (ii) consistent with the proxy public

key $\hat{pk}_i$, and (iii) encrypted in C_i under the key $H_0(Y_i)$.

We write $l = \text{HashToPrime}(crs, z, Y, D, \hat{pk})$, where the hash input binds the TLP instance (z, Y) to the delegated-share commitment D and the proxy public key $\hat{pk}$.

$\text{PreCom}(crs, N, z, Y, D, \hat{pk}; T) \rightarrow \pi^{\text{pre}}$: On input public parameters crs , RSA modulus N , puzzle base $z \in \mathbb{Z}_N^*$, target value $Y \in \mathbb{Z}_N^*$, share commitment D , proxy public key $\hat{pk}$, and time parameter T , the prover performs:

Compute the challenge prime

$$l \leftarrow \text{HashToPrime}(crs, z, Y, D, \hat{pk}).$$

Compute the Euclidean division

$$2^T = q \cdot l + r, \quad 0 \leq r < l.$$

Compute the quotient commitment

$$Q = z^q \pmod{N}.$$

Output the proof $\pi^{\text{pre}} = (Q, r)$.

$\text{PreCom.Verify}(crs, N, z, Y, D, \hat{pk}, \pi^{\text{pre}}) \rightarrow \{0, 1\}$: Given $\pi^{\text{pre}} = (Q, r)$, the verifier recomputes

$$l \leftarrow \text{HashToPrime}(crs, z, Y, D, \hat{pk})$$

and accepts if $Q^l \cdot z^r \equiv Y \pmod{N}$.

The pre-computation proof certifies that the published TLP instance is well-formed. Independently, a SNARK proof certifies that the delegated-share commitment D_i is consistent with the encoded share s'_i and the proxy public key $\hat{pk}_i$. This separation avoids the notation clash between the time-lock target and the share commitment, and it yields a clean verification interface for the subsequent algorithms.

4.2 Correctness Proof of TTPS

Theorem 4.1: *If the Original Signer (OS) and Proxy Signers (PS_i) honestly execute the TTPS scheme, the timed delegation commitment and the proxy aggregate signature will be successfully verified.*

Proof sketch The claim follows from the correctness of the timed delegation phase and the correctness

of the aggregate signing equation. In particular, **Pre-Com.Verify** and **SNARK.Verify** accept honestly generated delegation tuples, while the final verification equation is obtained by combining Lagrange interpolation with the relation $sk = sk \cdot H(m_w, K) + k$. Details are deferred to Appendix A. $\square$

4.3 Concrete Construction

This subsection presents a concrete construction of the timed threshold proxy signature scheme (TTPS). The construction consists of three phases: (i) System Initialization, (ii) Time-Locked Delegation, and (iii) Proxy Signing and Verification.

All signature-related group operations (e.g., g^x and multiplicative products) are computed in the order- q subgroup $\langle g \rangle \subseteq \mathbb{Z}_p^*$, so we omit explicit reductions modulo p in group equations. All scalar arithmetic for secret keys, polynomials, and hash outputs used arithmetically is performed in $\mathbb{Z}_q$, so we omit explicit reductions modulo q . All time-lock puzzle computations involving the RSA modulus N are understood as operations in $\mathbb{Z}_N^*$, so we also omit explicit reductions modulo N . Whenever the trapdoor is used (e.g., for exponent reduction and inverses), computations are understood in $\mathbb{Z}_{\phi(N)}$, so we omit explicit reductions modulo $\phi(N)$.

The formal algorithms are given below. We first define the SNARK relation used in the timed delegation phase:

$$R_K = \left\{ ((\hat{p}k_i, D_i, Y_i, C_i); \hat{sk}_i, s'_i, \nu, \rho_i) : \begin{array}{l} D_i = g^{s'_i} h^{\rho_i} \wedge s'_i = \hat{sk}_i \cdot 2^{4\lambda} + \nu \wedge \\ \hat{p}k_i = g^{\hat{sk}_i} \wedge C_i = \text{Enc}_{H_0(Y_i)}(s'_i, \rho_i) \end{array} \right\}.$$

Algorithm 1 Setup($1^\lambda, T_t, n, t$)

Require: Security parameter λ , time-lock parameter T_t , number of signers n , threshold t

Ensure: Common reference string crs , public parameters pp

- 1: $(crs_{vt}, pp_{\text{snark}}) \leftarrow \text{SNARK.Setup}(1^\lambda, R_K)$
 - 2: $\tau_t \leftarrow 2^{T_t}$
 - 3: $pp \leftarrow (pp_{\text{snark}}, p, q, g, h, H, H_0)$
 - 4: $crs \leftarrow (crs_{vt}, N, \tau_t, n, t)$
 - 5: **return** (crs, pp)
-

Setup($1^\lambda, T_t, n, t$) $\rightarrow (crs, pp)$: On input the security parameter λ and the release time T_t for time-locked shares, this algorithm initializes both the signature layer and the verifiable time-lock component. It outputs global public parameters pp (including the cyclic group parameters (p, q, g) with g generating an order- q subgroup of $\mathbb{Z}_p^*$, and the public parameters produced by **SNARK.Setup**, a hash function H) and a common reference string $crs = (crs_{vt}, N, n, t)$. Here $N = p_N q_N$ is the RSA modulus used by the time-lock puzzle, and $\tau_t := 2^{T_t}$ denotes the time parameter corresponding to the release time T_t .

Algorithm 2 Kgen(pp)

Require: Public parameters pp

Ensure: Key pairs $\{(pk_i, sk_i)\}_{i=1}^n$, OS key pair (pk, sk) , warrant m_w

- 1: **for** $i = 1$ to n **do**
 - 2: $sk_i \xleftarrow{\$} \mathbb{Z}_q^*, pk_i \leftarrow g^{sk_i}$
 - 3: **end for**
 - 4: OS generate (pk, sk) and warrant m_w , where m_w includes identities, pk , threshold t , signer number n , and validity interval $[T_1, T_2]$
 - 5: Publish m_w on the electronic bulletin board
 - 6: **return** $(\{(pk_i, sk_i)\}_{i=1}^n, (pk, sk), m_w)$
-

Kgen(pp) $\rightarrow (\{(pk_i, sk_i)\}_{i \in [n]}, m_w, (pk, sk))$: Let $\mathbb{PS} = \{\text{PS}_1, \text{PS}_2, \dots, \text{PS}_n\}$ be a group of n proxy signers, $\mathbb{PS}$ represents the set of identity information of the actual proxy signers. The original signer (OS) and each proxy signer PS_i ($i \in [n]$) generate their own key pairs (pk_i, sk_i) . The OS generates the authorization warrant m_w , which contains the identities of OS and PS_i , the threshold parameter t , the number of proxy signers n , and the signing time interval $[T_1, T_2]$. The warrant m_w is then published on the Electronic Bulletin Board (BBS).

