An Extended Hierarchy of Security Notions for Threshold Signature Schemes and Automated Analysis of Protocols That Use Them

Cas Cremers[1], Aleksi Peltonen[1], and Mang Zhao[2]

[1]CISPA Helmholtz Center for Information Security, {cremers,aleksi.peltonen}@cispa.de
[2]Independent Researcher, mang.zhao@hotmail.com

Version 1.0
November 26, 2024

Abstract

Despite decades of work on threshold signature schemes, there is still limited agreement on their desired properties and threat models. In this work we significantly extend and repair previous work to give a unified syntax for threshold signature schemes and a new hierarchy of security notions for them. Moreover, our new hierarchy allows us to develop an automated analysis approach for protocols that use threshold signatures, which can discover attacks on protocols that exploit the details of the security notion offered by the used scheme, which can help choose the correct security notion (and scheme that fulfills it) that is required for a specific protocol.

Unlike prior work, our syntax for threshold signatures covers both non-interactive and interactive signature schemes with any number of key generation and signing rounds, and our hierarchy of security notions additionally includes elements such as various types of corruption and malicious key generation. We show the applicability of our hierarchy by selecting representative threshold signature schemes from the literature, extracting their core security features, and categorizing them according to our hierarchy. As a side effect of our work, we show through a counterexample that a previous attempt at building a unified hierarchy of unforgeability notions does not meet its claimed ordering, and show how to repair it without further restricting the scope of the definitions.

Based on our syntax and hierarchy, we develop the first systematic, automated analysis method for higher-level protocols that use threshold signatures. We use a symbolic analysis framework to abstractly model threshold signature schemes that meet security notions in our hierarchy, and implement this in the Tamarin prover. Given a higher-level protocol that uses threshold signatures, and a security notion from our hierarchy, our automated approach can find attacks on such protocols that exploit the subtle differences between elements of our hierarchy. Our approach can be used to formally analyze the security implications of implementing different threshold signature schemes in higher-level protocols.

1 Introduction

A threshold signature scheme allows a group of signers to collaboratively sign messages in a secure and decentralized manner, even if some of them are unavailable or have been compromised. Specifically, a *t-out-of-n* scheme requires that at least t signers collaborate to produce a signature for a group with n members. Each group member can sign messages using their private key pair and once at least t partial signatures have been collected, they can be combined into a group signature. The resulting signature verifies under the group's shared public key, regardless of which subset of signers participated in creating it.

The concept of threshold signatures was first proposed in the late 1980s [29] for distributed key generation and signing. Recently, the idea has re-gained attention due to its potential applications in modern blockchain and cryptocurrency systems [53, 54]. As a result, many new threshold signature schemes have been introduced [5, 6, 16, 23, 27, 35, 36, 37, 42, 44, 48] and various advanced security properties have been proposed [2, 4, 11, 13, 14, 23, 24, 25, 32, 42, 45, 48].

The conventional security requirement for signature schemes is existential unforgeability against chosen message attacks (EUF-CMA), which informally states that an adversary cannot forge valid message-signature pairs without knowing the corresponding secret keys. In the context of threshold signatures, this means that, for any message, a subset of less than t signers should not be able to create a group signature that pass verification under their shared public key. The exact definition is, however, subject to debate, and various models with subtle differences have been proposed in the literature.

For example, while many definitions [1, 4, 5, 6, 13, 16, 23, 24, 36, 42, 48] only consider unforgeability for messages, recent work [11] extends the range to so-called *leader requests*, which also include additional associated data. Furthermore, different adversary models give the adversary varying compromise capabilities, such as malicious key generation [4, 5, 24, 35, 42], and static [11, 13, 25, 36, 37, 44, 48] or adaptive [2, 6, 16, 23, 27] signer corruption. These discrepancies between definitions mean that two schemes that meet the unforgeability requirements might not provide the same security guarantees in practice. Moreover, variations in the syntax definitions for threshold signatures have resulted in many proposed security models being incompatible with schemes that implement different features.

Our first goal is to address the issues caused by conflicting definitions and security notions. To this end, we develop a generalized syntax and new hierarchy of security notions for threshold signatures, which goes beyond prior work in both terms of scope (we cover non-interactive as well as interactive schemes with any number of rounds for key derivation and signing) and adversarial capabilities (we include several types of corruption and malicious key generation).

As part of constructing our syntax and hierarchy, we revisit a prior unforgeability hierarchy proposed by Bellare, Tessaro, and Zhu [10, 11] and show through a counterexample that the hierarchy incorrectly claims implied relations between two of their levels, and propose an updated version that solves the issue. Moreover, we show that their protocol transformation is better than advertised.

Our second goal is to enable automated analysis of higher-level protocols that use threshold signatures as a building block. Based on our new hierarchy, we develop the first automated analysis method for protocols deploying threshold signatures. In this part of our work, we use symbolic analysis methods, modeling a family of threshold signing protocols in the framework of the Tamarin prover. By modeling higher-level protocols in Tamarin together with our threshold signature models, our approach can be used to systematically find attacks that exploit the subtle differences in security properties, and to offer guidance in the selection of signature schemes.

Our methodology can also be used "in reverse": Given a higher-level protocol that uses threshold signatures, what exact guarantees should the threshold signature provide? This was also identified by Bellare, Tessaro, and Zhu in [10, 11] as an open problem: *"We stress that it is not clear which scenarios demand which notions in our hierarchy. This is especially true because we are still lacking formal analyses of full-fledged systems using threshold signatures, but it is not hard to envision a potential mismatch between natural expectations from such schemes and what they actually achieve."* Specifically, we envision our framework to be useful for protocol designers to evaluate the suitability of different threshold signature schemes for their work.

Contributions. Our main contributions are the following:

1. We define a unified syntax for threshold signature schemes that covers both non-interactive and interactive constructions, with any number of key generation and signing rounds. We identify differences between proposed syntax definitions and generalize their core functionalities, which allows us to define and categorize all different scheme types in one framework.
2. We propose a hierarchy of unforgeability notions for the unified syntax that extends and improves previous work. We identify and correct inaccurate security guarantees in prior attempts at building a unified hierarchy, and extend them with additional attributes that capture subtle differences in the security requirements of different scheme types, and revisit the guarantees proven for existing schemes in the context of our hierarchy.
3. We develop the first systematic, automated methodology for analyzing protocols that implement threshold signature schemes. Our framework captures the unforgeability properties of our new hierarchy and can be used to analyze the implications of using schemes that meet different security notions from our hierarchy in higher-level protocols. We implement our models for a state-of-the-art verification tool, the Tamarin prover.

Outline. First, we provide background and relevant definitions from related work in Section 2. In Section 3, we define a unified syntax for threshold signature schemes. In Section 4, we propose an extended hierarchy of unforgeability notions. In Section 5, we compare our hierarchy with previous work [10, 11]. In Section 6, we develop a formal analysis framework for automatic verification of protocols that implement threshold signatures. Finally, we conclude in Section 7. We provide additional preliminaries and the proofs to all theorems in this work in the supplementary material.

2 Background and Related Work

2.1 Unforgeability Definitions for Threshold Signatures

Since the inception of threshold signatures in the late 1980s [29], a great number of signature constructions have been proposed [5, 6, 16, 23, 27, 35, 36, 37, 42, 44, 48], most providing slightly different security guarantees [2, 4, 11, 13, 14, 23, 24, 25, 32, 42, 45, 48]. One of the most commonly adapted security definitions is the conventional notion of *existential unforgeability (against chosen message attacks)*. While almost all threshold signature designs are proven to achieve some form of existential unforgeability, the exact security guarantees that they achieve are often distinct. The main differences occur in three aspects: syntax, adversary capabilities, and assumptions on communication channels, which we address in turn below.

Syntax. Subtle variations in the syntax definitions of threshold signatures are a common reason for incompatible security models. For example, [6, 24] define a syntax and associated security model for threshold signatures with three message rounds in the signing phase. This directly excludes the possibility of applying their model to schemes with more than three rounds. Moreover, while many existing threshold signature definitions [4, 13] only consider two action phases, namely key generation and signing, modern so-called *echo schemes*, such as FROST2 [24], achieve non-interactive signing by distributing pre-processing tokens in a new pre-processing phase. Models that do not involve this additional phase, cannot cover the constructions that require it.

To solve this problem, Bellare, Tessaro, and Zhu [10, 11] propose a generic syntax definition and corresponding hierarchy of unforgeability levels. However, their definition is restricted to non-interactive threshold signature schemes and, consequently, excludes all interactive schemes, such as [16, 24, 36].

Adversary Capabilities. A crucial difference between threshold signatures and other stateless authentication protocols, such as digital signatures and message authentication codes, is that they are multi-party protocols that often require multi-round group actions. Consequently, the corruption of some signers at a given timepoint might impact the final security of the whole group. While the security goal in various unforgeability models might be similar, i.e., preventing adversaries from forging a signature for some message, the adversaries' capabilities considered in different models differ vastly, making the models themselves incompatible. Different adversary models consider e.g., static [11, 13, 25, 36, 37, 44, 48] or adaptive corruption [2, 6, 16, 23, 27], and honest or malicious key generation [4, 5, 24, 35, 42].

Assumptions on Communication Channels. The security of the peer-to-peer communication channels in threshold signature schemes have a crucial impact on the adversaries' knowledge of the transmitted messages and, therefore, the final security guarantees the protocol can provide. Depending on the security assumptions on these channels, the adversary model changes drastically. Specifically, some models require a secure channel for key generation, but not for message signing [6, 11, 13, 23, 24, 36, 42, 48], while others require authenticated [4, 5, 16] communication channels for all messages.

One of our goals is to provide a unifying framework for the preceding notions.

2.2 Other Security Notions

In addition to existential unforgeability, Bellare, Tessaro, and Zhu [10, 11] also propose a *strong unforgeability* notion for non-interactive threshold signatures. Informally, this notion prevents adversaries from forging new, "unseen" signatures. We briefly recall it in Section 2.3 and expand on the details in Section 4.2.

Besides unforgeability, several other security notions have also been proposed in the literature. These include, for example, robustness [38, 48], identified aborts [48], proactivity [13], accountability or traceability [14, 45], privacy [14], private accountability [14], and blindness [25]. Each of these notions provide certain security guarantees in some specific application scenarios. Since this work only focuses on the generalization of the conventional unforgeability hierarchy, the discussion about other security notions is considered out of scope.

2.3 Unforgeability Notions for Non-Interactive Threshold Signatures

Bellare, Tessaro, and Zhu [10, 11] propose the first generic syntax and unforgeability hierarchy for non-interactive threshold signatures. Below, we define notions and recall definitions that are relevant for our extended hierarchy.

Notation. We assume that all algorithms defined in this paper are parameterized implicitly by the security parameter. We write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. We write $\perp$ to denote a special error symbol that is not included in any set or list (unless specified), $*$ to denote a variable that is irrelevant, and $\mathbf{0}$ to denote an empty string. Let $x \xleftarrow{\$} Z$ denote sampling a variable x uniformly at random from a set Z and let $x \xleftarrow{\$} X(y)$ denote the execution of a probabilistic algorithm X with an input y followed by assigning the output to a variable x. We write $x \leftarrow X(y)$ if the algorithm X is deterministic. Let $\mathcal{L} \xleftarrow{+} x$ denote adding x into an (unordered) list or set $\mathcal{L}$ ($\mathcal{L} \leftarrow \mathcal{L} \cup \{x\}$) and let $\mathcal{L} \xleftarrow{-} x$ denote removing x from a set or list $\mathcal{L}$ ($\mathcal{L} \leftarrow \mathcal{L} \setminus \{x\}$). For a dictionary $\mathcal{D}$, let $\mathcal{D}[*] \leftarrow x$ denote initializing all elements to be x. We use the keyword **req** to indicate that a following condition must be satisfied and the keyword **parse** to indicate that a following expression must hold, and otherwise undo all executions in the current algorithm. We use $\|$ to denote string concatenation (e.g., $x_1 \| x_2$ denotes the concatenation of two strings x_1 and x_2) and $\|_{i \in I} x_i$ to denote the concatenation of strings x_i for all $i \in I$, ordered by i from the smallest to the largest. We write $x \xleftarrow{\|} y$ for $x \leftarrow x \| y$. We use $[\![exp]\!]$ to denote the evaluation of the expression exp that outputs 1 if exp is true and 0 otherwise.

Leader Requests and States. A threshold signature scheme lets a group of n stateful signers, identified by $i \in [n]$, collaboratively sign leader requests lr issued by a stateful leader, identified by 0.

Definition 1 (Leader Request [10, 11]). *The leader request lr is a collection of variables that include:*

- $lr.m \in \{0,1\}^\star$*: a message m to be signed.*
- $lr.SS \subseteq [n]$*: a set of signers SS that are expected to sign this request.*
- $lr.\mathsf{PP} : lr.SS \to \{0,1\}^*$*: an optional function that specifies pre-processing tokens for every signer $i \in lr.SS$; initialized with $\perp$.*

Note that [11, Section 3.1] defines that "leader request is mandated to specify a message $lr.\mathsf{msg}$ and a set $lr.\mathsf{SS} \subseteq [1..\mathsf{ns}]$ of servers from whom partial signatures are being requested" and for echo schemes "additionally specifies a function $lr.\mathsf{PP} : lr.\mathsf{SS} \to \{0,1\}^\star$". Here, we merge the definitions and assume $lr.\mathsf{PP} = \perp$ for non-echo schemes.

Definition 2 (Echo Scheme [10, 11]). *We say a threshold signature scheme is an* echo *scheme if leader requests lr specify the function $lr.\mathsf{PP} \neq \perp$.*

Each signer i is associated with a per-signer long-term state st_i.

Definition 3 (Signer States [10, 11]). *The states of every signer S_i are a collection of variables that include:*

- $\mathsf{st}_i.id = i$*: the identifier of the signer S_i.*
- $\mathsf{st}_i.sk$*: the signing key of the signer S_i; initialized with $\perp$.*
- $\mathsf{st}_i.VK$*: a dictionary that includes all signers' verification keys; initialized with $\mathsf{st}_i.VK[*] \leftarrow \perp$. (named $\mathsf{st}_i.\mathsf{aux}$ in [10, 11].)*
- $\mathsf{st}_i.gvk$*: the group verification key; initially $\perp$. (named $\mathsf{st}_i.vk$ in [10, 11].)*

Similarly, the stateful leader holds the long-term state st_0 for storing public information from signers. In contrast to signer's private long-term state, the leader's long-term state is public, as it only includes the signers' public information.

Definition 4 (Leader State [10, 11])**.** *The states of the leader L are a collection of variables that include:*

- $\mathsf{st}_0.VK$*: a dictionary that includes all signers' verification keys; initialized with* $\mathsf{st}_0.VK[*] \leftarrow \perp$.
- $\mathsf{st}_0.gvk$*: the group verification key; initialized with* $\perp$.
- $\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}$*: the dictionary that includes all signers' pre-processing tokens; initialized with* $\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[*] \leftarrow \emptyset$. *(Cf. "the leader updates its state* st_0 *to incorporate token pp" [10, 11, Section 3.1].)*

Unforgeability Hierarchy. The conventional definition of unforgeability prevents an adversary from forging a signature $\sigma^\star$ for a message $m^\star$, unless $(m^\star, \sigma^\star)$ is a *trivial forgery*. Bellare, Tessaro, and Zhu [10, 11] provide a unified classification for trivial forgeries, divided into five levels of unforgeability (EUF-CMA), denoted TS-UF-{0,1,2,3,4}, where TS-UF-3 is only defined for echo schemes. Intuitively, a message-signature pair $(m^\star, \sigma^\star)$ is considered as trivial forgery if:

- TS-UF-0: A partial signature for the message $m^\star$ was generated by at least one honest signer.
- TS-UF-1: A partial signature for the message $m^\star$ was generated by at least $t - c$ honest signers, where c is the number of corrupted signers.
- TS-UF-2: There exists a leader request lr for the message $m^\star$ which was answered by at least $t - c$ honest signers.
- TS-UF-3: There exists a leader request lr for the message $m^\star$ such that every honest signer $i \in lr.SS$ if and only if i either answered lr or the token pp_i associated with i in lr is maliciously generated.
- TS-UF-4: There exists a leader request lr for the message $m^\star$ such that every honest signer $i \in lr.\mathsf{SS}$ answered lr.

The definitions of these notions allow for static (but not adaptive) corruption, and assume secure communication channels among signers during the key generation phase. Bellare, Tessaro, and Zhu [10, 11] claim that these five levels are of increasing strength. However, in Section 5 we will show that TS-UF-4 security does not generally imply TS-UF-3 security for echo schemes. Our hierarchy will repair this implication, notably by strengthening the highest level.

While numerous existential unforgeability models have been proposed for threshold signatures in the literature, studies of strong unforgeability are very limited. To the best of our knowledge, the first and only definition for strong unforgeability (SUF-CMA) for threshold signatures is provided by Bellare, Tessaro, and Zhu [10, 11]. Informally, strong unforgeability prevents adversaries from forging a new "unseen" signature $\sigma^\star$ for a message $m^\star$. Similarly to existential unforgeability, their strong unforgeability definition is restricted to the non-interactive threshold signature and is classified into three levels: TS-SUF-{2,3,4}. Using a similar argument as for the existential unforgeability case, TS-SUF-4 does not imply TS-SUF-3 security. In Section 4.2 we propose our own strong unforgeability hierarchy for generalized threshold signatures.

We will return to the exact relation between the unforgeability part of our hierarchy and [10, 11] in Section 5.

2.4 Symbolic Modeling of Cryptographic Primitives

Traditionally, symbolic analysis of security protocols only included extremely coarse approximations of cryptographic primitives. For example, there existed only one symbolic model for any type of symmetric encryption. Recent works have shown that it is possible to develop families of symbolic models for some cryptographic primitives, including hashes [18], signatures [41], Diffie-Hellman groups [21], authenticated encryption [19], and key encapsulation mechanisms [20]. This has resulted in more precise attack-finding models capable of capturing subtle security flaws that have previously been undetectable by symbolic analysis. The symbolic analysis methodology for threshold signatures that we develop in this work is in the same spirit.

3 Unified Syntax for Threshold Signatures

In this section, we propose a unified syntax for threshold signature schemes. Our syntax reuses the definitions of leader requests (see Definition 1) and leader states (see Definition 4) from [10, 11], and extends the definition of signer states (see Definition 3) for interactive schemes. Specifically, the extended signer state definition includes variables for interactive signing in the long-term states and defines additional session states π_i^j for the j-th (sequential or parallel) interactive signing session. We use the abbreviations $S_i = \{\mathsf{st}_i\} \cup \{\pi_i^j\}_j$ and $S_i^j = \{\mathsf{st}_i\} \cup \{\pi_i^j\}$ to simplify presentation.

Definition 5 (Extended Signer States)**.** *Extending Definition 3, the state S_i of every signer i includes the following additional variables:*

- $\mathsf{st}_i.\mathsf{rnd} \in \{0, \ldots, u+1\}$*: the next round that the signer i will process in the key generation phase; initialized with* 0.
- $\pi_i^j.id = j$*: the session identifier of the signer session state π_i^j.*
- $\pi_i^j.\mathsf{rnd} \in \{0, \ldots, w+1\}$*: the next round that the signer i will process in the signing phase; initialized with* 0.

In Definition 6, we define (t, n)-threshold signature schemes with (u, v, w) rounds. Here, t and n respectively denote the threshold and the number of total signers, $u \geq 0$ and $w \geq 0$ respectively denote the communication rounds among signers during the key generation and signing phases, and $v \in \{0, 1\}$ specifies whether the optional pre-processing algorithm is necessary ($v = 1$) or not ($v = 0$). A signer in a threshold signature scheme with $u = 0$ (resp. $w = 0$) has no interaction with other signers in the key generation (resp. signing) phase and will only output outgoing messages to the leaders. $v = 1$ corresponds to an *echo scheme* in [10, 11]. As an example, we illustrate the well-known echo FROST2 [24] and non-echo BLS [13] constructions respectively as $(u, v, w) = (2, 1, 0)$ and $(2, 0, 0)$ threshold signature instances in Appendix B.

Definition 6. *A (t, n)-threshold signature scheme* $\mathsf{TS} = (\mathsf{KGen}, \mathsf{VkAgg}, \mathsf{SPP}, \mathsf{LPP}, \mathsf{LR}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Vrfy})$ *with (u, v, w) rounds is stateful protocol as follows:*

Key Generation: $\mathsf{KGen} = (\mathsf{KGen}^{(0)}, \ldots, \mathsf{KGen}^{(u)})$ *allows a signer to generate a (per-signer) signing and verification key pair and the group verification key. The interactive key generation algorithm includes the following sub-algorithms for $u \geq 1$:*

- $\|_{i' \in [n] \setminus \{i\}}\, m_{(i,i')}^{(1)} \xleftarrow{\$} \mathsf{KGen}^{(0)}(S_i)$*: allows a signer i to output a sequence of outgoing messages $m_{(i,i')}^{(1)}$ for other $(n-1)$ signers i', where $i' \in [n] \setminus \{i\}$.*
- $\|_{i' \in [n] \setminus \{i\}}\, m_{(i,i')}^{(y+1)} \xleftarrow{\$} \mathsf{KGen}^{(y)}(S_i, m)$ *for $y \in [u-1]$: allows a signer i to input an incoming message m and output a sequence of outgoing messages $m_{(i,i')}^{(y+1)}$ for other $(n-1)$ signers i', where $i' \in [n] \setminus \{i\}$.*
- $m_i \xleftarrow{\$} \mathsf{KGen}^{(u)}(S_i, m)$*: allows a signer i to input an incoming message m and output an outgoing message m_i to the leader.*

The non-interactive key generation algorithm only includes the following one algorithm for $u = 0$:

- $m_i \xleftarrow{\$} \mathsf{KGen}^{(0)}(S_i)$*: allows a signer i to output an outgoing message m_i to the leader.*

Verification Key Aggregation: $gvk \leftarrow \mathsf{VkAgg}(L, \{m_i\}_{i \in [n]})$ *inputs a set of signers' messages, deterministically initializes the leader's long-term state, and outputs a group verification key.*

Signer Pre-Processing: $pp_i \xleftarrow{\$} \mathsf{SPP}(S_i)$ *allows a signer i to output an outgoing message (so-called* pre-processing token[1]*) pp that is used for i to sign messages later, if $v = 1$ (i.e., echo). If $v = 0$ (i.e., non-echo), this algorithm is omitted and the invocation of this algorithm simply outputs $\perp$.*

Leader Pre-Processing: $\mathsf{LPP}(L, pp)$ *allows the leader to input a pre-processing token pp and to update its local state, if $v = 1$ (i.e., echo). If $v = 0$ (i.e., non-echo), this algorithm is omitted.*

Leader Signing-Request: $lr \xleftarrow{\$} \mathsf{LR}(L, SS, m)$ *allows the leader to input a set of signers SS and a message m to produce a leader signing request lr.*

[1]The outgoing messages output by the signer pre-processing algorithm are commonly called pre-processing tokens, which we will use in the remainder of this paper.

