From TS-SUF-2 to TS-SUF-4: Practical Security Enhancements for FROST2 Threshold Signatures

Will Wang, Syh-Yuan Tan, Ryan Chow, Chanson Chan, and Martin Zhao

Solv Protocol, Singapore
{will,ryan,chanson,martin}@solv.finance, sytansy@gmail.com

Abstract. Threshold signature schemes play a vital role in securing digital assets within blockchain and distributed systems. FROST2 stands out as a practical threshold Schnorr signature scheme, noted for its efficiency and compatibility with standard verification processes. However, under the one-more discrete logarithm assumption, with static corruption and centralized key generation settings, FROST2 has been shown by Bellare et al. (in CRYPTO 2022) to achieve only TS-SUF-2 security, which is a consequence of its vulnerability to TS-UF-3 attacks.
In this paper, we address this security limitation by presenting an enhanced variant of FROST2, namely, FROST2+ which achieves the TS-SUF-4 security level under the same computational assumptions as the original FROST2. FROST2+ strengthens FROST2 by integrating additional pre-processing token verifications that help mitigate TS-UF-3 and TS-UF-4 vulnerabilities while maintaining practical efficiency. We show that FROST2+ can achieve TS-SUF-4 security not only under the same conditions as the original FROST2 analysis, but also when initialized with a distributed key generation protocol such as PedPoP. Our benchmark using ZCash's FROST library shows that the performance of FROST2+ is comparable to FROST2 and about $64-79\%$ faster than FROST when precomputation is enabled.

Keywords: Schnorr · Threshold Signature · FROST · TS-SUF · Provable Security

1 Introduction

The Schnorr signature scheme [28] has emerged as a cornerstone for blockchain applications due to its efficient signing and verification processes, compact signatures, and support for aggregation. In this context, Schnorr-based threshold signatures are particularly critical for threshold signing in blockchain systems [24,20,1,27], with FROST [22] establishing itself as a benchmark and currently undergoing IETF standardization [12]. Furthermore, NIST's recent call for contributions on threshold multi-party schemes [9] has intensified research interest in FROST.

Research related to FROST can be categorized into two main areas: the development of new schemes that offer enhanced efficiency [15,8,6,23], and improvements in security [26,11,2,10,3,25,4]. The latter category can be further divided

into specific areas of security analysis focused on distributed key generation [11,15,6] and adaptive security [16,14,10,18,2,25,17,4]. Several works [8,6,13,14,18] delve deeper into the security notions relevant to threshold signature schemes.

The TS-(S)UF-i security framework [8,6] represents a recent effort to rigorously formalize the security of threshold signature schemes. This framework groups a message M along with its auxiliary signing information as a leader request (lr) and classifies unforgeability levels according to adversarial capabilities, ranging from the weakest (TS-UF-0, which resists less capable attackers) to the strongest (TS-UF-4, which resists more capable attackers). While the framework typically describes forgeries in terms of triviality, let $c < t$ denotes the number of corrupted signers, we outline the attackers' abilities at each level for clarity:

1. TS-UF-0: The threshold scheme can withstand an attacker who cannot obtain any partial signatures from honest signers for the target message M.
2. TS-UF-1: The threshold scheme can withstand an attacker at TS-UF-0 who also queries at most $t - c - 1$ partial signatures on M from honest signers.
3. TS-UF-2: The threshold scheme can withstand an attacker at TS-UF-1 who also queries leader requests (lr) on M and obtain at most $t - c - 1$ honest partial signatures per lr.
4. TS-UF-3: The threshold scheme can resist an attacker at TS-UF-2 who also corrupts pre-processing tokens (pp_i) for honest signers $\mathsf{id}_i \in lr.\mathcal{S}$ in the signing set, but not for all non-responding honest signers in the same set.
5. TS-UF-4: The threshold scheme can resist an attacker at TS-UF-3 who corrupts pre-processing tokens (pp_i) for all honest signers $\mathsf{id}_i \in lr.\mathcal{S}$ in the signing set, though it cannot obtain partial signatures from every honest signer in $lr.\mathcal{S}$.

where the descriptions of TS-UF-3 and TS-UF-4 refer to the adversary capabilities in echo schemes [6,8,13] like FROST. In such schemes, an honest signer id_i only produces a partial signature if the leader request (lr) contains id_i's honestly generated pre-processing token $pp_i \in lr.\mathsf{PP}(\mathsf{id}_i)$. The framework also introduces the concept of strong unforgeability (TS-SUF-i), which ties a request lr to each unique signature, beyond just the message M.

Moreover, the framework provides valuable insights into the relationships among security levels. For instance, it demonstrates that achieving TS-UF-0 security implies TS-UF-1 security, risking a security loss of $\binom{n}{t-1}$ for the total number of signers n and threshold t. Additionally, it shows that combining a threshold scheme achieving TS-UF-3 security with a standard digital signature scheme with strong existential unforgeability under chosen-message attacks (SUF-CMA) results in a threshold scheme with TS-UF-4 security.

Within this security framework, both FROST and FROST2 achieve TS-UF-0 (and thereby TS-UF-1) security in the distributed key generation (DKG) setting [22,15,6], which represents the baseline security level within the TS-UF-i hierarchy. Under the centralized key generation (CKG) setting, FROST achieves TS-SUF-3 security and can be further improved to attain the highest TS-SUF-4 security level by incorporating an additional digital signature scheme for each signer

[8,6,13]. However, this approach introduces additional computational overhead and implementation complexity. More critically for our work, FROST2 [15,8,6] has been shown to achieve only TS-SUF-2 security and does not qualified for this security upgrade.

The limitation of FROST2 to TS-SUF-2 security arises from a fundamental vulnerability to TS-UF-3 attacks, as demonstrated by Bellare et al. [8,6]. This attack is reminiscent of the rogue public key attack in multi-signature schemes, exploiting a weakness in FROST2's pre-processing token mechanism to forge a Schnorr signature. Consequently, this enables a forgery under an unauthorized signer set without needing to know the partial signatures of all honest signers, thereby compromising TS-UF-3 security (see Appendix A.2).

Enhancements to FROST2 have been proposed, but none have achieved TS-SUF-3 security or higher. For instance, ROAST [26] introduces token aggregation and signature share verification, improving robustness while acknowledging security reduction to the original FROST2. Its successor, Olaf [11], combines ROAST with a simplified DKG protocol (SimplPedPoP) from FROST's PedPoP, proving its TS-SUF-2 security. Recent theoretical results [14,17] indicate that FROST, FROST2, and FROST3 (or ROAST) are either unforgeable with full adaptive security in the algebraic group model (AGM), under the assumptions of Algebraic One-More Discrete Logarithm (AOMDL) and Low-Dimensional Vector Representation (LDVR); or adaptively secure with a maximum of $t/2$ corruptions, assuming AOMDL in the random oracle model (ROM).

FROST-Mask [10], which can be viewed as an extension of FROST2 with masking mechanisms applied to the pre-processing tokens and partial signatures, has been proven unforgeable with adaptive security under the ROM. Conversely, ms-FROST [2] and FaFROST [4] apply masking solely to the partial signature, proven to achieve TS-UF-4 and adp-TS-UF, respectively, with adaptive security under the weaker AGM.

The security notions TS-SUF-3 and TS-SUF-4 are important in practice. The TS-SUF-3 security protects against premature leakage of a pre-processing token, and the TS-SUF-4 security prevents an honest signer from being wrongfully framed for a forgery. In this work, we investigate whether FROST2, which improves efficiency over FROST and reduces communication rounds compared to several later three-round variants [15,3,25], can achieve the same TS-SUF-3 and TS-SUF-4 security levels as FROST in the ROM.

1.1 Our Contributions

We proposed an improved variant of FROST2 that achieves TS-SUF-4 security, the highest tier in Bellare et al.'s hierarchical framework, while maintaining the protocol's efficiency and its two-round structure.

The proposed FROST2+ addresses the TS-UF-3 vulnerability of FROST2 by introducing a lightweight token authentication mechanism for pre-processing tokens. In contrast to Bellare et al.'s ATS[TS, DS] transformation [8,6], which requires each signer to generate and manage an additional digital signature keypair, our approach authenticates the tokens using only the signing key share

and the secret exponents already presented in the pre-processing tokens (R, S). We establish the TS-SUF-4 security of FROST2+ under the one-more discrete logarithm (OMDL) assumption within the ROM, in the static corruption and centralized key generation (CKG) settings.

Subsequently, we show that our proposed schemes maintain their security levels when transitioning from the CKG setting to a distributed key generation (DKG) setting, specifically using PedPoP DKG as an instantiation. We also explain how to speed up the partial signing algorithm by pre-verifying the pre-processing tokens, which is not bind to the signing message. Moreover, similar to FROST's offline pre-processing, our token verification mechanism facilitates per-signer batching across multiple signing sessions. This design choice preserves the efficiency of FROST-style pre-processing, making our schemes particularly advantageous for high-throughput, leaderless environments where offline token preparation is critical.

Finally, we show that the proposed FROST2+, while offering stronger security, achieves better efficiency than FROST and matches the performance of FROST2, as detailed in our performance evaluation (Section 5).

Organization. This paper is structured as follows. In Section 2, we present the necessary mathematical tools. We then outline the proposed FROST2+ scheme in Section 3, followed by the security analysis in Section 4. Next, we present the performance evaluation in Section 5 and conclude the paper in Section 6.

2 Preliminaries

We briefly recall the required mathematical tools in this section.

2.1 One-More Discrete Logarithm (OMDL) Assumption

Definition 1. *Let $\mathbb{G}$ be a cyclic group of prime order p with generator g, and let* Chall *and* Dlog *be oracles such that:*

- Chall() *returns a random challenge $A = g^a$ for $a \xleftarrow{\$} \mathbb{Z}_p^*$*
- Dlog(B) *returns $b \in \mathbb{Z}_p^*$ such that $B = g^b$ for any $B \in \mathbb{G}$*

The OMDL *assumption states that for any probabilistic polynomial-time adversary $\mathcal{C}$ making at most $q - 1$ queries to* Dlog*, the advantage*

$$\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{C}) = \Pr\left[\begin{array}{l}(A_1, \ldots, A_q) \leftarrow \textsc{Chall}(), \\ (a_1, \ldots, a_q) \leftarrow \mathcal{C}^{\textsc{Chall},\textsc{Dlog}}(g) \\ \forall i : g^{a_i} = A_i \wedge \mathcal{C} \text{ never queried } \textsc{Dlog}(A_i)\end{array}\right]$$

is negligible.

2.2 Lagrange Coefficient

Let $f(\cdot)$ be a random $(t-1)$-degree polynomial such that $f(x_i) = s_i$. The polynomial $f(\cdot)$ can be reconstructed by having the knowledge of t-pair values of (x_i, s_i):

$$f(\cdot) = \sum_{x_i \in \mathcal{S}} s_i \Delta_{x_i,\mathcal{S}}(\cdot)$$

where $\mathcal{S} = \{x_1, x_2, \ldots, x_t\}$.

Definition 2. *The Lagrange coefficient $\Delta_{x_i,\mathcal{S}}$ is defined as:*

$$\Delta_{x_i,\mathcal{S}}(Z) = \prod_{x_j \in S, j \neq i} \frac{Z - x_j}{x_i - x_j}$$

where x_i and x_j are elements in the set $\mathcal{S}$.

2.3 Threshold Signature Scheme

To synchronize with Bellare et al.'s TS-(S)UF-i security framework [8,6], we adopt their scheme definition while making some adjustments, which includes the addition of a Setup algorithm and modifications to certain inputs for each algorithm. We define a non-interactive threshold signature scheme as TS={Setup, Kg, SPP, LPP, LR, PS, Agg, Vf, SVf}, characterized as follows:

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{par}$: The setup algorithm takes in a security parameter 1^λ and produces public parameters par, which are implicitly provided to all other algorithms.

$\mathsf{Kg}(\mathsf{par}, t, \mathsf{ns}) \rightarrow \{\mathsf{st}_i = (\tilde{X}, \mathsf{aux}, \mathsf{sk})\}_{i=1}^{\mathsf{ns}}$: The key generation algorithm takes in threshold t and number of signers ns such that $t \leq \mathsf{ns}$, initializes the state st_i for every signer $\mathsf{id}_i \in \{\mathsf{id}_1, \ldots, \mathsf{id}_{\mathsf{ns}}\}$ with joint public key $\tilde{X}$, public auxiliary information aux, and a signing key share sk.

$\mathsf{SPP}(\mathsf{st}_i) \rightarrow (pp, \mathsf{st}_i)$: The signing pre-processing algorithm takes as input the state st_i of signer id_i. It outputs a pre-processing token pp and updates the signer's state.

$\mathsf{LPP}(\mathsf{id}_i, pp, \mathsf{st}_0) \rightarrow \mathsf{st}_0$: The leader pre-processing algorithm takes the pre-processing token pp from signer id_i and the leader's state st_0, updating st_0 to incorporate the token.

$\mathsf{LR}(M, \mathcal{S}, \mathsf{st}_0) \rightarrow (lr, \mathsf{st}_0)$: This leader signing-request algorithm generates a leader request lr based on a message M, a signing set $\mathcal{S}$ with at least t signers, and the leader's state st_0.

$\mathsf{PS}(lr, \mathsf{st}_i) \to (psig_i, \mathsf{st}_i)$: The signer partial-signing algorithm takes the leader request lr, the state st_i of a signer id_i, generating a partial signature $psig_i$ and updating state st_i.

$\mathsf{Agg}(lr, \{psig_i\}_{\mathsf{id}_i \in lr.\mathcal{S}}, \mathsf{st}_0) \to (\sigma, \mathsf{st}_0)$: The leader partial-signature aggregation algorithm aggregates the partial signatures received from the signing set $\mathcal{S}$ into a final signature σ for the message M while updating the leader's state st_0.

$\mathsf{Vf}(\tilde{X}, M, \sigma) \to 0$ or 1: The verification algorithm checks the validity of the signature σ for the message M using the public key $\tilde{X}$, returning 1 if valid and 0 otherwise.

$\mathsf{SVf}(\tilde{X}, lr, \sigma) \to 0$ or 1: The strong verification algorithm checks the validity of the signature σ for the leader request lr using the public key $\tilde{X}$, returning 1 if valid and 0 otherwise.

2.4 Security Model

Table 1. Trivial forgery predicates for TS-(S)UF-i [8,6] with Cremers et al.'s recommendation [13].

Predicate	**Definition**
$\mathsf{tf}_0(M)$	$S_1(M) \neq \emptyset$
$\mathsf{tf}_1(M)$	$\lvert S_1(M)\rvert \geq t - \lvert CS\rvert$
$\mathsf{tf}_2(lr)$	$\lvert S_2(lr)\rvert \geq t - \lvert CS\rvert$
$\mathsf{tf}_3(lr)$	$\mathsf{tf}_2(lr) \wedge S_2(lr) = S_3(lr)$
$\mathsf{tf}_4(lr)$	$\mathsf{tf}_2(lr) \wedge S_2(lr) = S_3(lr) = S_4(lr)$
$\mathsf{tsf}_2(lr, \tilde{X}, \sigma)$	$\mathsf{tf}_2(lr) \wedge \mathsf{SVf}(\tilde{X}, lr, \sigma)$
$\mathsf{tsf}_3(lr, \tilde{X}, \sigma)$	$\mathsf{tf}_3(lr) \wedge \mathsf{SVf}(\tilde{X}, lr, \sigma)$
$\mathsf{tsf}_4(lr, \tilde{X}, \sigma)$	$\mathsf{tf}_4(lr) \wedge \mathsf{SVf}(\tilde{X}, lr, \sigma)$

To address a shortcoming in the original hierarchy of Bellare et al.'s hierarchy framework [6], where TS-(S)UF-4 does not imply TS-(S)UF-3, and vice versa, due to incomparable trivial forgery conditions, we adopt the strengthened trivial forgery condition in Table 1 for tf_4 as recommended by Cremers et al. [13]. Specifically, this strengthens tf_4 from requiring $S_2(lr) = S_4(lr)$ to $S_2(lr) = S_3(lr) = S_4(lr)$, ensuring that all honest signers in $lr.\mathcal{S}$ responded with honestly generated pre-processing tokens. The modification is purely definitional such that for echo schemes satisfying the standard pre-processing token check, it does not change the set of trivial forgeries. In particular, this holds for the ATS[TS,DS] transformation [8,6] whose partial signing only accepts signed pre-processing tokens, so the original and the strengthened tf_4 coincide and the existing TS-SUF-4 security proof for ATS[TS,DS] remains valid.

Games $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ $(i = 0, 1, 2, 3, 4)$ and $\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}$ $(i = 2, 3, 4)$

$\textsc{Init}(CS)$:
1: Require: $CS \subseteq \{\mathsf{id}_1, \ldots, \mathsf{id}_{\mathsf{ns}}\}$ and $|CS| < t$ ▷ Set of corrupted signers
2: $\mathsf{h} \xleftarrow{\$} \mathsf{TS.HF}; (\tilde{X}, aux, \mathsf{sk}_1, \ldots, \mathsf{sk}_{\mathsf{ns}}) \xleftarrow{\$} \mathsf{Kg}$ ▷ Set of hash functions TS.HF
3: $HS \leftarrow \{\mathsf{id}_1, \ldots, \mathsf{id}_{\mathsf{ns}}\} \setminus CS$ ▷ Set of honest signers
4: **for** $\mathsf{id}_i \in HS$ **do**
5: $\quad \mathsf{st}_i.\mathsf{sk} \leftarrow x_i;\ \mathsf{st}_i.\tilde{X} \leftarrow \tilde{X};\ \mathsf{st}_i.\mathsf{aux} \leftarrow aux$
6: **end for**
7: **return** $\tilde{X}, aux, \{x_i\}_{\mathsf{id}_i \in CS}$

$\text{PPO}(\mathsf{id}_i)$:
8: Require: $\mathsf{id}_i \in HS$
9: $(pp, \mathsf{st}_i) \xleftarrow{\$} \mathsf{SPP}(\mathsf{st}_i);\ PPL \leftarrow PPL \cup \{(\mathsf{id}_i, pp)\}$
10: **return** pp

$\textsc{PSign}\text{O}(\mathsf{id}_i, lr)$:
11: Require: $lr.\mathcal{S} \subseteq \{\mathsf{id}_1, \ldots, \mathsf{id}_{\mathsf{ns}}\}$, $M = lr.\mathsf{msg} \in \{0,1\}^*$ and $\mathsf{id}_i \in HS$
12: $(psig, \mathsf{st}_i) \xleftarrow{\$} \mathsf{PS}(lr, \mathsf{st}_i); L \leftarrow L \cup \{lr\}$
13: **if** $psig \neq \perp$ **then**
14: $\quad S_1(M) \leftarrow S_1(M) \cup \{\mathsf{id}_i\}; S_2(lr) \leftarrow S_2(lr) \cup \{\mathsf{id}_i\}$
15: **end if**
16: **return** $psig$

$\text{RO}[\mathsf{h}](x)$:
17: **return** $\mathsf{h}(x)$

$\textsc{Fin}(M, \sigma)$:
18: **for** all $lr \in L$ **do**
19: $\quad S_3(lr) \leftarrow \{\mathsf{id}_i \in HS \cap lr.\mathcal{S} : lr.\mathsf{PP}(\mathsf{id}_i) \in PPL\}; S_4(lr) \leftarrow HS \cap lr.\mathcal{S}$
20: **end for**
21: **if** $\neg\mathsf{Vf}(\tilde{X}, M, \sigma)$ **then**
22: $\quad$ **return** false;
23: **end if**
24: **if** Game $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ for $i = 0, 1$ **then**
25: $\quad$ **return** $\neg\mathsf{tf}_i(M)$
26: **end if**
27: **if** Game $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ for $i = 2, 3, 4$ **then**
28: $\quad$ **return** $\neg\exists lr(lr.\mathsf{msg} = M \wedge \mathsf{tf}_i(lr))$
29: **end if**
30: **if** Game $\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}$ **then**
31: $\quad$ **return** $\neg\exists lr(lr.\mathsf{msg} = M \wedge \mathsf{tsf}_i(lr, \tilde{X}, \sigma))$
32: **end if**

Fig. 1. Security games defining TS-(S)UF-i [6,8,13].