Algorithm 3 PKeyGen(sk, m_w)

Require: OS secret key sk , warrant m_w

Ensure: Delegation metadata K , proxy public key $\hat{pk}$, proxy shares $\{(\hat{sk}_i, \hat{pk}_i)\}_{i=1}^n$

- 1: $k \xleftarrow{\$} \mathbb{Z}_q^*, K \leftarrow g^k, \hat{sk} \leftarrow sk \cdot H(m_w, K) + k, \hat{pk} \leftarrow g^{\hat{sk}}$
- 2: Choose $a_1, \dots, a_{t-1} \xleftarrow{\$} \mathbb{Z}_q^*$

3: Define

$$f(x) = \hat{s}k + a_1x + a_2x^2 + \dots + a_{t-1}x^{t-1}$$

4: **for** $i = 1$ to n **do**

5: $\hat{s}k_i \leftarrow f(i)$, $\hat{p}k_i \leftarrow g^{\hat{s}k_i}$

6: **end for**

7: **return** $(K, \hat{p}k, \{(\hat{s}k_i, \hat{p}k_i)\}_{i=1}^n)$

PKeyGen $(sk, m_w) \rightarrow (\hat{s}k_i, \hat{p}k_i)$: Given OS's secret key sk , and a warrant m_w , the OS derives a proxy signing secret $\hat{s}k$ bound to m_w and a randomizer $K = g^k$ (for fresh $k \in \mathbb{Z}_q^*$). The value K is treated as public delegation metadata and is included in the final aggregate signature for verification. It sets $\hat{s}k = sk \cdot H(m_w, K) + k$ and $\hat{p}k = g^{\hat{s}k}$, then shares $\hat{s}k$ into per-signer shares using a degree- $(t-1)$ polynomial f with $f(0) = \hat{s}k$, and publishes/distributes the corresponding $\hat{p}k_i = g^{\hat{s}k_i}$ to PS_i .

Algorithm 4 TimedDel($crs, \hat{s}k_i, \hat{p}k_i, T_t$)

Require: Common reference string crs , proxy share $\hat{s}k_i$, proxy public key $\hat{p}k_i$, time-lock parameter T_t

Ensure: Commitment $comm_i$ and proof π_i

- 1: $\nu \xleftarrow{\$} \{0, 1, \dots, 2^{4\lambda} - 1\}$, $s'_i \leftarrow \hat{s}k_i \cdot 2^{4\lambda} + \nu$
 - 2: $\rho_i \xleftarrow{\$} \mathbb{Z}_q$, $D_i \leftarrow g^{s'_i h^{\rho_i}}$, $z_i \xleftarrow{\$} \mathbb{Z}_N^*$, $Y_i \leftarrow z_i^{2^{T_t}}$, $K_i \leftarrow H_0(Y_i)$
 - 3: $C_i \leftarrow \text{Enc}_{K_i}(s'_i, \rho_i)$
 - 4: $\pi_i^{\text{pre}} \leftarrow \text{PreCom}(crs, N, z_i, Y_i, D_i, \hat{p}k_i; T_t)$
 - 5: $\pi_i^R \leftarrow \text{SNARK.Prove}(crs_{vt}, R_K, (\hat{p}k_i, D_i, Y_i, C_i), s'_i, \hat{s}k_i, \nu, \rho_i)$
 - 6: $comm_i \leftarrow (z_i, Y_i, C_i, D_i)$
 - 7: $\pi_i \leftarrow (\pi_i^{\text{pre}}, \pi_i^R)$
 - 8: **return** $(comm_i, \pi_i)$
-

TimedDel $(crs, \hat{s}k_i, \hat{p}k_i, T_t) \rightarrow (comm_i, \pi_i)$: The original signer first encodes the proxy share $\hat{s}k_i$ as $s'_i = \hat{s}k_i \cdot 2^{4\lambda} + \nu$ using fresh blinding ν . It then samples fresh commitment randomness $\rho_i \leftarrow \mathbb{Z}_q$ and computes the commitment $D_i = g^{s'_i h^{\rho_i}}$. Separately, it samples a TLP instance (z_i, Y_i) satisfying $Y_i = z_i^{2^{T_t}} \bmod N$, derives a symmetric key $K_i = H_0(Y_i)$, and encrypts the encoded share as $C_i = \text{Enc}_{K_i}(s'_i, \rho_i)$ (where $\text{Enc}(x)$ is a symmetric encryption scheme with correctness). The proof π_i^R certifies the same witness $(\hat{s}k_i, s'_i, \nu, \rho_i)$

simultaneously explains the commitment D_i and the proxy public key $\hat{p}k_i$, and that s'_i is correctly encrypted in C_i under the key derived from the TLP target Y_i .

Algorithm 5

PPSign($crs, \hat{p}k_i, comm_i, \pi_i, m, \mathbb{PS}, \tau_s$)

Require: Common reference string crs , proxy public key $\hat{p}k_i$, commitment $comm_i$, proof π_i , message m , signer set $\mathbb{PS}$, timestamp τ_s

Ensure: $(1, \hat{\sigma}_i)$ or $(0, \perp)$

- 1: Parse $comm_i$ as (z_i, Y_i, C_i, D_i) , π_i as $(\pi_i^{\text{pre}}, \pi_i^R)$
 - 2: **if** PreCom.Verify($crs, N, z_i, Y_i, D_i, \hat{p}k_i, \pi_i^{\text{pre}}$) = 1 **and** SNARK.Verify($crs_{vt}, R_K, (\hat{p}k_i, D_i, Y_i, C_i), \pi_i^R$) = 1 **and** $\tau_s \in [T_1, T_2]$ **then**
 - 3: $\hat{\sigma}_i \leftarrow \text{TExSign}(comm_i, m, \mathbb{PS}, \tau_s)$
 - 4: **return** $(1, \hat{\sigma}_i)$
 - 5: **else**
 - 6: **return** $(0, \perp)$
 - 7: **end if**
-