Signing: $\mathsf{Sign} = (\mathsf{Sign}^{(0)}, \ldots, \mathsf{Sign}^{(w)})$ *allows signers to interactively sign messages and includes the following sub-algorithms if* $w \geq 1$*:*

- $\|_{i' \in lr.SS \setminus \{i\}}\, m^{(1)}_{(i,i')} \xleftarrow{\$} \mathsf{Sign}^{(0)}(S_i, lr)$ *allows a signer* i *to start signing a leader request* lr*, and output a sequence of outgoing messages* $m^{(1)}_{(i,i')}$ *for every signer* i *with* $i' \in lr.SS \setminus \{i\}$*.*
- $\|_{i' \in lr.SS \setminus \{i\}}\, m^{(y+1)}_{(i,i')} \xleftarrow{\$} \mathsf{Sign}^{(y)}(S_i, m)$ *for* $y \in [w-1]$ *allows a signer* i *to input an incoming message* m *and to output a sequence of outgoing messages* $m^{(y+1)}_{(i,i')}$ *for every signer* i' *with* $i' \in lr.SS \setminus \{i\}$*.*
- $\varsigma_i \xleftarrow{\$} \mathsf{Sign}^{(w)}(S_i, m)$ *allows the signer* i *to input an incoming message* m *and to output a partial signature* ς_i*.*

or the following algorithm if $w = 0$*:*

- $\varsigma_i \xleftarrow{\$} \mathsf{Sign}^{(0)}(S_i, lr)$ *allows the signer* i *to sign a leader request* lr *and to output a partial signature* ς_i*.*

Signature Aggregation: $\sigma \leftarrow \mathsf{SigAgg}(L, lr, \{\varsigma_i\}_{i \in lr.SS})$ *allows a leader* L *to input a leader request* lr *and a set of partial signatures* ς_i *that are generated by signers* $i \in lr.SS$ *and to deterministically output a group signature* σ*.*

Verification: $0/1 \leftarrow \mathsf{Vrfy}(gvk, m, \sigma)$ *verifies whether* σ *is a valid group signature over the message* m *with respect to the group verification key* gvk *(outputs* 1*) or not (outputs* 0*).*

Optionally, TS *might include an additional* strong verification algorithm SVrfy *as follows:*

Strong Verification: $0/1 \leftarrow \mathsf{SVrfy}(gvk, lr, \sigma)$ *verifies whether* σ *is a valid group signature over the leader request* lr *with respect to the group verification key* gvk *(outputs* 1*) or not (outputs* 0*). Moreover, for every* gvk *and* lr*, there exists* at most *one* σ *such that* $\mathsf{SVrfy}(gvk, lr, \sigma) = 1$.

Correctness. Following the (strong) correctness definition in [10, 11], we define (strong) correctness for our generalized threshold signature in Appendix C.

Message vs Leader Request. Unlike historical definitions of threshold signatures that only sign a message m, our definition (following [10, 11]) signs a collection of data called a leader request lr (see Definition 1). In addition to a message $lr.m$, this can also include a set of expected signers $lr.SS$ and other values, depending on the scheme. We stress that our syntax can still cover the historical definitions, as every message m to be signed can be considered as a leader request lr with the message $lr.m = m$, and the expected set of all signers $lr.SS = [n]$.

Comparison with [10, 11]. Definition 6 extends the definition of threshold signatures in [10, 11, Section 3.1], differing from it in three main aspects: First, while [10, 11, Section 3.1] is only defined for non-interactive threshold signatures ($w = 0$), our definition also covers interactive threshold signatures ($w \geq 1$). Second, [10, 11, Section 3.1] only includes a compact key generation assumed to be "done by a trusted algorithm", whereas our key generation captures the underlying (possibly interactive) communication between signers, in particular for threshold signature schemes with $u \geq 1$. This modification allows our security model to capture more find-grained features, such as malicious key generation and insecure communication channels between signers. Third, unlike [10, 11, Section 3.1], our definition does *not* explicitly include "a set HF of functions from which the random oracle is drawn", making it more generic and compatible with post-quantum secure constructions for future analyses.

4 Extended Hierarchy of Security Notions

In this section, we propose a new unforgeability hierarchy for threshold signature schemes, which includes both (the weaker) existential unforgeability ($\mathsf{EUF\text{-}CMA}$) and strong unforgeability ($\mathsf{SUF\text{-}CMA}$). We depict our generic security game $\mathrm{Game}^{\mathtt{UF}}_{\mathsf{TS}}$ in Figure 2. We provide a detailed description for the $\mathsf{EUF\text{-}CMA}$ security hierarchy (i.e., $\mathtt{UF} = \mathsf{EUF\text{-}CMA}$), in Section 4.1 and extend it to the $\mathsf{SUF\text{-}CMA}$ security hierarchy (i.e., $\mathtt{UF} = \mathsf{SUF\text{-}CMA}$) in Section 4.2.

Minimal Assumptions on Security Models. In our security model, we assume that all honest signers employ unpredictable random number generators. Furthermore, we assume that the leader is honest during the key generation phase to prevent a malicious leader from forging the group verification key and signing messages on behalf of the group. We also assume that the communication channels between signers and the leader are authenticated during the key generation phase. Otherwise, an adversary could impersonate all honest signers, so that the final group verification key produced by the leader relied solely on the secrets chosen by the adversary. However, we stress that there are *no assumptions* on the communication channels between signers and the leader in the signing phase. Moreover, our model allows signers to concurrently sign messages.

4.1 Existential Unforgeability

In our existential unforgeability model ($\mathrm{Game}_{\mathsf{TS}}^{\mathsf{EUF\text{-}CMA}}$ in Figure 2), an adversary is given access to four oracles that respectively simulate key generation, pre-processing tokens generation, message signing, and state corruption. The adversary wins if it can output a challenge message-signature pair that passes verification and is not a trivial forgery. To systematically measure the security guarantees for threshold signatures, we associate our unforgeability hierarchy with a quadruple of attributes.

Security Attributes. Our model is associated with a quadruple of attributes ($\mathtt{SiGu}, \mathtt{Corr}, \mathtt{KGCh}, \mathtt{SiCh}$), each representing a class of security guarantees from the literature (see Section 2). Concretely, we identify the following four attributes:

1. $\mathtt{SiGu}$: Guarantees obtained on the Signers,
2. $\mathtt{Corr}$: Adversary's capability to Corrupt signers,
3. $\mathtt{KGCh}$: Adversary's capability wrt. the Key Generation Channel, and
4. $\mathtt{SiCh}$: Adversary's capability wrt. the Signing Channel.

For each attribute, we identify several levels. A specific attribute level is denoted as $\mathit{lev}\colon \mathit{name}$, where name is a shorthand and $\mathit{lev} \geq 0$ is an integer that represents the level. $\mathit{lev} = 0$ represents the weakest security level (i.e., most restrictions for adversaries) and higher numbers indicate stronger security (i.e., fewer restrictions for adversaries). To emphasize the ordering within each attribute, we prefix every instance name by its level, e.g., $\mathtt{Att} = \mathit{lev}\colon \mathit{name}$. Thus, for a given attribute, the order on attribute instances directly corresponds to the order on the prefixed numbers, i.e., for two levels of the same attribute $\mathtt{Att}_1 = \mathit{lev}_1\colon \mathit{name}_1$ and $\mathtt{Att}_2 = \mathit{lev}_2\colon \mathit{name}_2$, we have that $\mathtt{Att}_1 > \mathtt{Att}_2$ if $\mathit{lev}_1 > \mathit{lev}_2$, and $\mathtt{Att}_1 \geq \mathtt{Att}_2$ if $\mathit{lev}_1 > \mathit{lev}_2$ or $\mathit{lev}_1 = \mathit{lev}_2$.

Attribute 1: Signer Guarantees. We identify five different levels of signer guarantees, denoted $\mathtt{SiGu} \in \{0\colon \mathsf{eM}, 1\colon \mathsf{tM}, 2\colon \mathsf{tLR}, 3\colon \mathsf{tLRhPP}, 4\colon \mathsf{aLRhPP}\}$. Each level is specified by the set of challenge message-signature pairs that are considered to be trivially forged in the model (see Figure 2, Line 13). In Table 1, we provide the formal definitions used to define each level: the challenge message-signature pair $(m^\star, \sigma^\star)$ output by an adversary is considered as a trivial forgery, if there exists a leader request lr with $lr.m = m^\star$ such that $\mathtt{tf}_{\mathtt{SiGu}}(lr)$.

The five levels can be described more intuitively as follows, where n_{ms} is the number of malicious signers at the end of the experiment, and t is the threshold:

- $\mathtt{SiGu} = 0\colon \mathsf{eM}$: At least one honest signer has involved in signing $lr.m$.
- $\mathtt{SiGu} = 1\colon \mathsf{tM}$: In addition to the $0\colon \mathsf{eM}$ guarantee, the number of honest signers who has involved in signing $lr.m$ is at least $(t - n_{\mathsf{ms}})$.
- $\mathtt{SiGu} = 2\colon \mathsf{tLR}$: At least $(t - n_{\mathsf{ms}})$ signers have involved in signing the leader request lr.
- $\mathtt{SiGu} = 3\colon \mathsf{tLRhPP}$: In addition to the $2\colon \mathsf{tLR}$ guarantee, for each honest signer i, we require that (i has involved in signing the leader request lr) if and only if (the leader request lr includes i and (the pre-processing token associated with i is honest, if TS is an echo scheme)).
- $\mathtt{SiGu} = 4\colon \mathsf{aLRhPP}$: In addition to the $3\colon \mathsf{tLRhPP}$ guarantee, all honest signers in the leader request have involved in signing the leader request lr.

For non-echo schemes ($v = 0$), the levels $3\colon \mathsf{tLRhPP}$ and $4\colon \mathsf{aLRhPP}$ are equivalent, because the leader requests in non-echo schemes include no pre-processing tokens.

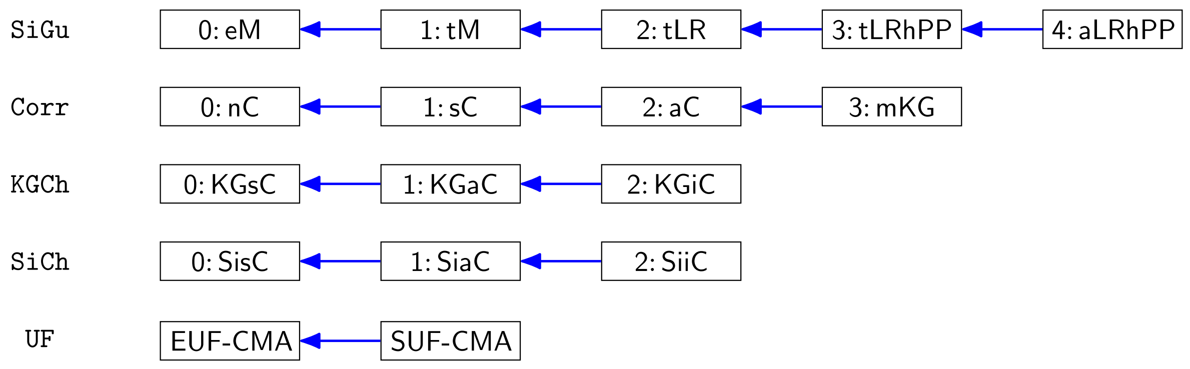

Figure 1: The implications (indicated by arrows) between the attributes in our hierarchy.

Attribute 2: Signer Corruption. Corr $\in \{0\colon \mathsf{nC}, 1\colon \mathsf{sC}, 2\colon \mathsf{aC}, 3\colon \mathsf{mKG}\}$ specifies the adversaries' capabilities to corrupt honest signers. While an honest signer must follow the protocol, a corrupted signer is considered to be malicious and can deviate from the protocol at will.

Our model incorporates existing definitions of state corruption in the literature and considers the following four levels of corruption:

- Corr $= 0\colon \mathsf{nC}$: Adversaries are *not* allowed to corrupt any signer in the model.
- Corr $= 1\colon \mathsf{sC}$: Adversaries must pre-choose a desired set of signers to be corrupted at the beginning of the security model and are allowed to corrupt these signers only after the completion of the group verification key aggregation. This level is often called "*static corruption*" in the literature.
- Corr $= 2\colon \mathsf{aC}$: Adversaries are allowed to corrupt any signer at any time after the completion of the group verification key aggregation. This level is often called "*adaptive corruption*" in the literature.
- Corr $= 3\colon \mathsf{mKG}$: Adversaries are allowed to corrupt any signer *at any time* in the security model, particularly, including the key generation phase. This level is often called "*malicious key generation*" in the literature.

Attribute 3: Key Generation Channel Security. KGCh $\in \{0\colon \mathsf{KGsC}, 1\colon \mathsf{KGaC}, 2\colon \mathsf{KGiC}\}$ specifies the adversaries' capabilities to interfere with communication channels *among signers*[2] during the key generation phase[3]. For a threshold signature with no key generation rounds among signers ($u = 0$), all levels of KGCh are equivalent.

Our model considers the following three levels of key generation channels:

- KGCh $= 0\colon \mathsf{KGsC}$: Adversaries can *neither* eavesdrop *nor* manipulate the communication channels among signers during the key generation phase. This level is often called a *secure channel*.
- KGCh $= 1\colon \mathsf{KGaC}$: Adversaries *can* eavesdrop but *cannot* manipulate the communication channels among signers during the key generation phase. This level is often called an *authenticated channel*.
- KGCh $= 2\colon \mathsf{KGiC}$: Adversaries can *both* eavesdrop *and* manipulate the communication channels among signers during the key generation phase. This level is often called an *insecure channel*.

Attribute 4: Signing Channel Security. SiCh $\in \{0\colon \mathsf{SisC}, 1\colon \mathsf{SiaC}, 2\colon \mathsf{SiiC}\}$ specifies the adversaries' capabilities to interfere with communication channels *among signers*[4] during the signing phase. For a threshold signature with no signing rounds among signers ($w = 0$), all levels of SiCh are equivalent.

Our model considers the following three levels of signing channels:

- SiCh $= 0\colon \mathsf{SisC}$: Adversaries can *neither* eavesdrop *nor* manipulate the communication channels among signers during the signing phase. This level is often called *secure channels*.

[2]We stress that the KGCh attribute only applies to the communication channels among signers without involving the leader, as we assume the communication channels between signers and the leader during the key generation phase are authenticated.

[3]For threshold signature schemes with multiple rounds in the key generation phase, we do *not* distinguish between the communication channels for each round; we only consider the strongest channel assumption (i.e., the lowest level). Similar arguments apply to the channels in the signing phase.

[4]The communication channels between the leader and every signer are considered *insecure*, as the adversaries are given access to signing oracle for any leader request.

Table 1: The trivial-forgery conditions $\mathtt{tf}_{\mathtt{SiGu}}$ and trivial-strong-forgery conditions $\mathtt{tsf}_{\mathtt{SiGu}}$ for $\mathtt{SiGu} \in \{0\colon \mathsf{eM}, 1\colon \mathsf{tM}, 2\colon \mathsf{tLR}, 3\colon \mathsf{tLRhPP}, 4\colon \mathsf{aLRhPP}\}$.

$\mathtt{tf}_{0:\mathsf{eM}}(lr)$	:	$\mathcal{D}_1[lr.m] \neq \emptyset$
$\mathtt{tf}_{1:\mathsf{tM}}(lr)$	:	$\lvert\mathcal{D}_1[lr.m]\rvert \geq t - \lvert\mathcal{L}_{\mathsf{MS}}\rvert$
$\mathtt{tf}_{2:\mathsf{tLR}}(lr)$	:	$\lvert\mathcal{D}_2[lr]\rvert \geq t - \lvert\mathcal{L}_{\mathsf{MS}}\rvert$
$\mathtt{tf}_{3:\mathsf{tLRhPP}}(lr)$	:	$\mathtt{tf}_{2:\mathsf{tLR}}(lr)$ **and** $\mathcal{D}_2[lr] = \mathcal{D}_3[lr]$
$\mathtt{tf}_{4:\mathsf{aLRhPP}}(lr)$	:	$\mathtt{tf}_{2:\mathsf{tLR}}(lr)$ **and** $\mathcal{D}_2[lr] = \mathcal{D}_3[lr] = \mathcal{D}_4[lr]$

$\mathtt{tsf}_{0:\mathsf{eM}}(lr, gvk, \sigma)$	:	$\mathtt{tf}_{0:\mathsf{eM}}(lr)$ **and** $\mathsf{SVf}(gvk, lr, \sigma)$
$\mathtt{tsf}_{1:\mathsf{tM}}(lr, gvk, \sigma)$	:	$\mathtt{tf}_{1:\mathsf{tM}}(lr)$ **and** $\mathsf{SVf}(gvk, lr, \sigma)$
$\mathtt{tsf}_{2:\mathsf{tLR}}(lr, gvk, \sigma)$	:	$\mathtt{tf}_{2:\mathsf{tLR}}(lr)$ **and** $\mathsf{SVf}(gvk, lr, \sigma)$
$\mathtt{tsf}_{3:\mathsf{tLRhPP}}(lr, gvk, \sigma)$	:	$\mathtt{tf}_{3:\mathsf{tLRhPP}}(lr)$ **and** $\mathsf{SVf}(gvk, lr, \sigma)$
$\mathtt{tsf}_{4:\mathsf{aLRhPP}}(lr, gvk, \sigma)$	:	$\mathtt{tf}_{4:\mathsf{aLRhPP}}(lr)$ **and** $\mathsf{SVf}(gvk, lr, \sigma)$

- $\mathtt{SiCh} = 1\colon \mathsf{SiaC}$: Adversaries *can* eavesdrop but *cannot* manipulate the communication channels among signers during the signing phase. This level is often called *authenticated channels.*
- $\mathtt{SiCh} = 2\colon \mathsf{SiiC}$: Adversaries can *both* eavesdrop *and* manipulate the communication channels among signers during the signing phase. This level is often called *insecure channels.*

We provide the full relations between the above attributes in Figure 1.

Security Model. At the beginning of our existential unforgeability model $\mathrm{Game}_{\mathsf{TS}}^{\mathtt{UF}}$ for $\mathtt{UF} = \mathsf{EUF\text{-}CMA}$ in Figure 2, the adversary $\mathcal{A}$ chooses a set $\mathcal{L}_{\mathsf{CS}}$ of signers to be corrupted, which makes particular significance to the security notions in our hierarchy with $\mathtt{Corr} = 1\colon \mathsf{sC}$, i.e., static corruption. Next, our model initializes two lists, $\mathcal{L}_{\mathsf{MS}}$ with the empty set and $\mathcal{L}_{\mathsf{HS}}$ with $[n]$, which respectively denotes the list of malicious and of honest signers, and a counter $\mathsf{ctr}_{\mathsf{Sess}}$ with 0, which denotes the number of all signers' (concurrent) signing sessions.

In our model, session identifiers identify the leader request to be signed. In particular, sessions π_i^j and $\pi_{i'}^{j'}$ of signers i and i' will sign the same leader request, as long as $j = j'$[5]. Our experiment initializes eight dictionaries: $\mathcal{D}_{\mathsf{Trans}}^{\mathsf{KGen}}$ and $\mathcal{D}_{\mathsf{Trans}}^{\mathsf{Sign}}$ respectively record honest signers' output during the key generation and signing phase; $\mathcal{D}_{\mathsf{LR}}$ records the leader requests signed in every session; $\mathcal{D}_{\mathsf{PP}}$ records the pre-processing tokens output by every honest signers; $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$ record information for the trivial forgery test at the end of the experiment.

Afterwards, the adversary $\mathcal{A}$ is given access to the key generation and corruption oracles, followed by outputting a set of outgoing messages on behalf of all malicious signers. The experiment derives the group verification key gvk, if all necessary input messages, i.e., the recorded messages for honest signers and the by $\mathcal{A}$ output messages for malicious signers, are not $\perp$. The aggregated group verification key gvk must not be $\perp$.

In the end, the adversary $\mathcal{A}$ is given access to three oracles, respectively for state corruption, pre-processing tokens generation, and message signing, followed by outputting a challenge message-signature pair $(m^\star, \sigma^\star)$. The adversary $\mathcal{A}$ immediately loses if the challenge message-signature pair cannot pass the verification or at least t signers have been corrupted. The adversary $\mathcal{A}$ wins if the challenge message-signature pair is not a trivial forgery, defined by the attribute $\mathtt{SiGu}$ (see Attribute 1: Signer Guarantees), and loses otherwise.