This strengthening approach, corresponds to Cremers et al.'s SiGu-4:aLRhPP level, restores linear downward implications across the hierarchy (e.g., TS-(S)UF-

4 implies TS-(S)UF-3) without weakening the lower levels as done in the revised version [8]. We prefer this strengthened TS-(S)UF-i model as shown in Figure 1 over Cremers et al.'s full framework [13] because it provides a standard comparison to FROST and its related variants.

Definition 3. *Let* $\mathsf{X} \in \{\mathsf{suf}, \mathsf{uf}\}$ *and* $i \in \{0, 1, 2, 3, 4\}$. *A* TS *scheme is secure if there is no polynomial time algorithm* $\mathcal{F}$ *that can win the game* $\mathbf{G}_{\mathsf{TS}}^{\mathsf{ts\text{-}X\text{-}}i}$ *in time* $T(\mathcal{F})$ *with non-negligible advantage:*

$$\mathsf{Adv}_{\mathsf{TS}}^{\mathsf{ts\text{-}X\text{-}i}}(\mathcal{F}) = \Pr[\mathbf{G}_{\mathsf{TS}}^{\mathsf{ts\text{-}X\text{-}}i}(\mathcal{F}) = \mathsf{true}].$$

2.5 Generalized Forking Lemma Extension

This lemma is proposed by Bellare et al. [8,6] in proving the TS-SUF-2 security of FROST2. It extends the generalized Forking Lemma [7] where if $\mathcal{A}$ can only output index I within a given set $SS \subseteq [1..q]$, then the final bound on $\mathsf{acc}(\mathsf{Fork}^{\mathcal{A}})$ depends only on $|S|$ instead of q. This reduces the security loss from $1/q$ to $1/|SS|$.

Lemma 1. *Let* $q \geq 1$ *be an integer,* $SS \subseteq [1..q]$ *be a set, and* $\mathbb{H}$ *be a set. Let* $\mathcal{A}$ *be a randomized algorithm that on input* $x, h_1, \ldots, h_q$ *outputs a pair* (i_{sig}, Out), *where* $i_{sig} \in \{\perp\} \cup SS$ *and Out is a side output. Let GrGen be a randomized algorithm that generates* x. *The accepting probability of* $\mathcal{A}$ *is defined as* $\mathsf{acc}(\mathcal{A}) = \Pr_{x \leftarrow \mathbf{GrGen},\, h_1,\ldots,h_q \leftarrow \mathbb{H}}\big[(i_{sig}, \mathsf{Out}) \leftarrow \mathcal{A}(x, h_1, \ldots, h_q) : i_{sig} \neq \perp\big]$.

The accepting probability of $\mathsf{Fork}^{\mathcal{A}}$ *is defined as*

$$\mathsf{acc}(\mathsf{Fork}^{\mathcal{A}}) = \Pr_{x \leftarrow \mathbf{GrGen}}\big[\alpha \leftarrow \mathsf{Fork}^{\mathcal{A}}(x) : \alpha \neq \perp\big].$$

Then $\mathsf{acc}(\mathsf{Fork}^{\mathcal{A}}) \geq \mathsf{acc}(\mathcal{A})^2/|SS|$.

2.6 Generalized Multi-Instance Multi-Forking Lemma

We extend the generalized multi-instance forking lemma of Bagherzandi et al. [5] to allow multiple forks per instance. The statement is average-case over parameters generated by GrGen, in the same style as the original lemma. In essence, we modify the success count of each local instance from 1 to n, analogous to Gennaro et al.'s multi-reset lemma [19] built upon Bellare and Neven's generalized reset lemma [7].

Let $\mathbb{H}$ be a finite set and let q, N, n be positive integers. Consider an algorithm $\mathcal{A}$ that, on input par, internal coins ρ, and a random-oracle tape $h = (h_1, \ldots, h_q) \in \mathbb{H}^q$, outputs an index set I and side outputs $(\{\mathsf{out}_i\}_{i \in I}, \theta)$. If $\mathcal{A}$ fails, it outputs $(I, \{\mathsf{out}_i\}_{i \in I}, \theta)$ with $I = \emptyset$. Otherwise we assume $|I| = N$, treating I as a multiset and allowing repetitions, and we assume that if $I = \{i_1, \ldots, i_N\}$ then $i_1 \leq \cdots \leq i_N$. Note that we adopt the simplified output notations [11] that has an extra θ, which can be viewed as a suffix in every out_i.

Let $\Omega = \{(\rho, h_1, \ldots, h_q) \mid \rho \in \{0,1\}^*,\ h_j \in \mathbb{H}\}$ be the set of all possible random tapes. For a fixed par, let $S(\mathsf{par}) = \{f \in \Omega \mid \mathcal{A}(\mathsf{par}, f) \text{ succeeds}\}$. Define $\varepsilon(\mathsf{par}) = \Pr_{f \leftarrow \Omega}[\mathcal{A}(\mathsf{par}, f) \text{ succeeds}]$ and let $\hat{\varepsilon} = \Pr_{\mathsf{par} \leftarrow \mathsf{GrGen}(1^\lambda),\, f \leftarrow \Omega}[\mathcal{A}(\mathsf{par}, f) \text{ succeeds}]$.

Let $\mathcal{O} = \{(h_{i_m}, \mathsf{out}_{i_m})\}_{m=1}^{N}$ be the side outputs from the initial execution, and let $\mathcal{O}' = \emptyset$ initially. The forking algorithm $\mathsf{MIMFork}^{\mathcal{A}}_{N,n}(\mathsf{par})$ performs N local rewinding processes, one for each occurrence i_m in the ordered multiset I. For each target occurrence i_m, it resamples the random-oracle suffix starting at i_m until it collects n additional side outputs whose random-oracle responses at i_m are pairwise distinct and also distinct from the original response h_{i_m}.

Algorithm 1 Generalized Multi-Instance Multi-Forking Algorithm $\mathsf{MIMFork}^{\mathcal{A}}_{N,n}(\mathsf{par})$

Pick internal coins ρ for $\mathcal{A}$ at random
$(h_1, \ldots, h_q) \xleftarrow{\$} \mathbb{H}^q$
$(I, \{\mathsf{out}_i\}_{i \in I}, \theta) \leftarrow \mathcal{A}(\mathsf{par}, (h_1, \ldots, h_q); \rho)$
if $I = \emptyset$ **then**
    return $\perp$
end if
Parse I as a multiset $\{i_1, i_2, \ldots, i_N\}$ such that $i_1 \le i_2 \le \cdots \le i_N$
$\mathcal{O} \leftarrow \{(h_{i_m}, \mathsf{out}_{i_m})\}_{m=1}^{N}$; $\mathcal{O}' \leftarrow \emptyset$; $L \leftarrow \left\lceil \frac{8Nq}{\hat{\varepsilon}} \ln\left(\frac{8Nn}{\hat{\varepsilon}}\right) \right\rceil$; $T_{\max} \leftarrow nL$
for $m = 1$ **to** N **do**
    $\mathcal{O}'_m \leftarrow \emptyset$; $c_m \leftarrow 0$
    while $|\mathcal{O}'_m| < n$ **and** $c_m < T_{\max}$ **do**
        $c_m \leftarrow c_m + 1$; $h_{i_m}^{(m,c_m)}, \ldots, h_q^{(m,c_m)} \xleftarrow{\$} \mathbb{H}^{q - i_m + 1}$
        $Q^{(m,c_m)} \leftarrow \left(h_1, \ldots, h_{i_m - 1}, h_{i_m}^{(m,c_m)}, \ldots, h_q^{(m,c_m)}\right)$
        $(I', \{\mathsf{out}'_i\}_{i \in I'}, \theta') \leftarrow \mathcal{A}(\mathsf{par}, Q^{(m,c_m)}; \rho)$
        if $i_m \in I'$ **then**
            if $h_{i_m}^{(m,c_m)} \notin \{h_{i_m}\} \cup \{h' \mid (h', \cdot) \in \mathcal{O}'_m\}$ **then**
                $\mathcal{O}'_m \leftarrow \mathcal{O}'_m \cup \{(h_{i_m}^{(m,c_m)}, \mathsf{out}'_{i_m})\}$
            end if
        end if
    end while
    if $|\mathcal{O}'_m| < n$ **then**
        return $\perp$
    end if
    $\mathcal{O}' \leftarrow \mathcal{O}' \cup \mathcal{O}'_m$
end for
return $(\mathcal{O}, \mathcal{O}')$

Lemma 2 (Generalized Multi-Instance Multi-Forking Lemma). *Let $\mathbb{H}$ be a finite set and let $q, N, n \in \mathbb{N}$ with $N, n \ge 1$. Let $\mathcal{A}$ be a randomized algorithm that, on input par and a vector $h \in \mathbb{H}^q$, returns an index set I and side outputs $(\{\mathsf{out}_i\}_{i \in I}, \theta)$. Define $\hat{\varepsilon}$ as above. Assume $|\mathbb{H}| > \frac{16Nnq}{\hat{\varepsilon}}$. Then the*

accepting probability of $\mathsf{MIMFork}^{\mathcal{A}}_{N,n}$ *is*

$$\mathsf{acc}\left(\mathsf{MIMFork}^{\mathcal{A}}_{N,n}\right) = \Pr\left[\mathsf{MIMFork}^{\mathcal{A}}_{N,n}(\mathsf{par}) \neq \bot\right] \geq \frac{\hat{\varepsilon}}{8},$$

where the probability is over $\mathsf{par} \leftarrow \mathsf{GrGen}(1^\lambda)$ *and the coins of* $\mathsf{MIMFork}^{\mathcal{A}}_{N,n}$. *Moreover, whenever the algorithm returns* $(\mathcal{O}, \mathcal{O}')$, *for every occurrence* i_m *the collection* $\mathcal{O}'_m$ *contains* n *pairs whose first components are pairwise distinct and distinct from* h_{i_m}.

Proof. For a fixed par, let $\varepsilon = \varepsilon(\mathsf{par})$ and $S = S(\mathsf{par})$ where $\Pr[S] = \varepsilon$ by definition. For every $f \in S$, let $I(f) = \{i_1(f), \ldots, i_N(f)\}$, where $i_1(f) \leq \cdots \leq i_N(f)$. For $j \in \{1, \ldots, q\}$, let $f_j = (\rho, h_1, \ldots, h_{j-1})$ denote the prefix before the j-th random-oracle response, and let $s_j = (h_j, \ldots, h_q)$ denote the corresponding suffix. For every $m \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, q\}$, define $A^j_m = \{f \in S \mid i_m(f) = j\}$ and $B^j_m = \left\{f \in \Omega \;\middle|\; \Pr_{s_j}[(f_j \| s_j) \in A^j_m] \geq \frac{\varepsilon}{2Nq}\right\}$. Let $E_m = \bigcup_{j=1}^{q}(A^j_m \cap B^j_m)$. By part (3) of the Splitting Lemma [5, Lemma 2], for every m,

$$\Pr[E_m \mid S] = \sum_{j=1}^{q} \Pr[A^j_m \cap B^j_m \mid S] \geq \sum_{j=1}^{q}\left(\Pr[A^j_m \mid S] - \frac{\varepsilon/(2Nq)}{\Pr[S]}\right) = 1 - \frac{1}{2N}. \tag{1}$$

Hence, by the union bound, it gives

$$\Pr\left[S \cap \bigcap_{m=1}^{N} E_m\right] \geq \varepsilon\left(1 - \sum_{m=1}^{N} \Pr[\overline{E_m} \mid S]\right) \geq \frac{\varepsilon}{2}. \tag{2}$$

Condition on $S \cap \bigcap_{m=1}^{N} E_m$. Fix an occurrence m and suppose that fewer than n distinct forks have been collected. Since the original tape belongs to E_m, we have $f \in B^{i_m}_m$, and hence a fresh suffix yields an execution in $A^{i_m}_m$ with probability at least $\varepsilon/(2Nq)$. Let $H_{\mathsf{forbid}} = \{h_{i_m}\} \cup \{h' \mid (h', \cdot) \in \mathcal{O}'_m\}$. Since $|\mathcal{O}'_m| < n$, we have $|H_{\mathsf{forbid}}| \leq n$, and conditioned on the current state of the local rewinding process, the probability that a fresh trial produces an execution in $A^{i_m}_m$ and yields a new response is at least $p \geq \frac{\varepsilon}{2Nq} - \frac{n}{|\mathbb{H}|} > \frac{\varepsilon}{2Nq} - \frac{\hat{\varepsilon}}{16Nq} > \frac{\varepsilon}{4Nq}$, where the second inequality uses $|\mathbb{H}| > 16Nnq/\hat{\varepsilon}$, and the last uses $\varepsilon > \hat{\varepsilon}/2$ for the good parameters.

Thus, as long as fewer than n forks have been collected, every rewinding trial has conditional probability at least $\varepsilon/(4Nq)$ of producing a new valid fork. Recall that $T_{\max} = nL$, where now $L = \left\lceil \frac{8Nq}{\hat{\varepsilon}} \ln\left(\frac{8Nn}{\hat{\varepsilon}}\right)\right\rceil$. Partition the nL trials into n consecutive blocks of L trials. Since each trial has conditional success probability at least $p > \varepsilon/(4Nq) > \hat{\varepsilon}/(8Nq)$, the probability that a fixed block produces no new fork is at most $(1-p)^L \leq \exp(-pL) \leq \frac{\hat{\varepsilon}}{8Nn}$.

Therefore, let Fail_m be the event that the m-th occurrence fails to collect n distinct forks. By the union bound over the n blocks and then over the N

occurrences,

$$\Pr\left[\mathsf{Fail}_{\mathsf{any}} \;\middle|\; S \cap \bigcap_{m=1}^{N} E_m\right] \leq N \cdot n \cdot \frac{\hat{\varepsilon}}{8Nn} = \frac{\hat{\varepsilon}}{8}, \tag{3}$$

where $\mathsf{Fail}_{\mathsf{any}} = \bigcup_{m=1}^{N} \mathsf{Fail}_m$. Since we are conditioning on a good parameter with $\varepsilon > \hat{\varepsilon}/2$, we have $\hat{\varepsilon}/8 < \varepsilon/4$. Combining (2) and (3), for every par with $\varepsilon(\mathsf{par}) > \hat{\varepsilon}/2$, we have $\mathsf{frk}(\mathsf{par}) \geq \frac{\varepsilon}{2} - \frac{\varepsilon}{4} = \frac{\varepsilon}{4}$.

Finally, average over $\mathsf{par} \leftarrow \mathsf{GrGen}(1^\lambda)$. Let $P' = \{\mathsf{par} \mid \varepsilon(\mathsf{par}) > \hat{\varepsilon}/2\}$. For $\mathsf{par} \in P'$, we have $\mathsf{frk}(\mathsf{par}) \geq \frac{\varepsilon(\mathsf{par})}{4}$ while for $\mathsf{par} \notin P'$, we use the trivial bound $\mathsf{frk}(\mathsf{par}) \geq 0$. Therefore,

$$\begin{aligned}\mathsf{acc}\left(\mathsf{MIMFork}^{\mathcal{A}}_{N,n}\right) &\geq \frac{1}{|P|}\sum_{\mathsf{par}\in P'}\frac{\varepsilon(\mathsf{par})}{4} \\ &= \frac{1}{4}\left(\hat{\varepsilon} - \frac{1}{|P|}\sum_{\mathsf{par}\notin P'}\varepsilon(\mathsf{par})\right) \geq \frac{1}{4}\left(\hat{\varepsilon} - \frac{\hat{\varepsilon}}{2}\right) = \frac{\hat{\varepsilon}}{8}.\end{aligned}$$

□

3 Our Constructions

In this section we present FROST2+ (shown in Figure 2) as a TS-SUF-4 variant of FROST2 from Bellare et al. [6]. Different from Bellare et al.'s ATS[TS, DS] transformation [6], which requires each signer to manage an extra digital signature key-pair to authenticate the pre-processing token ($R = g^r, S = g^s$), ours reuses only the existing signing key share and the secret exponents already presented in the token. That said, FROST2+ has the same construction as FROST2 except in the SPP and PS algorithms. The SPP now broadcasts not only the pre-processing token (R, S) but also a short authentication transcript $(u, V = g^v)$ with:

$$u = v + \mathsf{st}_i.\mathsf{sk} \cdot e + re^2 + se^3$$

which is to be verified during PS. A signer aborts PS whenever a peer's (u, V) verification fails. The correctness follows directly from the equations such that if all parties are honest, then for each authentication transcript (u, V), we have $g^u = VX_i^e R^{e^2} S^{e^3}$. In the signing phase, the partial signatures satisfy $z_i = r_i + ds_i + c\Delta_{\mathsf{id}_i,\mathcal{S}}(0)\mathsf{st}_i.\mathsf{sk}$. Aggregating $z = \sum_{\mathsf{id}_i \in \mathcal{S}} z_i$ gives $g^z = \tilde{R}\tilde{X}^c$ and Vf returns 1. Moreover, as discussed in the subsequent sections, when considering a batch verification of k tokens $(R_1, S_1, \ldots, R_k, S_k)$, the transcript size remains constant.