Algorithm 6 TExSign($comm_i, m, \mathbb{PS}, \tau_s$)

Require: Commitment $comm_i$, message m , signer set $\mathbb{PS}$, timestamp τ_s

Ensure: Partial proxy signature $\hat{\sigma}_i$ or $\perp$

- 1: Parse $comm_i$ as (z_i, Y_i, C_i, D_i)
 - 2: $Y'_i \leftarrow \text{TLP.Solve}(z_i, N, T_t)$
 - 3: $K'_i \leftarrow H_0(Y'_i)$, $(s'_i, \rho_i) \leftarrow \text{Dec}_{K'_i}(C_i)$
 - 4: **if** $g^{s'_i h^{\rho_i}} \neq D_i$ **then**
 - 5: **return** $\perp$
 - 6: **end if**
 - 7: $\hat{s}k_i \leftarrow \left\lfloor \frac{s'_i}{2^{4\lambda}} \right\rfloor$
 - 8: $k_i \xleftarrow{\$} \mathbb{Z}_q^*$, $r_i \leftarrow g^{k_i}$
 - 9: Broadcast r_i to all signers in $\mathbb{PS}$, collect $\{r_j\}_{j \in \mathbb{PS}}$
 - 10: $\tilde{r} \leftarrow \prod_{j \in \mathbb{PS}} r_j$
 - 11: $w_i \leftarrow \prod_{\substack{j \in \mathbb{PS} \\ j \neq i}} \frac{0 - j}{i - j}$
 - 12: $\hat{\sigma}_i \leftarrow k_i + (\hat{s}k_i \cdot w_i + s k_i) \cdot H(\tilde{r}, m, \mathbb{PS}, \tau_s)$
 - 13: **return** $\hat{\sigma}_i$
-

PPSign $(crs, \hat{p}k_i, comm_i, \pi_i) \rightarrow ((1, \Sigma)/(0, \perp))$: Each proxy signer PS_i begins by validating the delegated commitment tuple $(comm_i, \pi_i)$ and its accompanying proofs. After the release time T_1 ,

PS_i invokes TExSign to recover the proxy signing share $\hat{sk}_i$. The signing phase then unfolds in two communication rounds. In Round 1, every PS_i chooses a fresh nonce $k_i \xleftarrow{\$} \mathbb{Z}_q^*$, computes $r_i = g^{k_i}$, and broadcasts r_i to the members of $\mathbb{PS}$; once all nonces are available, each signer derives the aggregate nonce $\tilde{r} = \prod_{j \in \mathbb{PS}} r_j$. In Round 2, every PS_i forms its partial signature $\hat{\sigma}_i$ and forwards it to a designated aggregator, who gathers at least t valid partial signatures, computes $\hat{\sigma} = \sum_{i \in \mathbb{PS}} \hat{\sigma}_i$, and outputs $\Sigma = (K, \tilde{r}, \hat{\sigma})$.

Algorithm 7 $\text{PVerify}(m_w, m, \Sigma, \mathbb{PS}, \tau_s)$

Require: Warrant m_w , message m , aggregate signature Σ , signer set $\mathbb{PS}$, timestamp τ_s

Ensure: 1 (accept) or 0 (reject)

```

1: Parse  $\Sigma$  as  $(K, \tilde{r}, \hat{\sigma})$ 
2: if  $\tau_s \notin [T_1, T_2]$ , or  $|\mathbb{PS}| < t$  then
3:   return 0
4: end if
5: if  $g^{\hat{\sigma}} = \tilde{r} \left( K \cdot pk^{H(m_w, K)} \cdot \prod_{i \in \mathbb{PS}} pk_i \right)^{H(\tilde{r}, m, \mathbb{PS}, \tau_s)}$ 
   then
6:   return 1
7: else
8:   return 0
9: end if
```

$\text{PVerify}(m_w, m, \Sigma, \mathbb{PS}, \tau_s) \rightarrow (1/0)$: The verifier V collects at least t valid partial signatures. On input the warrant m_w , message m , an aggregate proxy signature Σ , and the participating identity set $\mathbb{PS}$, the verifier V outputs 1 if verification succeeds (i.e., the signature is consistent with an authorized set of at least t proxy signers and the warrant constraints), and outputs 0 otherwise.

4.4 Proof of Security

Theorem 4.2 (UF-CMA Security): *Assume the Random Oracle Model (ROM) and the Discrete Logarithm (DL) Assumption hold. Then the proposed Timed Threshold Proxy Signature (TTPS) scheme is unforgeable against adaptive chosen-message attacks (UF-CMA).*

Proof sketch. The proof reduces UF-CMA security to the discrete-logarithm assumption for the Schnorr-type aggregate signing layer. Any valid aggregate proxy signature satisfies a Fiat–Shamir equation of the

form

$$g^{\hat{\sigma}} = \tilde{r} \cdot Y_{\mathbb{PS}}^c, \quad c = H(\tilde{r}, m, \mathbb{PS}, \tau_s).$$

Assume that a PPT adversary outputs a fresh valid forgery with non-negligible probability. The reduction embeds a DL challenge into the public key of a randomly chosen honest proxy signer, answers signing queries by honest signing or standard Schnorr simulation in the random oracle model, and uses the freshness condition to ensure that the final forgery involves the target signer with non-negligible probability. By the Forking Lemma, two accepting transcripts with the same $\tilde{r}$ and different hash challenges yield the discrete logarithm of the effective aggregate public key, from which the embedded secret key can be extracted after subtracting the known contributions of the other keys. This contradicts the DL assumption. Details are deferred to Appendix B. $\square$

Theorem 4.3 (Robustness): *If the delegated proxy secret is shared by a correct t -out-of- n secret sharing scheme, the commitment and encryption layers in TimedDel hide the encoded proxy shares, the delegation proof system is zero-knowledge and sound, and the underlying time-lock puzzle is secure before the release time, then the TTPS scheme satisfies Robustness (RB-CMA).*

Proof sketch. A successful RB-CMA adversary must either make an invalid delegation tuple pass verification, which breaks the soundness of the zk-SNARK for relation R_K , or produce an accepted aggregate containing an honest unqueried signer contribution, which yields a forgery against the underlying signing layer. Hence, robustness follows from SNARK soundness and DL-based unforgeability. Details are deferred to Appendix B. $\square$