The four oracles are defined as follows:

Oracle 1: Key Generation. The $\textsc{OKGen}(i, m)$ oracle simulates an honest signer i's key generation execution with an incoming message m. We require that the signer i must be honest and the key generation must be uncompleted. If the signer i has never executed key generation, it runs $\mathsf{KGen}^{(0)}(\mathsf{st}_i)$ for an outgoing message m'. Otherwise, the signer i starts to run the subsequent round key generation and the input message m must be non-$\perp$. If the communication channels among signers are secure or authenticated (indicated by $\mathtt{KGCh} \leq 1\colon \mathsf{KGaC}$), the input messages m should specify the incoming messages from malicious

[5]However, the reverse might not hold, as the signature generation might not involve all signers. Moreover, the same leader requests might be repeatedly signed by the same signer in different sessions.

```
$\underline{\mathrm{Game}^{\mathtt{UF}}_{\mathsf{TS}}(\mathcal{A}),\ \mathtt{UF} \in \{\text{EUF-CMA}, \text{SUF-CMA}\}:}$
$//$ initialize counters, lists, and dictionaries
$\mathcal{L}_{\mathsf{CS}} \xleftarrow{\$} \mathcal{A}();\ \mathcal{L}_{\mathsf{MS}} \xleftarrow{\$} \emptyset;\ \mathcal{L}_{\mathsf{HS}} \leftarrow [n];\ \mathsf{ctr}_{\mathsf{Sess}} \leftarrow 0;\ \mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[*], \mathcal{D}^{\mathsf{Sign}}_{\mathsf{Trans}}[*], \mathcal{D}_{\mathsf{LR}}[*] \leftarrow \perp;\ \mathcal{D}_{\mathsf{PP}}[*] \leftarrow \emptyset$
$\mathcal{D}_1[*], \mathcal{D}_2[*], \mathcal{D}_3[*], \mathcal{D}_4[*] \leftarrow \emptyset$ $//$ dictionaries for testing winning conditions
$\{m'_i\}_{i \in \mathcal{L}_{\mathsf{MS}}} \leftarrow \mathcal{A}^{\mathrm{OKGen},\mathrm{OCorrupt}}()$ $//$ key generation phase
$\textbf{req}\ \forall i \in \mathcal{L}_{\mathsf{HS}} : \mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(u+1, i, 0)] \neq \perp;\ \textbf{req}\ \forall i \in \mathcal{L}_{\mathsf{MS}} : m'_i \neq \perp$
$gvk \leftarrow \mathsf{VkAgg}(\mathsf{st}_0, \{\mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(u+1, i, 0)]\}_{i \in \mathcal{L}_{\mathsf{HS}}} \cup \{m'_i\}_{i \in \mathcal{L}_{\mathsf{MS}}});\ \textbf{req}\ gvk \neq \perp$
$(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\mathrm{OCorrupt},\mathrm{OPP},\mathrm{OSign}}(gvk)$ $//$ adversary outputs a challenge message-signature pair
$\textbf{foreach}\ lr \in \{\mathcal{D}_{\mathsf{LR}}[i] : i \in [\mathsf{ctr}_{\mathsf{Sess}}]\}\ \textbf{do}$ $//$ finalize for testing winning conditions
  $\mathcal{D}_1[lr.m] \leftarrow \mathcal{D}_1[lr.m] \cap \mathcal{L}_{\mathsf{HS}};\ \mathcal{D}_2[lr] \leftarrow \mathcal{D}_2[lr] \cap \mathcal{L}_{\mathsf{HS}}$
  $\mathcal{D}_3[lr] \leftarrow \{i \in lr.SS : lr.\mathsf{PP}(i) \in \mathcal{D}_{\mathsf{PP}}[i]\ \textbf{or}\ v = 0\} \cap \mathcal{L}_{\mathsf{HS}};\ \mathcal{D}_4[lr] \leftarrow lr.SS \cap \mathcal{L}_{\mathsf{HS}}$
$\textbf{if}\ \mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{false}\ \textbf{or}\ |\mathcal{L}_{\mathsf{MS}}| \geq t\ \textbf{then}$
  $\textbf{return false}$ $//$ loses if at least $t$ signers are corrupted
$\textbf{return}\ \neg(\exists lr\ \texttt{with}\ lr.m = m^\star\!:\ \mathtt{tf}_{\mathtt{SiGu}}(lr))$ $//$ for $\mathrm{Game}^{\text{EUF-CMA}}_{\mathsf{TS}}$ experiment
$\textbf{return}\ \neg(\exists lr\ \texttt{with}\ lr.m = m^\star\!:\ \mathtt{tsf}_{\mathtt{SiGu}}(lr, gvk, \sigma^\star))$ $//$ for $\mathrm{Game}^{\text{SUF-CMA}}_{\mathsf{TS}}$ experiment

$\underline{\mathrm{OKGen}(i, m)}$ $//$ key generation
$\textbf{req}\ i \in \mathcal{L}_{\mathsf{HS}}\ \textbf{and}\ \mathsf{st}_i.\mathsf{rnd} \leq u$
$\mathsf{rnd} \leftarrow \mathsf{st}_i.\mathsf{rnd};\ m' \leftarrow \perp$
$\textbf{if}\ \mathsf{rnd} = 0\ \textbf{then}$
  $m' \xleftarrow{\$} \mathsf{KGen}^{(0)}(\mathsf{st}_i)$
$\textbf{else}$ $//$ if $1 \leq \mathsf{rnd} \leq u$
  $\textbf{req}\ m \neq \perp$
  $\textbf{if}\ \mathtt{KGCh} \leq 1\!:\mathsf{KGaC}\ \textbf{then}$
    $\textbf{foreach}\ i' \in \mathcal{L}_{\mathsf{HS}} \setminus \{i\}\ \textbf{do}$
      $\textbf{req}\ \mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(\mathsf{rnd}, i', i)] \neq \perp$
      $m \xleftarrow{\|} \mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(\mathsf{rnd}, i', i)]$
  $m' \xleftarrow{\$} \mathsf{KGen}^{(\mathsf{rnd})}(\mathsf{st}_i, m)$
$\textbf{req}\ m' \neq \perp$
$\mathsf{rnd}{+}{+}$
$//$ record outputs
$\textbf{if}\ \mathsf{rnd} \leq u\ \textbf{then}$ $//$ if $\mathsf{KGen}$ is uncompleted
  $\textbf{parse}\ \|_{i' \in [n]\setminus\{i\}}\ m'_{(i,i')} \leftarrow m'$
  $\textbf{foreach}\ i' \in [n] \setminus \{i\}\ \textbf{do}$
    $\mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(\mathsf{rnd}, i, i')] \leftarrow m'_{(i,i')}$
$\textbf{else}$ $//$ if $\mathsf{KGen}$ has completed
  $\mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(\mathsf{rnd}, i, 0)] \leftarrow m'$
$//$ for secure $\mathsf{KGen}$ channels among signers
$\textbf{if}\ \mathtt{KGCh} = 0\!:\mathsf{KGsC}\ \textbf{and}\ \mathsf{rnd} \leq u\ \textbf{then}$
  $\textbf{return}\ \|_{i' \in \mathcal{L}_{\mathsf{MS}}}\ \mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(\mathsf{rnd}, i, i')]$
$//$ for authenticated or insecure $\mathsf{KGen}$ channels
$\textbf{return}\ m'$

$\underline{\mathrm{OCorrupt}(i)}$ $//$ state corruption
$\textbf{req}\ i \in \mathcal{L}_{\mathsf{HS}}\ \textbf{and}\ \mathtt{Corr} \geq 1\!:\mathsf{sC}$
$\textbf{req}\ \mathtt{Corr} \geq 3\!:\mathsf{mKG}\ \textbf{or}\ gvk \neq \perp$
$\textbf{req}\ \mathtt{Corr} \geq 2\!:\mathsf{aC}\ \textbf{or}\ i \in \mathcal{L}_{\mathsf{CS}}$
$\mathcal{L}_{\mathsf{HS}} \xleftarrow{-} i;\ \mathcal{L}_{\mathsf{MS}} \xleftarrow{+} i$
$\textbf{return}\ S_i$

$\underline{\mathrm{OPP}(i)}$ $//$ pre-processing phase
$\textbf{req}\ v = 1\ \textbf{and}\ i \in \mathcal{L}_{\mathsf{HS}}$
$pp \xleftarrow{\$} \mathsf{SPP}(\mathsf{st}_i);\ \mathcal{D}_{\mathsf{PP}}[i] \xleftarrow{+} pp$
$\textbf{return}\ pp$

$\underline{\mathrm{OSign}(i, j, m)}$ $//$ honest signing
$\textbf{req}\ i \in \mathcal{L}_{\mathsf{HS}}\ \textbf{and}\ j \leq \mathsf{ctr}_{\mathsf{Sess}} + 1$
$\textbf{req}\ \pi^j_i.\mathsf{rnd} \leq w\ \textbf{and}\ m \neq \perp$
$\textbf{if}\ j = \mathsf{ctr}_{\mathsf{Sess}} + 1\ \textbf{then}$ $//$ new sessions
  $\textbf{parse}\ lr \leftarrow m$
  $\textbf{req}\ lr.SS \subseteq [n]\ \textbf{and}\ lr.m \in \{0,1\}^\star$
  $\mathsf{ctr}_{\mathsf{Sess}}{+}{+};\ \mathcal{D}_{\mathsf{LR}}[\mathsf{ctr}_{\mathsf{Sess}}] \leftarrow lr$
$\mathsf{rnd} \leftarrow \pi^j_i.\mathsf{rnd};\ lr \leftarrow \mathcal{D}_{\mathsf{LR}}[j];\ SS \leftarrow lr.SS;\ m' \leftarrow \perp$
$\textbf{if}\ \mathsf{rnd} = 0\ \textbf{then}$
  $\textbf{req}\ m = lr$
$\textbf{else}$ $//$ if $1 \leq \mathsf{rnd} \leq w$
  $\textbf{if}\ \mathtt{SiCh} \leq 1\!:\mathsf{SiaC}\ \textbf{then}$
    $\textbf{foreach}\ i' \in \mathcal{L}_{\mathsf{HS}} \cap SS \setminus \{i\}\ \textbf{do}$
      $\textbf{req}\ \mathcal{D}^{\mathsf{Sign}}_{\mathsf{Trans}}[(j, \mathsf{rnd}, i', i)] \neq \perp$
      $m \xleftarrow{\|} \mathcal{D}^{\mathsf{Sign}}_{\mathsf{Trans}}[(j, \mathsf{rnd}, i', i)]$
$m' \xleftarrow{\$} \mathsf{Sign}^{(\mathsf{rnd})}(S^j_i, m);\ \textbf{req}\ m' \neq \perp$
$\mathcal{D}_1[lr.m], \mathcal{D}_2[lr] \xleftarrow{+} i;\ \mathsf{rnd}{+}{+}$
$//$ record outputs
$\textbf{if}\ \mathsf{rnd} \leq w\ \textbf{then}$ $//$ if $\mathsf{Sign}$ is uncompleted
  $\textbf{parse}\ \|_{i' \in SS\setminus\{i\}}\ m'_{(i,i')} \leftarrow m'$
  $\textbf{foreach}\ i' \in SS \setminus \{i\}\ \textbf{do}$
    $\mathcal{D}^{\mathsf{Sign}}_{\mathsf{Trans}}[(j, \mathsf{rnd}, i, i')] \leftarrow m'_{(i,i')}$
$//$ for secure $\mathsf{Sign}$ channels among signers
$\textbf{if}\ \mathtt{SiCh} = 0\!:\mathsf{SisC}\ \textbf{and}\ \mathsf{rnd} \leq w\ \textbf{then}$
  $\textbf{return}\ \|_{i' \in \mathcal{L}_{\mathsf{MS}}}\ \mathcal{D}^{\mathsf{Sign}}_{\mathsf{Trans}}[(j, \mathsf{rnd}, i, i')]$
$//$ for authenticated or insecure $\mathsf{Sign}$ channels
$\textbf{return}\ m'$
```

Figure 2: The (t, n)-$(\mathtt{SiGu}, \mathtt{Corr}, \mathtt{KGCh}, \mathtt{SiCh})$-$\mathtt{UF}$ game $\mathrm{Game}^{\mathtt{UF}}_{\mathsf{TS}}(\mathcal{A})$ for an adversary $\mathcal{A}$ that breaks TS with (u, v, w) rounds. The trivial-forgery conditions $\mathtt{tf}_{\mathtt{SiGu}}$ and trivial-strong-forgery conditions $\mathtt{tsf}_{\mathtt{SiGu}}$ are defined in Table 1.

signers to i. We require the necessary input messages from all other honest signers i' to i to be non-$\bot$ and append them to the incoming message m. The key generation algorithm is executed with the (possibly) appended message m for an outgoing message m'. We require $m' \neq \bot$.

If the signer has not completed key generation, the outgoing message m' is parsed and recorded as input messages of all other signers' next-round key generation. Otherwise, the outgoing message m' is recorded as the output to the leader. If the key generation is uncompleted over secure communication channels, the oracle only returns the input messages of malicious signers' next round key generation. Otherwise, the full outgoing message m' is returned.

Oracle 2: Pre-Processing. The $\mathrm{OPP}(i)$ oracle simulates the process that an honest signer i generates and transmits a pre-processing token pp to the leader. The oracle checks whether the threshold signature employs pre-processing algorithm, indicated by $v = 1$, and whether the signer i is honest. If the check passes, the oracle executes algorithm SPP with signer i's long-term state st_i for a pre-processing token pp, which is then recorded into the dictionary $\mathcal{D}_{\mathsf{PP}}[i]$ and output. Otherwise, the oracle simply exits without any output.

Oracle 3: Signing. The $\mathrm{OS{\scriptsize IGN}}(i, j, m)$ oracle simulates an honest signer i participating in signing the globally j-th leader request with incoming message m. Recall that all signers' sessions with the same identifier must sign the same leader request, while the reverse might not hold. The oracle requires that (1) the signer i must be honest, (2) the j-th session must exist or be the next one, (3) the j-th session must be uncompleted, and (4) the incoming message m must be non-$\bot$. If the input $j = \mathsf{ctr}_{\mathsf{Sess}} + 1$ indicates a new session, the input message m must be parsed as a leader request lr including a valid set of signers $lr.SS$ and a message $lr.m$. The oracle increments the session counter $\mathsf{ctr}_{\mathsf{Sess}}$ by 1 and records the leader request lr.

The cases split depending on the round index $\pi_i^j.\mathsf{rnd}$ included in the session state. If the round equals 0, then the oracle requires that the incoming message to be j-th leader request recorded in the dictionary $\mathcal{D}_{\mathsf{LR}}$. Otherwise, if the signing communication channels are authenticated, indicated by $\mathtt{SiCh} \leq 1\colon \mathsf{SiaC}$, the input message m must specifies all messages sent from malicious signers to the honest signer i. The oracle checks whether all necessary input messages from other honest signers i' to i have been produced and recorded, followed by appending them to the incoming message m. If the check fails, the oracle undoes executed computation and exits. In both cases, the oracle runs the next round signing algorithm with input m for an outgoing message m', which is required to be non-$\bot$. Then, the oracle records the signer i into two dictionaries $\mathcal{D}_1$ and $\mathcal{D}_2$. If the signer i has not completed this j-th signing session, then the outgoing message m' is parsed and recorded as other signers' next-round signing inputs. Otherwise, the outgoing message m' is recorded as the output, i.e., partial signature, to the leader. In the case that the signing phase is uncompleted over secure communication channels, the oracle only returns malicious signers' next round signing inputs. Otherwise, the full outgoing message m' is returned.

Oracle 4: State Corruption. The $\mathrm{OC{\scriptsize ORRUPT}}(i)$ oracle simulates the state corruption of an honest signer i. The oracle first checks whether the signer i is honest and whether the state corruption is allowed, indicated by $\mathtt{Corr} \geq 1\colon \mathsf{sC}$ (see Attribute 2: State Corruption). Next, the oracle checks whether malicious key generation is allowed, indicated by $\mathtt{Corr} \geq 3\colon \mathsf{mKG}$, or the key generation phase is done, indicated by $gvk \neq \bot$, which is the pre-condition for other levels of corruption with $\mathtt{Corr} < 3\colon \mathsf{mKG}$. Then, the oracle checks whether the adaptive corruption is allowed, indicated by $\mathtt{Corr} \geq 2\colon \mathsf{aC}$, or the signer i is pre-chosen at the beginning of the model in $\mathcal{L}_{\mathsf{CS}}$. If any check fails, the oracle simply exits. Otherwise, the oracle marks the honest signer i to be malicious by removing i from the honest-signer list $\mathcal{L}_{\mathsf{HS}}$ and adding i into the malicious-signer list, followed by returning all states of the signer i, including both the long-term state st_i and all existing session states $\{\pi_i^j\}_j$.

Definition 7. *Let* TS *denote a* (t, n)*-threshold signature scheme with* (u, v, w) *rounds. We say* TS *is* ϵ-$(\mathtt{SiGu}, \mathtt{Corr}, \mathtt{KGCh}, \mathtt{SiCh})$-$\mathsf{EUF}$-$\mathsf{CMA}$ *secure if the below defined advantage for all adversaries against the experiment* $\mathrm{Game}_{\mathsf{TS}}^{\mathsf{EUF\text{-}CMA}}$ *in Figure 2 is bounded by* ϵ*, i.e.,*

$$\mathsf{Adv}_{\mathsf{TS}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) := \Pr[\mathrm{Game}_{\mathsf{TS}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) = 1] \leq \epsilon$$

4.2 Strong Unforgeability

We define our generalized strong unforgeability hierarchy for interactive threshold signatures. We depict the strong unforgeability model $\mathrm{Game}_{\mathsf{TS}}^{\mathsf{SUF\text{-}CMA}}$ in Figure 2. Our strong unforgeability share the same

Table 2: Threshold signature constructions and their provable unforgeability in the literature. For schemes with $w = 0$, `SiCh` $= 2{:}\,\mathsf{SiiC}$ always holds. Schemes with $u = -$ only consider centralized key generation with a trusted dealer, which, in theory, could be replaced by some distributed key generation [39, 46].

Scheme	**(u,v,w)**	**Proof Model**	`UF`	`SiGu`	`Corr`	`KGCh`	`SiCh`
BLS / TGS(G) [13, 15]	$(2,0,0)$	Game [11, 13] Game [4]	EUF-CMA EUF-CMA	1: tM 1: tM	1: sC 3: mKG	0: KGsC 1: KGaC	2: SiiC 1: SiaC
SimpleTSig [24]	$(2,0,2)$	Game [24]	EUF-CMA	1: tM	1: sC*	0: KGsC	2: SiiC
FROST [42]	$(2,1,0)$	Simulation [42] Game [11]	EUF-CMA SUF-CMA	1: tM 3: tLRhPP	1: sC* 1: sC	0: KGsC 0: KGsC	2: SiiC 2: SiiC
FROST2 [24]	$(2,0,1)$ $(2,1,0)$	Game [24] Game [11]	EUF-CMA SUF-CMA	1: tM 2: tLR	1: sC* 1: sC	0: KGsC 0: KGsC	2: SiiC 2: SiiC
ROAST (FROST3) [48]	$(2,0,0)$	Game [48]	EUF-CMA	1: tM	1: sC	0: KGsC	2: SiiC
Sparkle+ [23]	$(-,0,2)$	Game [23]	EUF-CMA	1: tM	2: aC	0: KGsC	2: SiiC
GG18 [36]	$(4,0,9)$	Game [36]	EUF-CMA	1: tM	1: sC	0: KGsC	2: SiiC
"New" GG20 [16]	$(4,0,3)$ $(4,0,6)$	UC [16]	EUF-CMA	0: eM	2: aC	1: KGaC	1: SiaC
ASY20 [1, Section 6.1]	$(-,0,1)$	Game [1]	EUF-CMA	1: tM	2: aC	0: KGsC	2: SiiC
Twinkle [6]	$(-,0,2)$	Game [6]	EUF-CMA	1: tM	2: aC	0: KGsC	2: SiiC
HARTS [5]	$(9,0,0)$	Game [5]	EUF-CMA	1: tM	3: mKG	1: KGaC	1: SiaC

* The model only considers malicious key generation without state corruption. We merge this with the weaker static corruption case (`Corr` $= 1{:}\,\mathsf{sC}$).

syntax (Definition 6), quadruple of attributes (`SiGu`, `Corr`, `KGCh`, `SiCh`), and security model ($\mathrm{Game}_{\mathsf{TS}}^{\mathsf{SUF\text{-}CMA}}$) with the existential unforgeability, except for the following differences:

Differences in Syntax. A threshold signature scheme TS providing strong unforgeability must be equipped with a strong verification algorithm (SVrfy, see Definition 6), which is optional for achieving existential unforgeability. Recall that SVrfy can verify whether a signature σ is the unique possible signature of a message m under any group verification key gvk. Correspondingly, TS must additionally achieve strong correctness (see Definition 10).

Differences in Attributes. The definitions of `Corr`, `KGCh`, and `SiCh` for strong unforgeability are identical to the ones for existential unforgeability. For `SiGu`, the following difference holds: Let gvk denote the group verification key. The challenge message-signature pair $(m^\star, \sigma^\star)$ output by an adversary is considered as a trivial forgery, if there exists any leader request lr with $lr.m = m^\star$ such that in addition to $\mathtt{tf}_{\mathtt{SiGu}}(lr)$ the condition $\mathsf{SVrfy}(gvk, lr, \sigma^\star)$ holds.

Differences in Security Models. Reflected by the differences in attributes, the $\mathrm{Game}_{\mathsf{TS}}^{\mathsf{SUF\text{-}CMA}}$-model is identical to the $\mathrm{Game}_{\mathsf{TS}}^{\mathsf{EUF\text{-}CMA}}$-model, except for the final winning test. Namely, the execution of Line 13 is replaced by the one of Line 14.

4.3 Properties of Existing Threshold Signature Schemes

In Table 2, we select several representative threshold signature schemes from the literature. Moreover, we extract and categorize the core features of their existing provable security models according to our generic unforgeability hierarchy.

4.4 Linearity of Attribute Levels

The following theorems ensure the linearity of our unforgeability hierarchy. Roughly speaking, these theorems prove that a threshold signature that satisfies unforgeability with respect to some element of our hierarchy, also satisfies those with lower levels in any attribute. We provide the proof in Appendix E.

Theorem 1 (Unforgeability Hierarchy). *Let* TS *denote a* (t, n) *threshold signature with* (u, v, w) *rounds.*

- *If* TS *is* ϵ-$(\mathtt{SiGu}_1, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1)$-$\mathtt{UF}$ *secure, then* TS *is also* ϵ-$(\mathtt{SiGu}_2, \mathtt{Corr}_2, \mathtt{KGCh}_2, \mathtt{SiCh}_2)$-$\mathtt{UF}$ *secure, for any* $\mathtt{SiGu}_2 \leq \mathtt{SiGu}_1$*,* $\mathtt{Corr}_2 \leq \mathtt{Corr}_1$*,* $\mathtt{KGCh}_2 \leq \mathtt{KGCh}_1$*,* $\mathtt{SiCh}_2 \leq \mathtt{SiCh}_1$*,* $\mathtt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$*.*
- *If* TS *is* ϵ-$(\mathtt{SiGu}, \mathtt{Corr}, \mathtt{KGCh}, \mathtt{SiCh})$-$\mathsf{SUF\text{-}CMA}$ *secure, then* TS *is also* ϵ-$(\mathtt{SiGu}, \mathtt{Corr}, \mathtt{KGCh}, \mathtt{SiCh})$-$\mathsf{EUF\text{-}CMA}$ *secure.*

5 Revisiting the Hierarchy in [10, 11]

In our hierarchy we presented in the previous section, our first attribute, $\mathtt{SiGu}$, largely follows the security definitions proposed by Bellare, Tessaro, and Zhu in [10, 11], with a few notable exceptions. Namely, we use a modified definition for $\mathtt{SiGu} = 4\colon \mathsf{aLRhPP}$ and an extended application domain for all other levels. In this section, we explain and motivate the need for our modifications.

The abstract of [11] states: "*We give a unified syntax, and a hierarchy of definitions of security of increasing strength, for non-interactive threshold signature schemes. They cover both fully non-interactive schemes (these are ones that have a single-round signing protocol, the canonical example being threshold-BLS) and ones, like FROST, that have a prior round of message-independent pre-processing.*" As we show in this section, their hierarchy is *not* of increasing strength for all schemes in their stated scope, i.e., even for all echo schemes. Furthermore, we show that their ATS transformation (from their TS-(S)UF-3 to TS-(S)UF-4 notions) is better than advertised.

We communicated our results and suggestions to the authors of [11] on 18 May 2024, and they responded that they intended but forgot to further restrict the class of schemes for which their TS-(S)UF-3 is defined, and they would update their ePrint to clarify this. As of November 21st 2024, this planned update has not happened yet.

Indeed, further restricting TS-(S)UF-3 can exclude our counterexample. However, an alternative route to repairing the linearity, which we have taken in the definition of our own hierarchy, is to strengthen the highest level. This has the benefit of increasing the scope of the hierarchy, and in fact matches the guarantees achieved by the transformation of [10, 11], as we will show in the next subsections.

5.1 TS-(S)UF-i Security is not Linearly Increasing in Strength

We prove that the TS-(S)UF-i hierarchy in [10, 11] is not increasing in strength by constructing an echo scheme Π (specified in Appendix D) that meets TS-(S)UF-4 but not TS-(S)UF-3. The main trick for not achieving TS-UF-3 is that the signer i only checks whether i is included in the leader request, without checking whether the associated pre-processing token is honestly generated. Note that neglecting the precise check for the associated pre-processing token is not only a theoretical possibility, but is also error-prone in practice.

The following theorems prove that our construction Π is TS-(S)UF-4 but not TS-(S)UF-3 secure. We provide their full proofs in Appendix F to Appendix J.