3.1 Pre-Computations

Although the TS-SUF-i notions of Bellare et al. [6] assume a designated leader for collecting and distributing pre-processing tokens and signing requests to simplify security analysis, they explicitly cover scenarios in which corrupted signers take turns acting as the leader. Consequently, a leaderless design introduces no

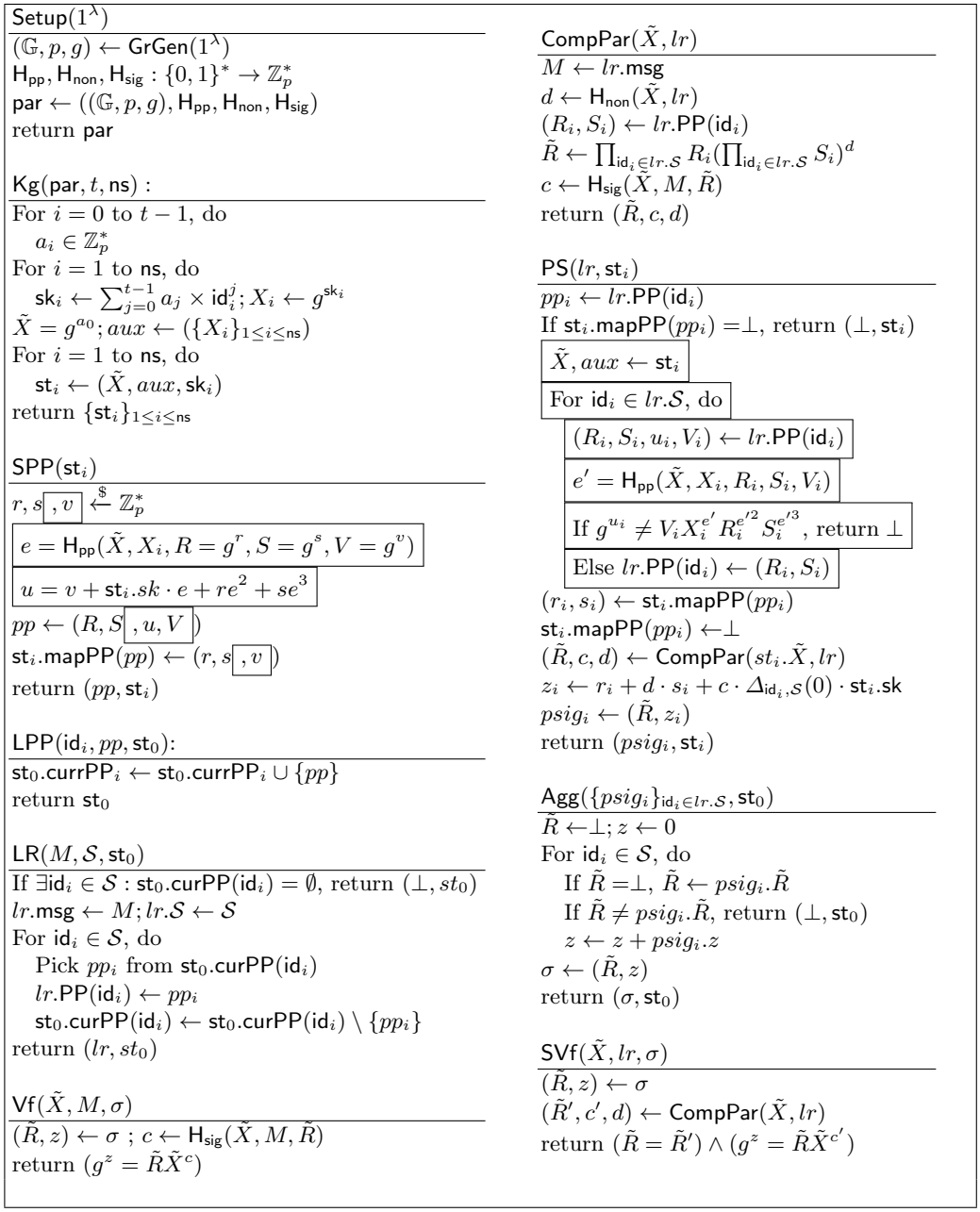


Fig. 2. The FROST2 and FROST2+ signature schemes where FROST2 contains all but the solid box. GrGen generates cyclic group parameter in which the discrete logarithm assumption holds, st is state, $\mathsf{ns} < p - 1$ is the number of signers, and t is the signing threshold. The function mapPP records dlogs of tokens pp, and currPP is the current pp collection.

unmodeled attack vectors. In particular, attacks captured by TS-UF-3 and TS-UF-4, where a corrupted leader forging tokens or requests, are already included in the TS-SUF-i adversary model. When an honest signer initiates the protocol, the same corruption bounds apply, preserving the original security guarantees.

If our proposed FROST2+ adopts this leaderless approach, signers can collect and verify pre-processing tokens as in the original FROST [22] prior to the partial signing (PS) phase. This pre-verification step improves runtime performance, as shown in Table 2, without compromising security.

3.2 Batch Verification

Similar to ATS[FROST,DS] [8,6] which can generate an offline signature on multiple tokens for batch verification, our FROST2+ generates an offline authentication transcript on multiple tokens for the same purpose. Specifically, when a signer id_i has $n(\geq 2)$ tokens to be verified, during SPP, the signer computes:

$$\{r_{i,j}, s_{i,j}\}_{j=1}^{n}, v \xleftarrow{\$} \mathbb{Z}_p^*,\ e = \mathsf{H}_{\mathsf{pp}}(\tilde{X}, X_i, \{R_{i,j}, S_{i,j}\}_{j=1}^{n}, V = g^v)$$
$$u = v + \mathsf{st}_i.\mathsf{sk} \cdot e + \sum_{j=1}^{n} r_{i,j} e^{j+1} + \sum_{j=1}^{n} s_{i,j} e^{j+1+n},\ pp \leftarrow (\{R_{i,j}, S_{i,j}\}_{j=1}^{n}, u, V)$$

where the transcript size (u, V) remains constant. Subsequently, the verification is done by checking whether:

$$g^u = V X_i^{e'} \prod_{j=1}^{n} R_{i,j}^{e'^{j+1}} \prod_{j=1}^{n} S_{i,j}^{e'^{j+1+n}}$$

holds where $e' = \mathsf{H}_{\mathsf{pp}}(\tilde{X}, X_i, \{R_{i,j}, S_{i,j}\}_{j=1}^{n}, V)$.

Enabling this batch verification feature does not affect the TS-SUF-4 security level, but the reduction loss because the run time of reduction is linear to n as shown in Section 4. If a tighter reduction is desirable, then one may need to explore the Multi-Instance Reset Lemma [21].

4 Security Analysis

We prove the TS-SUF-4 security of FROST2+ under the OMDL assumption in the random oracle model. Our proving strategy is to separate the security analysis into three parts which handle the events

1. $E_1 : \mathsf{Vf}(\tilde{X}, M^*, \sigma^*) \ \wedge \ (lr^* = \perp \ \vee \ |S_2(lr^*)| < t - |CS|)$,
2. $E_2 : \mathsf{Vf}(\tilde{X}, M^*, \sigma^*) \ \wedge \ lr^* \neq \perp \ \wedge S_2(lr^*) \neq S_3(lr^*)$,
3. $E_3 : \mathsf{Vf}(\tilde{X}, M^*, \sigma^*) \ \wedge \ lr^* \neq \perp \ \wedge \neg(S_2(lr^*) = S_3(lr^*) = S_4(lr^*))$,

respectively. We first present the security proofs in the centralized key generation setting to ease exposition, and show how to extend them to the decentralized key generation setting in Section 4.1.

Theorem 1. *Let $\mathcal{F}$ be a TS-SUF-4 forger against* FROST2+ *making at most q queries to the random oracle. Suppose the success probability of $\mathcal{F}$ is bounded by $\mathsf{Adv}^{\text{ts-suf-4}}_{\text{FROST2+}}(\mathcal{F}) \leq \Pr[E_1] + \Pr[E_2] + \Pr[E_3]$. Then, there exist three OMDL adversaries $\mathcal{B}_1, \mathcal{B}_2$ and $\mathcal{B}_3$ such that:*

$$\mathsf{Adv}^{\text{ts-suf-4}}_{\text{FROST2+}}(\mathcal{F}) \leq \sqrt{q \cdot \mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}_1)} + 8\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}_2) + 8\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}_3) + \mathsf{negl}$$

where $\mathcal{B}_1$ and $\mathcal{B}_3$ issue at most $2q_s + t$ queries to the Chall *oracle, while $\mathcal{B}_2$ issues at most $2q_s$ queries. Moreover, their running times are bounded above by*

$$\begin{aligned} T(\mathcal{B}_1) &\leq 2T(\mathcal{F}) + T(q), \\ T(\mathcal{B}_2) &\leq \frac{16\mathsf{ns}^2 q \ln(16\mathsf{ns}/\Pr[E_2])}{\Pr[E_2]} T(\mathcal{F}) + T(q), \text{ and} \\ T(\mathcal{B}_3) &\leq \frac{24q \ln(24/\Pr[E_3])}{\Pr[E_3]} \cdot T(\mathcal{F}) + T(q), \end{aligned}$$

respectively, where $t \leq \mathsf{ns}$, and $T(q)$ denotes the time required for the relevant group operations and exponentiations.

Proof. The proof follows from Lemmas 3, 4 and 7. □

Lemma 3 is intended for the event E_1, where the forgery must contain at least one forged $psig_i$. The proof is similar to the TS-SUF-2 proof of FROST2 [8, Lemma 5.2], except that we add an additional RO[H_{pp}] to handle the authentication transcript (u, V) and adjust the sequence of hash queries accordingly.

Lemma 3. *There exists an OMDL adversary $\mathcal{B}_1$ making at most $2q_s + t$ queries to the* Chall *oracle and q queries to the random oracle such that*

$$\Pr[E_1] \leq \sqrt{q \cdot \mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}_1) + \mathsf{negl}}.$$

Moreover, $\mathcal{B}_1$ runs in time roughly twice that of $\mathcal{F}$, plus the time $T(q)$ required to perform exponentiations and group operations.

Proof. Let $\mathcal{F}$ be an adversary as described in the theorem. Denote the output message-signature pair of $\mathcal{F}$ as $(M^*, sig^* = (\tilde{R}^*, z^*))$. Without loss of generality, we assume $\mathcal{F}$ always queries RO on $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, M^*, \tilde{R}^*)$ before $\mathcal{F}$ returns and always queries RO on $\mathsf{H}_{\mathsf{pp}}(\tilde{X}, X_i, R, S, V)$ and $\mathsf{H}_{\mathsf{non}}(\tilde{X}, lr)$ prior to the query PSign(id_i, lr) for some id_i and lr. This adds up to $2q_s$ additional RO queries, and we let $q = q_h + 2q_s + 1$ where $q_h = q_{pp} + q_{non} + q_{sig}$.

Let lr^* be the leader query whose nonce query $\mathsf{H}_{\mathsf{non}}(\tilde{X}, lr^*)$ is the last such query before the signature hash query $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, M^*, \tilde{R}^*)$ and which satisfies $\mathsf{SVf}(\tilde{X}, lr^*, \sigma^*) = \mathsf{true}$ (if none exist, $lr^* = \perp$). Denote the event E_1 as

$$\mathsf{Vf}(\tilde{X}, M^*, \sigma^*) \wedge (lr^* = \perp \vee |S_2(lr^*)| < t - |CS|).$$

We first construct an algorithm $\mathcal{C}$ compatible with the syntax in Lemma 1. The input of $\mathcal{C}$ consists of $2q_s + t$ uniformly random group elements $\mathbf{A} =$

$\{A_0,\ldots,A_{t-1}\},\mathbf{R}=\{R_1,\ldots,R_{q_s}\},\mathbf{S}=\{S_1,\ldots,S_{q_s}\}\in\mathbb{G}$ and uniformly random integers $h_1,\ldots,h_{2(q-q_{pp})},h_{2(q-q_{pp})+1},\ldots,h_{2q}\in\mathbb{Z}_p$. Also, $\mathcal{C}$ can access an oracle DLOG, which on input $X\in\mathbb{G}$ outputs $\mathsf{DL}_{\mathbb{G},g}(X)$. To start with, $\mathcal{C}$ initializes all the states $\mathsf{st}_0,\ldots,\mathsf{st}_{\mathsf{ns}}$. In addition, it initializes counters $\mathsf{ctr}_s,\mathsf{ctr}_h$ to 0, ctr_{pp} to $2(q-q_{pp})$ and a function dt to an empty table, which are used to record the DLOG query related to each (R_j,S_j). $\mathcal{C}$ also initializes $\mathsf{curLR}\leftarrow\emptyset$ to record all leader requests that appear during the game and initializes ctrPP to an empty table, which are used to record the counter corresponding to each token generated by honest signers. We also use a flag BadPPO to denote whether a bad event occurs, which is initially set to false. Then, $\mathcal{C}$ runs $\mathcal{F}$ with access to the oracles $\widetilde{\text{INIT}},\widetilde{\text{PPO}},\widetilde{\text{PSIGNO}},\widetilde{\text{RO}}$, which are simulated as follows.

$\widetilde{\text{INIT}}(CS)$: $\mathcal{C}$ initializes H to an empty table and sets $\tilde{X}\leftarrow A_0, X_i=\prod_{j=0}^{t-1}A_j^{\mathsf{id}_i^j}$ for $i\in[1..\mathsf{ns}]$, and $\mathsf{st}_i.\mathsf{sk}=\text{DLOG}(X_i)$ for $\mathsf{id}_i\in CS$. Finally, $\mathcal{C}$ returns $\tilde{X}$, $\mathsf{aux}=(X_1,\ldots,X_{\mathsf{ns}})$, $\{\mathsf{st}_i.\mathsf{sk}\}_{\mathsf{id}_i\in CS}$.

$\widetilde{\text{RO}}[\mathsf{H}_{\mathsf{pp}}](x)$: If $\mathsf{H}_{\mathsf{pp}}(x)\neq\perp$, $\mathcal{C}$ returns $\mathsf{H}_{\mathsf{pp}}(x)$. Otherwise, $\mathcal{C}$ parses x as $(\tilde{X},X_i,R,S,V)$. If the parsing fails or $\tilde{X}\neq A_0$, $\mathcal{C}$ sets $\mathsf{H}_{\mathsf{pp}}(x)\xleftarrow{\$}\mathbb{Z}_p^*$ and returns $\mathsf{H}_{\mathsf{pp}}(x)$. Otherwise, $\mathcal{C}$ sets $\mathsf{ctr}_{pp}\leftarrow\mathsf{ctr}_{pp}+1$ and $\mathsf{H}_{\mathsf{pp}}(x)\leftarrow h_{\mathsf{ctr}_{pp}}$ then returns $\mathsf{H}_{\mathsf{pp}}(x)$.

$\widetilde{\text{RO}}[\mathsf{H}_{\mathsf{non}}](x)$: If $\mathsf{H}_{\mathsf{non}}(x)\neq\perp$, $\mathcal{C}$ returns $\mathsf{H}_{\mathsf{non}}(x)$. Otherwise, $\mathcal{C}$ parses x as $(\tilde{X},lr)$. If the parsing fails or $\tilde{X}\neq A_0$, $\mathcal{C}$ sets $\mathsf{H}_{\mathsf{non}}(x)\leftarrow\mathbb{Z}_p^*$ and returns $\mathsf{H}_{\mathsf{non}}(x)$. Else, $\mathcal{C}$ sets $\mathsf{ctr}_h\leftarrow\mathsf{ctr}_h+1$ and $\mathsf{H}_{\mathsf{non}}(x)\leftarrow h_{2\mathsf{ctr}_h-1}$. It then computes $\tilde{R}\leftarrow\prod_{\mathsf{id}_i\in lr.\mathcal{S}}R_iS_i^{h_{2\mathsf{ctr}_h-1}}$, where $(R_i,S_i,u_i,V_i)\leftarrow lr.\mathsf{PP}(\mathsf{id}_i)$. If $\mathsf{H}_{\mathsf{sig}}(\tilde{X},lr.\mathsf{msg},\tilde{R})=\perp$, $\mathcal{C}$ sets $\mathsf{H}_{\mathsf{sig}}(\tilde{X},lr.\mathsf{msg},\tilde{R})=h_{2\mathsf{ctr}_h}$. In addition, $\mathcal{C}$ defines $\mathsf{mapLR}(\mathsf{ctr}_h)=lr$ and sets $\mathsf{curLR}\leftarrow\mathsf{curLR}\cup\{lr\}$. Finally, $\mathcal{C}$ returns $\mathsf{H}_{\mathsf{non}}(x)$.

$\widetilde{\text{RO}}[\mathsf{H}_{\mathsf{sig}}](x)$: If $\mathsf{H}_{\mathsf{sig}}(x)\neq\perp$, $\mathcal{C}$ returns $\mathsf{H}_{\mathsf{sig}}(x)$. Otherwise, $\mathcal{C}$ parses x as $(\widetilde{X},M,\tilde{R})$. If the parsing fails or $\widetilde{X}\neq X$, $\mathcal{C}$ sets $\mathsf{H}_{\mathsf{sig}}(x)\xleftarrow{\$}\mathbb{Z}_p^*$ and returns $\mathsf{H}_{\mathsf{sig}}(x)$. Else, $\mathcal{C}$ sets $\mathsf{ctr}_h\leftarrow\mathsf{ctr}_h+1$ and $\mathsf{H}_{\mathsf{sig}}(x)\leftarrow h_{2\mathsf{ctr}_h}$. Finally, $\mathcal{C}$ returns $\mathsf{H}_{\mathsf{sig}}(x)$.

$\widetilde{\text{PPO}}(\mathsf{id}_i)$ query: Same as in the game $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$, except that, in the simulation of SPP, $\mathcal{C}$ increments ctr_s and ctr_{pp} by 1, chooses $u_{\mathsf{ctr}_{pp}}\xleftarrow{\$}\mathbb{Z}_p^*$ and computes $V_{\mathsf{ctr}_{pp}}=g^{u_{\mathsf{ctr}_{pp}}}/X_i^{h_{\mathsf{ctr}_{pp}}}R_{\mathsf{ctr}_s}^{h^2_{\mathsf{ctr}_{pp}}}S_{\mathsf{ctr}_s}^{h^3_{\mathsf{ctr}_{pp}}}$. If $\mathsf{H}_{\mathsf{pp}}(\tilde{X},X_i,R_{\mathsf{ctr}_s},S_{\mathsf{ctr}_s},V_{\mathsf{ctr}_h})=\perp$, $\mathcal{C}$ sets $\mathsf{H}_{\mathsf{pp}}(\tilde{X},X_i,R_{\mathsf{ctr}_s},S_{\mathsf{ctr}_s},V_{\mathsf{ctr}_h})\leftarrow h_{\mathsf{ctr}_{pp}}$, $pp\leftarrow(R_{\mathsf{ctr}_s},S_{\mathsf{ctr}_s},u_{\mathsf{ctr}_h},V_{\mathsf{ctr}_h})$, $\mathsf{st}_i.\mathsf{mapPP}(pp)\leftarrow(0,0,0,0)$, and $\mathsf{ctrPP}(\mathsf{id}_i,pp)\leftarrow\mathsf{ctr}_s$. Else if $\mathsf{H}_{\mathsf{pp}}(\tilde{X},X_i,R_{\mathsf{ctr}_s},S_{\mathsf{ctr}_s},V_{\mathsf{ctr}_{pp}})\neq\perp$, BadPPO is set to true. If there exists $lr\in\mathsf{curLR}$ such that $lr.\mathsf{PP}(\mathsf{id}_i)=(R_{\mathsf{ctr}_s},S_{\mathsf{ctr}_s},u'_{\mathsf{ctr}_{pp}},V'_{\mathsf{ctr}_{pp}})$ for some valid $(u'_{\mathsf{ctr}_{pp}}\neq u_{\mathsf{ctr}_{pp}},V'_{\mathsf{ctr}_{pp}}\neq V_{\mathsf{ctr}_{pp}})$, then BadPPO is also set to true.

$\widetilde{\text{PSIGNO}}(\mathsf{id}_i,lr)$ query: Same as in the game $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$, except that, in the simulation of PS, if $\mathsf{st}_i.\mathsf{mapPP}(pp)\neq\perp$, $\mathcal{C}$ sets $z_i\leftarrow\text{DLOG}\big(R_jS_j^{d_i}X_i^{c\Delta_{\mathsf{id}_i,lr.\mathcal{S}}(0)}\big)$, where $j\leftarrow\mathsf{ctrPP}(\mathsf{id}_i,lr.\mathsf{PP}(\mathsf{id}_i))$. In addition, $\mathcal{C}$ sets $\mathsf{dt}(j)\leftarrow(\mathsf{id}_i,k,d_i,c\Delta_{\mathsf{id}_i,lr.\mathcal{S}}(0),z_i)$, where k denotes the index such that $\mathsf{H}_{\mathsf{non}}(\tilde{X},lr)$ is set to h_{2k-1} during the simulation.

After receiving the output $(lr^* = \{M^*, \mathcal{S}, \mathsf{PP}\}, \sigma^* = (\tilde{R}^*, z^*))$ from $\mathcal{F}$, $\mathcal{C}$ returns $\perp$ if $\mathsf{BadPPO} = \mathsf{true}$, or E_1 does not occur. Else, $\mathcal{C}$ finds the index i_{sig} such that $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, M^*, \tilde{R}^*)$ is set to $h_{i_{sig}}$ during the simulation. By our assumption on $\mathcal{F}$, we know such i_{sig} must exist. Then, $\mathcal{C}$ returns (i_{sig}, Out), where Out consists of all variables received or generated by $\mathcal{C}$.