Theorem 4.4 (Time-Lock Security): *Suppose the time-lock puzzle in TimedDel is secure for the time parameter T_1 , and the delegation proof system for relation R_K is zero-knowledge. Then the TTPS scheme satisfies time-lock security (TL-SC).*

Proof sketch. If a signature is produced with a time label outside $[T_1, T_2]$, then either it is rejected directly by the interval check, or it must correspond to a valid proxy contribution before maturity. The latter implies either early recovery of an unrevealed proxy share, breaking the time-lock puzzle, or a valid signing contribution without the corresponding delegated secret, contradicting unforgeability. Details are deferred to Appendix B. $\square$

Theorem 4.5 (Delegation Soundness). *If the underlying zk-SNARK for relation R_K is sound, then TTPS satisfies delegation soundness.*

Proof. Assume there exists a PPT adversary $\mathcal{A}$ such that $\text{Adv}_{\mathcal{A}}^{\text{DPS}}(\lambda)$ is non-negligible. Then with non-negligible probability, $\mathcal{A}$ outputs $(\vec{x}^*, \pi^*)$ with $\vec{x}^* \notin R_K$ but $\text{SNARK.Verify}(crs_{vt}, R_K, \vec{x}^*, \pi^*) = 1$. Construct adversary $\mathcal{B}$ against SNARK soundness: on input crs_{vt} , $\mathcal{B}$ runs $\mathcal{A}(crs_{vt}, pp)$ and outputs the same $(\vec{x}^*, \pi^*)$. Whenever $\mathcal{A}$ succeeds, $\mathcal{B}$ produces an accepting proof for a false statement in R_K , contradicting SNARK soundness. Hence,

$$\text{Adv}_{\mathcal{A}}^{\text{DPS}}(\lambda) \leq \text{Adv}_{\mathcal{B}}^{\text{sound}}(\lambda) \leq \text{negl}(\lambda).$$

Therefore, TTPS satisfies delegation soundness. $\square$

5 Performance Analysis

5.1 Performance of the Verifiable Time-Lock Puzzle

Table 2 reports the cost of the verifiable TLP primitive on a commodity laptop (AMD Ryzen 7 5800H) using a 2048-bit RSA modulus and calibrated delay parameter $T = 182685260$ (corresponding to 1800 s). In our implementation, we instantiate the underlying zk-SNARK with the Groth16 [18] proving system. The measured proving time is 2.08 s, while verification requires only 1 ms. According to the design of our pre-computation strategy, Under pre-compute, the circuit size is about 5.35×10^5 constraints and the proof size is 256 bytes, indicating that the online verification cost remains small and does not grow with the delay parameter during verification.

Table 2: Computation time of the time-lock puzzle

Stage / Component	Execution Time
Binding commitment D	2.883ms
HashToPrime l	1.389ms
Remainder computation $r = 2^T \bmod l$	0.006ms
Quotient reduction $q \bmod \phi(N)$	0.170ms
Quotient commitment $Q = u^q$	22.527ms
Residual proving overhead	2053.025ms
Total SNARK.Prove	2.08s
SNARK.Verify	1ms

These results show that the main proving overhead is paid offline, whereas public verification remains lightweight in practice.

5.2 Evaluation Methodology and Key Variables

As shown in Table 3, we distinguish the intended release delay from the measured cryptographic runtime. The delay is represented by the calibrated time-lock parameter T , where

$$T = \lceil \text{sqrps}(N) \cdot \text{delay_seconds} \rceil.$$

Accordingly, the reported runtimes measure the computational cost of the cryptographic procedures, while the intended waiting period is captured separately by T .

5.3 Performance of timed threshold proxy signature schemes

We summarize the theoretical computational cost of each stage of the proposed timed threshold proxy signature scheme in Table 4, and report measured runtimes under representative threshold settings in Table 5.

As shown in Table 4, the proxy key generation stage is primarily dominated by the Original Signer assisting in the derivation of proxy signing keys $\hat{sk}_i$; consequently, the cost increases with both n and t . In the timed delegation stage, the main overhead comes from producing the commitments comm_i and the associated proofs π_i , and therefore scales linearly in the number of proxy signers n . The partial signature generation stage measures the online cryptographic computation performed after the delegated proxy share becomes available according to the scheme semantics. Hence, it reflects the cost of local verification, signing arithmetic, and partial-signature generation, rather than the wall-clock release delay represented by the calibrated time-lock parameter. Finally, in the aggregate signature verification stage, the dominant operation is verifying the aggregate signature.

To illustrate the computational profile of the scheme under representative parameter choices, Table 5 reports the measured runtime of each cryptographic stage separately.

The data in table 5 support the expected scaling trends with respect to the key variables n and t . First, the timed delegation stage grows approximately linearly in n , consistent with the per-signer proof-generation overhead reported in

Table 3: Calibration of T to real-world time on target hardware

$ N $ (bits)	sqrps(N) (squarings/s)	Target delay (s)	Calibrated T (squarings)
2048	1.01492×10^5	60	6 089 509
2048	1.01492×10^5	300	30 447 544
2048	1.01492×10^5	1800	182 685 260

Table 4: The theoretical time cost of timed threshold proxy signatures

Stage	Theoretical Value
Proxy key generation stage	$T_H + (n + t)T_{\text{ex}} + (nt - n)T_m$
Timed delegation stage	$n \cdot T_{\text{SNARK.Prove}} + n \cdot T_{\text{commit}}$
Public puzzle verification (with pre-computation)	$T_{\text{verify}} = O(1)$
Public puzzle verification (without pre-computation)	$T_{\text{verify}} = O(T)$
Partial signature generation stage	$T_{\text{SNARK.Verify}} + T_{\text{Sign}}$
Aggregate signature verification stage	$tT_{\text{ex}} + (t + 2)T_m + 2T_H$

Note: n represents the number of signers, and t represents the threshold of the threshold signature.

table 2. Second, aggregate signature verification increases with t , matching the theoretical dependence shown in table 4. Third, the public puzzle-verification path remains in the millisecond range under pre-computation, and for a fixed signer-set size, its per-instance verification cost does not grow with the delay parameter T .

Overall, these results indicate that the proposed TTPS construction is practical for time-sensitive delegated signing scenarios. The timed delegation phase incurs an offline cost because the original signer must generate one proof per proxy signer, whereas the online public verification path remains lightweight after the delegation material has been published. This separation between offline proof generation and lightweight online verification is the main efficiency benefit of the proposed design.