Theorem 2. *Suppose* DS *underlying* Π *is a digital signature scheme and* $2 \leq t < n$*. If* DS *is UF-CMA-secure, then* Π *is TS-UF-4 secure.*

Theorem 3. *For any* $2 \leq t < n$*,* Π *is not TS-UF-3 secure.*

Theorem 4. *Suppose* DS *underlying* Π *is a digital signature scheme and* $2 \leq t < n$*. If* DS *is unique and SUF-CMA-secure, then* Π *is TS-SUF-4 secure.*

Theorem 5. *For any* $2 \leq t < n$*,* Π *is not TS-SUF-3 secure.*

Corollary 1. *Neither TS-UF-3 security nor TS-UF-4 security implies the other. The same holds for TS-SUF-3 security and TS-SUF-4 security.*

Remark 1. *In order to keep the linearity of the unforgeability hierarchy, we define our own* $\mathtt{SiGu} = 4\colon \mathsf{aLRhPP}$ *level in Section 4.1. Recall that* $S_i(lr)$ *in [10, 11] is equivalent to* $\mathcal{D}_i[lr]$ *in our model. In terms of signer guarantees definition, our* $\mathtt{SiGu} = 4\colon \mathsf{aLRhPP}$ *security strengthens TS-(S)UF-4 security by requiring not only* $S_2(lr) = S_4(lr)$ *but* $S_2(lr) = S_3(lr) = S_4(lr)$*. Intuitively, our* $(4\colon \mathsf{aLRhPP}, 1\colon \mathsf{sC}, 0\colon \mathsf{KGsC}, 2\colon \mathsf{KGiC})$-$\mathsf{EUF\text{-}CMA}$ *(resp. -*$\mathsf{SUF\text{-}CMA}$*) is a stronger replacement of TS-UF-4 (resp. TS-SUF-4) security.*

5.2 The ATS Transformation is Better Than Advertised

In addition to the unforgeability hierarchy for non-interactive threshold signature schemes, Bellare, Tessaro, and Zhu [10, 11] propose a transformation, ATS, from TS-(S)UF-3 security to TS-(S)UF-4 security. Although their unforgeability hierarchy does not realize linear relation, as we proved in Section 5.1, we surprisingly find that their ATS transformation is better than advertised. More concretely, while Bellare, Tessaro, and Zhu [10, 11] proved that if a threshold signature can guarantee that all signers who signed a leader request lr only if their respective associated pre-processing tokens included in lr are honestly generated (i.e., $S_2(lr) \subseteq S_3(lr)$, which is implied by $S_2(lr) = S_3(lr)$) except for negligible probability, the transformed threshold signature might lose this guarantee, as TS-(S)UF-4 only requires $S_2(lr) = S_4(lr)$ without the requirement on $S_2(lr) \subseteq S_3(lr)$. Below, we prove that such loss cannot happen. We adapt ATS to our threshold signature syntax (Definition 6) for our new $\mathsf{ATS}^{3:\mathsf{tLRhPP}}_{4:\mathsf{aLRhPP}}$ transformation and prove that it transforms the unforgeability attribute $\mathtt{SiGu} = 3{:}\,\mathsf{tLRhPP}$ to $\mathtt{SiGu} = 4{:}\,\mathsf{aLRhPP}$, which is stronger in terms of trivial forgery (Signer Guarantees) than TS-(S)UF-4 security (see Remark 1).

Theorem 6. *Let* TS *denotes a* (t, n) *threshold signature with* (u, v, w) *rounds. Let* DS *denotes a digital signature scheme. Let* $\mathsf{TS}' = \mathsf{ATS}^{3:\mathsf{tLRhPP}}_{4:\mathsf{aLRhPP}}[\mathsf{TS}, \mathsf{DS}]$ *denotes the* (t, n) *threshold signature with* $(u', v', w') = (\max(1, u), v, w)$ *rounds. If* DS *is* ϵ_{DS}*-SUF-CMA secure and* TS *is* ϵ_{TS}-$(3{:}\,\mathsf{tLRhPP}, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1)$-$\mathtt{UF}_1$ *secure, then* TS' *is* $\epsilon_{\mathsf{TS}'}$-$(4{:}\,\mathsf{aLRhPP}, \mathtt{Corr}_2, \mathtt{KGCh}_2, \mathtt{SiCh}_2)$-$\mathtt{UF}_2$ *secure, where* $\mathtt{Corr}_2 = \mathtt{Corr}_1$*,* $\mathtt{KGCh}_2 = \min(\mathtt{KGCh}_1, 1{:}\,\mathsf{KGaC})$*,* $\mathtt{SiCh}_2 = \mathtt{SiCh}_1$*,* $\mathtt{UF}_2 = \mathtt{UF}_1$ *and*

$$\epsilon_{\mathsf{TS}'} \leq n\epsilon_{\mathsf{DS}} + (n - t)\epsilon_{\mathsf{TS}}$$

6 Automated Analysis of Protocols That Use Threshold Signatures

We develop a formal analysis framework for automatic verification of protocols that implement threshold signature schemes. Specifically, our methodology captures the unforgeability properties of the extended hierarchy introduced in Section 4. First, we summarize relevant background and give a general intuition of modeling threshold signatures in the symbolic model. Second, we instantiate our models for the Tamarin prover [49]. Finally, we present attack-finding case studies conducted with our framework. Because our approach is symbolic, we cannot use it to provide computational proofs, but our approach can automatically find attacks (counterexamples). Furthermore, a symbolic verification result in our approach can be used as a starting point for a computational proof attempt. As of writing, automated symbolic approaches have shown to be more suitable for higher-level protocols than their computational relatives (e.g., EasyCrypt [33]), and provide a much higher level of automation, which enables systematic exploration of alternatives.

6.1 Methodology

Our framework is built for the *symbolic model* of cryptography, which represents protocol messages as abstract terms, and cryptographic functions as equational theories operating on terms. For example, signing a message m with a key k can be expressed as $sign(m, k)$. Using equational theories, we can specify the relation between signing and verification as $verify(sign(m, sk), m, pk(sk)) = true$, where $pk(sk)$ represents public-key derivation from a (secret) key sk.

Security properties in the symbolic model are expressed with some property specification language such as first-order logic. The adversary is typically modeled as an active network adversary, as defined by Dolev and Yao [30]. This means that the adversary has complete control of the network and can intercept, read, modify, or delete messages, but is limited to applying cryptographic functions on known terms.

The Tamarin prover [49] is a state-of-the-art verification tool for the automatic analysis of cryptographic protocols in the symbolic model. Given a protocol specification and a set of properties, the tool works through backwards reasoning to prove the protocol or provide a counterexample representing an attack. Protocols are formalized as collections of *labelled multiset rewriting rules* operating on facts. For example, asymmetric encryption of a message m with a public key pkR can be expressed as follows:

```
rule encrypt_message:
    [ Sender(pkR, m) ]          // Premise
  --[ SendMessage(m) ]->        // Action(s)
    [ Out(aenc(m, pkR)) ]       // Conclusion
```

The rule consists of three parts: a *premise*, *actions*, and a *conclusion*, each containing multisets of facts. Rules represent transitions that replace the facts in the premise with the ones in the conclusion, marking the process in a trace with action facts. The premise determines the required facts for the rule to be applicable, and the conclusion is the outcome of executing it. Our example rule consumes a sender state containing the public key of the receiver and a message, and outputs an encrypted message into the network.

Protocol properties are specified as fragments of first-order logic with timepoints. For example, we can express secrecy of the sent message as follows:

```
lemma secrecy_of_m:
    " All m #i . SendMessage(m)@i ==> not Ex #k. K(m)@k "
```

Informally, this states that for *any* message m that was sent at an arbitrary timepoint $\#i$, there does not exist a timepoint $\#k$ at which the adversary learns m. Since the adversary is unable to break cryptographic primitives, the property holds as long as the encryption key remains secret.

In our formalization of threshold signatures, we make use of TAMARIN's *restrictions* feature. Syntactically, restrictions are similar to security lemmas, but instead of properties, specify restrictions for TAMARIN's search algorithm. This allows us to avoid directly modeling complex cryptographic primitives by discarding traces that violate their intended properties.

For a more comprehensive list of TAMARIN's features and syntax, we refer the reader to [7, 8, 49].

6.2 A Symbolic Model of Threshold Signatures

In Section 4, we proposed a new hierarchy of unforgeability notions for threshold signature schemes. Next, we explain how the attributes in our hierarchy can be formalized in the symbolic model of cryptography for automated analysis of security properties. Note that determining or proving the hierarchy level of any specific signature scheme is out of scope for the framework. Instead, we represent each level of trivial forgery as an abstract signing operation that captures its relevant security properties and model the adversary capabilities by strengthening the default Dolev-Yao adversary.

Functions and Equational Theories. Recall from Section 6.1, that digital signatures in the symbolic model can be specified by an equational theory that defines the association between a message, a signature, and a key pair. In a threshold signature scheme, the group signature is created by aggregating partial signatures from individual signers belonging to a common group. Once at least t signatures are combined, the resulting signature can be verified with the group's public key, without ever constructing the shared secret key.

In the symbolic model, we can represent this with an abstract group entity that, given enough valid partial signatures, signs a message on behalf of its members. We define a function symbol $\mathit{ts\_sign}(m, \mathit{sig}, \mathit{sk}_g)$, which in addition to the message m and the group's signing key sk_g, contains the list of signers sig that provided partial signatures. This allows us to differentiate between *privacy-preserving* and *accountable* threshold signature schemes.

A *privacy-preserving* threshold signature scheme hides the threshold t from outsiders and does not reveal the identities of the parties that produced a valid signature. More specifically, a group's public key and a set of message-signature pairs should not reveal the threshold or the identities of the parties that produced any of the signatures. Privacy can further be divided into *privacy against the public* and *privacy against the signers*. While the former only requires the information to be protected from outsiders, the latter also hides the identities from inside parties.

To represent privacy-preserving signature schemes, we define an equational theory that ignores the list of signers in the verification process. Similar to traditional signatures, the verification succeeds if the function is given the correct message and public key.

$$\mathit{ts\_verify\_private}(\mathit{ts\_sign}(m, s, \mathit{sk}_g), m, \mathit{pk}(\mathit{sk}_g)) = \mathit{true}$$

In contrast, an *accountable* threshold signature scheme reveals the identities of the signing parties for any given message-signature pair. We model this by allowing the signature verifier to check whether a given subset of signers participated in producing a signature.

$$ts\_verify\_accountable(ts\_sign(m, s, sk_g), m, s, pk(sk_g)) = true$$

Any group member or outsider (including the adversary) is allowed to generate a threshold signature for a given key pair. However, no individual signer can produce a valid signature for the group, since the private key is only known by the abstract signing entity. We specify a set of restrictions for determining when the required conditions have been met. These restrictions represent the security guarantees of each level in our hierarchy.

6.3 Modeling Threshold Signatures for the Tamarin Prover

We instantiate our symbolic model of threshold signatures for the Tamarin prover [49]. Specifically, we model a family of (t, n) threshold signature schemes that achieves $(\texttt{SiGu}, \texttt{Corr}, \texttt{KGCh}, \texttt{SiCh})$-$\texttt{UF}$ security for:

- $\texttt{SiGu} \in \{0\colon \mathsf{eM}, 1\colon \mathsf{tM}, 2\colon \mathsf{tLR}, 3\colon \mathsf{tLRhPP}, 4\colon \mathsf{aLRhPP}\}$,
- $\texttt{Corr} \in \{0\colon \mathsf{nC}, 1\colon \mathsf{sC}, 2\colon \mathsf{aC}\}$,
- $\texttt{KGCh} \in \{0\colon \mathsf{KGsC}\}$,
- $\texttt{SiCh} \in \{0\colon \mathsf{SisC}, 1\colon \mathsf{SiaC}, 2\colon \mathsf{SiiC}\}$, and
- $\texttt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$

On a high level, our model consists of three parts: (1) group creation, (2) partial signing, and (3) group signing. We briefly summarize our main design choices and explain the limitations of our framework. All of our models and detailed execution instructions are available at [22].

Group Creation. Groups are created in an iterative process that can terminate with any group size $n \geq 1$ and threshold $1 \leq t \leq n$. We model the finished group as a persistent state that contains its identifier, threshold, size, private key, and a list of its members. The group size and threshold can both be restricted in the security lemmas for modeling a specific scenario or protocol.

Many threshold signature schemes [24, 28, 42] implement Pedersen's distributed key generation (DKG) [46] protocol or a variation of it. The protocol defines how a group of signers can create a shared secret without any individual member being able to construct it. On a high level, the protocol works by each participant choosing a secret value and distributing shares of it to the others using verifiable secret sharing. Once each participant has received a share from every other signer, the shares are combined to form a secret value. Any sufficiently large subset of these values can be combined to construct the shared secret. We refer the reader to Appendix N for additional details.

In our signature model, we abstract away the key generation process by giving each signer an honestly generated key pair (i.e., $\texttt{Corr} < 3\colon \mathsf{mKG}$ and $\texttt{KGCh} = 0\colon \mathsf{KGsC}$) and binding them together through group membership. However, in a separate key derivation model, we capture the missing attributes and verify a well-known weakness of distributed key generation: rogue-key attacks [12, 40]. We explain the attack and proposed solutions in Section 6.4.

Partial Signing. In our model, leader requests are created by a rule that selects an arbitrary subset of signers from a group to sign a message. For $\texttt{SiCh} = 0\colon \mathsf{SisC}$, we model the communication channel between the leader and signers as a secure channel, which the adversary can neither read nor manipulate. For $\texttt{SiCh} = 1\colon \mathsf{SiaC}$, we let the adversary read messages from the channel, and for $\texttt{SiCh} = 2\colon \mathsf{SiiC}$, we give the adversary full control of the channel.

We model state corruption ($\texttt{Corr} \in \{1\colon \mathsf{sC}, 2\colon \mathsf{aC}\}$) by letting signers leak their secrets to the adversary. Furthermore, uncompromised signers can choose to behave *honestly* or *maliciously*. An honest signer only signs a message if it receives a valid pre-processing token. A malicious signer ignores the token and signs any message.

Group Signing. We make use of Tamarin's restrictions to capture the different Signature Guarantee levels in our hierarchy (see Section 4.1 for details). Specifically, for each level of $\texttt{SiGu}$, we specify a signing rule that captures the specific subset of signers that provided partial signatures:

```
rule sign_sigu_i:
    let
        group_signature = ts_sign(msg, signers, skG)
    in
    [ !Group(~idG, t, n, skG, members)
    , Signatures(~idG, msg, leader_request,
                 signature_count, signers, signatures) ]
  --[ /* SiGu-i-specific restrictions */ ]->
    [ Out(<~idG, msg, group_signature>) ]
```

Security Guarantees. Following the hierarchy definitions in Section 4, we model different levels of security guarantees for group signatures.

Unforgeability. Our model captures both existential (EUF-CMA) and strong unforgeability (SUF-CMA). For EUF-CMA, we define a function and a corresponding equational theory, which give the adversary the ability to create multiple signatures that verify under the same group key.

Signer Guarantees. The adversary breaks $\texttt{SiGu} = \texttt{X}$ security by crafting a non-trivial forgery consisting of a message M and a valid signature for it. We summarize the guarantees provided by each level of SiGu security in Appendix M.

6.4 Case Studies

As initial case studies, we use our framework (1) to verify and fix a well-known rogue-key attack in distributed key generation, and (2) to show that the SiGu attribute only guarantees accountability with regards to the leader request for the highest level of strictness ($\texttt{SiGu} = 4{:}\,\mathsf{aLRhPP}$). We provide all our TAMARIN models at [22]. Furthermore, in Appendix O, we give examples of practical application scenarios for threshold signatures and discuss their specific security requirements. We briefly summarize the motivations and results of our initial case studies below.

Rogue-Key Attacks in Distributed Key Generation. As previously mentioned, Pedersen's distributed key generation (DGK) protocol [46], which is commonly implemented for threshold signature schemes, is vulnerable to a rogue-key attack. The attack is caused by a lack of prior association between signers, which allows a dishonest group member to select an arbitrary public key in the key derivation process. By compromising a large enough subset of the group, the adversary can then choose their secret share in a way that negates the other shares and, consequently, gain the ability to sign messages on behalf of the group.

In a small group with a high level of trust between parties, the attack is arguably not very consequential. However, in practice, there are application scenarios where an adversary can realistically achieve a sufficient level of compromise to perform the attack. Namely, if (1) the group of signers is large and the threshold is relatively small (i.e., $t \ll n$), or (2) the group frequently changes members and consequently has to perform re-keying. Both of these cases are often true in e.g., blockchain interoperability protocols [3, 17, 52, 57].

We model Pedersen's DKG protocol [46] for a group of three signers, and show that an adversary can perform the attack when $\texttt{Corr} = 3{:}\,\mathsf{mKG}$ and $\texttt{KGCh} = 2{:}\,\mathsf{KGiC}$. This is by no means a new finding; it has been documented and discussed in detail in the literature [9, 47]. However, modeling it in our framework allows us to verify that proposed mitigation methods (used by recent threshold signature schemes) prevent the attack, even in the presence of a strong adversary model.

Since rogue-key attacks rely on the adversary choosing a public key for which it does not know the corresponding secret key, it can be easily prevented by requiring participants to prove *knowledge of the secret key (KOSK)* [47]. By asking each signer to present a proof of having access to the secret key, a dishonest signer can be caught and the key derivation process can be aborted. This can be achieved through multiple different methods, two of which we confirm with our model (see Appendix N for additional details):

1. *Proof of possession* requires each signer to prove that they have access to the commitment corresponding to a secret value. For example, in FROST [42], each signer computes and distributes a zero-knowledge proof of their secret value.

2. *Key commitment* is a multi-round solution that requires each party to commit to their secret before sending or receiving anything. For example, each signer can be required to send a hash of their public key and only proceed once everyone has done so.

Proof Effort. In total, we analyze three variations of the protocol. First, we model the original version of Pedersen's algorithm to verify that our model captures a rogue-key attack in the default setting. Then, we modify it to create two adaptations that capture the mitigation methods discussed previously. For each model, we define two properties: a sanity check to ensure that it behaves as expected, and a security lemma that captures rogue-key attacks. Out of the six lemmas, TAMARIN proves five automatically with the help of user-defined goal rankings. Specifically, we make use of the *tactics* feature, which allows us to optimize the attack-finding search. Each model is approximately 300 lines of code in size, with an additional 50 lines of tactics. The total running time for the five lemmas is approximately 20 minutes on a 2023 MacBook Pro laptop. For the sixth lemma, we use manual exploration to find a trace that represents an attack. Finding the trace required initially significant effort, but once identified, made it easy to reproduce.

Limited Accountability for Leader Requests. Our second case study uses the signature model to discover a set of traces, which are approximately equivalent to the counterexamples used by Bellare, Tessaro, and Zhu in [10, 11] to show that FROST [42] and FROST2 [24] do not meet the requirements for TS-(S)UF-3 and TS-(S)UF-4 respectively. In threshold signature schemes with $\texttt{Corr} > 0\colon \mathsf{nC}$, $\texttt{SiCh} = 2\colon \mathsf{SiiC}$, and $\texttt{SiGu} \in \{2\colon \mathsf{tLR}, 3\colon \mathsf{tLRhPP}\}$, an adversary can corrupt a subset of $< t$ signers and manipulate the signing process to produce a group signature that seemingly was created by the members included in the leader requests, even though only a subset of them actually provided partial signatures. The attack is only defined for signature schemes with leader requests, since it does not violate the guarantees of weaker schemes.

Rather than modeling any particular protocol to discover these attacks, we write a set of security lemmas, which claim that in schemes with leader requests (i.e., $\texttt{SiGu} \in \{2\colon \mathsf{tLR}, 3\colon \mathsf{tLRhPP}, 4\colon \mathsf{aLRhPP}\}$) the following holds: whenever a group signature is produced, the subset of signers is equivalent to the subset of signers in the corresponding leader request. As expected, this fails for $\texttt{SiGu} \in \{2\colon \mathsf{tLR}, 3\colon \mathsf{tLRhPP}\}$, since the guarantees only limit the number of corrupted signers that are part of the leader request.

Proof Effort. The model uses TAMARIN's built-in pre-processor to generate threshold signature constructions that meet the requirements of the different hierarchy levels (see [22] for details and instructions). In total, this lets us generate 180 variations of it, each corresponding to a specific quadruple of attribute values. For this case study, we analyzed accountability lemmas in 54 signature schemes. Specifically, we generated signatures schemes corresponding to $\texttt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$, $\texttt{Corr} \in \{0\colon \mathsf{nC}, 1\colon \mathsf{sC}, 2\colon \mathsf{aC}\}$, $\texttt{SiCh} \in \{0\colon \mathsf{SisC}, 1\colon \mathsf{SiaC}, 2\colon \mathsf{SiiC}\}$, and $\texttt{SiGu} \in \{2\colon \mathsf{tLR}, 3\colon \mathsf{tLRhPP}, 4\colon \mathsf{aLRhPP}\}$. As in the previous example, each model defines two properties: a sanity check to ensure that it behaves as expected, and a security lemma to verify or falsify the accountability property. The size of our model is approximately 800 lines of code, with an additional 70 lines of tactics. The total running time for all 54 cases (including the sanity checks) in approximately 80 minutes on a 2023 MacBook Pro laptop.

7 Conclusions

We defined a unified syntax for threshold signature schemes and proposed a new hierarchy of unforgeability notions that combines, corrects, and extends previous work. Furthermore, we developed the first systematic, automated analysis framework for protocols deploying threshold signatures. Our methodology and protocol models are freely available from [22] and enable detecting protocol weaknesses caused by subtle differences between the levels of unforgeability in our hierarchy. It can be used for automatic attack finding in the symbolic model of cryptography, or as a general guideline in the selection of threshold signature schemes.

References

[1] Shweta Agrawal, Damien Stehle, and Anshu Yadav. *Round-Optimal Lattice-Based Threshold Signatures, Revisited.* Cryptology ePrint Archive, Paper 2022/634. 2022.

[2] Gennaro Avitabile, Vincenzo Botta, and Dario Fiore. *Extendable Threshold Ring Signatures with Enhanced Anonymity.* Cryptology ePrint Archive, Paper 2022/1568. 2022.

[3] *Axelar.* `https://axelar.network/`. Accessed: September 20, 2024.

[4] Renas Bacho and Julian Loss. *On the Adaptive Security of the Threshold BLS Signature Scheme.* Cryptology ePrint Archive, Paper 2022/534. 2022.

[5] Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. *HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures.* Cryptology ePrint Archive, Paper 2024/280. 2024.

[6] Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. *Twinkle: Threshold Signatures from DDH with Full Adaptive Security.* Cryptology ePrint Archive, Paper 2023/1482. 2023.

[7] David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. “Symbolically Analyzing Security Protocols using Tamarin”. In: *ACM SIGLOG News* 4.4 (2017).

[8] David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. “Tamarin: Verification of Large-Scale, Real World, Cryptographic Protocols”. In: *IEEE Security and Privacy Magazine* (2022).

[9] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. “Randomness Re-Use in Multi-Recipient Encryption Schemeas”. In: *Public Key Cryptography-PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6.* Springer. 2002.

[10] Mihir Bellare, Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. “Better than Advertised Security for Non-interactive Threshold Signatures”. In: *Advances in Cryptology – CRYPTO 2022.* Springer, 2022.

[11] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. *Stronger Security for Non-Interactive Threshold Signatures: BLS and FROST.* Cryptology ePrint Archive, Paper 2022/833. (Accessed: November 21, 2024). 2022.

[12] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. *On the (in)security of ROS.* Cryptology ePrint Archive, Paper 2020/945. 2020.

[13] Alexandra Boldyreva. “Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme”. In: *Public Key Cryptography — PKC 2003.* Springer, 2002.

[14] Dan Boneh and Chelsea Komlo. *Threshold Signatures with Private Accountability.* Cryptology ePrint Archive, Paper 2022/1636. 2022.

[15] Dan Boneh, Ben Lynn, and Hovav Shacham. “Short Signatures from the Weil Pairing”. In: *International conference on the theory and application of cryptology and information security.* Springer. 2001.

[16] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. *UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts.* Cryptology ePrint Archive, Paper 2021/060. 2021.

[17] *Chainflip.* `https://chainflip.io/`. Accessed: September 20, 2024.

[18] Vincent Cheval, Cas Cremers, Alexander Dax, Lucca Hirschi, Charlie Jacomme, and Steve Kremer. *Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses.* Cryptology ePrint Archive, Paper 2022/1314. 2022.