Analysis of $\mathcal{C}$. To apply Lemma 1, we define $SS := \{2k\}_{k\in[1..q-q_{pp}]}$ and let GrGen be the algorithm that samples $2q_s + t$ group elements uniformly from $\mathbb{G}$ and outputs them. From the simulation, the output index i_{sig} of $\mathcal{C}$ is always in SS. Moreover, it is clear that $\mathcal{C}$ simulates the game $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$ perfectly when all the inputs to $\mathcal{C}$ are uniformly sampled from their domain, which implies $\mathsf{acc}(\mathcal{C}) \geq \Pr[E_1] - \Pr[\mathsf{BadPPO}]$, where $\Pr[E_1]$ denotes the probability in the original $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$ game with $\mathcal{F}$ as in Lemma 1, and $\Pr[\mathsf{BadPPO}]$ denotes the probability that $\mathsf{BadPPO} = \mathsf{true}$ at the end of $\mathcal{C}$'s execution. Since each pair (R_j, S_j) is sampled uniformly from $\mathbb{G}$ and the corresponding authentication transcript (u, V) is generated by programming $\mathrm{RO}[\mathsf{H}_{\mathsf{pp}}]$, for each $\mathrm{PPO}(\mathsf{id}_i)$ query, the probability BadPPO is set to true is negligible and at most $q_h/p + |\mathsf{curLR}|/p = 2q_h/p$. Therefore, we have $\Pr[\mathsf{BadPPO}] \leq 2q_sq_h/p$. By Lemma 1,

$$\mathsf{acc}(\mathsf{Fork}^{\mathcal{C}}) \geq \frac{(\Pr[E_1] - 2q_sq_h/p)^2}{q - q_{pp}} \geq \frac{\Pr[E_1]^2}{q} - \frac{4\Pr[E_1]q_sq_h}{p \cdot q} \geq \frac{\Pr[E_1]^2}{q} - \frac{2q_h}{p}.$$

Construct $\mathcal{B}_1$ *from* $\mathsf{Fork}^{\mathcal{C}}$. We now construct the OMDL adversary $\mathcal{B}_1$ using $\mathsf{Fork}^{\mathcal{C}}$ and the available DLOG oracle. To begin, $\mathcal{B}_1$ queries INIT and then queries CHALL oracle $2q_s + t$ times to generate the three lists of group elements $(\mathbf{A}, \mathbf{R}, \mathbf{S})$ as the input to run $\mathsf{Fork}^{\mathcal{C}}$. Without loss of generality, we can assume all the OMDL challenges are distinct, since otherwise $\mathcal{B}_1$ can solve them trivially. All DLOG queries from $\mathsf{Fork}^{\mathcal{C}}$ are forwarded by $\mathcal{B}_1$ to the DLOG oracle in the game $\mathbf{G}^{\mathsf{omdl}}_{\mathbb{G}}$. Denote by $\mathsf{BadHash}$ the event that any two of the scalars $h_1, h'_1, \ldots, h_q, h'_q$ generated in the execution of $\mathsf{Fork}^{\mathcal{C}}$ are equal. Since these scalars are sampled uniformly from $\mathbb{Z}_p^*$, we have $\Pr[\mathsf{BadHash}] \leq 2q^2/p$.

It remains to show that if $\mathsf{Fork}^{\mathcal{C}}$ returns $(i_{sig}, \mathsf{Out}, \mathsf{Out}')$ and $\mathsf{BadHash}$ does not occur, $\mathcal{B}_1$ wins the game $\mathbf{G}^{\mathsf{omdl}}_{\mathbb{G}}$. Absorbing all strictly negligible terms into a function negl, we have

$$\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}_1) \geq \mathsf{acc}(\mathsf{Fork}^{\mathcal{C}}) - \Pr[\mathsf{BadHash}] \geq \frac{\Pr[E_1]^2}{q} - \frac{3q^2}{p} = \frac{\Pr[E_1]^2}{q} - \mathsf{negl}.$$

We use the notations introduced in the description of $\mathcal{C}$ to denote the variables in Out and append a prime $(\cdot)'$ to denote the variables in Out'. By the execution of $\mathsf{Fork}^{\mathcal{C}}$, we have $(\tilde{X}, M^*, \tilde{R}^*) = (\tilde{X}', M^{*\prime}, \tilde{R}^{*\prime})$ and $\tilde{X} = A_0$. Since $i_{sig} \in SS$, let $k^* = i_{sig}/2$ where $\mathsf{mapLR}(k^*) = lr^*$. Furthermore, if $\mathsf{mapLR}(k^*) = \perp$, then $lr^* = \perp$ as well.

We first show how to compute the discrete logarithm of $A_0, \ldots, A_{t-1}$. Denote these discrete logarithms by $a_0, \ldots, a_{t-1}$ and define $f(x) := \sum_{i=0}^{t-1} a_i x^i$. Since $\mathsf{BadHash}$ does not occur, we have $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, M^*, \tilde{R}^*) = h_{i_{sig}} \neq h'_{i_{sig}} =$

$\mathsf{H_{sig}}'(\tilde{X}, M^*, \tilde{R}^*)$. As $g^{z^*} = \tilde{R}^* g^{a_0 h_{i_{sig}}}$ and $g^{z^{*'}} = \tilde{R}^* g^{a_0 h'_{i_{sig}}}$, $\mathcal{B}_1$ computes $f(0) = a_0 = \frac{z^* - z^{*'}}{h_{i_{sig}} - h'_{i_{sig}}}$. Define $T_{\mathsf{dt}} := \{j : (i,k,d,c,z) \leftarrow \mathsf{dt}(j),\ k = k^*\}$. For each $j \in T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}$, let $(i,k,d,c,z) \leftarrow \mathsf{dt}(j)$ and $(i',k',d',c',z') \leftarrow \mathsf{dt}'(j)$, then $g^z = R_j S_j^d X_i^c$ and $g^{z'} = R_j S_j^{d'} X_i^{c'}$.

Since $\mathsf{BadPPO} = \mathsf{false}$ during both executions of $\mathcal{C}$, we know that (R_j, S_j, u_j, V_j) was returned by a query $\mathrm{PPO}(\mathsf{id}_i)$ prior to the query $\mathsf{H_{sig}}(\tilde{X}, M^*, \tilde{R}^*)$ in the first execution of $\mathcal{C}$. Because the two executions of $\mathcal{C}$ are identical up to the query $\mathsf{H_{sig}}(\tilde{X}, M^*, \tilde{R}^*)$, we have $\mathsf{id}'_i = \mathsf{id}_i$. Moreover, we know $d = h_k = h_{k^*} = h_{k'} = d'$. Therefore, $\mathcal{B}_1$ can compute $f(\mathsf{id}_i) = \mathsf{DL}_{\mathbb{G},g}(X_i) = \frac{z - z'}{c - c'}$.

Let $D := \{\mathsf{id}_i\}_{j \in T_{\mathsf{dt}} \cap T_{\mathsf{dt}'},\ (i,k,d,c,z) \leftarrow \mathsf{dt}(j)}$. If $S_2(lr^*) < t - |CS|$ occurs in the first execution of $\mathcal{C}$, then $|T_{\mathsf{dt}}| = |S_2(lr^*)| < t - |CS|$. Therefore, we have $|D| = |T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}| < t - |CS|$. Consequently, $\mathcal{B}_1$ can pick an arbitrary set $D' \subseteq HS \setminus D$ of size $(t - |CS| - |T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}| - 1)$ and for each $\mathsf{id}_i \in D'$, $\mathcal{B}_1$ queries the DLOG oracle on X_i. Thus $\mathcal{B}_1$ knows the value of $f(\mathsf{id}_i)$ for every $\mathsf{id}_i \in CS \cup D \cup D' \cup \{0\}$. Since $|CS \cup D \cup D' \cup \{0\}| = t$, $\mathcal{B}_1$ can recover $a_0, \ldots, a_{t-1}$ by Lagrange interpolation.

We now show how to compute the discrete log of $R_1, S_1, \ldots, R_{q_s}, S_{q_s}$. Denote these discrete logarithms by $r_1, s_1, \ldots, r_{q_s}, s_{q_s}$. From the execution of $\mathcal{C}$, we know $\mathsf{dt}(j) = (i,k,d,c,z) \neq \perp$ if and only if $\mathcal{C}$ queries DLOG on $R_j S_j^d X_i^c$. Therefore, we denote the DLOG query on $R_j S_j^d X_i^c$ as the DLOG query associated with $\mathsf{dt}(j)$. For each $j \in [1..q_s]$, there are the following cases.

Case 0: Both $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ are $\perp$. In this case, $\mathcal{B}_1$ queries $r_j \leftarrow \mathrm{DLOG}(R_j)$ and $s_j \leftarrow \mathrm{DLOG}(S_j)$.

Case 1: Exactly one of $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ is not $\perp$. Without loss of generality, assume $\mathsf{dt}(j) = (i,k,d,c,z)$, which implies $g^z = R_j S_j^d X_i^c$. $\mathcal{B}_1$ queries $s_j \leftarrow \mathrm{DLOG}(S_j)$ and computes $r_j = z - d \cdot s_j - c \cdot f(\mathsf{id}_i)$.

For all the following cases, both $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ are not $\perp$ and we denote $(i,k,d,c,z) \leftarrow \mathsf{dt}(j)$ and $(i',k',d',c',z') \leftarrow \mathsf{dt}'(j)$.

Case 2: $k \neq k'$ or $k = k' > k^*$. In this case, we have $d = h_k \neq h'_{k'} = d'$ and $g^z = R_j S_j^d X_i^c$, $g^{z'} = R_j S_j^{d'} X_i^{c'}$. Therefore, $\mathcal{B}_1$ computes $s_j = \frac{z - cf(\mathsf{id}_i) - z' + c' f(\mathsf{id}_i)}{d - d'}$ and $r_j = z - d \cdot s_j - c \cdot f(\mathsf{id}_i)$.

Case 3: $k = k' = k^*$. In this case, $\mathcal{B}_1$ computes r_j, s_j the same as Case 1.

Case 4: $k = k' < k^*$. $\mathcal{B}_1$ computes r_j, s_j the same as Case 1. Also, in this case we have $d = d'$ and $c = c'$. Therefore, $\mathcal{B}_1$ queries the DLOG oracle once in order to simulate the DLOG queries associated with $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$.

We now count the number of DLOG queries made by $\mathcal{B}_1$. $\mathcal{B}_1$ queries the DLOG oracle $|CS|$ times to simulate the queries $\mathrm{DLOG}(X_i)$ made by $\mathcal{C}$ for each $i \in CS$. If $S_2(lr^*) < t - |CS|$ occurs, $\mathcal{B}_1$ queries the DLOG oracle $|D'|$ times to compute $a_0, \ldots, a_{t-1}$. For each $j \in [1..q_s]$, $\mathcal{B}_1$ queries DLOG twice to simulate the queries associated with $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ and to compute r_j, s_j in cases 0, 1, 2, 4, and three times in **Case** 3. Since the condition of **Case 3** is equivalent to $j \in T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}$, the total DLOG queries made by $\mathcal{B}_1$ is $2q_s + |T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}| + |CS| + |D'| = 2q_s + t - 1$. Therefore, $\mathcal{B}_1$ breaks the OMDL assumption. □

Lemma 4 below is devoted to the event E_2, which captures forgeries under the condition $S_2(lr) \neq S_3(lr)$. Since $\mathsf{st.mapPP}(pp_i) \neq \perp$ is enforced in the partial-signing algorithm PS, an honest signer generates a pp_i (and $S_3(lr)$ grows) whenever a $psig_i$ is requested (and $S_2(lr)$ grows), so forgeries with $S_3(lr) \subset S_2(lr)$ will not occur. Therefore, we only consider the case $S_2(lr) \subset S_3(lr)$. In other words, E_2 captures the attack in which at least one pp_i is honestly generated but the corresponding $psig_i$ is forged. Our proof idea is to embed OMDL challenges into the honest signer's pp_i, which would fall in $S_3(lr) \setminus S_2(lr)$ when E_2 occurs. We show that by using the Lemma 2 with an approach inspired by the mix forking technique [11], all secret exponents in the forger's token authentication transcripts (u, V) (i.e., those belonging to $lr^*.\mathcal{S} \setminus S_3(lr^*)$) can be recovered, and an OMDL challenge in $S_3(lr^*) \setminus S_2(lr^*)$ can be solved without querying the DLOG oracle.

Lemma 4. *There exists an OMDL adversary $\mathcal{B}_2$ issuing at most $2q_s$ queries to the CHALL oracle and q queries to random oracle such that*

$$\Pr[E_2] \;\leq\; 8\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}_2) + \mathsf{negl}.$$

Moreover, $\mathcal{B}_2$ runs in time $T(\mathcal{B}_2) \leq \frac{16\mathsf{ns}^2 q \ln(16\mathsf{ns}/\Pr[E_2])}{\Pr[E_2]} \cdot T(\mathcal{F}) + T(q)$ where ns is the number of signers, and $T(q)$ is the time required to perform exponentiations and group operations.

Proof. Let $\mathcal{F}$ be an adversary as described in the theorem. Denote the output message-signature pair of $\mathcal{F}$ as $(M^*, sig^* = (\tilde{R}^*, z^*))$. Without loss of generality, we assume $\mathcal{F}$ always queries RO on $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, M^*, \tilde{R}^*)$ before $\mathcal{F}$ returns and always queries RO on $\mathsf{H}_{\mathsf{pp}}(\tilde{X}, X_i, R, S, V)$ and $\mathsf{H}_{\mathsf{non}}(\tilde{X}, lr)$ prior to the query $\textsc{PSign}\mathsf{O}(\mathsf{id}_i, lr)$ for some id_i and lr. This adds up to $2q_s$ additional RO queries, and we let $q = q_h + 2q_s + 1$ where $q_h = q_{pp} + q_{non} + q_{sig}$.

Let lr^* be the leader query whose nonce query $\mathsf{H}_{\mathsf{non}}(\tilde{X}, lr^*)$ is the last such query before the signature hash query $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, M^*, \tilde{R}^*)$ and which satisfies $\mathsf{SVf}(\tilde{X}, lr^*, \sigma^*) = \mathsf{true}$. Denote the event E_2 as

$$E_2 : \mathsf{Vf}(vk, M^*, \sigma^*) \;\wedge\; lr^* \neq \perp \;\wedge\; S_2(lr^*) \subset S_3(lr^*).$$

To begin, we construct an algorithm $\mathcal{C}$ which is compatible with the syntax in Lemma 2. The input of $\mathcal{C}$ consists of $2q_s$ random elements $\mathbf{R} = \{R_1, \ldots, R_{q_s}\}$, $\mathbf{S} = \{S_1, \ldots, S_{q_s}\}$ from $\mathbb{G}$, and uniformly random integers $h_1, \ldots, h_q \in \mathbb{Z}_p^*$. Also, $\mathcal{C}$ has access to an oracle DLOG, which on input $X \in \mathbb{G}$ outputs $\mathsf{DL}_{\mathbb{G},g}(X)$. $\mathcal{C}$ initializes the signers' states $\mathsf{st}_0, \ldots, \mathsf{st}_{\mathsf{ns}}$, counters $\mathsf{ctr}_h = 0$ and $\mathsf{ctr}_s = 0$. It then runs $\mathcal{F}$, answering its oracle queries as follows.

- $\widetilde{\textsc{Init}}(CS)$: $\mathcal{C}$ picks $a_0, \ldots, a_{t-1} \xleftarrow{\$} \mathbb{Z}_p$, sets $f(x) = \sum_{i=0}^{t-1} a_i x^i$, and defines $\tilde{X} = g^{f(0)}$, $X_i = g^{f(\mathsf{id}_i)}$ for $i \in [1..\mathsf{ns}]$. For $\mathsf{id}_i \in CS$ it stores $\mathsf{st}_i.\mathsf{sk} = f(\mathsf{id}_i)$. It returns $(\tilde{X}, (X_1, \ldots, X_{\mathsf{ns}}), \{\mathsf{st}_i.\mathsf{sk}\}_{\mathsf{id}_i \in CS})$.
- $\widetilde{\mathrm{RO}}[\mathsf{H}_{\mathsf{pp}}](x)$: If $\mathsf{H}_{\mathsf{pp}}(x) \neq \perp$, $\mathcal{C}$ returns $\mathsf{H}_{\mathsf{pp}}(x)$. Otherwise, increments ctr_h to set $\mathsf{H}_{\mathsf{pp}}(x) \leftarrow h_{\mathsf{ctr}_h}$ and returns $\mathsf{H}_{\mathsf{pp}}(x)$.

- $\widetilde{\mathrm{RO}}[\mathsf{H}_{\mathsf{non}}](x)$: If $\mathsf{H}_{\mathsf{non}}(x) \neq \perp$, $\mathcal{C}$ returns $\mathsf{H}_{\mathsf{non}}(x)$. Otherwise, increments ctr_h to set $\mathsf{H}_{\mathsf{non}}(x) \leftarrow h_{\mathsf{ctr}_h}$ and returns $\mathsf{H}_{\mathsf{non}}(x)$.
- $\widetilde{\mathrm{RO}}[\mathsf{H}_{\mathsf{sig}}](x)$: If $\mathsf{H}_{\mathsf{sig}}(x) \neq \perp$, $\mathcal{C}$ returns $\mathsf{H}_{\mathsf{sig}}(x)$. Otherwise, increments ctr_h to set $\mathsf{H}_{\mathsf{non}}(x) \leftarrow h_{\mathsf{ctr}_h}$ and returns $\mathsf{H}_{\mathsf{non}}(x)$.
- $\widetilde{\mathrm{PPO}}(\mathsf{id}_i)$: Same as in the game $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$, except that, in the simulation of SPP, $\mathcal{C}$ selects $u \xleftarrow{\$} \mathbb{Z}_p$, increments ctr_h and ctr_s to set $e = h_{\mathsf{ctr}_h}$, and computes $V = g^u / X_i^e R_{\mathsf{ctr}_s}^{e^2} S_{\mathsf{ctr}_s}^{e^3}$. $\mathcal{C}$ sets $\mathsf{H}_{\mathsf{pp}}(\tilde{X}, X_i, R_{\mathsf{ctr}_s}, S_{\mathsf{ctr}_s}, V) \leftarrow e$ and adds $pp_i \leftarrow (R_{\mathsf{ctr}_s}, S_{\mathsf{ctr}_s}, u, V)$ to $\mathsf{st}_i.\mathsf{curPP}$.
- $\widetilde{\mathrm{PSIGNO}}(\mathsf{id}_i, lr)$: Same as in the game $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$, except that, in the simulation of PS, if $\mathsf{st}_i.\mathsf{mapPP}(pp_{\mathsf{id}_i}) \neq \perp$ and all $pp_j \in lr.\mathsf{PP}$ pass the verifications, $\mathcal{C}$ retrieves $(R, S) \leftarrow pp_{\mathsf{id}_i}$ and other $(R_j, S_j) \in lr.\mathsf{PP}$ to compute $\tilde{R} = \prod_{i \in lr.\mathcal{S}} R_i (\prod_{i \in lr.\mathcal{S}} S_i)^d$ where $d \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, lr)$. $\mathcal{C}$ then removes pp_{id_i} from $\mathsf{st}_i.\mathsf{curPP}$ and returns $z_i \leftarrow \textsc{Dlog}\big(RS^d\big) + c\Delta_{\mathsf{id}_i, lr.\mathcal{S}}(0)\, f(\mathsf{id}_i)$ where $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, lr.\mathsf{msg}, \tilde{R})$.