6 Related Work

Existing work related to TTPS mainly falls into three lines: timed signatures, threshold signatures, and threshold proxy signatures.

In timed cryptography, early foundations such as [19] and [13] established computational delay as a cryptographic primitive, and later work such as [20] extended this line toward richer functionality. More recently, [12] introduced public verifiability for timed signatures. However, these constructions do not address threshold delegation, and direct circuitization of sequential squaring

still incurs overhead that scales with the delay parameter, which limits practical deployment in delay-sensitive settings.

Threshold signatures provide robustness for distributed signing, as illustrated by [9], [21], [22], and [23]. This line of work has substantially improved adaptive security, setup assumptions, and signer-set expressiveness. Nevertheless, its primary focus is distributed key management and signing correctness, rather than cryptographically enforced activation time. As a result, existing threshold-signature constructions cannot directly support delayed delegation or publicly verifiable time-bounded authorization.

Proxy-signature research studies delegation under explicit authorization, starting from [2] and continuing with threshold or system-oriented variants such as [24], [10], [11], and [6]. These schemes improve delegation efficiency or application integration, but they generally rely on static authorization or revocation-based control rather than cryptographically enforced delay. Related timed proxy-signature work, including [8], moves closer to our setting, yet still does not provide a unified construction that simultaneously supports threshold delegation, resistance to premature share accumulation, and efficient public verification of puzzle solvability.

As a result, these three lines of work address temporal control, distributed signing, and delegation only in a largely separate manner. Table 6 summarizes this gap more directly.

Table 5: The time cost of timed threshold proxy signatures with different threshold values (n, t)

Stage	(5, 3)	(8, 4)	(10, 5)
Proxy key generation stage	9.705 ms	14.99 ms	19.21 ms
Timed delegation stage	10.41 s	16.65 s	20.82 s
Public puzzle verification (with pre-computation)	3.00 ms	4.00 ms	5.00 ms
Public puzzle verification (without pre-computation)	2.091 s	2.788 s	3.485 s
Partial signature generation stage	3.00 ms	3.00 ms	3.00 ms
Aggregate signature verification stage	3.714 ms	4.91 ms	6.11 ms

Table 6: Comparison of Functionalities and Security Objectives

Metric	Guo et al. [10]	Hu et al. [6]	Das et al. [9]	Tang et al. [11]	Xu et al. [12]	Ours (TTPS)
Functionalities						
Proxy Delegation	✓	✓	×	✓	×	✓
Time-Lock Mechanism	×	×	×	×	✓	✓
Puzzle Verifiability	×	×	×	×	✓ ($O(T)$)	✓ ($O(1)$)
Security Objectives						
Unforgeability	✓	✓	✓	✓	✓	✓
Robustness	×	✓	✓	✓	✓	✓
Corruption attacks	×	×	×	×	✓	✓
Key Irrecoverability	×	×	×	×	✓	✓

7 Conclusion

This paper addresses a gap in existing delegated-signing systems by introducing a formally defined Timed Threshold Proxy Signature (TTPS) scheme. TTPS combines threshold proxy signing with a time-lock mechanism so that delegated proxy shares remain computationally unrecoverable before a designated release time. To improve efficiency, we incorporate a pre-computation strategy that reduces the per-instance public verification cost of the time-lock component from linear dependence on the delay parameter T to $O(1)$. Our analysis shows that the resulting framework supports unforgeability, robustness, delayed-share security before release, and delegation soundness under the stated assumptions. Experimental results further indicate that the online public verification path remains lightweight in practice, while the heavier cost is concentrated in the offline delegation phase. Future work includes tightening the trust model and exploring post-quantum instantiations of the proposed framework.

References

- [1] Hsu, C.-L., Wu, T.-S., Wu, T.-C.: New non-repudiable threshold proxy signature scheme with known signers. *Journal of Systems and Software* **58**(2), 119–124 (2001)
- [2] Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures: Delegation of the power to sign messages. *IEICE transactions on fundamentals of electronics, communications and computer sciences* **79**(9), 1338–1354 (1996)
- [3] Desmedt, Y., Frankel, Y.: Shared generation of authenticators and signatures. In: *Annual International Cryptology Conference*, pp. 457–469 (1991). Springer
- [4] Reichert, B.M., Obelheiro, R.R.: Software supply chain security: a systematic literature review. *International Journal of Computers and Applications* **46**(10), 853–867 (2024)
- [5] Samuel, J., Mathewson, N., Cappos, J., Dingledine, R.: Survivable key compromise in software update systems. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pp. 61–72 (2010)

- [6] Hu, J., Zhang, J.: Cryptanalysis and improvement of a threshold proxy signature scheme. *Computer Standards & Interfaces* **31**(1), 169–173 (2009)
- [7] Branco, R., Gueron, S.: Blinded random corruption attacks. In: 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 85–90 (2016). IEEE
- [8] Özden, D., Yayla, O.: Verifiable timed proxy signatures and multi-signatures (2025)
- [9] Das, S., Ren, L.: Adaptively secure bls threshold signatures from ddh and co-cdh. In: Annual International Cryptology Conference, pp. 251–284 (2024). Springer
- [10] Guo, Q., Zhang, D., Chang, L., Liu, X., Song, J.: Design and implementation of proxy-protected proxy signature based on sm2. *Chin. J. Netw. Inf. Secur* **3**, 47–54 (2017)
- [11] Tang, F., Xu, T., Peng, J., Gan, N.: Tppbft: A scalable pbft based on threshold proxy signature for iot-blockchain applications. *IEEE Internet of Things Journal* **11**(9), 15434–15449 (2023)
- [12] Zhou, X., He, D., Ning, J., Luo, M., Huang, X.: Efficient construction of verifiable timed signatures and its application in scalable payments. *IEEE Transactions on Information Forensics and Security* **18**, 5345–5358 (2023)
- [13] Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
- [14] Xin, J., Papadopoulos, D.: “check-before-you-solve”: Verifiable time-lock puzzles. In: 2025 IEEE Symposium on Security and Privacy (SP), pp. 2037–2056 (2025). IEEE
- [15] Gordon, D.: Discrete logarithm problem. In: *Encyclopedia of Cryptography, Security and Privacy*, pp. 678–679. Springer, Berlin, Heidelberg (2025)
- [16] Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: *Providing Sound Foundations for Cryptography: on the Work of Shafi Goldwasser and Silvio Micali*, pp. 329–349 (2019)
- [17] WhiteHat, B., Baylina, J., Bellés, M.: Baby jubjub elliptic curve. *Ethereum Improvement Proposal*, EIP-2494 **29**, 27 (2020)
- [18] Groth, J.: On the size of pairing-based non-interactive arguments. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 305–326 (2016). Springer
- [19] Boneh, D., Naor, M.: Timed commitments. In: *Annual International Cryptology Conference*, pp. 236–254 (2000). Springer
- [20] Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: *Annual International Cryptology Conference*, pp. 620–649 (2019). Springer
- [21] Bacho, R., Loss, J.: On the adaptive security of the threshold bls signature scheme. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 193–207 (2022)
- [22] Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: hints: Threshold signatures with silent setup. In: 2024 IEEE Symposium on Security and Privacy (SP), pp. 3034–3052 (2024). IEEE
- [23] Baird, L., Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: Threshold signatures in the multiverse. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 1454–1470 (2023). IEEE
- [24] Hwang, M.-S., Lu, E.J.-L., Lin, I.-C.: A practical (t, n) threshold proxy signature scheme based on the rsa cryptosystem. *IEEE Transactions on Knowledge and Data Engineering* **15**(6), 1552–1560 (2003)