[19] Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. *Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security.* Cryptology ePrint Archive, Paper 2023/1246. 2023.

[20] Cas Cremers, Alexander Dax, and Niklas Medinger. *Keeping Up with the KEMs: Stronger Security Notions for KEMs and Automated Analysis of KEM-based Protocols.* Cryptology ePrint Archive, Paper 2023/1933. 2023.

[21] Cas Cremers and Dennis Jackson. “Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman”. In: *2019 IEEE 32nd Computer Security Foundations Symposium.* IEEE, 2019.

[22] Cas Cremers, Aleksi Peltonen, and Mang Zhao. *Symbolic Verification of Threshold Signatures: Tamarin Models.* `https://github.com/FormalTSS/SymbolicTSSModels`. 2024.

[23] Elizabeth Crites, Chelsea Komlo, and Mary Maller. *Fully Adaptive Schnorr Threshold Signatures.* Cryptology ePrint Archive, Paper 2023/445. 2023.

[24] Elizabeth Crites, Chelsea Komlo, and Mary Maller. *How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures.* Cryptology ePrint Archive, Paper 2021/1375. 2021.

[25] Elizabeth Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. *Snowblind: A Threshold Blind Signature in Pairing-Free Groups.* Cryptology ePrint Archive, Paper 2023/1228. 2023.

[26] Anders Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. “Securing DNSSEC Keys via Threshold ECDSA from Generic MPC”. In: *Computer Security – ESORICS 2020.* Springer, 2020.

[27] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bunz, and Ling Ren. *Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold.* Cryptology ePrint Archive, Paper 2023/598. 2023.

[28] Sourav Das and Ling Ren. *Adaptively Secure BLS Threshold Signatures from DDH and co-CDH.* Cryptology ePrint Archive, Paper 2023/1553. 2023.

[29] Yvo Desmedt and Yair Frankel. “Threshold Cryptosystems”. In: *Advances in Cryptology — CRYPTO’ 89 Proceedings*. Springer, 1990.

[30] Danny Dolev and Andrew Yao. “On the security of public key protocols”. In: *IEEE Transactions on Information Theory* 29.2 (1983).

[31] *drand*. `https://drand.love/`. Accessed: September 20, 2024.

[32] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. “On the Security of Two-Round Multi-Signatures”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019.

[33] *EasyCrypt*. `https://www.easycrypt.info/`. Accessed: October 2, 2024.

[34] Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler. *tlock: Practical Timelock Encryption from Threshold BLS*. Cryptology ePrint Archive, Paper 2023/189. 2023.

[35] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. “hinTS: Threshold Signatures with Silent Setup”. In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.

[36] Rosario Gennaro and Steven Goldfeder. *Fast Multiparty Threshold ECDSA with Fast Trustless Setup*. Cryptology ePrint Archive, Paper 2019/114. 2019.

[37] Rosario Gennaro and Steven Goldfeder. *One Round Threshold ECDSA with Identifiable Abort*. Cryptology ePrint Archive, Paper 2020/540. 2020.

[38] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. “Robust Threshold DSS Signatures”. In: *Advances in Cryptology—EUROCRYPT’96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings 15*. Springer. 1996.

[39] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. “Secure Distributed Key Generation for Discrete-Log Based Cryptosystems”. In: *Advances in Cryptology—EUROCRYPT’99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer. 1999.

[40] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. “Secure Distributed Key Generation for Discrete-Log Based Cryptosystems”. In: *Journal of Cryptology* 20 (2007).

[41] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. “Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’19. London, United Kingdom: ACM, 2019.

[42] Chelsea Komlo and Ian Goldberg. *FROST: Flexible Round-Optimized Schnorr Threshold Signatures*. Cryptology ePrint Archive, Paper 2020/852. 2020.

[43] Wenting Li, Haibo Cheng, Ping Wang, and Kaitai Liang. “Practical Threshold Multi-Factor Authentication”. In: *IEEE Transactions on Information Forensics and Security* 16 (2021).

[44] Yehuda Lindell. *Simple Three-Round Multiparty Schnorr Signing with Full Simulatability*. Cryptology ePrint Archive, Paper 2022/374. 2022.

[45] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. “Accountable-Subgroup Multisignatures: Extended Abstract”. In: *Proceedings of the 8th ACM Conference on Computer and Communications Security*. CCS ’01. Philadelphia, PA, USA: ACM, 2001.

[46] Torben Pryds Pedersen. “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”. In: *Advances in Cryptology – CRYPTO ’91*. Springer, 1992.

[47] Thomas Ristenpart and Scott Yilek. “The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks”. In: *Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26*. Springer. 2007.

[48] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. *ROAST: Robust Asynchronous Schnorr Threshold Signatures*. Cryptology ePrint Archive, Paper 2022/550. 2022.

[49] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. “Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties”. In: *2012 IEEE 25th Computer Security Foundations Symposium*. 2012.

[50] Kiarash Sedghighadikolaei and Attila Altay Yavuz. *A Comprehensive Survey of Threshold Digital Signatures: NIST Standards, Post-Quantum Cryptography, Exotic Techniques, and Real-World Applications*. 2023. arXiv: `2311.05514 [cs.CR]`.

[51] *The ZF FROST Book*. `https://frost.zfnd.org/`. Accessed: September 20, 2024.

[52] *THORChain*. `https://thorchain.org/`. Accessed: September 20, 2024.

[53] Gang Wang and Mark Nixon. *InterTrust: Towards an Efficient Blockchain Interoperability Architecture with Trusted Services*. Cryptology ePrint Archive, Paper 2021/1513. 2021.

[54] Gang Wang, Qin Wang, and Shiping Chen. “Exploring Blockchains Interoperability: A Systematic Survey”. In: *ACM Comput. Surv.* 55.13s (2023).

[55] *Web3Auth*. `https://web3auth.io/docs/infrastructure`. Accessed: September 20, 2024.

[56] *Zengo Wallet*. `https://zengo.com/security-in-depth/`. Accessed: September 20, 2024.

[57] *ZetaChain*. `https://www.zetachain.com/`. Accessed: September 20, 2024.

[58] Guy Zyskind, Avishay Yanai, and Alex “Sandy” Pentland. *Unstoppable Wallets: Chain-assisted Threshold ECDSA and its Applications*. Cryptology ePrint Archive, Paper 2023/832. 2023.

Supplementary Material

In the following sections, we provide supplementary material that includes additional examples, proofs, and clarifying explanations. In Appendix A, we provide additional preliminaries. In Appendix B, we show that, in our syntax, FROST2 [24] is a $(u, v, w) = (2, 1, 0)$ scheme and BLS [13] is a $(u, v, w) = (2, 0, 0)$ scheme. In Appendix C, we define (strong) correctness for our generalized threshold signature syntax. In Appendix D, we provide a counterexample for TS-(S)UF-4 $\not\Rightarrow$ TS-(S)UF-3 to prove that the TS-(S)UF-i security hierarchy in [10, 11] does not provide a linear strength relation. In Appendix E–K, we give proofs for the theorems and corollaries presented in Section 5. In Appendix L, we define a $\mathsf{ATS}^{\mathsf{3:tLRhPP}}_{\mathsf{4:aLRhPP}}[\mathsf{TS}, \mathsf{DS}]$ transformation. In Appendix M, we summarize the security guarantees provided by each level of `SiGu` in our hierarchy. In Appendix N, we explain Pedersen's protocol for distributed key generation and the rogue-key attack that it is vulnerable to. Finally, in Appendix O, we give examples of practical application scenarios for threshold signatures and discuss their security requirements.

A Additional Preliminaries

A.1 Digital Signatures

Definition 8. *A digital signature scheme over message space $\mathcal{M}$, is a tuple of algorithms* $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *as defined below. In this work, we assume $\mathcal{M} = \{0,1\}^\star$, i.e., the set of binary strings with limited length.*

Key Generation: $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}}) \xleftarrow{\$} \mathsf{DS.KGen}()$ *outputs a key pair $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}})$ consisting of a public verification key and private signing key.*

Signing: $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk^{\mathsf{DS}}, m)$ *inputs a signing key sk^{DS} and a message $m \in \mathcal{M}$, and outputs a signature σ.*

Verification: $0/1 \leftarrow \mathsf{DS.Vrfy}(vk^{\mathsf{DS}}, m, \sigma)$ *verifies whether σ is a valid signature over the message m with respect to the verification key vk^{DS} (outputs 1) or not (outputs 0).*

We say a DS is δ-correct, if for every $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}}) \xleftarrow{\$} \mathsf{DS.KGen}()$ and every message $m \in \mathcal{M}$, we have

$$\Pr[0 \leftarrow \mathsf{DS.Vrfy}(vk^{\mathsf{DS}}, m, \mathsf{DS.Sign}(sk^{\mathsf{DS}}, m))] \leq \delta$$

In particular, we call a DS *(perfectly) correct* if $\delta = 0$. Moreover, we say a digital signature scheme DS is *unique*, if for every $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}}) \xleftarrow{\$} \mathsf{DS.KGen}()$ and every message $m \in \mathcal{M}$, there exists at most one signature σ such that

$$\mathsf{DS.Vrfy}(vk^{\mathsf{DS}}, m, \sigma) = 1$$

In terms of the security notation, we recall the standard definitions for *existential* and *strong unforgeability against chosen message attack* ($\mathsf{EUF\text{-}CMA}/\mathsf{SUF\text{-}CMA}$).

```
Game_DS^EUF-CMA(A):
1  $\mathcal{L} \leftarrow \emptyset$
2  $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}}) \xleftarrow{\$} \mathsf{DS.KGen}()$
3  $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\textsc{OSign}}(vk^{\mathsf{DS}})$
4  if $m^\star \in \mathcal{L}$
5      return 0
6  return $[\![\mathsf{DS.Vrfy}(vk^{\mathsf{DS}}, m^\star, \sigma^\star)]\!]$
OSign(m)
7  $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk^{\mathsf{DS}}, m)$
8  $\mathcal{L} \xleftarrow{+} m$
9  return $\sigma$
```

```
Game_DS^SUF-CMA(A):
1  $\mathcal{L} \leftarrow \emptyset$
2  $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}}) \xleftarrow{\$} \mathsf{DS.KGen}()$
3  $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\textsc{OSign}}(vk^{\mathsf{DS}})$
4  if $(m^\star, \sigma^\star) \in \mathcal{L}$
5      return 0
6  return $[\![\mathsf{DS.Vrfy}(vk^{\mathsf{DS}}, m^\star, \sigma^\star)]\!]$
OSign(m)
7  $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk^{\mathsf{DS}}, m)$
8  $\mathcal{L} \xleftarrow{+} (m, \sigma)$
9  return $\sigma$
```

Figure 3: $\mathsf{EUF\text{-}CMA}$ and $\mathsf{SUF\text{-}CMA}$ experiments for $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$.

Definition 9. *Let* $\mathsf{DS} = (\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy})$ *be a digital signature scheme with message space $\mathcal{M}$. We say* DS *is ϵ-*$\mathsf{EUF\text{-}CMA}$ *secure (resp. ϵ-*$\mathsf{SUF\text{-}CMA}$ *secure), if for every adversary $\mathcal{A}$, we have*

$$\epsilon^{\mathsf{euf\text{-}cma}}_{\mathsf{DS}}(\mathcal{A}) := \Pr[\mathrm{Game}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A}) = 1] \leq \epsilon$$

$$\epsilon_{\mathsf{DS}}^{\mathsf{suf\text{-}cma}}(\mathcal{A}) := \Pr[\mathrm{Game}_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}}(\mathcal{A}) = 1] \leq \epsilon$$

where the experiments $\mathrm{Game}_{\mathsf{DS}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A})$ *and* $\mathrm{Game}_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}}(\mathcal{A})$ *are defined in Figure 3.*

B Examples of the Unified Syntax

We provide two examples for how existing threshold signature schemes can be expressed in our unified syntax (see Definition 6). The first one is an echo scheme, and the second one is a non-echo scheme.

Example 1. *The FROST2 construction [24] can be expressed as a* $(2, 1, 0)$*-round threshold signature scheme (see Figure 4) by extending each signer's state with the following temporal variables:*

- $\mathsf{st}_i.\mathcal{L}_{\mathsf{PP}}$*: a list that includes signer i' pre-processing tokens; initialized with* $\mathsf{st}_i.\mathcal{L}_{\mathsf{PP}} \leftarrow \emptyset$*.*
- $\mathsf{st}_i.f$*: a polynomial function; initialized with* $\perp$*.*
- $\mathsf{st}_i.\phi_{(i',k)}, \forall i' \in [n], k \in \{0, \ldots, t-1\}$*: a temporal value; initialized with* $\perp$*.*
- $\mathsf{st}_i.f_i^i$*: a temporal value; initialized with* $\perp$*.*

Example 2. *The BLS construction [13] can be expressed as a* $(2, 0, 0)$*-round threshold signature scheme (see Figure 5) by extending each signer's state with the following temporal variables:*

- $\mathsf{st}_i.a_{(i',k)}, \forall k \in \{0, \ldots, t-1\}$*: a temporal value; initialized with* $\perp$*.*
- $\mathsf{st}_i.s_{(i',i)}, \forall i' \in [n]$*: a temporal value; initialized with* $\perp$*.*

C Correctness Definition

In order to simplify the correctness definition for a (t, n)-threshold signature scheme TS with (u, v, w) rounds, we use $(\{m_i\}_{i\in[n]}, \mathsf{trans}) \xleftarrow{\$} \mathsf{KGen}(\{S_i\}_{i\in[n]})$ to denote an honest key generation over authenticated channels, i.e., the following sequential execution if $u \geq 1$:

$$\begin{aligned}
\mathop{\Big\|}_{i'\in[n]\setminus\{i\}} m_{(i,i')}^{(1)} &\xleftarrow{\$} \mathsf{KGen}^{(0)}(S_i), \quad \forall i \in [n] \\
\mathop{\Big\|}_{i'\in[n]\setminus\{i\}} m_{(i,i')}^{(y+1)} &\xleftarrow{\$} \mathsf{KGen}^{(y)}(S_i, \mathop{\Big\|}_{i'\in[n]\setminus\{i\}} m_{(i',i)}^{(y)}), \quad \forall i \in [n], y \in [u-1] \\
m_i &\xleftarrow{\$} \mathsf{KGen}^{(u)}(S_i, \mathop{\Big\|}_{i'\in[n]\setminus\{i\}} m_{(i',i)}^{(u)}), \quad \forall i \in [n] \\
\mathsf{trans} &\leftarrow \mathop{\Big\|}_{\substack{i\in[n] \\ i'\in[n]\setminus\{i\} \\ 0\leq y\leq u-1}} m_{(i,i')}^{(y)}
\end{aligned}$$

or the following execution if $u = 0$:

$$\begin{aligned}
m_i &\xleftarrow{\$} \mathsf{KGen}^{(u)}(S_i), \quad \forall i \in [n] \\
\mathsf{trans} &\leftarrow \mathbf{0}
\end{aligned}$$

Moreover, we use $(\{\varsigma_i\}_{i\in lr.SS}, \mathsf{trans}) \xleftarrow{\$} \mathsf{Sign}(lr)$ to denote an honest interaction among signers $i \in lr.SS$ for signing a message $lr.m$ using pre-processing tokens $lr.\mathsf{PP}(i)$, i.e., the following sequential execution if $w \geq 1$:

$$\begin{aligned}
\mathop{\Big\|}_{i'\in lr.SS\setminus\{i\}} m_{(i,i')}^{(0)} &\xleftarrow{\$} \mathsf{Sign}^{(0)}(S_i, lr), \quad \forall i \in lr.SS \\
\mathop{\Big\|}_{i'\in lr.SS\setminus\{i\}} m_{(i,i')}^{(y+1)} &\xleftarrow{\$} \mathsf{Sign}^{(y)}(S_i, \mathop{\Big\|}_{i'\in lr.SS\setminus\{i\}} m_{(i',i)}^{(y)}), \quad \forall i \in lr.SS, \forall y \in [w-1] \\
\varsigma_i &\xleftarrow{\$} \mathsf{Sign}^{(w)}(S_i, \mathop{\Big\|}_{i'\in lr.SS\setminus\{i\}} m_{(i',i)}^{(w)}), \quad \forall i \in lr.SS \\
\mathsf{trans} &\leftarrow \mathop{\Big\|}_{\substack{i\in lr.SS \\ i'\in lr.SS\setminus\{i\} \\ 0\leq y\leq w-1}} m_{(i,i')}^{(y)}
\end{aligned}$$

```
$\mathsf{KGen}^{(0)}(\mathsf{st}_i)$:
$a_{(i,0)}, \dots, a_{(i,t-1)} \xleftarrow{\$} \mathbb{Z}_q$
Define function $f_i(x) = \sum_{j=0}^{t-1} a_{(i,j)} x^j$
$\mathsf{st}_i.f \leftarrow f$
$k_i \xleftarrow{\$} \mathbb{Z}_q$; $R_i \leftarrow g^k$
// $\Phi$ is a context string to prevent replay attacks.
$c_i \leftarrow H(i, \Phi, g^{a(i,0)}, R_i)$
$\mu_i \leftarrow k_i + a_{(i,0)} \cdot c_i$; $\sigma_i \leftarrow (R_i, \mu_i)$
foreach $k \in \{0, \dots, t-1\}$ do
    $\phi_{(i,k)} \leftarrow g^{a(i,k)}$
$\mathsf{st}_i.\phi_{(i,0)} \leftarrow \phi_{(i,0)}$
$C_i \leftarrow (\phi_{(i,0)}, \dots, \phi_{(i,t-1)})$
foreach $i' \in [n] \setminus \{i\}$
    $m_{(i,i')} \leftarrow (C_i, \sigma_i)$
return $\|_{i' \in [n] \setminus \{i\}} m_{(i,i')}$

$\mathsf{KGen}^{(1)}(\mathsf{st}_i, m)$
parse $\|_{l \in [n] \setminus \{i\}} (C_l, \sigma_l) \leftarrow m$
foreach $l \in [n] \setminus \{i\}$
    $(\phi_{(l,0)}, \dots, \phi_{(l,t-1)}) \leftarrow C_l$
    $(R_l, \mu_l) \leftarrow \sigma_l$
    $c_l \leftarrow H(l, \Phi, \phi_{(l,0)}, R_l)$
    req $R_l = g^{\mu_l} \cdot \phi_{(l,0)}^{-c_l}$
    foreach $k \in \{0, \dots, t-1\}$ do
        $\mathsf{st}_i.\phi_{(l,k)} \leftarrow \phi_{(l,k)}$
foreach $i' \in [n] \setminus \{i\}$
    $m_{(i,i')} \leftarrow (l, \mathsf{st}_i.f(l))$
$\mathsf{st}_i.f_i^i \leftarrow \mathsf{st}_i.f(i)$; $\mathsf{st}_i.f \leftarrow \bot$
return $\|_{i' \in [n] \setminus \{i\}} m_{(i,i')}$

$\mathsf{KGen}^{(2)}(\mathsf{st}_i, m)$
parse $\|_{l \in [n] \setminus \{i\}} (i, f_l^i) \leftarrow m$
foreach $l \in [n] \setminus \{i\}$
    req $g^{f_l^i} = \prod_{k=0}^{t-1} (\mathsf{st}_i.\phi_{(l,k)})^{(i^k \bmod q)}$
    foreach $k \in [t-1]$
        $\mathsf{st}_i.\phi_{(l,k)} \leftarrow \bot$
$\mathsf{st}_i.sk \leftarrow \mathsf{st}_i.f_i^i + \sum_{l \in [n] \setminus \{i\}} f_l^i$
$\mathsf{st}_i.gvk \leftarrow \prod_{i' \in [n]} \mathsf{st}_i.\phi_{(i',0)}$
$m' \leftarrow (\mathsf{st}_i.vk, \mathsf{st}_i.\phi_{(i,0)})$
$\mathsf{st}_i.f_i^i \leftarrow \bot$
foreach $i' \in [n]$ do
    $\mathsf{st}_i.VK[i'] \leftarrow \mathsf{st}_i.\phi_{(i',0)}$
    $\mathsf{st}_i.\phi_{(i',0)} \leftarrow \bot$
return $m'$

$\mathsf{VkAgg}(\mathsf{st}_0, \{m_i\}_{i \in [n]})$
foreach $i \in [n]$
    parse $\{vk_i, \phi_{(i,0)}\} \leftarrow m_i$
    $\mathsf{st}_0.VK[i] \leftarrow vk_i$
$\mathsf{st}_0.gvk \leftarrow \prod_{i \in [n]} \phi_{(i,0)}$
return $\mathsf{st}_0.gvk$

$\mathsf{SPP}(\mathsf{st}_i)$
$(d_i, e_i) \xleftarrow{\$} \mathbb{Z}_q^\star \times \mathbb{Z}_q^\star$; $(D_i, E_i) \leftarrow (g^{d_i}, g^{e_i})$
$\mathsf{st}_i.\mathcal{L}_{\mathsf{PP}} \xleftarrow{+} (D_i, E_i)$
return $(i, D_i, E_i)$

$\mathsf{LPP}(\mathsf{st}_0, m)$
parse $(i, D, E) \leftarrow m$
req $i \in [n]$ and $D, E \in \mathbb{G}^\star$
$\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i] \xleftarrow{+} (D, E)$
return

$\mathsf{LR}(\mathsf{st}_0, SS, m)$
req $\forall i \in SS : \mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i] \neq \emptyset$
$lr.SS \leftarrow SS$; $lr.m \leftarrow m$
foreach $i \in SS$ do
    pick some $pp_i$ from $\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i]$
    $lr.\mathsf{PP}(i) \leftarrow pp_i$; $\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i] \xleftarrow{-} pp_i$
return $lr$

$\mathsf{Sign}^{(0)}(S_i, lr)$
req $lr.m \in \{0,1\}^\star$ and $lr.\mathsf{PP}(i) \in \mathsf{st}_i.\mathcal{L}_{\mathsf{PP}}$
foreach $i' \in lr.SS$
    parse $(D_{i'}, E_{i'}) \leftarrow lr.\mathsf{PP}(i')$
    req $(D_{i'}, E_{i'}) \in \mathbb{G}^\star \times \mathbb{G}^\star$
$\rho \leftarrow \mathsf{H}_1(\mathsf{st}_i.gvk, lr)$; $R \leftarrow \prod_{i' \in lr.SS} D_{i'} \cdot (E_{i'})^{\rho_{i'}}$; $c \leftarrow$
$\mathsf{H}_2(\mathsf{st}_i.gvk, R, lr.m)$
$z_i \leftarrow d_i + (e_i \cdot \rho_i) + \lambda_i \cdot \mathsf{st}_i.sk \cdot c$ // $\lambda_i$ is the $i$-th Lagrange
coefficient determined by $lr.SS$
$\mathsf{st}_i.\mathcal{L}_{\mathsf{PP}} \xleftarrow{-} lr.\mathsf{PP}(i)$
return $z_i$

$\mathsf{SigAgg}(\mathsf{st}_0, lr, \{\varsigma_i\}_{i \in lr.SS})$
foreach $i \in lr.SS$
    parse $(D_i, E_i) \leftarrow lr.\mathsf{PP}(i)$
    req $(D_i, E_i) \in \mathbb{G}^\star \times \mathbb{G}^\star$
$\rho \leftarrow \mathsf{H}_1(\mathsf{st}_0.gvk, lr)$
$R \leftarrow \prod_{i \in lr.SS} D_i \cdot (E_i)^\rho$; $z \leftarrow \sum_{i \in lr.SS} \varsigma_i$
return $(R, z)$

$\mathsf{Vrfy}(gvk, m, \sigma)$
parse $(R, z) \leftarrow \sigma$
$c \leftarrow \mathsf{H}_2(gvk, R, m)$
return $[\![R \cdot gvk^c = g^z]\!]$

$\mathsf{SVrfy}(gvk, lr, \sigma)$
parse $(R, z) \leftarrow \sigma$
foreach $i \in lr.SS$
    parse $(D_i, E_i) \leftarrow lr.\mathsf{PP}(i)$
    req $(D_i, E_i) \in \mathbb{G}^\star \times \mathbb{G}^\star$
$\rho \leftarrow \mathsf{H}_1(gvk, lr)$; $c \leftarrow \mathsf{H}_2(gvk, R', m)$
$R' \leftarrow \prod_{i \in lr.SS} D_i \cdot (E_i)^\rho$
return $[\![R = R']\!]$ and $[\![R \cdot gvk^c = g^z]\!]$
```