Finally, $\mathcal{F}$ outputs its forgery $(lr^* = \{M^*, \mathcal{S}, \mathsf{PP}\}, \sigma^* = (\tilde{R}^*, z^*))$ and wins the game $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$. $\mathcal{C}$ returns $(I, \{\mathsf{out}_i\}_{i \in I}, \theta)$ where I are the set of random oracle indices such that out_i contains $h_i \leftarrow \mathsf{H}_{\mathsf{pp}}(\tilde{X}, X_i, R_i, S_i, V_i)$ for $\mathsf{id}_i \in lr^*.\mathcal{S} \setminus S_3(lr^*)$ and $\theta = (lr^*, \sigma^*)$.

Analysis of $\mathcal{C}$. There are two possible forging cases under the condition $S_2(lr^*) \subset S_3(lr^*)$: (1) $lr^*.\mathcal{S} \setminus S_3(lr^*) = \emptyset$; and (2) $lr^*.\mathcal{S} \setminus S_3(lr^*) \neq \emptyset$. We first present the analysis for case (1) and subsequently explain how to extend it to handle case (2).

Case (1): When $lr^*.\mathcal{S} \setminus S_3(lr^*) = \emptyset$, $\mathcal{F}$ forges without generating any pre-processing tokens such that $S_3(lr^*) = S_4(lr^*) = lr^*.\mathcal{S}$. We show that there is an OMDL adversary $\mathcal{B}_2$ which uses $\mathcal{C}$ as the helper algorithm to compute the discrete logarithms of (R^*, S^*) for an $\mathsf{id}^* \in S_3(lr^*) \setminus S_2(lr^*)$. $\mathcal{B}_2$ queries $2q_s$ challenges $\mathbf{R} = \{R_1, \ldots, R_{q_s}\}$, $\mathbf{S} = \{S_1, \ldots, S_{q_s}\}$ from $\textsc{Chall}$ and selects random $h_1, \ldots, h_q \in \mathbb{Z}_p^*$ as the input to $\mathcal{C}$. The $\textsc{Dlog}$ queries from $\mathcal{C}$ are relayed by $\mathcal{B}_2$ to its $\textsc{Dlog}$ of $\mathbf{G}^{\mathsf{omdl}}_{\mathbb{G}}$. From $(I, \{\mathsf{out}_i\}_{i \in I}, \theta) \leftarrow \mathcal{C}$, $\mathcal{B}_2$ needs only θ to obtain $((z^*, c^*, f(0)), \tilde{R}^* = \prod_{\mathsf{id}_j \in lr^*.\mathcal{S}} R_j (\prod_{\mathsf{id}_j \in lr^*.\mathcal{S}} S_j)^{d^*})$ and compute the sum $\alpha = \sum_{\mathsf{id}_j \in lr^*.\mathcal{S}} r_j + d^* \cdot s_j = z^* - c^* f(0)$.

Subsequently, for all (R_j, S_j) in the set $D = S_3(lr^*) \setminus S_2(lr^*)$, $\mathcal{B}_2$ queries $\{r_j \leftarrow \textsc{Dlog}(R_j)\}_{\mathsf{id}_j \in D}$ and $\{s_j \leftarrow \textsc{Dlog}(S_j)\}_{\mathsf{id}_j \in D \setminus \{\mathsf{id}^*\}}$ where $\mathsf{id}^* \in D$ is the targeted identity who owns (R^*, S^*). For those (R_j, S_j) belonging to $\mathsf{id}_j \in S_2(lr^*)$, they have been queried to PSIGNO and $\mathcal{B}_2$ knows $z_j = r_j + d^* \cdot s_j + c^* \Delta_{\mathsf{id}_j, lr^*.\mathcal{S}}(0) f(\mathsf{id}_j)$ in which $\mathcal{B}_2$ can query $s_j \leftarrow \textsc{Dlog}(S_j)$ and computes $r_j = z_j - c^* \Delta_{\mathsf{id}_j, lr^*.\mathcal{S}}(0) f(\mathsf{id}_j) - d^* \cdot s_j$. Therefore, $lr^*.\mathcal{S} = D \cup S_2(lr^*)$ and $\mathcal{B}_2$ can extract $s^* \leftarrow \mathsf{DL}_{\mathbb{G},g}(S^*)$ by computing:

$$s^* = \frac{\alpha - \sum_{\mathsf{id}_j \in lr^*.\mathcal{S}} r_j - d^* \sum_{\mathsf{id}_j \in lr^*.\mathcal{S} \setminus \{\mathsf{id}^*\}} s_j}{d^*}.$$

For the OMDL challenges not involved in the forgery $(R_j, S_j) \notin lr^*.\mathsf{PP}(\mathsf{id}_j)$ where $\mathsf{id}_j \in HS$, they are either queried or have not queried to PSIGNO. If

queried, $\mathcal{B}_2$ knows the corresponding partial signature z_j in which it can query $s_j \leftarrow \textsc{Dlog}(S_j)$ and compute $r_j = z_j - f(\mathsf{id}_j)c_j\Delta_{\mathsf{id}_j,lr.\mathcal{S}}(0) - d_j s_j$. Else, $\mathcal{B}_2$ queries $r_j \leftarrow \textsc{Dlog}(R_j)$ and $s_j \leftarrow \textsc{Dlog}(S_j)$.

We now count the number of Dlog queries made by $\mathcal{B}_2$. For each challenge pair in $\{R_j = g^{r_j}, S_j = g^{s_j}\}_{j=1}^{q_s}$, $\mathcal{B}_2$ queries Dlog twice except the $s^* \leftarrow \mathsf{DL}_{\mathbb{G},g}(S^*)$ used by the targeted honest signer $\mathsf{id}^* \in S_3(lr^*) \setminus S_2(lr^*)$. Hence the total Dlog queries made by $\mathcal{B}_2$ equals $2q_s - 1$, breaking the OMDL assumption. Denote $\mathsf{BadHash}$ as the event of $h_i = h_j$ for some $i \neq j \leq q$. Since every hash value is selected independently, it is $\Pr[\mathsf{BadHash}] = 1/p$. Absorbing all strictly negligible terms into a function negl, we have:

$$\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}_2) \geq \Pr[\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}(\mathcal{F}) = \mathsf{true}] - \Pr[\mathsf{BadHash}] \geq \Pr[E_2] - \mathsf{negl},$$

and $T(\mathcal{B}_2) \approx T(\mathcal{F}) + T(q)$.

Case (2): Now, we consider the case where $\mathcal{F}$ forges with at least one preprocessing token in $lr^*.\mathsf{PP}$ generated by itself, i.e., $lr^*.\mathcal{S} \setminus S_3(lr^*) \neq \emptyset$ where $S_3(lr^*) \subset S_4(lr^*) \subseteq lr^*.\mathcal{S}$. In this case, the output of $\mathcal{C}$'s single run is not sufficient for solving the OMDL problem. This is because $\mathcal{C}$ does not know the discrete logarithms of $\{R_i, S_i\}_{\mathsf{id}_i \in lr^*.\mathcal{S} \setminus S_3(lr^*)}$ generated by $\mathcal{F}$. To overcome this obstacle, we need a helper algorithm $\mathcal{C}'$.

Construct $\mathcal{C}'$ from $\mathsf{MIMFork}^{\mathcal{C}}_{N,2}$. We now construct the helper algorithm $\mathcal{C}'$, which is a syntactically valid OMDL adversary, using $\mathsf{MIMFork}^{\mathcal{C}}_{N,2}$ with $N = |I| = |lr^*.\mathcal{S} \setminus S_3(lr^*)|$. $\mathcal{C}'$ initiates $\mathcal{C}$ the same as $\mathcal{B}_2$ did in the previous case. Similar to the proof in case (1), from $(\mathcal{O}, \mathcal{O}') \leftarrow \mathsf{MIMFork}^{\mathcal{C}}_{N,2}$, $\mathcal{C}'$ also computes the values $(\alpha, \{r_j \leftarrow \textsc{Dlog}(R_j)\}_{\mathsf{id}_j \in S_3(lr^*)}, \{s_j \leftarrow \textsc{Dlog}(S_j)\}_{\mathsf{id}_j \in S_3(lr^*) \setminus \{\mathsf{id}^*\}})$. Now, $\mathcal{C}'$ needs to additionally handle the transcripts $\{(e_{i,j} = h_{i,j}, v_i + r_i e_{i,j}^2 + s_i e_{i,j}^3 = u_{i,j} - f(\mathsf{id}_i)e_{i,j}, V_i) \leftarrow pp_{i,j}\}_{j=0}^{2}$ for $\mathsf{id}_i \in D$ where $D = lr^*.\mathcal{S} \setminus S_3(lr^*)$. These yields a solvable system of linear equations, allowing $\mathcal{C}'$ to extract the discrete logarithms $\{r_i \leftarrow \mathsf{DL}_{\mathbb{G},g}(R_i), s_i \leftarrow \mathsf{DL}_{\mathbb{G},g}(S_i)\}_{\mathsf{id}_i \in D}$. As $lr^*.\mathcal{S} = D \cup S_3(lr^*)$, $\mathcal{C}'$ can compute

$$\mathsf{DL}_{\mathbb{G},g}(S^*) \to s^* = \frac{\alpha - \sum_{\mathsf{id}_i \in lr^*.\mathcal{S}} r_i - d^* \sum_{\mathsf{id}_i \in lr^*.\mathcal{S} \setminus \{\mathsf{id}^*\}} s_i}{d^*}.$$

$\mathcal{C}'$ then solves those OMDL challenges not involved in the forgery by querying the Dlog oracle. Since a hash collision occurs with a negligible probability only, by Lemma 2, we have:

$$\mathsf{acc}(\mathcal{C}') \geq \mathsf{acc}(\mathsf{MIMFork}^{\mathcal{C}}_{N,2}) - \mathsf{negl} \geq \hat{\varepsilon}/8 - \mathsf{negl} \geq \Pr[E_2]/8 - \mathsf{negl}.$$

However, after broadcasting a pp_i in any of the $|I| \cdot T_{max}$ rewinds in $\mathsf{MIMFork}^{\mathcal{C}}_{N,2}$, $\mathcal{F}$ still can issue PSignO queries till it forges a signature. So, $\mathcal{C}'$ does not solve OMDL because the $|I| \cdot T_{max}$ rewinds make $\mathcal{C}$ issues more than $2q_s$ Dlog queries to answer these additional PSignO queries.

Construct $\mathcal{B}_2$ from $\mathcal{C}'$. We now construct an OMDL adversary $\mathcal{B}_2$ which is defined the same as $\mathcal{C}'$ except that, starting from the first rewind of each instance $i \in I$, $\mathcal{B}_2$ aborts any executions of $\mathcal{C}$ after it has obtained the pp_i. This avoids $\mathcal{B}_2$ from issuing too many DLOG queries as done by $\mathcal{C}'$. Moreover, $\mathcal{B}_2$ has the same success probability as $\mathcal{C}'$ because the θ required to compute α is available after the first run ended completely, that is, before a rewind begins. Since $N = |I| = |lr^*.\mathcal{S} \setminus S_3(lr^*)| \leq \mathsf{ns} - 1$, and every $i \in I$ collected $n = 2$ forks, $\mathcal{B}_2$ uses time:

$$\begin{aligned} T(\mathcal{B}_2) \leq T(\mathcal{C}') &\leq |I| \cdot T_{max} \cdot T(\mathcal{C}) + T(q) \leq |I| \cdot T_{max} \cdot T(\mathcal{F}) + T(q) \\ &\leq \mathsf{ns} \cdot \frac{8 \cdot \mathsf{ns} \cdot 2q \ln(8 \cdot \mathsf{ns} \cdot 2/\Pr[E_2])}{\Pr[E_2]} \cdot T(\mathcal{F}) + T(q) \\ &\leq \frac{16\mathsf{ns}^2 q \ln(16\mathsf{ns}/\Pr[E_2])}{\Pr[E_2]} \cdot T(\mathcal{F}) + T(q). \end{aligned}$$

This completes the proof. □

Up to this point, FROST2+ has achieved the TS-SUF-3 security whose forgeries are covered by the events E_1 and E_2. If we apply the ATS[TS,DS] transformation framework as in FROST [8,6], we obtain an ATS[FROST2+, DS] scheme that achieves TS-SUF-4 security. However, the transformation comes with the cost of managing an extra DS key pair, longer pp_i and slower signing time. To avoid these, we give a direct proof instead, which is the proof for event E_3. As $S_4(lr) = HS \cap lr^*.\mathcal{S}$ and the event E_2 has covered $S_2(lr) \neq S_3(lr)$, we only need to deal with the forgeries under $S_2(lr) = S_3(lr) \subset S_4(lr)$ in E_3.

Before presenting the direct proof, we would like to introduce a leader request forger $\mathcal{F}_{lr}$ of FROST2+ which possesses the same ability as that of a normal TS-SUF-4 forger $\mathcal{F}$, but instead of producing a valid signature (σ^*, lr^*) as the output, $\mathcal{F}_{lr}$ produces a valid (i.e. $S_3(lr^*) \subset S_4(lr^*)$) leader request lr^* as the output. Denote $A \Longrightarrow B$ as A implies B, that is, B is reducible to A. Below, we show that $\mathcal{F}_{lr} \Longleftrightarrow \mathcal{F}$ when a forgery falls under $S_3(lr) \subset S_4(lr)$, that is, when at least a pre-processing token is forged.

Theorem 2 **($\mathcal{F}_{lr} \Longleftrightarrow \mathcal{F}$).** *For FROST2+, a TS-SUF-4 forger $\mathcal{F}$ which output a forgery that falls under the event*

$$E: \ \mathsf{Vf}(\tilde{X}, M^*, \sigma^*) \ \wedge \ lr^* \neq \perp \ \wedge S_3(lr^*) \subset S_4(lr^*)$$

is equivalent to a leader request forger $\mathcal{F}_{lr}$.

Proof. The proof follows from Lemmas 5 and 6 which show that $\mathcal{F}$ and $\mathcal{F}_{lr}$ are equivalent under E such that they are reducible to each other. □

Lemma 5 **($\mathcal{F} \Longrightarrow \mathcal{F}_{lr}$).** *Let $\mathcal{F}$ be a TS-SUF-4 forger for FROST2+, and $\mathcal{F}_{lr}$ be a leader request forger in FROST2+. There exists an $\mathcal{F}_{lr}$ which wins $\mathbf{G}^{\mathsf{ts\text{-}lr}}_{FROST2+}$ such that*

$$\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{FROST2+}(\mathcal{F}_{lr}) \geq \Pr[E] - \mathsf{negl},$$

where E is the event of Theorem 2. Moreover, the running time of $\mathcal{F}_{lr}$ is at most $T(\mathcal{F}) + O(q)$ where $O(q)$ is the wrapper overhead that relays oracle queries.

Proof. It is straight forward to see that $\mathcal{F}_{lr}$ can simulate the environment perfectly for $\mathcal{F}$ by acting as the middle-man in between its underlying FROST2+ oracles and $\mathcal{F}$. When $\mathcal{F}$ outputs a valid TS-SUF-4 forgery (σ^*, lr^*), $\mathcal{F}_{lr}$ uses lr^* as its output. In fact, $\mathcal{F}_{lr}$ can return lr^* before answering the PSIGNO query on lr^*, that is, whenever it sees that a valid lr^* contains pp_i such that $S_3(lr^*) \subset S_4(lr^*)$. Therefore, $\mathcal{F}_{lr}$ runs in time at most that of $\mathcal{F}$. □

Lemma 6 ($\mathcal{F}_{lr} \Longrightarrow \mathcal{F}$)**.** *Let $\mathcal{F}$ be a TS-SUF-4 forger for FROST2+, and $\mathcal{F}_{lr}$ be a leader request forger in FROST2+. There exists an $\mathcal{F}$ which wins $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{FROST2+}$ such that*

$$\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{FROST2+}(\mathcal{F}_{lr}) \leq \Pr[E] \leq 8t \cdot \mathsf{Adv}^{\mathsf{ts\text{-}suf\text{-}4}}_{FROST2+}(\mathcal{F})$$

where E is the event of Theorem 2 and $t \leq \mathsf{ns}$. Moreover, $\mathcal{F}$ runs in time $T(\mathcal{F}) \leq \frac{24tq\ln(24t/\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{FROST2+}(\mathcal{F}_{lr}))}{\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{FROST2+}(\mathcal{F}_{lr})} \cdot T(\mathcal{F}_{lr}) + T(q)$ where q is the number of hash queries issued and $T(q)$ is the time required to perform exponentiations and group operations. The number of $\mathcal{F}$'s queries to each oracle is at most that of $\mathcal{F}_{lr}$.

Proof. We show that if there exists a leader request forger $\mathcal{F}_{lr}$, then we can construct a TS-SUF-4 forger $\mathcal{F}$ to win in $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$ such that the output forgery (lr^*, σ^*) falls under the event:

$$E: \mathsf{Vf}(\tilde{X}, M^*, \sigma^*) \ \wedge \ lr^* \neq \perp \ \wedge S_3(lr) \subset S_4(lr).$$

$\mathcal{F}$ constructs a wrapper $\mathcal{C}$ whose task is only to relay all oracle queries from $\mathcal{F}_{lr}$ to $\mathcal{F}$, which then replies by querying its own oracles. To begin, $\mathcal{F}$ forwards the $\textsc{Init}(\widetilde{CS})$ query from $\mathcal{C}$ to its underlying INIT oracle and returns the required parameter $(\tilde{X}, \mathsf{aux}, \{x_i\}_{\mathsf{id}_i \in \widetilde{CS}})$. Without loss of generality, we assume $|\widetilde{CS}| = t-1$. If $|\widetilde{CS}| < t-1$, $\mathcal{F}$ removes a random $\widetilde{CS}$ of size $t-1-|\widetilde{CS}|$ from HS and queries $\textsc{Init}(\widetilde{CS} \cup \widetilde{CS})$ but does not return the signing key share of $\mathsf{id}_j \in \widetilde{CS}$ to $\mathcal{C}$.

When $\mathcal{F}_{lr}$ returns a valid $lr^* = (M, \mathcal{S}, \{pp_i = (R_i, S_i, u_i, V_i)\}_{\mathsf{id}_i \in \mathcal{S}})$ during the first run, $\mathcal{C}$ knows $(I, \mathsf{out}_{i^*}, \theta = \perp)$ with $I = \{i^*\} = S_4(lr^*) \setminus S_3(lr^*)$ corresponds to a query $e_{i^*} = \mathsf{H}_{\mathsf{pp}}(\tilde{X}, X_{i^*}, R_{i^*}, S_{i^*}, V_{i^*})$ such that $pp_{i^*} \in lr^*$. If $1 < |I| \leq \mathsf{ns}$, $\mathcal{C}$ selects a random index $i^* \in I$. Since $\widetilde{CS} \in HS$ is chosen randomly by $\mathcal{C}$ and is hidden from the view of $\mathcal{F}_{lr}$, the probability of $\mathcal{C}$ getting an invalid $i^* \in I$ is at most $\Pr[i^* \in \widetilde{CS}] = \frac{|\widetilde{CS}|}{|HS|} \leq \frac{t-1}{t}$. Then $\mathcal{C}$ returns $(e_{i^*}, \mathsf{out}_{i^*})$.