Appendix A Correctness of the TTPS

A.1 Correctness of Timed Delegation

We prove that, if the original signer follows Algorithm 4 honestly, then both the pre-computation proof and the share-consistency proof are accepted, and the proxy signer later recovers the correct delegated share.

For a fixed signer PS_i , let

$$l_i = \text{HashToPrime}(crs, z_i, Y_i, D_i, \hat{p}k_i),$$

and let the Euclidean division of the time exponent be

$$2^{T_i} = q_i l_i + r_i, \quad 0 \leq r_i < l_i.$$

By construction,

$$Q_i = z_i^{q_i} \pmod{N}, \quad \pi_i^{pre} = (Q_i, r_i).$$

Hence, the verifier computes

$$Q_i^{l_i} \cdot z_i^{r_i} = (z_i^{q_i})^{l_i} \cdot z_i^{r_i} = z_i^{q_i l_i + r_i} = z_i^{2^{T_i}} \equiv Y_i.$$

Therefore,

$$\text{PreCom.Verify}(crs, N, z_i, Y_i, D_i, \hat{p}k_i, \pi_i^{pre}) = 1.$$

Next, consider the SNARK proof. By the definition of the relation R_K , the witness $(s'_i, \hat{s}k_i, \nu, \rho_i)$ satisfies $D_i = g^{s'_i} h^{\rho_i}$, $s'_i = \hat{s}k_i \cdot 2^{4\lambda} + \nu$, $g^{\hat{s}k_i} = \hat{p}k_i$, $C_i = \text{Enc}_{K_i}(s'_i, \rho_i)$. Therefore, for an honestly generated witness,

$$\text{SNARK.Verify}(crs_{vt}, R_K, (\hat{p}k_i, D_i, Y_i, C_i), \pi_i^R) = 1$$

by completeness of the SNARK system.

After the release time, the proxy signer solves the puzzle:

$$Y'_i = \text{TLP.Solve}(z_i, N, T_i) = z_i^{2^{T_i}} \equiv Y_i \pmod{N}.$$

Thus, the derived key is identical:

$$H_0(Y'_i) = H_0(Y_i) = K_i.$$

Hence, decryption succeeds and returns the same encoded share:

$$\text{Dec}_{H_0(Y'_i)}(C_i) = \text{Dec}_{K_i}(C_i) = (s'_i, \rho_i).$$

The signer then checks

$$g^{s'_i} h^{\rho_i} = D_i,$$

which holds by construction.

Finally, since

$$s'_i = \hat{s}k_i \cdot 2^{4\lambda} + \nu \quad \text{with} \quad 0 \leq \nu < 2^{4\lambda},$$

we recover the delegated proxy share uniquely as

$$\hat{s}k_i = \left\lfloor \frac{s'_i}{2^{4\lambda}} \right\rfloor.$$

Moreover,

$$g^{\hat{s}k_i} = \hat{p}k_i,$$

so the recovered share is consistent with the published proxy public key. Therefore, the timed delegation phase is correct.

A.2 Correctness of Proxy Aggregate Signature

Assume at least t proxy signers in the set $\mathbb{PS}$ honestly recover their delegated shares and produce partial proxy signatures

$$\hat{\sigma}_i = k_i + (\hat{s}k_i \cdot w_i + sk_i) \cdot H(\tilde{r}, m, \mathbb{PS}, \tau_s),$$

where

$$r_i = g^{k_i}, \quad \tilde{r} = \prod_{i \in \mathbb{PS}} r_i.$$

The aggregate signature is

$$\hat{\sigma} = \sum_{i \in \mathbb{PS}} \hat{\sigma}_i.$$

Therefore,

$$\hat{\sigma} = \sum_{i \in \mathbb{PS}} k_i + H(\tilde{r}, m, \mathbb{PS}, \tau_s) \left(\sum_{i \in \mathbb{PS}} \hat{s}k_i w_i + \sum_{i \in \mathbb{PS}} sk_i \right).$$

Exponentiation by g yields

$$g^{\hat{\sigma}} = g^{\sum_{i \in \mathbb{PS}} k_i} \cdot g^{H(\tilde{r}, m, \mathbb{PS}, \tau_s) (\sum_{i \in \mathbb{PS}} \hat{s}k_i w_i + \sum_{i \in \mathbb{PS}} sk_i)}$$

$$\begin{aligned}
&= \left(\prod_{i \in \mathbb{PS}} g^{k_i} \right) \cdot \left(g^{\sum_{i \in \mathbb{PS}} \hat{s}k_i w_i} \cdot g^{\sum_{i \in \mathbb{PS}} sk_i} \right)^{H(\tilde{r}, m, \mathbb{PS}, \tau_s)} \\
&= \tilde{r} \cdot \left(g^{\hat{s}k} \cdot \prod_{i \in \mathbb{PS}} pk_i \right)^{H(\tilde{r}, m, \mathbb{PS}, \tau_s)},
\end{aligned}$$

where the interpolation coefficients w_i satisfy

$$\sum_{i \in \mathbb{PS}} \hat{s}k_i w_i = \hat{s}k.$$

From the proxy-key generation algorithm,

$$\hat{s}k = sk \cdot H(m_w, K) + k.$$

Hence

$$g^{\hat{s}k} = (g^{sk})^{H(m_w, K)} \cdot g^k = pk^{H(m_w, K)} \cdot K.$$

Substituting this into the previous equation gives

$$g^{\hat{\sigma}} = \tilde{r} \cdot \left(K \cdot pk^{H(m_w, K)} \cdot \prod_{i \in \mathbb{PS}} pk_i \right)^{H(\tilde{r}, m, \mathbb{PS}, \tau_s)}.$$

This is exactly the verification equation in PVerify. Therefore, the aggregate proxy signature produced by honest signers is always accepted.