Figure 4: The FROST2 construction [24] is a $(2, 1, 0)$-round threshold signature scheme. We use $\mathbb{G}$ to denote the underlying cyclic group with generator g and prime order q.

```
$\mathsf{KGen}^{(0)}(\mathsf{st}_i)$:
$a_{(i,0)}, ..., a_{(i,t-1)}, b_{(i,0)}, ..., b_{(i,t-1)} \xleftarrow{\$} \mathbb{Z}_q$
Define function $f_i(x) = \sum_{j=0}^{t-1} a_{(i,j)} x^j$
Define function $f'_i(x) = \sum_{j=0}^{t-1} b_{(i,j)} x^j$
foreach $k \in \{0, ..., t-1\}$ do
    $C_{(i,k)} \leftarrow g^{a_{(i,k)}} h^{b_{(i,k)}} \mod p$
$C_i \leftarrow \{C_{(i,k)}\}_{k \in \{0,...,t-1\}}$
foreach $i' \in [n]$ do
    $s_{(i,i')} \leftarrow f_i(i') \mod q$
    $s'_{(i,i')} \leftarrow f'_i(i') \mod q$
$\mathsf{st}_i.sk \leftarrow a_{(i,0)}$
foreach $k \in \{0, ..., t-1\}$ do
    $\mathsf{st}_i.a_{(i,k)} \leftarrow a_{(i,k)}$
return $\|_{i' \in [n] \setminus \{i\}} \left(s_{(i,i')}, s'_{(i,i')}, C_i\right)$

$\mathsf{KGen}^{(1)}(\mathsf{st}_i, m)$
parse $\|_{i' \in [n] \setminus \{i\}} \leftarrow \left(s_{(i',i)}, s'_{(i',i)}, C_{i'}\right) \leftarrow m$
foreach $i' \in [n] \setminus \{i\}$
    parse $\{C_{(i',k)}\}_{k \in \{0,...,t-1\}} \leftarrow C_{i'}$
    req $g^{s_{(i',i)}} h^{(s'_{i',i})} = \prod_{k=0}^{t-1} (C_{(i',k)})^{i^k} \mod p$
    $\mathsf{st}_i.s_{(i',i)} \leftarrow s_{(i',i)}$
foreach $k \in \{0, ..., t-1\}$ do
    $A_{(i,k)} \leftarrow g^{\mathsf{st}_i.a_{(i,k)}} \mod p$
    $\mathsf{st}_i.a_{(i,k)} \leftarrow \perp$
    $m_{(i,i')} \leftarrow \|_{k \in \{0,...,t-1\}} A_{(i,k)}$
return $\|_{i' \in [n] \setminus \{i\}} m_{(i,i')}$

$\mathsf{KGen}^{(2)}(\mathsf{st}_i, m)$
parse $\|_{i' \in [n] \setminus \{i\}, k \in \{0,...,t-1\}} A_{(i',k)} \leftarrow m$
foreach $i' \in [n] \setminus \{i\}$
    req $g^{\mathsf{st}_i.s_{(i',i)}} = \prod_{k=0}^{t-1} (A_{(i',k)})^{i^k} \mod p$
    $\mathsf{st}_i.s_{(i',i)} \leftarrow \perp$
    $\mathsf{st}_i.VK[i'] \leftarrow A_{(i',0)}$
$\mathsf{st}_i.gvk \leftarrow \prod_{i' \in [n]} A_{(i',0)} \mod p$
return $A_{(i,0)}$

$\mathsf{VkAgg}(\mathsf{st}_0, \{m_i\}_{i \in [n]})$
foreach $i \in [n]$
    $\mathsf{st}_0.VK[i] \leftarrow m_i$
$\mathsf{st}_0.gvk \leftarrow \prod_{i \in [n]} m_i \mod p$
return $\mathsf{st}_0.gvk$

$\mathsf{SPP}(\mathsf{st}_i)$
return $\perp$

$\mathsf{LPP}(\mathsf{st}_0, m)$
return

$\mathsf{LR}(\mathsf{st}_0, SS, m)$
$lr.SS \leftarrow SS$; $lr.m \leftarrow m$
return $lr$

$\mathsf{Sign}^{(0)}(S_i, lr)$
req $lr.m \in \{0,1\}^\star$ and $i \in lr.SS$
$\varsigma_i \leftarrow \mathsf{H}_1(lr.m)^{\mathsf{st}.sk}$
return $\varsigma_i$

$\mathsf{SigAgg}(\mathsf{st}_0, lr, \{\varsigma_i\}_{i \in lr.SS})$
foreach $i \in lr.SS$ do
    req $e(\mathsf{st}_0.VK[i], \mathsf{H}_1(lr.m)) = e(g, \varsigma_i)$
$\sigma \leftarrow \prod_{i \in lr.SS} (\varsigma_i)^{\lambda_i}$ // $\lambda_i$ is the $i$-th Lagrange coefficient determined by $lr.SS$
return $\sigma$

$\mathsf{Vrfy}(gvk, m, \sigma)$
return $[\![ e(gvk, \mathsf{H}_1(m)) = e(g, \sigma) ]\!]$
```

Figure 5: The BLS construction [13] is a $(2, 0, 0)$-round threshold signature scheme. p is a prime order and g is an element of order q in $\mathbb{Z}_p^\star$, where q is a large prime dividing $p - 1$. h is an element in the subgroup of $\mathbb{Z}_p^\star$ that is generated by g. $e : \mathbb{Z}_p^\star \times \mathbb{Z}_p^\star \to \mathbb{G}_T$ is a bilinear map for some cyclic group $\mathbb{G}_T$ with prime order p.

```
$\mathrm{Game}^{\mathsf{Corr}}_{\mathsf{TS}}(\mathcal{A})$ and $\mathrm{Game}^{\mathsf{sCorr}}_{\mathsf{TS}}(\mathcal{A})$:
    $\mathsf{win} \leftarrow \mathsf{false}$
    // key generation and group key aggregation
    $(\{m_i\}_{i\in[n]}, \mathsf{trans}) \xleftarrow{\$} \mathsf{KGen}(\{S_i\}_{i\in[n]})$
    $gvk \leftarrow \mathsf{VkAgg}(L, \{m_i\}_{i\in[n]})$
    $() \leftarrow \mathcal{A}^{\mathrm{OPP},\mathrm{OSign}}(\{m_i\}_{i\in[n]}, \mathsf{trans}, \mathcal{O})$
    return $\mathsf{win}$

$\mathrm{OPP}(i)$ // signer $i$ generates pre-processing tokens
    req $i \in [n]$ and $v = 1$ // only for echo scheme
    $pp \xleftarrow{\$} \mathsf{SPP}(S_i)$; req $pp \neq \bot$
    $\mathsf{LPP}(L, pp)$
    return $pp$

$\mathrm{OSign}(m, SS)$ // signers sign messages
    req $SS \subseteq [n]$ and $|SS| \geq t$
    $lr \xleftarrow{\$} \mathsf{LR}(L, SS, m)$
    req $lr \neq \bot$ // leader must produce a valid leader request
    if $lr.m \neq m$ or $lr.SS \neq SS$ then
        $\mathsf{win} \leftarrow \mathsf{true}$
    $(\{\varsigma_i\}_{i\in lr.SS}, \mathsf{trans}) \xleftarrow{\$} \mathsf{Sign}(lr)$
    $\sigma \leftarrow \mathsf{SigAgg}(L, lr, \{\varsigma_i\}_{i\in lr.SS})$
    if $\mathsf{Vrfy}(gvk, m, \sigma) = \mathsf{false}$ then // for $\mathrm{Game}^{\mathsf{Corr}}_{\mathsf{TS}}$
    if $\mathsf{SVrfy}(gvk, lr, \sigma) = \mathsf{false}$ then // for $\mathrm{Game}^{\mathsf{sCorr}}_{\mathsf{TS}}$
        $\mathsf{win} \leftarrow \mathsf{true}$
    return $(lr, \{\varsigma_i\}_{i\in lr.SS}, \mathsf{trans}, \sigma)$
```

Figure 6: The (strong) correctness game $\mathrm{Game}^{\mathsf{Corr}}_{\mathsf{TS}}(\mathcal{A})$ and $\mathrm{Game}^{\mathsf{sCorr}}_{\mathsf{TS}}(\mathcal{A})$ for an adversary $\mathcal{A}$ that breaks TS.

or the following execution if $w = 0$:

$$\varsigma_i \xleftarrow{\$} \mathsf{Sign}^{(0)}(S_i, lr), \quad \forall i \in lr.SS$$
$$\mathsf{trans} \leftarrow \mathbf{0}$$

Following the (strong) correctness definition in [10, 11], we define (strong) correctness for our generalized threshold signature as follows:

Definition 10 (Correctness)**.** *Let* $\mathsf{TS} = (\mathsf{KGen}, \mathsf{VkAgg}, \mathsf{SPP}, \mathsf{LPP}, \mathsf{LR}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Vrfy})$ *denote a* (t, n)*-threshold signature scheme with* (u, v, w) *rounds. We say* TS *is* δ-correct *(resp.* strongly-correct*) if the below defined advantage for all adversaries against the game* $\mathrm{Game}^{\mathsf{Corr}}_{\mathsf{TS}}$ *(resp.* $\mathrm{Game}^{\mathsf{sCorr}}_{\mathsf{TS}}$*) in Figure 6 is bounded by* δ*, i.e.,*

$$\mathsf{Adv}^{\mathsf{XXX}}_{\mathsf{TS}}(\mathcal{A}) := \Pr[\mathrm{Game}^{\mathsf{XXX}}_{\mathsf{TS}}(\mathcal{A}) = 1] \leq \delta, \quad \mathsf{XXX} \in \{\mathsf{Corr}, \mathsf{sCorr}\}$$

We say TS *is* (perfectly) correct *(resp.* strongly-correct*) if* $\delta = 0$.

D Counterexample: TS-(S)UF-4 $\not\Rightarrow$ TS-(S)UF-3

First, we recall necessary definitions and technical background from [10, 11] in D.1, and explain the rationale behind our counterexample in D.2. Then, following the syntax in [10, 11], we depict a non-interactive (t, n)-threshold signature scheme Π in Figure 7. Finally, we prove that Π is a counterexample for TS-UF-4 $\not\Rightarrow$ TS-UF-3 in Theorem 2 and 3, and for TS-SUF-4 $\not\Rightarrow$ TS-SUF-3 in Theorem 4 and 5. We summarize our results in Corollary 1.

D.1 Technical Background

The unforgeability models in [10, 11] assume static corruption, secure key generation channels, and insecure signing channels (i.e., $(\mathtt{Corr}, \mathtt{KGCh}, \mathtt{SiCh}) = (1{:}\,\mathsf{sC}, 0{:}\,\mathsf{KGsC}, 2{:}\,\mathsf{SiiC})$ in our model). Furthermore, in contrast to our syntax, they only consider threshold signature schemes with $(u, v, 0)$ rounds and use a single algorithm Kg to denote honest key generation over secure channels. The output of Kg is the group verification gvk, some auxiliary information, and all signers' secret keys $sk_1, \ldots, sk_n$. Moreover, the algorithms in [10, 11] define states explicitly, i.e., an algorithm that inputs a state will also explicitly output the state. The models include the following variables:

- CS denotes the corrupted (and therefore malicious) signers; equivalent to $\mathcal{L}_{\mathsf{MS}}$ in our model.
- PP_i denotes the pre-processing tokens that were honestly generated by an honest signer i; equivalent to $\mathcal{D}_{\mathsf{PP}}[i]$ in our model.
- HS denotes the set of honest signers; equivalent to $\mathcal{L}_{\mathsf{HS}}$ in our model.
- $S_2(lr)$ denotes the honest signers who have signed the leader request lr; equivalent to $\mathcal{D}_2[lr]$ in our model.

- $S_3(lr) = \{i \in HS \cap lr.SS : lr.\mathsf{PP}(i) \in \mathsf{PP}_i\}$ denotes the set of honest signers i that are included in the leader request lr and are associated with honestly generated pre-processing tokens; equivalent to $\mathcal{D}_3[lr]$ in our model.
- $S_4(lr) = HS \cap lr.SS$ denotes the set of honest signers i that are included in the leader request; equivalent to $\mathcal{D}_4[lr]$ in our model.

Trivial forgery for TS-(S)UF-i with $i \in \{2, 3, 4\}$ is formally defined [10, 11, Section 3.2] as follows:

- $\mathtt{tf}_2(lr) \Leftrightarrow |S_2(lr)| \geq t - |CS|$.
- $\mathtt{tf}_3(lr) \Leftrightarrow \mathtt{tf}_2(lr)$ **and** $S_2(lr) = S_3(lr)$.
- $\mathtt{tf}_4(lr) \Leftrightarrow \mathtt{tf}_2(lr)$ **and** $S_2(lr) = S_4(lr)$.
- $\mathtt{tsf}_2(lr, gvk, sig) \Leftrightarrow \mathtt{tf}_2(lr)$ **and** $\mathsf{SVrfy}(gvk, lr, sig)$
- $\mathtt{tsf}_3(lr, gvk, sig) \Leftrightarrow \mathtt{tf}_3(lr)$ **and** $\mathsf{SVrfy}(gvk, lr, sig)$
- $\mathtt{tsf}_4(lr, gvk, sig) \Leftrightarrow \mathtt{tf}_4(lr)$ **and** $\mathsf{SVrfy}(gvk, lr, sig)$

Intuitively, a trivial forgery that triggers $S_2(lr) = S_4(lr)$ does not necessarily trigger $S_2(lr) = S_3(lr)$, although $S_3(lr) \subseteq S_4(lr)$ for all lr. Indeed, the trick behind our counterexample Π is to trigger $S_4(lr) \not\subseteq S_3(lr)$ for some lr.

D.2 The Rationale Behind TS-UF-4 $\not\Rightarrow$ TS-UF-3

The rationale behind our counterexample is that the ranges of $S_2(lr)$ in $\mathtt{tf}_3(lr)$ and $\mathtt{tf}_4(lr)$ are incomparable. Note that $\mathtt{tf}_3(lr)$ and $\mathtt{tf}_4(lr)$ respectively require that

$$S_2(lr) = S_3(lr) \Leftrightarrow S_3(lr) \subseteq S_2(lr) \subseteq S_3(lr)$$
$$S_2(lr) = S_4(lr) \Leftrightarrow S_4(lr) \subseteq S_2(lr) \subseteq S_4(lr)$$

Recall that $S_3(lr) \subseteq S_4(lr)$ holds for every lr. We observe that the upper bound requirement of $S_2(lr)$ in $\mathtt{tf}_3(lr)$ is tighter than the one in $\mathtt{tf}_4(lr)$, while the lower bound requirement of $S_2(lr)$ in $\mathtt{tf}_4(lr)$ is tighter than the one in $\mathtt{tf}_3(lr)$. Consequently, while [11, Appendix A] uses the $\mathsf{RTS}_3[\mathsf{DS}]$ example to prove TS-UF-3 $\not\Rightarrow$ TS-UF-4 by triggering "upper bound separation" $S_2(lr) \subseteq S_3(lr) \subsetneq S_4(lr)$, our example Π is used to prove TS-UF-4 $\not\Rightarrow$ TS-UF-3 by triggering "lower bound separation" $S_3(lr) \subsetneq S_4(lr) \subseteq S_2(lr)$.

D.3 The Definition of Π

In Figure 7, we depict our threshold signature construction Π for (t, n) with $2 \leq t \leq n$. Below, we give additional details.

States. Our threshold signature Π requires the following additional state values:

- $\mathsf{st}_i.\mathsf{ctr}$ for all $i \in [n]$: the signer state st_i includes a counter $\mathsf{st}_i.\mathsf{ctr}$, which counts the pre-processing tokens produced by i; initialized with 0.

Algorithms. Π makes use of a digital signature $\mathsf{DS} = (\mathsf{DS.Kg}, \mathsf{DS.Sign}, \mathsf{DS.Vf})$:

- The key generation algorithm $\Pi.\mathsf{Kg}$ executes $\mathsf{DS.Kg}$ of the underlying digital signature scheme DS n times to obtain a key pair (vk_i, sk_i) for every $i \in [n]$. It sets the group verification key vk to $(vk_1, \ldots, vk_n)$ and returns $(vk, \epsilon, sk_1, \ldots, sk_n)$.
- The signer pre-processing algorithm $\Pi.\mathsf{SPP}$ inputs the signer's state st, increments the counter $\mathsf{st.ctr}$ by 1, and returns the (incremented) counter $\mathsf{st.ctr}$ and the full state st.
- The leader pre-processing algorithm[6] $\Pi.\mathsf{LPP}$ inputs a signer index i, a pre-processing token pp, and the leader state st_0. The leader sate st_0 stores pp into the $\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i]$ locally. Finally, the leader state st_0 is returned.

[6]We note that [11] defines the input of the LPP algorithm in two ways: Sometimes it only contains the pre-processing token pp and the leader state st_0 (e.g., [11, Figure 1]), other times it also includes the signer index i (e.g., [11, Figure 4]). Here, we use the syntax in which LPP has three inputs. However, the two definitions are easily exchangeable, as we can always include the index of signer i into the pre-processing token pp by setting $\mathsf{pp}' = (i, \mathsf{pp})$. Choosing either definition does not influence the security analyses or counterexamples in this work.

```
$\Pi.\mathsf{Kg}$ // the centralized key generation algorithm
for $i = 1, ..., n$
    $(vk_i, sk_i) \xleftarrow{\$} \mathsf{DS.Kg}$
$vk \leftarrow (vk_1, ..., vk_n)$
return $(vk, \epsilon, sk_1, ..., sk_n)$

$\Pi.\mathsf{SPP}(\mathsf{st})$
$\mathsf{st.ctr} \leftarrow \mathsf{st.ctr} + 1$ // the counter $\mathsf{st.ctr}$ was initialized with 0
return $(\mathsf{st.ctr}, \mathsf{st})$

$\Pi.\mathsf{LPP}(i, \mathsf{pp}, \mathsf{st}_0)$
$\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i] \xleftarrow{+} \mathsf{pp}$
return $\mathsf{st}_0$

$\Pi.\mathsf{LR}(M, SS, \mathsf{st}_0)$
$lr.m \leftarrow M,\ lr.SS \leftarrow SS$
for $i \in lr.SS$ do
    pick some $\mathsf{pp} \in \mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i]$
    $lr.\mathsf{PP}(i) \leftarrow \mathsf{pp}$; $\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i] \xleftarrow{-} \mathsf{pp}$
return $(lr, \mathsf{st}_0)$

$\Pi.\mathsf{PS}(lr, \mathsf{st})$// signer's signing algorithm, equals our $\mathsf{Sign}^{(0)}$
if $(\mathsf{st}.id \notin lr.SS$ or $|lr.SS| < t)$ then return $(\perp, \mathsf{st})$
if $(lr.\mathsf{PP}(\mathsf{st}.id) \notin \mathbb{N}_+)$ then return $(\perp, \mathsf{st})$
$psig \xleftarrow{\$} \mathsf{DS.Sign}(\mathsf{st}.sk, lr)$
return $(psig, \mathsf{st})$

$\Pi.\mathsf{Agg}(lr, \{psig_i\}_{i \in lr.SS}, \mathsf{st}_0)$
$sig \leftarrow (lr, lr.SS, \{psig_i\}_{i \in lr.SS})$
return $(sig, \mathsf{st}_0)$

$\Pi.\mathsf{Vf}(vk, M, sig)$
$(vk_1, .., vk_n) \leftarrow vk,\ (lr, F, \{psig_i\}_{i \in F}) \leftarrow sig$
if $(lr.m \neq M$ or $F \not\subseteq lr.SS)$ then return false
$T \leftarrow \{i \in F : \mathsf{DS.Vf}(vk_i, lr, psig_i)\}$
return $(T = lr.SS = F$ and $|T| \geq t)$

$\Pi.\mathsf{SVf}(vk, lr, sig)$
$(lr', F, \{psig_i\}_{i \in F}) \leftarrow sig$
return $(\Pi.\mathsf{Vf}(vk, lr.m, sig)$ and $lr = lr')$
```

Figure 7: Our threshold signature constructions Π. $\mathbb{N}_+ = \{1, \dots\}$ denotes the set of positive natural numbers. The text highlighted with a gray rectangle indicates the trick for not achieving TS-UF-3 security.

- The leader request algorithm $\Pi.\mathsf{LR}$ inputs a message M, a set of signers SS, and the leader state st_0. It then creates a new leader request by setting the message $lr.m$ to M, the set of signers $lr.SS$ to SS, and the map $lr.\mathsf{PP}(i)$ to some pre-processing token $\mathsf{pp}_i \in \mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i]$ for every $i \in lr.SS$. The selected pp_i is then removed from $\mathsf{st}_0.\mathcal{D}_{\mathsf{PP}}[i]$. Finally, it returns the created leader request lr and the leader state st_0.
- The partial signing algorithm $\Pi.\mathsf{PS}$ (which equals to our $\mathsf{Sign}^{(0)}$ algorithm) inputs a leader request lr and a signer state st. If the signer index $\mathsf{st}.id$ is not included in the set $lr.SS$ or the size of set $lr.SS$ is smaller than the threshold t, it returns $\perp$ and the signer state st. The same is true if the pre-processing token $lr.\mathsf{PP}(\mathsf{st}.id)$ of the signer $\mathsf{st}.id$ is not a positive natural number. Otherwise, it runs the signing algorithm $\mathsf{DS.Sign}$ of the underlying digital signature DS by using the signing key $\mathsf{st}.sk$ for signing the leader request lr. Finally, it returns the produced partial signature $psig$ and the signer state st.
- The aggregation algorithm $\Pi.\mathsf{Agg}$ inputs a leader request lr, a collection of partial signatures $psig_i$ of every signer $i \in lr.SS$, and the leader state st_0. It generates a group signature sig by merging the leader request lr, the set of signer $lr.SS$, and the collection of partial signatures $\{psig_i\}_{i \in lr.SS}$.

Finally, it returns sig and the leader state st_0.