$\mathcal{F}$ executed $(\mathcal{O}, \mathcal{O}') \leftarrow \mathsf{MIMFork}^{\mathcal{C}}_{1,3}$ to get four transcripts

$$(\tilde{X}, X_{i^*}, R_{i^*}, S_{i^*}, V_{i^*}, u_{i^*}, e_{i^*}) \leftarrow \mathcal{O}, (\{u_{i^*,j}, e_{i^*,j}\}_{j=1}^{3}) \leftarrow \mathcal{O}'$$

which yields a solvable system of linear equations and allows $\mathcal{F}$ to extract $\mathsf{st}_{i^*}.\mathsf{sk} = x_{i^*} \leftarrow \mathsf{DL}_{\mathbb{G},g}(X_{i^*}), r_{i^*} \leftarrow \mathsf{DL}_{\mathbb{G},g}(R_i), s_{i^*} \leftarrow \mathsf{DL}_{\mathbb{G},g}(S_{i^*})$. Thus, $\mathcal{F}$ knows $|\widetilde{CS}| + 1 = t$ pairs of $(\mathsf{id}_i, \mathsf{st}_i.\mathsf{sk} = x_i = f(\mathsf{id}_i))$ and can extract the shared signing key $f(0)$ through Lagrange interpolation. $\mathcal{F}$ then utilizes $f(0)$ to generate a signature σ^* on lr^* and output (σ^*, lr^*) to win its $\mathbf{G}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}$ game.

Throughout the game, all q hash values are produced by $\mathcal{F}$'s hash oracles and we assume hash collision does not occur, or $\mathcal{F}$ can win without $\mathcal{F}_{lr}$. Since $\mathcal{F}$ simulates the environment perfectly, by Lemma 2, we have:

$$\mathsf{Adv}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}(\mathcal{F}) = \mathsf{acc}(\mathsf{MIMFork}^{\mathcal{C}}_{1,3}) \geq (1-\Pr[i^* \in \widetilde{CS}])\frac{\Pr[E]}{8} = \frac{\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{\mathsf{FROST2+}}(\mathcal{F}_{lr})}{8t}$$

and $\mathcal{F}$ runs in time $T(\mathcal{F}) \leq \frac{24tq\ln(24t/\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{\mathsf{FROST2+}}(\mathcal{F}_{lr}))}{\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{\mathsf{FROST2+}}(\mathcal{F}_{lr})} \cdot T(\mathcal{F}_{lr}) + T(q)$. □

Now we are ready to prove security under event E_3 using an approach similar to that of Lemma 3, by constructing an OMDL challenger B_3 that uses the forger $\mathcal{F}_{lr}$, which is equivalent to the TS-SUF-4 forger $\mathcal{F}$. A careful reader will notice that we do not rely on $\mathcal{F}$. This is deliberate and required for proof soundness, not for reduction tightness. In particular, if we used $\mathcal{F}$, we would exhaust the DLOG quota when answering PSIGNO queries, preventing us from performing the computation needed (in Case 3 of Lemma 3) to extract the discrete logarithms of (R, S) used to construct lr^*.

Lemma 7. *There exists an OMDL adversary $\mathcal{B}_3$ issuing at most $2q_s + t$ queries to the CHALL oracle and q queries to random oracle such*

$$\Pr[E_3] \leq 8\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}_3) + \mathsf{negl}.$$

Moreover, $\mathcal{B}_3$ runs in time $T(\mathcal{B}_3) \leq \frac{24q\ln(24/\Pr[E_3])}{\Pr[E_3]} \cdot T(\mathcal{F}) + T(q)$ where ns is the number of signers, and $T(q)$ is the time required to perform exponentiations and group operations.

Proof. The proof setup is similar to that of Lemma 3 but considering the event

$$E_3 : \mathsf{Vf}(\tilde{X}, M^*, \sigma^*) \ \wedge \ lr^* \neq \perp \ \wedge \neg(S_2(lr) = S_3(lr) = S_4(lr)).$$

As event E_1 has covered $|S_2(lr^*)| < t$ and E_2 has covered $S_2(lr^*) \neq S_3(lr^*)$, we only need to deal with the forgeries under $S_2(lr^*) = S_3(lr^*) \subset S_4(lr^*)$ in event E_3 such that $|S_2(lr^*)| = |S_3(lr^*)| \geq t$. Furthermore, there must be at least $t+1$ honest signers in $lr^*.\mathcal{S}$ where t fall under $S_2(lr^*) = S_3(lr^*)$ and one falls under $S_4(lr^*) \setminus S_3(lr^*)$. Notice that $E_3 \subset E$ for the event E in Theorem 2. By Lemma 5, from any TS-SUF-4 forger $\mathcal{F}$ that wins under E we can construct a leader request forger $\mathcal{F}_{lr}$ with

$$\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{\mathsf{FROST2+}}(\mathcal{F}_{lr}) \ \geq \ \Pr[E_3] - \mathsf{negl}, \qquad T(\mathcal{F}_{lr}) \ \leq \ T(\mathcal{F}) + O(q),$$

where the $O(q)$ overhead is that of the wrapper that relays the oracle queries of $\mathcal{F}$ and outputs lr^* as soon as it sees $S_3(lr^*) \subset S_4(lr^*)$.

We simulate the key generation and oracles the same as that of Lemma 3 but here, $\mathcal{C}$ runs $\mathcal{F}_{lr}$ instead of $\mathcal{F}$, and takes in $h_1, \ldots, h_q$ only as the hash values. Unlike $\mathcal{F}$ which issues a PSIGNO query for lr^* to generate partial signatures before forging, $\mathcal{F}_{lr}$ outputs lr^* directly without issuing this specific query in its winning condition. Running $\mathcal{F}_{lr}$ inside $\mathcal{C}$ (rather than $\mathcal{F}$) is what avoids

exhausting the DLOG quota on the PSIGNO query for lr^*, exactly as discussed before the lemma.

After receiving the output $(lr^* = \{M^*, \mathcal{S}, \mathsf{PP} = \{pp_i = (R_i, S_i, u_i, V_i)\}_{\mathsf{id}_i \in \mathcal{S}}\})$ from $\mathcal{F}_{lr}$, $\mathcal{C}$ returns $\perp$ if $\mathsf{BadPPO} = \mathsf{true}$, or E_3 does not occur. Else, denote $I_{\mathcal{C}} = S_4(lr) \setminus S_3(lr)$, $\mathcal{C}$ selects an index $i^* \in I_{\mathcal{C}}$ such that i^* corresponds to $e_{i^*} = \mathsf{H}_{\mathsf{pp}}(\tilde{X}, X_{i^*}, R_{i^*}, S_{i^*}, V_{i^*})$ with $pp_{i^*} \in lr^*$. Then, $\mathcal{C}$ returns $(e_{i^*}, \mathsf{out}_{i^*}) \leftarrow \mathcal{O}$. Let $\hat{\varepsilon} := \mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{\mathsf{FROST2+}}(\mathcal{F}_{lr})$ denote the success probability of $\mathcal{C}$.

Construct $\mathcal{C}'$ from $\mathsf{MIMFork}^{\mathcal{C}}_{1,3}$. We can construct a helper algorithm $\mathcal{C}'$ which is a syntactically valid OMDL adversary using $\mathsf{MIMFork}^{\mathcal{C}}_{1,3}$ and the available DLOG oracle. To begin, $\mathcal{C}'$ queries its INIT and then queries CHALL oracle $2q_s + t$ times to generate the three lists of group elements $(\mathbf{A}, \mathbf{R}, \mathbf{S})$ as the input to run $\mathsf{MIMFork}^{\mathcal{C}}_{1,3}$. All DLOG queries from $\mathsf{MIMFork}^{\mathcal{C}}_{1,3}$ are forwarded by $\mathcal{C}'$ to the DLOG oracle in the game $\mathbf{G}^{\mathsf{omdl}}_{\mathbb{G}}$.

We first show how to compute the discrete logarithm of $A_0, \ldots, A_{t-1}$. From $(\mathcal{O}, \mathcal{O}') \leftarrow \mathsf{MIMFork}^{\mathcal{C}}_{1,3}$, $\mathcal{C}'$ obtains $\{e_{i^*}, pp_{i^*} = (R_{i^*}, S_{i^*}, u_{i^*}, V_{i^*}) \leftarrow \mathsf{out}_{i^*}\} \leftarrow \mathcal{O}$ and $\{e_{i^*,j}, pp_{i^*} = (R_{i^*}, S_{i^*}, u_{i^*,j}, V_{i^*})\}_{1 \le j \le 3} \leftarrow \mathcal{O}'$. This yields a solvable system of linear equations, allowing $\mathcal{C}'$ to extract the discrete logarithms $f(\mathsf{id}_{i^*}) = x_{i^*} \leftarrow \mathsf{DL}(X_{i^*}), r_{i^*} \leftarrow \mathsf{DL}(R_{i^*})$ and $s_{i^*} \leftarrow \mathsf{DL}(S_{i^*})$. Without loss of generality, we assume $|CS| = t - 1$ and so $\mathcal{C}'$ knows $|CS| + 1 = t$ pairs of $(\mathsf{id}_i, f(\mathsf{id}_i))$. If $|CS| < t - 1$, $\mathcal{C}'$ can compute $f(\mathsf{id}_i) = x_i \leftarrow \text{DLOG}(X_i)$ for a random set $\mathsf{id}_i \in D \subset S_3(lr^*)$ of size $|D| = t-1-|CS|$. $\mathcal{B}_3$ can interpolate the key generation polynomial $f(\cdot)$ to extract all $a_0 \leftarrow \mathsf{DL}(A_0), \ldots, a_{t-1} \leftarrow \mathsf{DL}(A_{t-1})$. Since lr^* have not been queried to PSIGNO, for the OMDL challenges used as tokens $(R_i, S_i) \in S_3(lr^*) \setminus D$, $\mathcal{C}'$ simply queries the DLOG oracle to solve them. For those $(R_i, S_i) \in D$ and $(R_i, S_i) \notin S_3(lr^*)$ generated through the PPO queries before $\mathcal{F}_{lr}$ returns lr^*, they are either not queried, or queried once to PSIGNO or DLOG. $\mathcal{C}'$ queries DLOG oracle to solve them as explained in the cases 0, 1, 2 and 4 in Lemma 3.

However, $\mathcal{C}'$ does not break the OMDL assumption because during the T_{max} rewinds, $\mathcal{F}_{lr}$'s view changes after recomputing $u_{i^*,j}$ and it may continue issuing PSIGNO queries which then exceeds the DLOG quota.

Construct $\mathcal{B}_3$ from $\mathcal{C}'$. We now construct an OMDL adversary using $\mathcal{B}_3$ which is defined the same as $\mathcal{C}'$ except that, starting from the first rewind of the instance $i^* \in I$, $\mathcal{B}_3$ aborts any execution of $\mathcal{C}$ after it has obtained the $\{e_{i^*,j}, pp_{i^*} = (R_{i^*}, S_{i^*}, u_{i^*,j}, V_{i^*})\}_{1 \le j \le 3} \leftarrow \mathcal{O}'$. This avoids $\mathcal{B}_3$ from issuing too many DLOG queries as done by $\mathcal{C}'$. Moreover, $\mathcal{B}_3$ has the same success probability as $\mathcal{C}'$ such that $\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}_3) = \mathsf{acc}(\mathcal{C}')$. Therefore, $\mathcal{B}_3$ extracted all $2q_s + t$ discrete logarithms in the OMDL challenges by using only $2q_s + t - 1$ DLOG queries. Absorbing all strictly negligible terms such as $\Pr[\mathsf{BadPPO}]$ and hash collision into a function

negl, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}_3) = \mathsf{acc}(\mathcal{C}') &\geq \mathsf{acc}(\mathsf{MIMFork}^{\mathcal{C}}_{1,3}) - \mathsf{negl} \geq \hat{\varepsilon}/8 - \mathsf{negl} \qquad //\text{by Lemma 2}\\ &\geq \frac{\mathsf{Adv}^{\mathsf{ts\text{-}lr}}_{\mathsf{FROST2+}}(\mathcal{F}_{lr})}{8} - \mathsf{negl}\\ &\geq \frac{\Pr[E_3]}{8} - \mathsf{negl} \qquad //\text{by Lemma 5}\end{aligned}$$

and $\mathcal{B}_3$ runs in time:

$$\begin{aligned}T(\mathcal{B}_3) \leq T(\mathcal{C}') &\leq |I| \cdot T_{max} \cdot T(\mathcal{F}_{lr}) + T(q)\\ &\leq N \cdot \frac{8Nnq\ln(8Nn/\hat{\varepsilon})}{\hat{\varepsilon}} \cdot T(\mathcal{F}_{lr}) + T(q)\\ &\leq \frac{24q\ln(24/\Pr[E_3])}{\Pr[E_3]} \cdot T(\mathcal{F}) + T(q),\end{aligned}$$

where the last line uses $\hat{\varepsilon} \geq \Pr[E_3] - \mathsf{negl}$ and $T(\mathcal{F}_{lr}) \leq T(\mathcal{F}) + O(q)$ with the resulting $O(q) \cdot \frac{24q\ln(24/\Pr[E_3])}{\Pr[E_3]}$ overhead absorbed into $T(q)$. □

4.1 Security with Distributed Key Generation

Our amendments on the pre-processing token generation and verification mechanism are orthogonal to key generation, allowing seamless integration with FROST2 when instantiated with secure distributed key generation (DKG) protocols such as PedPoP [22,6]. This follows because SPP/PS only use the existing key share $\mathsf{st}_i.\mathsf{sk}$ and token exponents (r, s), and do not involve the DKG commitment/PoP messages. To ensure completeness, we demonstrate that our proposed scheme maintain TS-SUF-4 security in the ROM when it is instantiated in the DKG setting using PedPoP (in Appendix B).

Essentially, the oracle simulations are similar to that in Theorem 1, except for the key generation. We first show how to build an OMDL adversary $\mathcal{B}'_1$ for a forgery under event E_1 in the DKG setting. Intuitively, we construct the OMDL adversary $\mathcal{B}'_1$ using the mix forking technique [11] as follows. First, we define a wrapper $\mathcal{C}$ that is same as that in Lemma 3 but its $\widetilde{\textsc{Init}}$ simulates the PedPoP protocol. Then, an algorithm $\mathcal{D}$ runs $\mathsf{Fork}^{\mathcal{C}}$ to extract the signing key $f(0)$. Next, we define $\mathcal{D}'$ which is syntactically a valid OMDL adversary but does not break the OMDL assumption. $\mathcal{D}'$ runs $\mathsf{MIMFork}^{\mathcal{D}}_{|CS|,1}$ and solves all the OMDL challenges. Finally, we present $\mathcal{B}'_1$, which uses $\mathcal{D}'$ as a helper to break the OMDL assumption.

We do not discuss the DKG-counterpart $\mathcal{B}'_2$ for the $\mathcal{B}_2$ in Lemma 4 which knows the signing key shares of every honest signer and it can run the PedPoP protocol honestly. Since $\mathcal{B}_1$ in Lemma 3 and $\mathcal{B}_3$ in Lemma 7 share the same game simulation strategy, the DKG-counterpart $\mathcal{B}'_3$ also adopts the same key generation simulation as $\mathcal{B}'_1$'s. However, $\mathcal{B}'_3$ does not need to make use of the mix forking technique. Specifically, $\mathcal{B}'_3$ runs $\mathsf{MIMFork}^{\mathcal{C}}_{1,3}$ as in Lemma 7 to extract $f(\mathsf{id}_{i^*})$ and subsequently the $f'(\mathsf{id}_{i^*})$. This allows $\mathcal{B}'_3$ to perform Lagrange interpolation and solve an extra OMDL challenges without querying $\textsc{Dlog}$.

Theorem 3. *Let $\mathcal{F}$ be a TS-SUF-4 forger against* FROST2+ *with PedPoP making at most q queries to the random oracle. Suppose the success probability of $\mathcal{F}$ is bounded by $\mathsf{Adv}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}(\mathcal{F}) \leq \Pr[E_1] + \Pr[E_2] + \Pr[E_3]$. Then, there exist three OMDL adversaries $\mathcal{B}'_1, \mathcal{B}'_2$ and $\mathcal{B}'_3$ such that:*

$$\mathsf{Adv}^{\mathsf{ts\text{-}suf\text{-}4}}_{\mathsf{FROST2+}}(\mathcal{F}) \leq \sqrt{8q \cdot \mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}'_1)} + 8\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}'_2) + 8\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}'_3) + \mathsf{negl}$$

where $\mathcal{B}'_1$ and $\mathcal{B}'_3$ makes at most $2q_s + t$ queries to the Chall *oracle, and $\mathcal{B}'_2$ makes at most $2q_s$ queries. Moreover, the running times of $\mathcal{B}'_1, \mathcal{B}'_2$ and $\mathcal{B}'_3$ are bounded above by*

$$T(\mathcal{B}'_1) \leq \frac{128t^2q^2\ln(64tq/\Pr[E_1]^2)}{\Pr[E_1]^2} \cdot T(\mathcal{F}) + T(q),$$
$$T(\mathcal{B}'_2) \leq \frac{16\mathsf{ns}^2q\ln(16\mathsf{ns}/\Pr[E_2])}{\Pr[E_2]} \cdot T(\mathcal{F}) + T(q), \text{ and}$$
$$T(\mathcal{B}'_3) \leq \frac{24q\ln(24/\Pr[E_3])}{\Pr[E_3]} \cdot T(\mathcal{F}) + T(q),$$

respectively, where $t \leq \mathsf{ns}$, and $T(q)$ denotes the time required for the relevant group operations and exponentiations.

Proof. We show how to construct the $\mathcal{B}'_1$ OMDL adversary before showing the $\mathcal{B}'_3$ OMDL adversary which shares the same proving strategy.

The $\mathcal{B}'_1$ Adversary. We define an algorithm $\mathcal{D}$ which can be viewed as the DKG-counterpart of $\mathcal{B}_1$ in Lemma 3. It constructs a wrapper $\mathcal{C}$ and simulates the oracles in a similar way. However, $\mathcal{C}$ now defines $f_i(Z)$ for every honest signer instead of querying the OMDL oracles to implicitly define $f(Z)$. $\mathcal{C}$ also defines an additional RO[H_{kg}] oracle to handle the hash queries incurred by PedPoP:

RO[H_{kg}](x) : If $\mathsf{H}_{\mathsf{kg}}(x) \neq \perp$, return $\mathsf{H}_{\mathsf{kg}}(x)$. Else, $\mathcal{C}$ sets $\mathsf{ctr}_{pp} \leftarrow \mathsf{ctr}_{pp} + 1$ and $\mathsf{H}_{\mathsf{kg}}(x) \leftarrow h_{\mathsf{ctr}_{pp}}$ then returns $\mathsf{H}_{\mathsf{kg}}(x)$.