Appendix B Proof of Security

In this consolidated appendix, we provide the detailed formal proofs for the core security properties of our proposed Timed Threshold Proxy Signature (TTPS) scheme. Specifically, we first prove its existential unforgeability under chosen-message attacks in Appendix B.1. Subsequently, we demonstrate its robustness against premature recovery in Appendix B.2, followed by the proof of time-lock security against malicious proxy collusions in Appendix B.3.

B.1 Proof of Unforgeability

Theorem 4.2 (UF-CMA Security): *Assume the Random Oracle Model (ROM) and the Discrete Logarithm (DL) Assumption hold. Then the proposed Timed Threshold Proxy Signature (TTPS) scheme is unforgeable against adaptive chosen-message attacks (UF-CMA).*

Proof. Let $(g, X = g^x)$ be a DL challenge. We construct $\mathcal{B}$ that solves DL using a successful UF-CMA adversary $\mathcal{A}$.

Setup. $\mathcal{B}$ picks a target index $i^* \xleftarrow{\$} [n]$ and sets $pk_{i^*} := X = g^x$. For every $j \in [n] \setminus \{i^*\}$, it samples $sk_j \xleftarrow{\$} \mathbb{Z}_q^*$ and sets $pk_j = g^{sk_j}$. $\mathcal{B}$ generates the OS key pair (sk, pk) honestly, samples $k \xleftarrow{\$} \mathbb{Z}_q^*$, sets $K = g^k$, and publishes m_w .

Since $\mathcal{B}$ knows sk , it computes

$$\hat{s}k = sk \cdot H(m_w, K) + k,$$

chooses a degree- $(t-1)$ polynomial f with $f(0) = \hat{s}k$, and derives all proxy shares $\hat{s}k_i = f(i)$ and proxy public keys $\hat{p}k_i = g^{\hat{s}k_i}$. For every $i \in [n]$, $\mathcal{B}$ runs TimedDel honestly (since it knows each $\hat{s}k_i$) to produce $(comm_i, \pi_i)$. It sends $(pp, crs, pk, m_w, \{\hat{p}k_i\}, \{(comm_i, \pi_i)\})$ to $\mathcal{A}$.

Corruption oracle $\mathcal{O}_c(i)$. If $i = i^*$, $\mathcal{B}$ aborts. Otherwise, it returns sk_i and moves i to $\mathcal{PC}$. Since $|\mathcal{PC}| \leq t-1$ and i^* is uniform, the probability that i^* is never corrupted is at least $\frac{n-t+1}{n}$.

Signing oracle $\mathcal{O}_p(i, m)$. For $i \neq i^*$, $\mathcal{B}$ knows both $\hat{s}k_i$ and sk_i , so it signs honestly.

For $i = i^*$, $\mathcal{B}$ knows $\hat{s}k_{i^*}$ but not $sk_{i^*} = x$. Let $\mathbb{PS}$ be the signer set of the current session and define

$$P_{i^*} = \hat{p}k_{i^*}^{w_{i^*}} \cdot pk_{i^*},$$

where $w_{i^*} = \prod_{j \in \mathbb{PS}, j \neq i^*} \frac{0-j}{i^*-j}$ is the Lagrange coefficient. Note that $P_{i^*} = g^{\hat{s}k_{i^*} w_{i^*} + sk_{i^*}}$ is the effective public key of signer i^* ; $\mathcal{B}$ can compute it from $\hat{p}k_{i^*}$, w_{i^*} , and $pk_{i^*} = X$.

$\mathcal{B}$ first samples $k_j \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $r_j = g^{k_j}$ honestly for all $j \in \mathbb{PS} \setminus \{i^*\}$. It then samples $z, c \xleftarrow{\$} \mathbb{Z}_q$ and sets

$$r_{i^*} = g^z \cdot P_{i^*}^{-c}, \quad \tilde{r} = r_{i^*} \cdot \prod_{j \in \mathbb{PS} \setminus \{i^*\}} r_j.$$

It programs the random oracle so that $H(\tilde{r}, m, \mathbb{PS}, \tau_s) = c$, and sets $\hat{\sigma}_{i^*} = z$.

To verify consistency: $g^{\hat{\sigma}_{i^*}} = g^z = r_{i^*} \cdot P_{i^*}^c$, which matches the real distribution. For every other honest signer $j \in \mathbb{PS} \setminus \{i^*\}$, $\mathcal{B}$ computes $\hat{\sigma}_j = k_j + (\hat{s}k_j w_j + sk_j) \cdot c$ honestly (since it knows all required values). The combined result is returned to $\mathcal{A}$.

Extraction. Suppose $\mathcal{A}$ outputs a forgery $(m^*, \Sigma^*, \mathbb{PS}^*)$ such that PVerify accepts and $(\mathbb{PS}^* \cap \mathcal{PH}) \cap \mathcal{PQ}[m^*] = \emptyset$. If $i^* \notin \mathbb{PS}^*$, $\mathcal{B}$ aborts. Since the freshness condition requires at least one honest unqueried signer in $\mathbb{PS}^*$, and i^* is chosen independently and uniformly, this happens with probability $\geq 1/n$.

The verification equation gives

$$g^{\hat{\sigma}^*} = \tilde{r}^* \cdot Y_{\mathbb{PS}^*}^{c^*},$$

where $Y_{\mathbb{PS}^*} = K \cdot pk^{H(m_w, K)} \cdot \prod_{i \in \mathbb{PS}^*} pk_i$ and $c^* = H(\tilde{r}^*, m^*, \mathbb{PS}^*, \tau_s^*)$.