- The verification algorithm $\Pi.\mathsf{Vf}$ inputs a group verification key vk, a message M, and a group signature sig. It first parses the group verification key vk into n per-signer verification keys $vk_1, \dots, vk_n$, and parses the group signature sig into a leader request lr, a set F, and a collection of partial signatures $\{psig_i\}_{i \in F}$. If the message $lr.m$ in the leader request is not equal to the input message M, or the set F is not a subset of $lr.SS$, it returns false. Otherwise, it initializes a set T that includes all signer $i \in F$ such that the verification $\mathsf{DS.Vf}$ passes with the per-signer verification key vk_i, the message $lr.m$, and the signature $psig_i$. Finally, it returns true if $T = lr.SS = F$ and the size of T is great than or equal t, and false otherwise.
- The strong verification algorithm $\Pi.\mathsf{SVf}$ inputs a group verification key vk, a leader request lr, and a group signature sig. It first parses the group signature sig into a leader request lr', a set F, and a collection of partial signatures. If $\Pi.\mathsf{Vf}(vk, lr.m, sig)$ returns true and the parsed leader request lr' equals the input one lr, it returns true. Otherwise, it returns false.

E Proof of Theorem 1

Proof. Theorem 1 makes the following two claims for a (t, n) threshold signature TS with (u, v, w) rounds:

Claim 1: If TS is ϵ-$(\mathtt{SiGu}_1, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1)$-$\mathtt{UF}$ secure, then it is also ϵ-$(\mathtt{SiGu}_2, \mathtt{Corr}_2, \mathtt{KGCh}_2, \mathtt{SiCh}_2)$-$\mathtt{UF}$ secure, for any $\mathtt{SiGu}_2 \le \mathtt{SiGu}_1$, $\mathtt{Corr}_2 \le \mathtt{Corr}_1$, $\mathtt{KGCh}_2 \le \mathtt{KGCh}_1$, $\mathtt{SiCh}_2 \le \mathtt{SiCh}_1$, $\mathtt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$.

Claim 2: If TS is ϵ-$(\mathtt{SiGu}, \mathtt{Corr}, \mathtt{KGCh}, \mathtt{SiCh})$-$\mathsf{SUF\text{-}CMA}$ secure, then it is also ϵ-$(\mathtt{SiGu}, \mathtt{Corr}, \mathtt{KGCh}, \mathtt{SiCh})$-$\mathsf{EUF\text{-}CMA}$ secure.

E.1 Proof of Claim 1

We prove the following four cases and conclude the proof of Claim 1 by combing them together:

Case 1: $\mathtt{SiGu}_2 \le \mathtt{SiGu}_1$, $\mathtt{Corr}_2 = \mathtt{Corr}_1$, $\mathtt{KGCh}_2 = \mathtt{KGCh}_1$, $\mathtt{SiCh}_2 = \mathtt{SiCh}_1$

Case 2: $\mathtt{SiGu}_2 = \mathtt{SiGu}_1$, $\mathtt{Corr}_2 \le \mathtt{Corr}_1$, $\mathtt{KGCh}_2 = \mathtt{KGCh}_1$, $\mathtt{SiCh}_2 = \mathtt{SiCh}_1$

Case 3: $\mathtt{SiGu}_2 = \mathtt{SiGu}_1$, $\mathtt{Corr}_2 = \mathtt{Corr}_1$, $\mathtt{KGCh}_2 \le \mathtt{KGCh}_1$, $\mathtt{SiCh}_2 = \mathtt{SiCh}_1$

Case 4: $\mathtt{SiGu}_2 = \mathtt{SiGu}_1$, $\mathtt{Corr}_2 = \mathtt{Corr}_1$, $\mathtt{KGCh}_2 = \mathtt{KGCh}_1$, $\mathtt{SiCh}_2 \le \mathtt{SiCh}_1$

Case 1. We prove Case 1 by reduction. If there exists an adversary $\mathcal{A}$ that can break $(\mathtt{SiGu}_2, \mathtt{Corr}_2, \mathtt{KGCh}_2, \mathtt{SiCh}_2)$-$\mathtt{UF}$ security for any $\mathtt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$, then we can construct an adversary $\mathcal{B}$ that breaks $(\mathtt{SiGu}_1, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1)$-$\mathtt{UF}$, with $\mathtt{SiGu}_2 \le \mathtt{SiGu}_1$, $\mathtt{Corr}_2 = \mathtt{Corr}_1$, $\mathtt{KGCh}_2 = \mathtt{KGCh}_1$, $\mathtt{SiCh}_2 = \mathtt{SiCh}_1$. The adversary $\mathcal{B}$ invokes $\mathcal{A}$, forwards all queries from $\mathcal{A}$ to its challenger, and forwards the reply from its challenger to $\mathcal{A}$.

If $\mathtt{UF} = \mathsf{EUF\text{-}CMA}$, $\mathcal{A}$ outputs a challenge message-signature pair $(m^\star, \sigma^\star)$ and wins if (1) $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$, (2) $|\mathcal{L}_{\mathsf{MS}}| < t$, and (3) $\neg\big(\exists lr \text{ with } lr.m = m^\star\text{: } \mathtt{tf}_{\mathtt{SiGu}_2}(lr)\big)$. $\mathcal{B}$ forwards the challenge message-signature pair $(m^\star, \sigma^\star)$ to its challenger and wins if (1) $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$, (2) $|\mathcal{L}_{\mathsf{MS}}| < t$, and (3) $\neg\big(\exists lr$ with $lr.m = m^\star\text{: } \mathtt{tf}_{\mathtt{SiGu}_1}(lr)\big)$. We only need to prove that

$$
\begin{aligned}
&\Big(\neg\big(\exists lr \text{ with } lr.m = m^\star : \mathtt{tf}_{\mathtt{SiGu}_2}(lr)\big)\Big) \Rightarrow \Big(\neg\big(\exists lr \text{ with } lr.m = m^\star : \mathtt{tf}_{\mathtt{SiGu}_1}(lr)\big)\Big) \\
\Leftrightarrow&\big(\forall lr \text{ with } lr.m = m^\star : \neg\mathtt{tf}_{\mathtt{SiGu}_2}(lr)\big) \Rightarrow \big(\forall lr \text{ with } lr.m = m^\star : \neg\mathtt{tf}_{\mathtt{SiGu}_1}(lr)\big) \\
\Leftrightarrow&\forall lr \text{ with } lr.m = m^\star : \neg\mathtt{tf}_{\mathtt{SiGu}_2}(lr) \Rightarrow \neg\mathtt{tf}_{\mathtt{SiGu}_1}(lr) \\
\Leftrightarrow&\forall lr \text{ with } lr.m = m^\star : \mathtt{tf}_{\mathtt{SiGu}_1}(lr) \Rightarrow \mathtt{tf}_{\mathtt{SiGu}_2}(lr)
\end{aligned}
$$

Note that for all leader requests lr it always holds that

- $\mathtt{tf}_{4:\mathsf{aLRhPP}}(lr) \Leftrightarrow \mathtt{tf}_{3:\mathsf{tLRhPP}}(lr)$, because $\mathtt{tf}_{4:\mathsf{aLRhPP}}(lr) \Leftrightarrow \Big(\mathtt{tf}_{2:\mathsf{tLR}}(lr)$ and $\mathcal{D}_2[lr] = \mathcal{D}_3[lr] = \mathcal{D}_4[lr]\Big) \Rightarrow \Big(\mathtt{tf}_{2:\mathsf{tLR}}(lr)$ and $\mathcal{D}_2[lr] = \mathcal{D}_3[lr]\Big) \Leftrightarrow \mathtt{tf}_{3:\mathsf{tLRhPP}}(lr)$
- $\mathtt{tf}_{3:\mathsf{tLRhPP}}(lr) \Rightarrow \mathtt{tf}_{2:\mathsf{tLR}}(lr)$, because $\mathtt{tf}_{3:\mathsf{tLRhPP}}(lr) \Leftrightarrow \Big(\mathtt{tf}_{2:\mathsf{tLR}}(lr)$ and $\mathcal{D}_2[lr] = \mathcal{D}_3[lr]\Big) \Rightarrow \mathtt{tf}_{2:\mathsf{tLR}}(lr)$

- $\texttt{tf}_{2:\mathsf{tLR}}(lr) \Rightarrow \texttt{tf}_{1:\mathsf{tM}}(lr)$, because $\mathcal{D}_1[lr.m] = \cup_{lr' \text{ with } lr'.m=lr.m}\mathcal{D}_2[lr'] \supseteq \mathcal{D}_2[lr]$ and $|\mathcal{D}_2[lr]| \geq t - |\mathcal{L}_{\mathsf{MS}}|$, which implies that $\texttt{tf}_{2:\mathsf{tLR}}(lr) \Leftrightarrow \Big(|\mathcal{D}_2[lr]| \geq t - |\mathcal{L}_{\mathsf{MS}}|\Big) \Rightarrow \Big(|\mathcal{D}_1[lr.m]| \geq |\mathcal{D}_2[lr]| \geq t - |\mathcal{L}_{\mathsf{MS}}|\Big) \Leftrightarrow \texttt{tf}_{1:\mathsf{tM}}(lr)$
- $\texttt{tf}_{1:\mathsf{tM}}(lr) \Rightarrow \texttt{tf}_{0:\mathsf{eM}}(lr)$, because $\texttt{tf}_{1:\mathsf{tM}}(lr) \Leftrightarrow \Big(|\mathcal{D}_1[lr]| \geq t - |\mathcal{L}_{\mathsf{MS}}| > 0\Big) \Rightarrow \Big(\mathcal{D}_1[lr] \neq \emptyset\Big) \Leftrightarrow \texttt{tf}_{0:\mathsf{eM}}(lr)$

This means that for all leader requests lr, we always have that $\texttt{tf}_{\texttt{SiGu}_1}(lr) \Rightarrow \texttt{tf}_{\texttt{SiGu}_2}(lr)$ for all $\texttt{SiGu}_2 \leq \texttt{SiGu}_1$ due to the transitivity. Thus, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins.

If $\texttt{UF} = \mathsf{SUF\text{-}CMA}$, $\mathcal{A}$ outputs a challenge message-signature pair $(m^\star, \sigma^\star)$ and wins if (1) $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$, (2) $|\mathcal{L}_{\mathsf{MS}}| < t$, and (3) $\neg\big(\exists lr \text{ with } lr.m = m^\star\text{: } \texttt{tsf}_{\texttt{SiGu}_2}(lr, gvk, \sigma^\star)\big)$. $\mathcal{B}$ forwards the challenge message-signature pair $(m^\star, \sigma^\star)$ to its challenger and wins if (1) $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$, (2) $|\mathcal{L}_{\mathsf{MS}}| < t$, and (3) $\neg\big(\exists lr \text{ with } lr.m = m^\star\text{: } \texttt{tsf}_{\texttt{SiGu}_1}(lr, gvk, \sigma^\star)\big)$. We only need to prove that

$$\begin{aligned}
&\Big(\neg\big(\exists lr \text{ with } lr.m = m^\star : \texttt{tsf}_{\texttt{SiGu}_2}(lr, gvk, \sigma^\star)\big)\Big)\\
&\qquad\qquad \Rightarrow \Big(\neg\big(\exists lr \text{ with } lr.m = m^\star : \texttt{tsf}_{\texttt{SiGu}_1}(lr, gvk, \sigma^\star)\big)\Big)\\
\Leftrightarrow&\big(\forall lr \text{ with } lr.m = m^\star : \neg\texttt{tsf}_{\texttt{SiGu}_2}(lr, gvk, \sigma^\star)\big)\\
&\qquad\qquad \Rightarrow \big(\forall lr \text{ with } lr.m = m^\star : \neg\texttt{tsf}_{\texttt{SiGu}_1}(lr, gvk, \sigma^\star)\big)\\
\Leftrightarrow&\forall lr \text{ with } lr.m = m^\star : \neg\texttt{tsf}_{\texttt{SiGu}_2}(lr, gvk, \sigma^\star) \Rightarrow \neg\texttt{tsf}_{\texttt{SiGu}_1}(lr, gvk, \sigma^\star)\\
\Leftrightarrow&\forall lr \text{ with } lr.m = m^\star : \texttt{tsf}_{\texttt{SiGu}_1}(lr, gvk, \sigma^\star) \Rightarrow \texttt{tsf}_{\texttt{SiGu}_2}(lr, gvk, \sigma^\star)
\end{aligned}$$

Note that for all leader requests lr it always holds that

$$\begin{aligned}
&\texttt{tsf}_{\texttt{SiGu}_1}(lr, gvk, \sigma)\\
\Leftrightarrow&\texttt{tf}_{\texttt{SiGu}_1}(lr) \text{ and } \mathsf{SVf}(gvk, lr, \sigma)\\
\Rightarrow&\texttt{tf}_{\texttt{SiGu}_2}(lr) \text{ and } \mathsf{SVf}(gvk, lr, \sigma)\\
\Leftrightarrow&\texttt{tsf}_{\texttt{SiGu}_2}(lr, gvk, \sigma)
\end{aligned}$$

This means that for all leader requests lr, we always have that $\texttt{tsf}_{\texttt{SiGu}_1}(lr, gvk, \sigma^\star) \Rightarrow \texttt{tsf}_{\texttt{SiGu}_2}(lr, gvk, \sigma^\star)$ for all $\texttt{SiGu}_2 \leq \texttt{SiGu}_1$. Thus, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins.

Case 2. We prove Case 2 by reduction. If there exists an adversary $\mathcal{A}$ that can break $(\texttt{SiGu}_2, \texttt{Corr}_2, \texttt{KGCh}_2, \texttt{SiCh}_2)\text{-}\texttt{UF}$ security for any $\texttt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$, then we can construct an adversary $\mathcal{B}$ that breaks $(\texttt{SiGu}_1, \texttt{Corr}_1, \texttt{KGCh}_1, \texttt{SiCh}_1)\text{-}\texttt{UF}$, with $\texttt{SiGu}_2 = \texttt{SiGu}_1$, $\texttt{Corr}_2 \leq \texttt{Corr}_1$, $\texttt{KGCh}_2 = \texttt{KGCh}_1$, $\texttt{SiCh}_2 = \texttt{SiCh}_1$. The adversary $\mathcal{B}$ invokes $\mathcal{A}$, forwards all queries from $\mathcal{A}$ to its challenger, and forwards the reply from its challenger to $\mathcal{A}$. The only exception is the queries to the corruption oracle OCorrupt. $\mathcal{B}$ first runs all checks in the OCorrupt oracle for $\mathcal{A}$. If no check fails, then $\mathcal{B}$ forwards the query to its challenger, otherwise exits.

Finally, $\mathcal{A}$ outputs a challenge message-signature pair $(m^\star, \sigma^\star)$ and $\mathcal{B}$ forwards it to its challenger. Note that $\mathcal{B}$ can perfectly simulate the challenger to $\mathcal{A}$ and wins if and only if $\mathcal{A}$ wins, which concludes the proof for Case 2.

Case 3. We prove Case 3 by reduction. If there exists an adversary $\mathcal{A}$ that can break $(\texttt{SiGu}_2, \texttt{Corr}_2, \texttt{KGCh}_2, \texttt{SiCh}_2)\text{-}\texttt{UF}$ security for any $\texttt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$, then we can construct an adversary $\mathcal{B}$ that breaks $(\texttt{SiGu}_1, \texttt{Corr}_1, \texttt{KGCh}_1, \texttt{SiCh}_1)\text{-}\texttt{UF}$, with $\texttt{SiGu}_2 = \texttt{SiGu}_1$, $\texttt{Corr}_2 = \texttt{Corr}_1$, $\texttt{KGCh}_2 \leq \texttt{KGCh}_1$, $\texttt{SiCh}_2 = \texttt{SiCh}_1$. We only consider $\texttt{KGCh}_2 < \texttt{KGCh}_1$, since the case otherwise is trivial.

The adversary $\mathcal{B}$ simulates the dictionary $\mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}$ by itself. Note that $2\text{: }\mathsf{KGiC} \geq \texttt{KGCh}_1 > \texttt{KGCh}_2 \geq 0\text{: }\mathsf{KGsC}$. $\mathcal{B}$ can always view and record all messages in the model. Moreover, $\mathcal{B}$ uses $\mathsf{rnd}_1, \ldots, \mathsf{rnd}_n$ to count the index of the next key generation round. These counts are incremented whenever $\mathsf{st}_i.\mathsf{rnd}$ should be incremented.

The adversary $\mathcal{B}$ invokes $\mathcal{A}$, forwards all queries from $\mathcal{A}$ to its challenger, and forwards the reply from its challenger to $\mathcal{A}$. The only exception is the queries to the key generation oracle $\text{OKGen}(i, m)$. $\mathcal{B}$ first checks whether $\mathsf{rnd}_i \geq 1$ and $\texttt{KGCh}_1 = 2\text{: }\mathsf{KGiC}$. If the check passes, $\mathcal{B}$ appends the input message m with $\|_{i' \in \mathcal{L}_{\mathsf{HS}} \setminus \{i\}} \mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(\mathsf{rnd}, i', i)]$ to m. Then, no matter whether the check passes or not, the adversary $\mathcal{B}$ forwards i and the (possibly appended) message m to its challenger.

When $\mathcal{B}$ receives a reply m' from its challenger, $\mathcal{B}$ checks whether $\texttt{KGCh}_2 = 0\text{: }\mathsf{KGsC}$ and $\mathsf{rnd}_i \leq u$. If the condition is true, then $\mathcal{B}$ returns $\|_{i' \in \mathcal{L}_{\mathsf{MS}}} \mathcal{D}^{\mathsf{KGen}}_{\mathsf{Trans}}[(\mathsf{rnd}, i, i')]$. Otherwise, $\mathcal{B}$ returns m'.

Finally, $\mathcal{A}$ outputs a challenge message-signature pair $(m^\star, \sigma^\star)$ and $\mathcal{B}$ forwards it to its challenger. Note that $\mathcal{B}$ can perfectly simulate the challenger to $\mathcal{A}$ and wins if and only if $\mathcal{A}$ wins, which concludes the proof for Case 3.

Case 4. We prove Case 4 by reduction, which is very close to the proof of Case 3. If there exists an adversary $\mathcal{A}$ that can break $(\texttt{SiGu}_2, \texttt{Corr}_2, \texttt{KGCh}_2, \texttt{SiCh}_2)$-$\texttt{UF}$ security for any $\texttt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$, then we can construct an adversary $\mathcal{B}$ that breaks $(\texttt{SiGu}_1, \texttt{Corr}_1, \texttt{KGCh}_1, \texttt{SiCh}_1)$-$\texttt{UF}$, with $\texttt{SiGu}_2 = \texttt{SiGu}_1$, $\texttt{Corr}_2 = \texttt{Corr}_1$, $\texttt{KGCh}_2 = \texttt{KGCh}_1$, $\texttt{SiCh}_2 \leq \texttt{SiCh}_1$. We only consider $\texttt{SiCh}_2 < \texttt{SiCh}_1$, since the case otherwise is trivial.

The adversary $\mathcal{B}$ simulates the dictionary $\mathcal{D}^{\mathsf{Sign}}_{\mathsf{Trans}}$ by itself. Note that $2\colon \mathsf{KGiC} \geq \texttt{SiCh}_1 > \texttt{SiCh}_2 \geq 0\colon \mathsf{KGsC}$. $\mathcal{B}$ can always view and record all messages in the model. Moreover, $\mathcal{B}$ uses $\mathsf{rnd}_{i,j}$ to count the index of the next signing round of the session state π_i^j. These counts are incremented whenever $\pi_i^j.\mathsf{rnd}$ should be incremented.

The adversary $\mathcal{B}$ invokes $\mathcal{A}$, forwards all queries from $\mathcal{A}$ to its challenger, and forwards the reply from its challenger to $\mathcal{A}$. The only exception is the queries to the key generation oracle $\textsc{OSign}(i, j, m)$. $\mathcal{B}$ first checks whether $\mathsf{rnd}_{i,j} \geq 1$ and $\texttt{SiCh}_1 = 2\colon \mathsf{KGiC}$. If the check passes, $\mathcal{B}$ appends the input message m with $\|_{i' \in \mathcal{L}_{\mathsf{HS}} \cap lr.SS \setminus \{i\}} \mathcal{D}^{\mathsf{Sign}}_{\mathsf{Trans}}[(j, \mathsf{rnd}, i', i)]$, where lr is the leader request that is expected to be signed in signer i's j-th session. Then, no matter whether the check passes or not, the adversary $\mathcal{B}$ forwards (i, j) and the (possibly appended) message m to its challenger.

When $\mathcal{B}$ receives a reply m' from its challenger, it checks whether $\texttt{SiCh}_2 = 0\colon \mathsf{SisC}$ and $\mathsf{rnd}_{i,j} \leq w$. If the condition is true, then $\mathcal{B}$ returns $\|_{i' \in \mathcal{L}_{\mathsf{MS}}} \mathcal{D}^{\mathsf{Sign}}_{\mathsf{Trans}}[(j, \mathsf{rnd}, i, i')]$. Otherwise, $\mathcal{B}$ returns m'.

Finally, $\mathcal{A}$ outputs a challenge message-signature pair $(m^\star, \sigma^\star)$ and $\mathcal{B}$ forwards it to its challenger. Note that $\mathcal{B}$ can perfectly simulate the challenger to $\mathcal{A}$ and wins if and only if $\mathcal{A}$ wins, which concludes the proof for Case 4.

Conclusion of Claim 1. The proof is claim 1 is concluded by combining the above four cases: for any $\texttt{UF} \in \{\mathsf{EUF\text{-}CMA}, \mathsf{SUF\text{-}CMA}\}$

$$\begin{aligned} &\epsilon\text{-}(\texttt{SiGu}_1, \texttt{Corr}_1, \texttt{KGCh}_1, \texttt{SiCh}_1)\text{-}\texttt{UF} \text{ security} \\ \Rightarrow\ &\epsilon\text{-}(\texttt{SiGu}_2, \texttt{Corr}_1, \texttt{KGCh}_1, \texttt{SiCh}_1)\text{-}\texttt{UF} \text{ security} \\ \Rightarrow\ &\epsilon\text{-}(\texttt{SiGu}_2, \texttt{Corr}_2, \texttt{KGCh}_1, \texttt{SiCh}_1)\text{-}\texttt{UF} \text{ security} \\ \Rightarrow\ &\epsilon\text{-}(\texttt{SiGu}_2, \texttt{Corr}_2, \texttt{KGCh}_2, \texttt{SiCh}_1)\text{-}\texttt{UF} \text{ security} \\ \Rightarrow\ &\epsilon\text{-}(\texttt{SiGu}_2, \texttt{Corr}_2, \texttt{KGCh}_2, \texttt{SiCh}_2)\text{-}\texttt{UF} \text{ security} \end{aligned}$$

E.2 Proof of Claim 2

We prove Claim 2 by reduction. If there exists an adversary $\mathcal{A}$ that can break $(\texttt{SiGu}, \texttt{Corr}, \texttt{KGCh}, \texttt{SiCh})$-$\mathsf{EUF\text{-}CMA}$ security, then we can construct an adversary $\mathcal{B}$ that breaks $(\texttt{SiGu}, \texttt{Corr}, \texttt{KGCh}, \texttt{SiCh})$-$\mathsf{SUF\text{-}CMA}$. The adversary $\mathcal{B}$ invokes $\mathcal{A}$, forwards all queries from $\mathcal{A}$ to its challenger, and forwards the reply from its challenger to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a challenge message-signature pair $(m^\star, \sigma^\star)$ and wins if (1) $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$, (2) $|\mathcal{L}_{\mathsf{MS}}| < t$, and (3) $\neg\big(\exists lr \text{ with } lr.m = m^\star\colon \texttt{tf}_{\texttt{SiGu}}(lr)\big)$. $\mathcal{B}$ forwards the challenge message-signature pair $(m^\star, \sigma^\star)$ to its challenger and wins if (1) $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$, (2) $|\mathcal{L}_{\mathsf{MS}}| < t$, and (3) $\neg\big(\exists lr$ with $lr.m = m^\star\colon \texttt{tsf}_{\texttt{SiGu}}(lr)\big)$. We only need to prove that

$$\begin{aligned} &\Big(\neg\big(\exists lr \text{ with } lr.m = m^\star : \texttt{tf}_{\texttt{SiGu}}(lr)\big)\Big) \Rightarrow \Big(\neg\big(\exists lr \text{ with } lr.m = m^\star : \texttt{tsf}_{\texttt{SiGu}}(lr)\big)\Big) \\ \Leftrightarrow &\big(\forall lr \text{ with } lr.m = m^\star : \neg\texttt{tf}_{\texttt{SiGu}}(lr)\big) \Rightarrow \big(\forall lr \text{ with } lr.m = m^\star : \neg\texttt{tsf}_{\texttt{SiGu}}(lr)\big) \\ \Leftrightarrow &\forall lr \text{ with } lr.m = m^\star : \neg\texttt{tf}_{\texttt{SiGu}}(lr) \Rightarrow \neg\texttt{tsf}_{\texttt{SiGu}}(lr) \\ \Leftrightarrow &\forall lr \text{ with } lr.m = m^\star : \texttt{tsf}_{\texttt{SiGu}}(lr) \Rightarrow \texttt{tf}_{\texttt{SiGu}}(lr) \end{aligned}$$

Note that this holds by definition. Thus, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins. □

F Proof of Theorem 2

We prove that our construction Π achieves TS-UF-4 security following the syntax and definitions in [10, 11].