Particularly, in $\widetilde{\text{Init}}(CS)$, let honest signers be the set $HS = \{\mathsf{id}_i, \ldots, \mathsf{id}_{\mathsf{ns}}\} \setminus CS$, $\mathcal{C}$ queries $\{A_\ell = g^{a_\ell}\}_{\ell=0}^{t-1} \leftarrow$ Chall. $\mathcal{C}$ selects $\{y_i\}_{\mathsf{id}_i \in HS} \xleftarrow{\$} \mathbb{Z}_p^*$ to implicitly define the random polynomials $f_i(Z) = y_i(\sum_{\ell=0}^{t-1} a_\ell Z^\ell)$ for every honest signer $\mathsf{id}_i \in HS$. Firstly, for every $\mathsf{id}_i \in HS$, $\mathcal{C}$ increments ctr_{pp} and chooses $\bar{z}_i \xleftarrow{\$} \mathbb{Z}_p^*$ to compute $\bar{R}_i = g^{\bar{z}_i}/A_0^{y_i h_{\mathsf{ctr}_{pp}}}$ and sets $\mathsf{H}_{\mathsf{kg}}(A_{i,0} = A_0^{y_i}, A_{i,0}, \bar{R}_i) \leftarrow h_{\mathsf{ctr}_{pp}}$. $\mathcal{C}$ broadcasts $((\bar{R}_i, \bar{z}_i), \mathbf{C} = (A_{i,0} = A_0^{y_i}, \ldots, A_{i,t-1} = A_{t-1}^{y_i}))$ as the proof of possession (PoP). Secondly, $\mathcal{C}$ queries $f'(\mathsf{id}_j) \leftarrow$ Dlog$(\prod_{\ell=0}^{t-1} A_\ell^{\mathsf{id}_j^\ell})$ and sends $x_{i,j} = y_i f'(\mathsf{id}_j)$ on behalf of every honest signer $\mathsf{id}_i \in HS$ to every corrupted signer $\mathsf{id}_j \in CS$. In return, $\mathcal{C}$ receives $x_{j,i} = f_j(\mathsf{id}_i)$ from $\mathsf{id}_j \in CS$ for every $\mathsf{id}_i \in HS$. So far, $\mathcal{C}$ issued t Chall queries and $|CS| < t$ Dlog queries.

As $\mathcal{C}$ simulates the DKG perfectly, all PoP broadcast by HS are verified and every corrupted signer $\mathsf{id}_j \in CS$ obtains their signing key share $x_j =$

$f(\mathsf{id}_j) = \sum_{i=1}^{\mathsf{ns}} f_i(\mathsf{id}_j) = \sum_{i=1}^{\mathsf{ns}} x_{i,j}$ as expected, and the joint public key is $\tilde{X} = g^{a_0 \sum_{\mathsf{id}_i \in HS} y_i + \sum_{\mathsf{id}_j \in CS} a_{j,0}} = \prod_{\mathsf{id}_i \in HS} A_{i,0} \prod_{\mathsf{id}_j \in CS} A_{j,0}$. Furthermore, from $\mathcal{F}$'s view, each signer's state st is indistinguishable from that generated in the CKG setting, so the oracle simulations by $\mathcal{D}$ after the key generation stage are identical to that by $\mathcal{B}_1$ in Lemma 3.

Therefore, from $(i_{sig}, \mathsf{Out}) \leftarrow \mathsf{Fork}^{\mathcal{C}}$, $\mathcal{D}$ can extract the shared signing key $f(0) = a_0 \sum_{\mathsf{id}_i \in HS} y_i + \sum_{\mathsf{id}_j \in CS} a_{j,0}$ and singing key shares $\{f(\mathsf{id}_i)\}_{\mathsf{id}_i \in S_2(lr^*)}$. These are not sufficient for $\mathcal{D}$ to solve its OMDL challenges. Specifically, the OMDL solutions required here are no longer the $f(\cdot)$ coefficients (the OMDL solutions of $\mathcal{B}_1$ in Lemma 3), but the $f'(\cdot)$ coefficients. Moreover, the values $\{a_{j,0} = f_j(0)\}_{\mathsf{id}_j \in CS}$ are not known to $\mathcal{D}$.

Construct helper $\mathcal{D}'$ from $\mathsf{MIMFork}^{\mathcal{D}}_{|CS|,1}$. Denote the output of $\mathcal{D}$ as $(I, \{\mathsf{out}_i\}_{i \in I}, \theta)$ where out_i is the internal state of $\mathcal{D}$ up to the i-th random oracle response such that $I = (i_1, \ldots, i_{|CS|})$ are the set of random oracle indices for $h_i \leftarrow \mathsf{H}_{\mathsf{kg}}(\mathsf{id}_i, A_{i,0}, A_{i,0}, \bar{R}_i)$ during $\widetilde{\text{INIT}}$ and $\theta = f(0)$. We now construct a helper algorithm $\mathcal{D}'$ which is a syntactically valid OMDL adversary and it runs $\mathsf{MIMFork}^{\mathcal{D}}_{|CS|,1}$. $\mathcal{D}'$ generates uniformly random $h_1, \ldots, h_{2q} \in \mathbb{Z}_p^*$ and queries the $2q_s + t$ challenges $(\mathbf{A}, \mathbf{R}, \mathbf{S}) \leftarrow$ CHALL to initiate $\mathsf{MIMFork}^{\mathcal{D}}_{|CS|,1}$. All DLOG queries from $\mathcal{D}$ are relayed to $\mathcal{D}'$.

Therefore, from $(\mathcal{O}, \mathcal{O}') \leftarrow \mathsf{MIMFork}^{\mathcal{D}}_{|CS|,1}$, $\mathcal{D}'$ obtains $f(0)$ from the first run of $\mathcal{D}$, and extracts the $\{f_j(0) = a_{j,0}\}_{\mathsf{id}_j \in CS}$ from corrupted signers' PoPs $((\bar{R}_j, \bar{z}_j = \bar{r}_j + h_j a_{j,0}) \leftarrow \mathcal{O}, (\bar{R}_j, \bar{z}'_j = \bar{r}_j + h'_j a_{j,0}) \leftarrow \mathcal{O}')$. Now, $\mathcal{D}'$ can compute $f'(0) = a_0 = \left(f(0) - \sum_{\mathsf{id}_j \in CS} f_j(0)\right) / \sum_{\mathsf{id}_i \in HS} y_i$. With $|CS| + 1 = t$ values of $(f'(0), \{f'(\mathsf{id}_j)\}_{\mathsf{id}_j \in CS})$, $\mathcal{D}'$ can recover $f'(\cdot)$ and then $\{a_\ell \leftarrow \mathsf{DL}_{\mathbb{G},g}(A_\ell)\}_{\ell=1}^{t-1}$ through Lagrange interpolation. If $|CS| < t-1$, the interpolation still works because $|S_2(lr^*)| < t - |CS|$, and $\mathcal{D}'$ can compute $f'(\mathsf{id}_i) = \left(f(\mathsf{id}_i) - \sum_{\mathsf{id}_j \in CS} f_j(\mathsf{id}_i)\right) / \sum_{\mathsf{id}_i \in HS} y_i$ from $\mathcal{C}$'s internal state for $\mathsf{id}_i \in S_2(lr^*)$. Hence, $\mathcal{D}'$ solved all t challenges $\{a_\ell \leftarrow \mathsf{DL}(A_\ell)\}_{\ell=0}^{t-1}$ used during $\widetilde{\text{INIT}}$ and it can also solve the $2q_s$ challenges $\{r_\ell \leftarrow \mathsf{DL}(R_\ell), s_\ell \leftarrow \mathsf{DL}(S_\ell)\}_{\ell=1}^{q_s}$ used in PPO queries as $\mathcal{B}_1$ done in Lemma 3. Since the BadPPO and hash collision happen with a negligible probability, the success probability of $\mathcal{D}'$ is:

$$\mathsf{acc}(\mathcal{D}') \geq \mathsf{acc}(\mathsf{MIMFork}^{\mathcal{D}}_{|CS|,1}) - \mathsf{negl} \geq \mathsf{acc}(\mathsf{Fork}^{\mathcal{C}})/8 - \mathsf{negl} \geq \Pr[E_1]^2/8q - \mathsf{negl}$$

However, rewinding $\mathcal{D}$ for at most $|I| \cdot T_{max}$ times making $\mathcal{D}'$ issued more than $2q_s$ DLOG queries to answer PSIGNO queries and so $\mathcal{D}'$ does not solve OMDL.

Construct $\mathcal{B}'_1$ from $\mathcal{D}'$. We now construct an OMDL adversary $\mathcal{B}'_1$ which is defined the same as $\mathcal{D}'$ except that $\mathcal{B}'_1$ aborts any executions of $\mathcal{C}$ after it obtains all PoPs $(\bar{R}_j, \bar{z}_j)$ which $\mathcal{F}$ sent during the $\widetilde{\text{INIT}}$ query. This makes sure $\mathcal{D}'$'s internal state recorded the required $(\theta, \{(\bar{R}_j, \bar{z}_j, \bar{z}'_j)\}_{\mathsf{id}_j \in CS})$, yet not issuing any

additional DLOG query during the rewinds as $\mathcal{D}'$ suffered from. Therefore, $\mathcal{B}'_1$ has the same success probability as $\mathcal{D}'$ and it runs in time:

$$\begin{aligned} T(\mathcal{B}'_1) \leq T(\mathcal{D}') &\leq |I| \cdot T_{max} \cdot T(\mathcal{D}) + T(q) \\ &\leq t \cdot T_{max} \cdot 2T(\mathcal{F}) + T(q) \\ &\leq \frac{8t^2 q \ln(8t/\mathsf{acc}(\mathsf{Fork}^{\mathcal{C}}))}{\mathsf{acc}(\mathsf{Fork}^{\mathcal{C}})} \cdot 2T(\mathcal{F}) + T(q) \\ &\leq \frac{128t^2q^2 \ln(64tq/\Pr[E_1]^2)}{\Pr[E_1]^2} \cdot T(\mathcal{F}) + T(q) \end{aligned}$$

such that $\mathsf{Adv}^{\mathsf{omdl}}_{\mathbb{G}}(\mathcal{B}'_1) \geq \Pr[E_1]^2/8q - \mathsf{negl}$.

The $\mathcal{B}'_3$ Adversary. As explained previously, $\mathcal{B}'_3$ simulates the distributed key generation polynomial $f(Z) = f'(Z)\sum_{\mathsf{id}_i \in HS} y_i + \sum_{\mathsf{id}_j \in CS} f_j(Z)$ the same as that of $\mathcal{B}'_1$ and proceeds to simulate the oracles as in Lemma 7. From $\{\mathcal{O}, \mathcal{O}'\} \leftarrow \mathsf{MIMFork}^{\mathcal{C}}_{1,3}$, (the helper algorithm $\mathcal{C}'$ and subsequently) $\mathcal{B}'_3$ can extract $f(\mathsf{id}_{i^*})$ to compute the value $f'(\mathsf{id}_{i^*}) = \left(f(\mathsf{id}_{i^*}) - \sum_{\mathsf{id}_j \in CS} f_j(\mathsf{id}_{i^*})\right) / \sum_{\mathsf{id}_i \in HS} y_i$ where $\{f_j(\mathsf{id}_{i^*})\}_{\mathsf{id}_j \in CS}$ can be obtained from $\mathcal{C}$'s internal state. Without loss of generality, assume $|CS| = t - 1$ and $\mathcal{B}'_3$ knows exactly $|CS| + 1 = t$ point-value pairs $(\mathsf{id}_i, f'(\mathsf{id}_i))$. If $|CS| < t - 1$, $\mathcal{B}_3$ samples a random subset $D \subset S_3(lr^*)$ of size $t - 1 - |CS|$ to compute $f'(\mathsf{id}_i) = x_i \leftarrow \text{DLOG}(\prod_{\ell=0}^{t-1} A_\ell^{\mathsf{id}_i^\ell})$ for each $\mathsf{id}_i \in D$. Using these t point-value pairs, $\mathcal{B}'_3$ can interpolate $f'(\cdot)$ to extract all coefficients $a_0 \leftarrow \mathsf{DL}(A_0), \ldots, a_{t-1} \leftarrow \mathsf{DL}(A_{t-1})$. For the OMDL challenges used as tokens (R_i, S_i) in the game, $\mathcal{B}'_3$ solves them identically to Lemma 7. Consequently, the success probability and running time of $\mathcal{B}'_3$ match those established in Lemma 7. □

5 Performance Evaluations

We benchmark the proposed FROST2+ against ZCash's FROST implementation (GitHub commit `3ffc19d`) [29] on a laptop with an Intel i9-14900HX (2.20 GHz), 32 GB RAM, running Windows 11 Home Single Edition. To support our (and FROST2's) CompPar algorithm, which differs from the original FROST design, we modified the functions `binding_factor_preimages` and `compute_group_commitment` in `frost-core/src/lib.rs`.

We benchmark the related schemes by measuring only their signing throughput. Variants such as ROAST [26] and Olaf [11] build on FROST2 and add robustness mechanisms. When those mechanisms are not invoked, their signing cost is essentially that of FROST2. Similarly, ATS[FROST,DS] is FROST plus an additional (Schnorr) signature per pre-processing token, so it is dominated by the measured FROST cost. We therefore benchmark FROST, FROST2, ms-FROST2 and FROST2+, which represent the main computational differences in Table 2, using the Ed25519 curve across various predefined $(t = 0.7n, n)$ parameter sets. For each configuration, we executed 100 rounds and recorded the

average time per signature generation. Each timing experiment was conducted twice: once with pre-computations enabled and once with them disabled. When pre-computations were enabled, the timings for token verification and the pre-processing round were excluded from the signature generation time; when disabled, they were included. The full numerical results are provided in Table 3 (Appendix C), and Figure 3 visualizes the signatures-per-second for the four schemes. We also present a comparison of the algorithmic complexity which focuses on scalar multiplications, and security of our schemes against FROST and its two-round variants in Table 2. This analysis highlights the core computational differences in scalar operations, with pre-processing contributions denoted in parentheses.

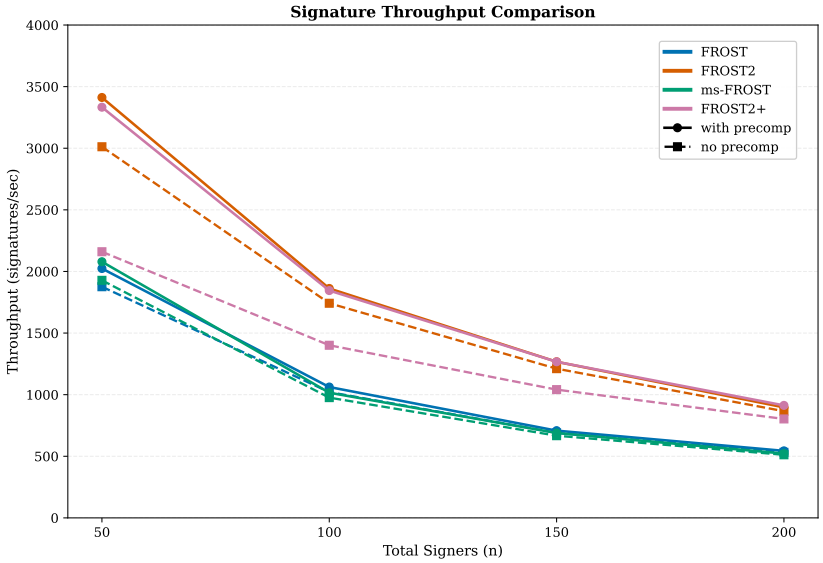


Fig. 3. Signatures generated per seconds for FROST, FROST2, ms-FROST, FROST2+.

When pre-computations are enabled, FROST2+ achieves throughput nearly identical (within -2.3% to 1.8%) to FROST2 across all group sizes, demonstrating that its additional features come with negligible overhead. Besides, FROST2+ is always 64.7% $= \frac{3333-2024}{2024} \times 100$ (at $n = 50$) to 79% $= \frac{1267-708}{708} \times 100$ (at $n = 150$) faster than FROST with pre-computations, confirming the better scalability of the underlying algorithms in FROST2 and FROST2+. Specifically, at $n = 200$ signers, FROST2+ reaches 913 signatures/sec, marginally surpassing FROST2 (897 sig/s) and outperforming FROST (544 sig/s). ms-FROST shows a similar performance to FROST, suggesting that the hashing operation is also an important factor for signing speed.

When pre-computations are disabled, FROST2+ exhibits a noticeable performance degradation compared to FROST2 (e.g., 2160 vs. 3012 sig/s at $n =$

Table 2. Comparison of complexity andsecurity for the proposed FROST+ against FROST and its variants with two signing rounds.

Scheme	Sign					Agg	Vf	Security			
	Round	$\|pp\|$	*smul*	$\mathbb{G}$	$\mathbb{Z}_p$	*smul*	*smul*	Notion	Model	Corrupt	Kg
FROST [22]	2	2	$(2)+t$	2	1	t	2	TS-UF-1	ROM	Static	D
FROST [6]	2	2	$(2)+t$	3	1	0	2	TS-SUF-3	ROM	Static	C
FROST** [14,17]	2	2	$(2)+t$	2	1	t	2	adp-TS-UF	AGM	Adaptive	C
ATS[TS, DS]* [6]	2	4	$(3)+3t$	4	2	0	2	TS-SUF-4	ROM	Static	C
FROST2 [15]	2	2	$(2)+1$	2	1	1	2	TS-UF-1	ROM	Static	D
FROST2 [6]	2	2	$(2)+1$	3	1	0	2	TS-SUF-2	ROM	Static	C
ROAST [26]	2	2	$(2)+1$	2	1	1	2	TS-SUF-2	ROM	Static	C
Olaf [11]	2	2	$(2)+1$	2	1	1	2	TS-SUF-2	ROM	Static	D
Arctic [23]	2	1	$(1)+t^2+t$	1	1	t	2	UF	ROM	Static	C,D
ms-FROST [2]	2	2	$(2)+1$	3	1	0	2	adp-TS-UF-4	AGM	Adaptive	C
FaFROST [4]	2	2	$(2)+1$	3	1	0	2	adp-TS-UF	AGM	Adaptive	C
FROST-Mask [10]	2	2	$(2)+1$	3	1	1	2	adp-UF	ROM	Adaptive	C
FROST2** [17]	2	2	$(2)+1$	3	1	0	2	adp-TS-UF	AGM	Adaptive	C
FROST3** [17]	2	2	$(2)+1$	2	1	1	2	adp-TS-UF	AGM	Adaptive	C
FROST2+	2	4	$(3)+4t+1$	4	2	0	2	TS-SUF-4	ROM	Static	C,D
FROST2+$^{\$}$	2	4	$(3+4t)+1$	4	2	0	2	TS-SUF-4	ROM	Static	C,D

*Assume DS is the Schnorr signature scheme. $^{\$}$with pre-computations.
** Maximum of $t/2$ adaptive corruptions if in ROM.
Note: *smul* : scalar multiplication, $\mathbb{G}, \mathbb{Z}_p$: numbers of group and field element sent per signer, Kg: key generation, C: centralized, D: decentralized, t : threshold, (num) : op. in pre-proc. round

50). This suggests that the optimizations in FROST2+ rely heavily on the pre-computations to achieve competitive throughput, and without them the additional operations from authenticating tokens impose a measurable cost. Nonetheless, even without pre-computations, FROST2+ outperforms FROST for all tested n, showing the efficiency of the lighter CompPar algorithm.

6 Conclusion

We presented an enhanced FROST2 scheme, namely, FROST2+ with proven TS-SUF-4 security under OMDL assumption in the ROM with static corruption and (de)centralized key generation settings. FROST2+ is the only ROM-based TS-SUF-4 scheme to date apart from Bellare et al.'s ATS[FROST,DS] where the primitive FROST used there is at least 64% slower than FROST2+ for a signing group with threshold $t = 0.7n$.