By the General Forking Lemma, $\mathcal{B}$ rewinds $\mathcal{A}$ at this random-oracle query and obtains, with non-negligible probability, two accepting transcripts

$$(\tilde{r}^*, \hat{\sigma}^*, c^*) \quad \text{and} \quad (\tilde{r}^*, \hat{\sigma}^{**}, c^{**}), \quad c^* \neq c^{**}.$$

Dividing the two verification equations:

$$g^{\hat{\sigma}^* - \hat{\sigma}^{**}} = Y_{\mathbb{PS}^*}^{c^* - c^{**}},$$

so $\mathcal{B}$ extracts

$$y_{\mathbb{PS}^*} = (\hat{\sigma}^* - \hat{\sigma}^{**})(c^* - c^{**})^{-1} \pmod{q}.$$

By definition,

$$y_{\mathbb{PS}^*} = \log_g Y_{\mathbb{PS}^*} = \hat{s}k + \sum_{i \in \mathbb{PS}^*} sk_i.$$

Since $\mathcal{B}$ knows $\hat{s}k$ and every sk_j for $j \neq i^*$, it computes

$$x = sk_{i^*} = y_{\mathbb{PS}^*} - \hat{s}k - \sum_{j \in \mathbb{PS}^* \setminus \{i^*\}} sk_j \pmod{q}.$$

Success probability. $\mathcal{B}$ does not abort if (i) i^* is never corrupted (probability $\geq \frac{n-t+1}{n}$) and (ii) $i^* \in \mathbb{PS}^*$ (probability $\geq \frac{1}{n}$). Combined with the Forking Lemma,

$$\epsilon' \geq \frac{n-t+1}{n^2} \cdot \text{frk}(\epsilon, q_H),$$

where q_H is the number of random-oracle queries. Under the DL assumption ϵ' is negligible, hence ϵ must be negligible, proving UF-CMA security. $\square$

B.2 Proof of Robustness

Theorem 4.3 (Robustness): *If the delegated proxy secret is shared by a correct t -out-of- n secret sharing scheme, the commitment and encryption layers in TimedDel hide the encoded proxy shares, the delegation proof system is zero-knowledge and sound, and the underlying time-lock puzzle is secure before the release time, then the TTPS scheme satisfies Robustness (RB-CMA).*

Proof. Assume, toward contradiction, that there exists a PPT adversary $\mathcal{A}$ that wins the RB-CMA experiment with non-negligible probability. Then $\mathcal{A}$ outputs a value $\hat{s}k^*$ such that

$$g^{\hat{s}k^*} = K \cdot pk^{H(m_w, K)}$$

while having obtained fewer than t valid delegated proxy shares.

By construction, the delegated proxy secret $\hat{s}k$ is shared by a degree- $(t-1)$ polynomial $f(x)$ with

$f(0) = \hat{s}k$. Hence, any set of fewer than t shares does not determine $\hat{s}k$ uniquely. Therefore, in order to reconstruct $\hat{s}k$, the adversary must extract additional information about one or more unrecovered shares from the public delegation material.

However, each delegated share is protected in three layers. First, the encoded share s'_i is hidden inside the commitment

$$D_i = g^{s'_i} h^{\rho_i},$$

which does not reveal s'_i under the hiding property of the commitment. Second, the same encoded share is encrypted in

$$C_i = \text{Enc}_{H_0(Y_i)}(s'_i, \rho_i),$$

so recovering (s'_i, ρ_i) from C_i before the release time would break the security of the time-lock puzzle and the underlying symmetric encryption. Third, the proof π_i reveals no witness information beyond the validity of the delegation tuple by the zero-knowledge property of the proof system.

Therefore, any adversary that reconstructs $\hat{s}k$ from fewer than t valid shares either:

- (1) breaks the threshold secret-sharing requirement by recovering the constant term of a degree- $(t-1)$ polynomial from fewer than t points; or
- (2) breaks the hiding / zero-knowledge / time-lock protection of the delegation layer. All such events occur only with negligible probability. Hence, the TTPS scheme satisfies RB-CMA security. $\square$

B.3 Proof of Time-lock Security

Theorem 4.4 (Time-Lock Security): *Suppose the time-lock puzzle in TimedDel is secure for the time parameter T_1 , and the delegation proof system for the relation $\mathcal{R}_K$ is zero-knowledge. Then the TTPS scheme satisfies time-lock security (TL-SC).*

Proof. We prove the theorem via a sequence of hybrid games, denoted Game_i for $i \in \{0, 1, 2\}$, where S_i is the event that the experiment outputs 1 in Game_i .

- **Game₀ (Real):** This is the real experiment (3.2) with honest algorithms.
- **Game₁:** Identical to Game_0 , except that all delegation-related proofs in the public transcript are generated using the zero-knowledge simulator. By the zero-knowledge property, the adversary's view is computationally indistinguishable from Game_0 .

$$|\Pr[S_1] - \Pr[S_0]| \leq n \cdot \text{Adv}_{\text{ZS}}^{\text{ZK}}(\lambda).$$

- **Game₂:** In Game₁, for every honest proxy signer, the time-lock instance in $comm_i$ is replaced by an instance that is independent of the encoded secret share (i.e., $comm_i$ is a random time-lock puzzle instance). By the security of the time-lock puzzle, any PPT adversary running in time $T < T_1$ cannot distinguish Game₂ from Game₁ except with negligible probability:

$$|\Pr[S_2] - \Pr[S_1]| \leq n \cdot \text{Adv}^{\text{TLP}}(\lambda).$$

In Game₂, the adversary's view before time T_1 is independent of the hidden proxy key shares, except for public information. Thus, $\Pr[S_2] \leq \text{negl}(\lambda)$. Combining the above, we have

$$\begin{aligned} \text{Adv}_{\mathcal{A},T}^{\text{TL-SC}}(\lambda) = \Pr[S_0] &\leq n \cdot \text{Adv}_{\text{ZS}}^{\text{ZK}}(\lambda) \\ &+ n \cdot \text{Adv}^{\text{TLP}}(\lambda) + \text{negl}(\lambda). \end{aligned}$$

Consequently, the view of any adversary who adaptively corrupts up to $t - 1$ proxy nodes remains independent of the uncorrupted nodes' secret shares prior to time T_1 . This formally guarantees that the TTPS scheme is resistant against premature share accumulation via malicious collusions, directly addressing the corruption vulnerabilities discussed in the introduction.

Which proves the theorem. $\square$