References

1. Arch Network: Arch's signature scheme model (frost + roast). https://docs.arch.network/learn/fundamentals/arch-signature-scheme-model-frost-roast (2024), documentation page, last updated 2024; accessed Nov 18, 2025

2. Bacho, R., Chen, Y., Loss, J., Tessaro, S., Zhu, C.: Adaptively secure partially non-interactive threshold schnorr signatures in the AGM. Cryptology ePrint Archive, Paper 2025/1953 (2025), https://eprint.iacr.org/2025/1953
3. Bacho, R., Das, S., Loss, J., Ren, L.: Adaptively secure three-round threshold schnorr signatures from ddh. In: Advances in Cryptology – CRYPTO 2025: 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2025, Proceedings, Part VI. p. 390–422. Springer-Verlag, Berlin, Heidelberg (2025). https://doi.org/10.1007/978-3-032-01887-8_13, https://doi.org/10.1007/978-3-032-01887-8_13
4. Baecker, R., Gerhart, P., Calsi, D.L., Russo, L., Schröder, D., Yerukhimovich, A.: Fully adaptive FROST in the algebraic group model from falsifiable assumptions. Cryptology ePrint Archive, Paper 2025/1950 (2025), https://eprint.iacr.org/2025/1950
5. Bagherzandi, A., Cheon, J.H., Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In: Proceedings of the 15th ACM Conference on Computer and Communications Security. p. 449–458. CCS '08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1455770.1455827, https://doi.org/10.1145/1455770.1455827
6. Bellare, M., Crites, E., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. pp. 517–550. Springer Nature Switzerland, Cham (2022)
7. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. p. 390–399. CCS '06, Association for Computing Machinery, New York, NY, USA (2006). https://doi.org/10.1145/1180405.1180453, https://doi.org/10.1145/1180405.1180453
8. Bellare, M., Tessaro, S., Zhu, C.: Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Paper 2022/833 (2022), https://eprint.iacr.org/2022/833
9. Brandão, L.T.A.N., Peralta, R.: NIST First Call for Multi-Party Threshold Schemes (NISTIR 8214C, draft). NIST Interagency Report (NISTIR) – Draft 8214C, National Institute of Standards and Technology (NIST) (2023), https://csrc.nist.gov/publications/detail/nistir/8214c/draft
10. Chen, Y.: Round-efficient adaptively secure threshold signatures with rewinding. IACR Communications in Cryptology **2**(2) (2025). https://doi.org/10.62056/ah893z10k
11. Chu, H., Gerhart, P., Ruffing, T., Schröder, D.: Practical schnorr threshold signatures without the algebraic group model. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. pp. 743–773. Springer Nature Switzerland, Cham (2023)
12. Connolly, D., Komlo, C., Goldberg, I., Wood, C.A.: Rfc 9591 the flexible round-optimized schnorr threshold (frost) protocol for two-round schnorr signatures (2024)
13. Cremers, C., Peltonen, A., Zhao, M.: An extended hierarchy of security notions for threshold signature schemes and automated analysis of protocols that use them. Cryptology ePrint Archive, Paper 2024/1920 (2024), https://eprint.iacr.org/2024/1920
14. Crites, E., Katz, J., Komlo, C., Tessaro, S., Zhu, C.: On the adaptive security of frost. In: Tauman Kalai, Y., Kamara, S.F. (eds.) Advances in Cryptology – CRYPTO 2025. pp. 480–511. Springer Nature Switzerland, Cham (2025)

15. Crites, E., Komlo, C., Maller, M.: How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Paper 2021/1375 (2021), https://eprint.iacr.org/2021/1375
16. Crites, E., Komlo, C., Maller, M.: Fully adaptive schnorr threshold signatures. In: Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I. p. 678–709. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-38557-5_22, https://doi.org/10.1007/978-3-031-38557-5_22
17. Crites, E., Komlo, C., Maller, M.: On the adaptive security of key-unique threshold signatures. Cryptology ePrint Archive, Paper 2025/943 (2025), https://eprint.iacr.org/2025/943
18. Crites, E., Stewart, A.: A plausible attack on the adaptive security of threshold schnorr signatures. In: Tauman Kalai, Y., Kamara, S.F. (eds.) Advances in Cryptology – CRYPTO 2025. pp. 457–479. Springer Nature Switzerland, Cham (2025)
19. Gennaro, R., Leigh, D., Sundaram, R., Yerazunis, W.: Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In: Lee, P.J. (ed.) Advances in Cryptology – ASIACRYPT 2004 (Lecture Notes in Computer Science, vol. 3329). Springer, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30539-2_20
20. Harman, S., Shishmarev, M., Nash, T., Lennon, D., Janse van Vuuren, S.: Chainflip protocol whitepaper. Tech. rep., Auxilio Ventures (2022), https://www.auxilioventures.com/wp-content/uploads/2022/08/Chainflip-Whitepaper-May-2020.pdf
21. Kiltz, E., Masny, D., Pan, J.: Optimal security proofs for signatures from identification schemes. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. pp. 33–61. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
22. Komlo, C., Goldberg, I.: Frost: Flexible round-optimized schnorr threshold signatures. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) Selected Areas in Cryptography. pp. 34–65. Springer International Publishing, Cham (2021)
23. Komlo, C., Goldberg, I.: Arctic: Lightweight and stateless threshold schnorr signatures. In: Jager, T., Pan, J. (eds.) Public-Key Cryptography – PKC 2025. pp. 234–267. Springer Nature Switzerland, Cham (2025)
24. Mccorry, P., Bakshi, S., Dembo, A., Miller, A.: sbtc: Design of a trustless two-way peg for bitcoin. Tech. rep., Trustless Computing Foundation (2023), https://sbtc.tech/sbtc.pdf
25. Niot, G., Reichle, M., Takemure, K.: Adaptively-secure three-round threshold schnorr from DL. Cryptology ePrint Archive, Paper 2025/1941 (2025), https://eprint.iacr.org/2025/1941
26. Ruffing, T., Ronge, V., Jin, E., Schneider-Bensch, J., Schröder, D.: Roast: Robust asynchronous schnorr threshold signatures. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 2551–2564. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3548606.3560583, https://doi.org/10.1145/3548606.3560583
27. Safe Research: safe-frost. https://github.com/safe-research/safe-frost (2024), gitHub repository; accessed Nov 18, 2025
28. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**, 161–174 (1991). https://doi.org/10.1007/BF00196725
29. Zcash Foundation: Speeding up frost with multi-scalar multiplication, https://zfnd.org/speeding-up-frost-with-multi-scalar-multiplication/

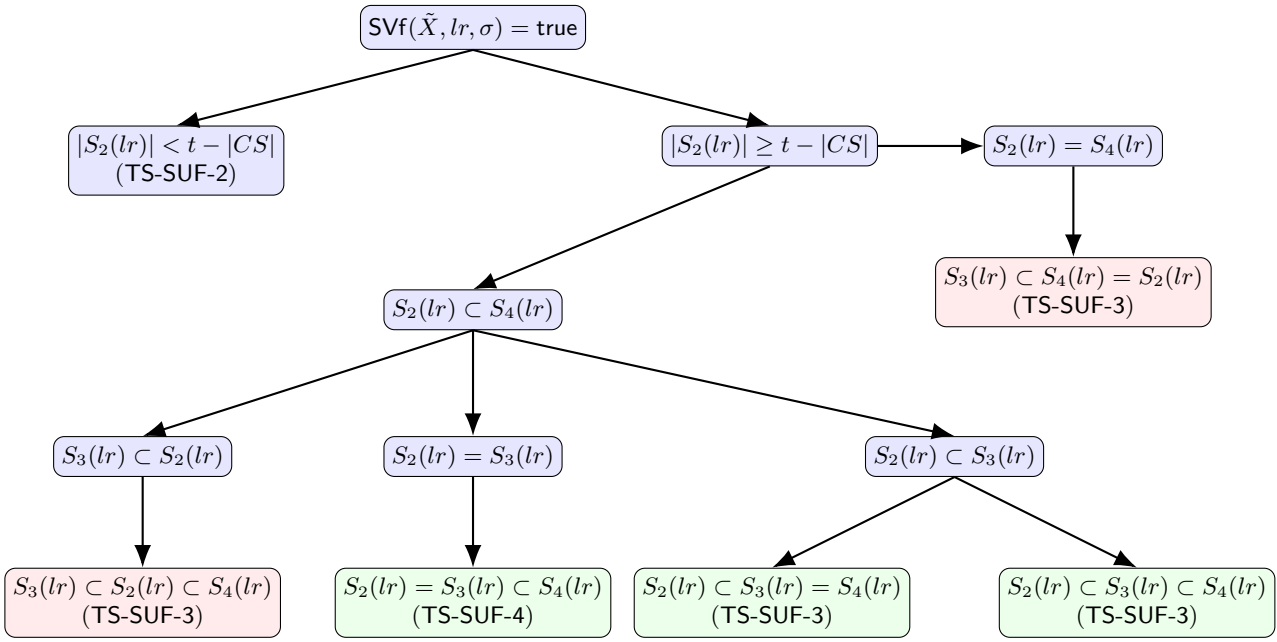


Fig. 4. Decision tree for queries comparisons and forgery outcomes. ▭ marks the algorithmically prohibited forgery types (i.e., $\mathsf{st.mapPP}(pp_i) \neq \perp$ enforced in PS) while ▭ marks the possible forgery types.

A Attacks on FROST2

A.1 Types of Forgeries

Referring to the security model in Section 2.4, we recall the conditions under which a (strong) forgery can be achieved in a threshold signature scheme that is also an echo scheme [8,6] (e.g., FROST2) in Figure 4.

A.2 TS-UF-3 Attack

In Bellare et al.'s work [8,6], they provide a security proof of TS-SUF-2 and also a TS-UF-3 attack for FROST2. This indicates that the best possible security level FROST2 can achieve is only TS-SUF-2. We recall Bellare et al.'s TS-UF-3 attack in the setting below:

- Total signers are $\{P_1, P_2, P_3, P_4\}$.
- Signers $\{P_3, P_4\}$ corrupted.
- A signing request on signer set $lr.\mathcal{S} = \{1, 2, 3\}$ for a message $lr.\mathsf{msg} = M$.

- The pre-processing token $(R_1, S_1), (R_2, S_2)$ are known to the corrupted signers before they generate theirs.
- First signer's partial signature $z_1 = r_1 + ds_1 + c\Delta_{1,\mathcal{S}}(0)x_1$ can be obtained.
- Expected combined signature is $z = \mathsf{dlog}(\tilde{R}) + cx = \sum_{i \in \mathcal{S}} r_i + ds_i + c\Delta_{i,\mathcal{S}}(0)x_i$

The attack procedure is as follows:

1. P_3 generates its pre-processing token as

$$R_3 = R_1^{\frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)}-1} R_2^{-1}, S_3 = S_1^{\frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)}-1} S_2^{-1}.$$

   where $\mathcal{S}' = \{1, 3, 4\}$. The resulting group pre-processing token will be:

$$\tilde{R} = R_1^{\frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)}} S_1^{d\frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)}} = \prod_{i \in \mathcal{S}} R_i S_i^d$$

   where $\mathsf{dlog}(\tilde{R}) = r_1 \frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)} + ds_1 \frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)}$ and $d = \mathsf{H}_{\mathsf{non}}(\tilde{X}, lr)$.
2. P_3 computes $z_1' = r_1 \frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)} + ds_1 \frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)} + c\Delta_{1,\mathcal{S}'}(0)x_1 = \frac{\Delta_{1,\mathcal{S}'}(0)}{\Delta_{1,\mathcal{S}}(0)} z_1$
3. P_3 and P_4 forge a valid combined signature for M under $lr.\mathcal{S}$ as

$$z' = \mathsf{dlog}(\tilde{R}) + cx = z_1' + c\Delta_{3,\mathcal{S}'}(0)x_3 + c\Delta_{4,\mathcal{S}'}(0)x_4.$$

This is a valid TS-UF-3 attack because $S_2 = \{1\}$ and unequal to $S_3 = \{1, 2\}$, i.e., $S_2(lr) \subset S_3(lr) = S_4(lr)$ in Figure 4. We note that this attack does not work on our schemes because the attacker's pre-processing token (R_3, S_3) is a combination of the honest signers' pre-processing token whose discrete logarithms are not known. Consequently, a valid authentication transcript (u, V) for (R_3, S_3) cannot be generated and the honest signers abort during PSIGNO.

A.3 TS-UF-4 Attack

In Bellare et al.'s work [8,6], they demonstrate a TS-UF-4 attack for FROST but not FROST2. Here, we mount the attack on FROST2 as follows:

- Total signers are $\{P_5, P_{10}, P_{11}, P_{15}, P_{20}\}$.
- Signers $\{P_5, P_{10}\}$ corrupted.
- A signing request on signer set $lr.\mathcal{S} = \{11, 15, 20\}$ for a message $lr.\mathsf{msg} = M$.
- The pre-processing token $lr.\mathsf{PP} = \{(R_{11}, S_{11}), (R_{15} = g^{r_{15}}, S_{15} = g^{s_{15}}), (R_{20} = g^{r_{20}}, S_{20} = g^{s_{20}})\}$) and $r_{15}, s_{15}, r_{20}, s_{20} \in \mathbb{Z}_p^*$ are known to the corrupted signers before they generate theirs.
- First signer's partial signature $z_{11} = r_{11} + d_{11}s_{11} + c\Delta_{11,\mathcal{S}}(0)x_{11}$ can be obtained.
- Expected combined signature is $z = \mathsf{dlog}(\tilde{R}) + cx = \sum_{i \in \mathcal{S}} r_i + ds_i + c\Delta_{i,\mathcal{S}}(0)x_i$

The attack procedure is as follows:

1. Let $\mathcal{S}' = \{5, 10, 11\}$, the Lagrange coefficients are the same for:

$$\Delta_{11,lr.\mathcal{S}} = \frac{15 \cdot 20}{(15-11)(20-11)} = \frac{25}{3} = \frac{5 \cdot 10}{(5-11)(10-11)} = \Delta_{11,\mathcal{S}'}.$$

   The resulting group pre-processing token will be:

$$\tilde{R} = R_{11} S_{11}^{d} g^{r_{15}+ds_{15}+r_{20}+ds_{20}} = \prod_{i \in \mathcal{S}} R_i S_i^{d}$$

   where $\mathsf{dlog}(\tilde{R}) = r_{11} + ds_{11} + r_{15} + ds_{15} + r_{20} + ds_{20}$ and $d = \mathsf{H}_{\mathsf{non}}(\tilde{X}, lr)$.
2. P_5 and P_{10} forge a valid combined signature for M under $lr.\mathcal{S}$ as

$$z' = \mathsf{dlog}(\tilde{R}) + cx = z_{11} + r_{15} + ds_{15} + r_{20} + ds_{20} + c\Delta_{5,\mathcal{S}'}(0)x_5 + c\Delta_{10,\mathcal{S}'}(0)x_{10}.$$

This is a valid TS-UF-4 attack because $S_2 = \{11\} = S_3$ but unequal to $S_4 = \{11, 15, 20\}$, i.e., $S_2(lr) = S_3(lr) \subset S_4(lr)$ in Figure 4. We note that this attack does not work on our schemes for the similar reason in Section A.2. Although the attacker knows the discrete logarithms for the pre-processing tokens $(R_{15}, S_{15}), (R_{20}, S_{20})$, it does not know the honest signers' signing key shares which are required to generate the authentication transcripts $(u_{15}, V_{15}), (u_{20}, V_{20})$ for the tokens. Therefore, the honest signer will abort during partial signing because of invalid $(u_{15}, V_{15}), (u_{20}, V_{20})$.

B PedPoP

We recall PedPoP key generation [22,6] in Figure 5.

C Detailed Benchmark Results

The detailed benchmark data is in Table 3.

PedPop.Kg(par, t, ns)

1. Every signer id_i chooses a polynomial $f_i(Z) \xleftarrow{\$} \mathbb{Z}_p^{t-1}$:

$$f_i(Z) = a_{i,0} + a_{i,1}Z + \cdots + a_{i,t-1}Z^{t-1}$$

and computes $A_{i,k} = g^{a_{i,k}}$ for $k = 0, \ldots, t-1$. Let $x_i = a_{i,0}$ and $X_{i,0} = A_{i,0}$. Every id_i selects $\bar{r}_i \xleftarrow{\$} \mathbb{Z}_p$ and sets $\bar{R}_i \leftarrow g^{\bar{r}_i}$ to compute $\bar{z}_i \leftarrow \bar{r}_i + h_i x_i$ where $h_i \leftarrow \mathsf{H}_{\mathsf{kg}}(\mathsf{id}_i, X_{i,0}, X_{i,0}, \bar{R}_i)$. They broadcast $((\bar{R}_i, \bar{z}_i), \mathbf{C}_i)$ where the $\mathbf{C}_i = (A_{i,0}, \ldots, A_{i,t-1})$ is a commitment vector.
2. Given $((\bar{R}_j, \bar{z}_j), \mathbf{C}_j)$ from each other signer id_j, every signer computes:

$$h_j \leftarrow \mathsf{H}_{\mathsf{kg}}(\mathsf{id}_j, A_{j,0}, A_{j,0}, \bar{R}_j)$$

and verifies whether:

$$\bar{R}_j A_{j,0}^{h_j} = g^{\bar{z}_j}.$$

If any checks fail, they disqualify the corresponding signer and continue to the next step.
3. Every id_i computes secret shares $\bar{x}_{i,j} = f_i(\mathsf{id}_j)$ and privately sends $\bar{x}_{i,j}$ to signer id_j for $j = 1, \ldots, \mathsf{ns}$.
4. Each signer id_j verifies the shares they received from other signers by checking:

$$g^{\bar{x}_{i,j}} = \prod_{k=0}^{t-1} A_{i,k}^{\mathsf{id}_j^k}$$

If the check fails for some index i, id_j broadcasts a complaint against id_i.
5. For each complaint by signer id_j against id_i, id_i broadcasts the share $\bar{x}_{i,j}$. If any revealed share fails to satisfy the verification equation, or if id_i does not broadcast a share for a complaining signer, then id_i is disqualified and their shares are set to 0.
6. The secret share for every id_j is then:

$$\bar{x}_j = \sum_{i=1}^{\mathsf{ns}} \bar{x}_{i,j}.$$

7. If $X_{i,0} = X_{j,0}$ for any $i \neq j$, then abort. Else, output the joint public key as:

$$\tilde{X} = \prod_{i=1}^{\mathsf{ns}} X_{i,0}$$

Fig. 5. The PedPoP algorithm [22,6].

Table 3. Performance comparison of threshold signature schemes (excluding DKG setup). (t, n) denotes the threshold and number of signers.

Scheme	**Pre-comp**	(t, n)	**Signature Gen.** (ms)	**Throughput** (sig/sec)	**Security**
FROST	✓	$(35, 50)$	0.494	2,024	TS-SUF-3
		$(70, 100)$	0.941	1,062	
		$(105, 150)$	1.412	708	
		$(140, 200)$	1.838	544	
	✗	$(35, 50)$	0.533	1,876	
		$(70, 100)$	0.979	1,021	
		$(105, 150)$	1.449	690	
		$(140, 200)$	1.879	532	
FROST2	✓	$(35, 50)$	0.293	3,412	TS-SUF-2
		$(70, 100)$	0.537	1,862	
		$(105, 150)$	0.789	1,267	
		$(140, 200)$	1.115	897	
	✗	$(35, 50)$	0.332	3,012	
		$(70, 100)$	0.574	1,742	
		$(105, 150)$	0.826	1,211	
		$(140, 200)$	1.155	866	
ms-FROST	✓	(35,50)	0.481	2079	adp-TS-UF-4
		(70,100)	0.984	1016	
		(105,150)	1.458	689	
		(140,200)	1.914	522	
	✗	(35,50)	0.519	1928	
		(70,100)	1.025	976	
		(105,150)	1.498	667	
		(140,200)	1.954	512	
FROST2+	✓	$(35, 50)$	0.300	3,333	TS-SUF-4
		$(70, 100)$	0.542	1,845	
		$(105, 150)$	0.789	1,267	
		$(140, 200)$	1.095	913	
	✗	$(35, 50)$	0.463	2,160	
		$(70, 100)$	0.714	1,401	
		$(105, 150)$	0.961	1,041	
		$(140, 200)$	1.245	803	