Proof. The proof is given by reduction. Namely, if there exists an adversary $\mathcal{A}$ that breaks TS-UF-4 security of Π, then we can construct an adversary $\mathcal{B}$ that breaks UF-CMA security of the underlying DS by invoking $\mathcal{A}$ in the following steps.

- $\mathcal{B}$ initializes n counters $\mathsf{ctr}_1, \ldots, \mathsf{ctr}_n$ with 0, and lists L, $S_2(lr)$, and $S_4(lr)$ for all leader requests lr with empty set $\emptyset$.
- When $\mathcal{A}$ queries the INIT oracle with input CS that denotes the set of corrupted signers, $\mathcal{B}$ checks whether $CS \subseteq [1..n]$ and $|CS| < t$ and aborts if the check fails. If the check passes, $\mathcal{B}$ sets the set of honest signers to be $HS \leftarrow [1..n] \setminus CS$. Next, $\mathcal{B}$ queries the INIT oracle and receives a verification key $vk^\star$. Then, $\mathcal{B}$ samples an index $i^\star \in HS$ uniformly at random, and sets $vk_{i^\star} = vk^\star$. Afterwards, $\mathcal{B}$ samples all other signers' signing and verification key pairs honestly by itself, i.e., by running $(vk_i, sk_i) \xleftarrow{\$} \mathsf{DS.Kg}$ for all $i \in [1..n] \setminus \{i^\star\}$. Finally, $\mathcal{B}$ sets $vk \leftarrow (vk_1, \ldots, vk_n)$ and returns $(vk, \epsilon, \{sk_i\}_{i \in CS})$ to $\mathcal{A}$. Recall that the model must start with this oracle, which can be queried only once.
- When $\mathcal{A}$ queries the PPO oracle with input i that denotes the index of a signer, $\mathcal{B}$ first checks whether i is an honest signer by checking whether $i \in HS$ or not. If the check passes, $\mathcal{B}$ increments ctr_i by 1, i.e., $\mathsf{ctr}_i \leftarrow \mathsf{ctr}_i + 1$ and returns the new value ctr_i. Otherwise, $\mathcal{B}$ aborts.
- When $\mathcal{A}$ queries the PSIGNO oracle with inputs i and lr, where i denotes the index of a signer and lr denotes a leader request, $\mathcal{B}$ first checks whether $lr.SS \subseteq [1..n]$, $lr.m \in \{0,1\}^\star$, and $i \in HS$. If any check fails, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ stores lr into the list L ($L \leftarrow L \cup \{lr\}$). Afterwards, if $i = i^\star$, $\mathcal{B}$ first checks whether $i^\star \in lr.SS$, $lr.SS \geq t$, and $lr.\mathsf{PP}(i^\star) \in \mathbb{N}_+$. If any check fails, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ queries its SIGNO oracle with input lr and obtains a signature $psig$. If $i \neq i^\star$, $\mathcal{B}$ runs the honest signing algorithm $psig \xleftarrow{\$} \mathsf{DS.Sign}(sk_i, lr)$ by itself, as $\mathcal{B}$ knows the corresponding private signing keys. In both cases, if $psig \neq \perp$, $\mathcal{B}$ stores i into the list $S_2(lr)$ ($S_2(lr) \leftarrow S_2(lr) \cup \{i\}$). Finally, $\mathcal{B}$ returns $psig$.
- When $\mathcal{A}$ queries the RO oracle with any input x, $\mathcal{B}$ simulates an honest random oracle by itself, since Π does not use of any random oracles.
- When $\mathcal{A}$ queries the FIN oracle with inputs $M^\star$ and $sig^\star$, $\mathcal{B}$ sets $S_4(lr) \leftarrow HS \cap lr.SS$ for all $lr \in L$. Next, $\mathcal{B}$ checks whether $\Pi.\mathsf{Vf}(vk, M^\star, sig^\star) = \mathsf{true}$. If the checks fails, $\mathcal{B}$ returns false.

  If the check passes, $\mathcal{B}$ further checks the expression
$$\begin{aligned} & \text{not } \exists lr\ (lr.m = M^\star \textbf{ and } \mathtt{tf}_4(lr)) \\ \Leftrightarrow\ & \text{not } \exists lr\ (lr.m = M^\star \textbf{ and } |S_2(lr)| \geq t - |CS| \textbf{ and } S_2(lr) = S_4(lr)). \end{aligned}$$
  If the express is true, $\mathcal{B}$ parses $(lr^\star, F^\star, \{psig_i^\star\}_{i \in F^\star}) \leftarrow sig^\star$. Note that this parsing was executed in $\Pi.\mathsf{Vf}(vk, M^\star, sig^\star) = \mathsf{true}$ and therefore always works. Finally, if $i^\star \notin F^\star$, $\mathcal{B}$ directly loses. Otherwise, $\mathcal{B}$ invokes its FIN oracle with input $(lr^\star, psig_{i^\star})$.

Final Analysis. Now, we calculate $\mathcal{B}$'s winning probability. First, note that $\Pi.\mathsf{Vf}(vk, M^\star, sig^\star) = \mathsf{true}$ and $(lr^\star, F^\star, \{psig_i^\star\}_{i \in F^\star}) = sig^\star$. Therefore, it must hold that

- $lr^\star.m = M^\star$
- $F^\star = lr^\star.SS$
- $\forall i \in lr^\star.SS = F^\star$: $\mathsf{DS.Vf}(vk_i, lr^\star, psig_i^\star) = \mathsf{true}$
- $|lr^\star.SS| = |F^\star| \geq t$

Recall that $|CS| < t$. This means that there must be at least one honest signer in the set $lr^\star.SS$.

$$\begin{aligned} & \text{not } \exists lr\ (lr.m = M^\star \textbf{ and } |S_2(lr)| \geq t - |CS| \textbf{ and } S_2(lr) = S_4(lr)) \\ \Leftrightarrow\ & \nexists lr\ (lr.m = M^\star \textbf{ and } |S_2(lr)| \geq t - |CS| \textbf{ and } S_2(lr) = S_4(lr)) \\ \Leftrightarrow\ & \forall lr\ \Big(\neg(lr.m = M^\star \textbf{ and } |S_2(lr)| \geq t - |CS| \textbf{ and } S_2(lr) = S_4(lr))\Big) \\ \Leftrightarrow\ & \forall lr\ \Big((lr.m \neq M^\star \textbf{ or } |S_2(lr)| < t - |CS| \textbf{ or } S_2(lr) \neq S_4(lr))\Big) \\ \Leftrightarrow\ & \forall lr\ \Big((lr.m \neq M^\star \textbf{ or } |S_2(lr)| < t - |CS| \textbf{ or } S_2(lr) \neq HS \cap lr.SS)\Big) \\ \Leftrightarrow\ & \forall lr \text{ with } lr.m = M^\star\ \Big((|S_2(lr)| < t - |CS| \textbf{ or } S_2(lr) \neq HS \cap lr.SS)\Big) \\ \Rightarrow\ & |S_2(lr^\star)| < t - |CS| \textbf{ or } S_2(lr^\star) \neq HS \cap lr^\star.SS \qquad (*) \end{aligned}$$

This indicates that if the adversary $\mathcal{A}$ wins, then the above expression $(*)$ must be true. We then consider the following two cases.

Case 1: $|S_2(lr^\star)| < t - |CS|$**.** The number of honest signers who have signed $lr^\star$ and the number of corrupted signers must be smaller than t. Recall that $|lr^\star.SS| \geq t$. This means that there must be at least one signer $j^\star \in lr^\star.SS$ such that

1. $j^\star$ has never signed $lr^\star$, and
2. $j^\star$ is not corrupted (and therefore honest).

Recall that $\mathcal{B}$ samples $i^\star \in HS \subseteq [n]$ uniformly at random and that $\mathcal{A}$ cannot distinguish the index $i^\star$ from the indices of all other honest signers, since $\mathcal{B}$ simulates the TS-UF-4 security model honestly. The probability that $i^\star = j^\star$ must be at least $\frac{1}{|HS|}$, i.e.,

$$\Pr[i^\star = j^\star] \geq \frac{1}{|HS|} \geq \frac{1}{n}.$$

Note that if $i^\star = j^\star$, $\mathcal{B}$ wins in the UF-CMA model. Note also that the event $i^\star = j^\star$ and the event $\mathcal{A}$ wins TS-UF-4 model are independent. Thus, we have that

$$\Pr[\mathcal{B} \text{ wins}] = \Pr[(i^\star = j^\star) \textbf{ and } (\mathcal{A} \text{ wins})] = \Pr[i^\star = j^\star] \cdot \Pr[\mathcal{A} \text{ wins}] \geq \frac{1}{n} \Pr[\mathcal{A} \text{ wins}]$$

Case 2: $S_2(lr^\star) \neq HS \cap lr^\star.SS$**.** We further divide this case into two sub-cases.

Case 2.1: $\exists j^\star \in S_2(lr^\star)$ **but** $j^\star \notin HS \cap lr^\star.SS$**.** Note that $j^\star \in S_2(lr^\star)$ indicates that $j^\star \in HS$. Thus, $j^\star \notin HS \cap lr^\star.SS$ means that $j^\star \notin lr^\star.SS$. This further means that $j^\star$ has signed the leader request $lr^\star$, where $j^\star \notin lr^\star.SS$. However, this sub-case is impossible, since the $\Pi.\mathsf{PS}$ algorithm always check whether the signer is included in the request $lr^\star.SS$ (see Line 35).

Case 2.2: $\exists j^\star \in HS \cap lr^\star.SS$ **but** $j^\star \notin S_2(lr^\star)$**.** This indicates that the honest signer $j^\star$ in the set $lr^\star.SS$ has never signed the leader request $lr^\star$. Similar to the argument in Case 1, we have that

$$\Pr[\mathcal{B} \text{ wins}] \geq \frac{1}{n} \Pr[\mathcal{A} \text{ wins}]$$

In summary, if $\mathcal{A}$ can win in the TS-UF-4 model against Π with non-negligible probability, then $\mathcal{B}$ can also win in the UF-CMA model against DS underlying Π with non-negligible probability, which concludes the proof. □

G Proof of Theorem 3

We prove that our construction Π does not achieve TS-UF-3 security following the syntax and definitions in [10, 11].

Proof. For simple presentation, we only explain the case for $(t, n) = (2, 3)$. However, the counterexample can be easily generalized to any other (t, n). Consider the adversary $\mathcal{A}$ with the following queries:

1. $\mathcal{A}$ queries the INIT oracle with input $CS = \emptyset$ and receives (vk, ϵ, ϵ), where $vk = (vk_1, vk_2, vk_3)$.
2. $\mathcal{A}$ queries the PPO oracle with input $i = 1$ and receives the value 1.
3. $\mathcal{A}$ creates a leader request $lr^\star$ as follows:
    - $lr^\star.m$: any valid message (e.g., "000").
    - $lr^\star.SS = \{1, 2\}$
    - $lr^\star.\mathsf{PP}(1) = lr^\star.\mathsf{PP}(2) = 1$
4. $\mathcal{A}$ queries the PSIGNO oracle with input $(1, lr^\star)$. This allows $\mathcal{A}$ to receive a partial signature $psig_1$ signed by the signer $i = 1$.
5. $\mathcal{A}$ queries the PSIGNO oracle with input $(2, lr^\star)$. This allows $\mathcal{A}$ to receive a partial signature $psig_2$ signed by the signer $i = 2$.

6. $\mathcal{A}$ queries the FIN oracle with input $(M^\star, sig^\star)$, where $M^\star = lr^\star.m$ and $sig^\star = (lr^\star, lr^\star.SS, \{psig_1, psig_2\})$.

Final Analysis. The adversary $\mathcal{A}$ wins the TS-UF-3 model if the following conditions holds:

1. $\Pi.\mathsf{Vf}(vk, M^\star, sig^\star) = \mathsf{true}$, and
2. not $\exists lr\ (lr.m = M^\star$ **and** $\mathtt{tf}_3(lr))$

The first condition holds trivially. The second expression can be further unfolded as

$$\begin{aligned}
&\text{not } \exists lr\ (lr.m = M^\star \textbf{ and } \mathtt{tf}_3(lr)) \\
\Leftrightarrow\ &\text{not } \exists lr\ (lr.m = M^\star \textbf{ and } |S_2(lr)| \geq t - |CS| \textbf{ and } S_2(lr) = S_3(lr)) \\
\Leftrightarrow\ &\nexists lr\ (lr.m = M^\star \textbf{ and } |S_2(lr)| \geq t - |CS| \textbf{ and } S_2(lr) = S_3(lr)) \\
\Leftrightarrow\ &\forall lr\ \Big(\neg(lr.m = M^\star \textbf{ and } |S_2(lr)| \geq t - |CS| \textbf{ and } S_2(lr) = S_3(lr))\Big) \\
\Leftrightarrow\ &\forall lr\ \Big((lr.m \neq M^\star \textbf{ or } |S_2(lr)| < t - |CS| \textbf{ or } S_2(lr) \neq S_3(lr))\Big) \\
\Leftrightarrow\ &\forall lr \text{ with } lr.m = M^\star\ \Big((|S_2(lr)| < t - |CS| \textbf{ or } S_2(lr) \neq S_3(lr))\Big) \qquad (**)
\end{aligned}$$

Note that the adversary $\mathcal{A}$ queries PSIGNO only for one leader request $lr^\star$. For all $lr \neq lr^\star$, it always holds that $S_2(lr) = \emptyset$. Note also that $|CS| = 0 < 2 = t$. The above expression $(**)$ always holds for all $lr \neq lr^\star$ because $|S_2(lr)| = 0 < 2 = 2 - 0 = t - |CS|$.

Moreover, recall that PP_i denotes the pre-processing tokens generated by the signer i. Since $\mathcal{A}$ only queries PPO(i) oracle once with input $i = 1$, we have $\mathsf{PP}_1 = \{1\}$ and $\mathsf{PP}_2 = \mathsf{PP}_3 = \emptyset$. For the leader request $lr^\star$, we then further have that

- $S_2(lr^\star) = \{1, 2\}$, and
- $S_3(lr^\star) = \{i \in HS \cap lr^\star.SS : lr^\star.\mathsf{PP}(i) \in \mathsf{PP}_i\} = \{i \in \{1,2\} : lr^\star.\mathsf{PP}(i) \in \mathsf{PP}_i\} = \{1\}$

The above expression $(**)$ is therefore also holds for $lr^\star$, because $S_2(lr^\star) = \{1,2\} \neq \{1\} = S_3(lr^\star)$. In summary, $\mathcal{A}$ wins the TS-UF-3 model with probability 1. □

H Proof of Theorem 4

We prove that our construction Π achieves TS-SUF-4 security following the syntax and definitions in [10, 11].

Proof. We first prove that our $\Pi.\mathsf{SVf}$ algorithm in Figure 7 is well-defined, i.e., for any vk and lr there exists at most one signature sig such that $\Pi.\mathsf{SVf}(vk, lr, sig) = \mathsf{true}$. Suppose that there exists a vk, a lr, and two signatures $sig_1 = (lr^{(1)}, F^{(1)}, \{psig_i^{(1)}\}_{i \in F^{(1)}})$ and $sig_2 = (lr^{(2)}, F^{(2)}, \{psig_i^{(2)}\}_{i \in F^{(2)}})$, such that $\Pi.\mathsf{SVf}(vk, lr, sig_1) = \Pi.\mathsf{SVf}(vk, lr, sig_2) = \mathsf{true}$. Note that $\Pi.\mathsf{SVf}(vk, lr, sig_i) = \mathsf{true}$ for $i \in \{1, 2\}$ if and only if $\Pi.\mathsf{Vf}(vk, lr.m, sig_i) = \mathsf{true}$ and $lr = lr^{(i)}$. By $lr = lr^{(i)}$ for $i \in \{1,2\}$, it must hold that $lr^{(1)} = lr^{(2)} = lr$. By $\Pi.\mathsf{Vf}(vk, lr.m, sig_i) = \mathsf{true}$ for $i \in \{1,2\}$, it must hold

- $lr.SS = F^{(i)}$ for $i \in \{1,2\}$, which implies that $lr.SS = F^{(1)} = F^{(2)}$, and
- $lr.SS = \{j \in F^{(i)} : \mathsf{DS.Vf}(vk_j, lr, psig_j^{(i)}) = \mathsf{true}\}$, which implies that $\forall j \in lr.SS : \mathsf{DS.Vf}(vk_j, lr, psig_j^{(1)}) = \mathsf{DS.Vf}(vk_j, lr, psig_j^{(2)}) = \mathsf{true}$. Due to the uniqueness of the underlying DS, we have $\{psig_i^{(1)}\}_{i \in F^{(1)}} = \{psig_i^{(2)}\}_{i \in F^{(2)}}$.

Combing the statements above, it holds that $sig_1 = sig_2$. Thus, our $\Pi.\mathsf{SVf}$ algorithm is well-defined.

We then prove TS-SUF-4 security of our protocol Π. Note the trivial forgery formulation $\mathtt{tsf}_4(lr, vk, sig) = \mathtt{tf}_4(lr)$ **and** $\mathsf{SVf}(vk, lr, sig)$. An adversary that can break TS-SUF-4 security of Π must output $M^\star$ and $sig^\star = (lr^\star, F^\star, \{psig_i^\star\}_{i \in F^\star})$ with $\mathsf{Vf}(vk, M^\star, sig^\star) = \mathsf{true}$ such that

$$\text{not } \exists lr\big(lr.m = M^\star \textbf{ and } \mathtt{tf}_4(lr) \textbf{ and } \mathsf{SVf}(vk, lr, sig^\star)\big).$$

Due to the TS-UF-4 security of Π in Theorem 2, we know that for all $M^\star$ and $sig^\star = (lr^\star, F^\star, \{psig_i^\star\}_{i \in F^\star})$ chosen by the adversary with $\mathsf{Vf}(vk, M^\star, sig^\star) = \mathsf{true}$, it always holds that

$$\exists lr\big(lr.m = M^\star \textbf{ and } \mathtt{tf}_4(lr)\big).$$

Thus, an adversary that can break TS-SUF-4 security of Π must output $M^\star$ and $sig^\star = (lr^\star, F^\star, \{psig_i^\star\}_{i\in F^\star})$ with $\mathsf{Vf}(vk, M^\star, sig^\star) = \mathsf{true}$ such that

$$\forall lr\big(lr.m = M^\star \textbf{ and } \mathtt{tf}_4(lr)\big) : \mathsf{SVf}(vk, lr, sig^\star) = \mathsf{false}. \qquad (***)$$

We prove that the equation $(***)$ never holds by contradiction: Suppose that the equation $(***)$ holds. Recall from the definition of $\Pi.\mathsf{SVf}$ in Figure 7 that $\Pi.\mathsf{SVf}(vk, lr, sig^\star) \Leftrightarrow \Pi.\mathsf{Vf}(vk, lr.m, sig^\star)$ **and** $lr = lr^\star$. Since $\mathsf{Vf}(vk, M^\star, sig^\star) = \mathsf{true}$, we can observe that the equation $(***)$ is equivalent to

$$\begin{aligned} & \forall lr\big(lr.m = M^\star \textbf{ and } \mathtt{tf}_4(lr)\big) : lr \neq lr^\star \\ \Leftrightarrow\ & \forall lr\big(lr.m = M^\star \textbf{ and } \mathtt{tf}_2(lr) \textbf{ and } S_2(lr) = S_4(lr)\big) : lr \neq lr^\star \\ \Leftrightarrow\ & \forall lr\big(lr.m = M^\star \textbf{ and } |S_2(lr)| \geq t - |CS| \textbf{ and } S_2(lr) = S_4(lr)\big) : lr \neq lr^\star. \end{aligned}$$

Note that

$$\begin{aligned} & \Pi.\mathsf{Vf}(vk, M^\star, sig^\star) \\ \Rightarrow\ & \{i \in F^\star : \mathsf{DS.Vf}(vk_i, lr^\star, psig_i^\star)\} = lr^\star.SS = F^\star \textbf{ and } lr^\star.m = M^\star \textbf{ and } |lr^\star.SS| \geq t \\ \Rightarrow\ & \forall i \in lr^\star.SS : \mathsf{DS.Vf}(vk_i, lr^\star, psig_i^\star) \textbf{ and } lr^\star.m = M^\star \textbf{ and } |lr^\star.SS| \geq t. \end{aligned}$$

Due to the uniqueness of the underlying DS, we know that there exists at most one partial signature $psig_i^\star$ that can be verified under the per-signer verification key vk_i and the leader request $lr^\star$. Due to the SUF-CMA security of the underlying DS, we know that the adversary can not forge the signature $psig_i^\star$ for any honest signer i. This means that the adversary must query the signing oracle for asking every signer $i \in S_4(lr^\star)$ to sign $lr^\star$, which indicates that $S_4(lr^\star) \subseteq S_2(lr^\star)$. Moreover, $S_2(lr^\star) \subseteq S_4(lr^\star)$ holds trivially, as $\Pi.\mathsf{PS}$ algorithm always check whether the signer is included in the request $lr^\star.SS$ (see Line 35). So, we have that $S_4(lr^\star) = S_2(lr^\star)$. The proof is concluded by the contradiction $lr^\star$, because

- $lr^\star.m = M^\star$,
- $S_4(lr^\star) = S_2(lr^\star)$, and
- $|S_2(lr^\star)| \geq t - |CS|$, since $|lr^\star.SS| \geq t$ and $|lr^\star.SS| = |lr^\star.SS \cap HS| + |lr^\star.SS \cap CS| \leq |S_4(lr^\star)| + |CS|$.

□

I Proof of Theorem 5

Proof. TS-SUF-3 security implies TS-UF-3 security. Π is not TS-SUF-3 secure due to Theorem 3. □

J Proof of Corollary 1

Proof. We summarize the conclusions of our proofs and the proofs in [11].

- TS-UF-3 security does not imply TS-UF-4 security [11, Proposation A.1].
- TS-UF-4 security does not imply TS-UF-3 security (see Theorem 2 and Theorem 3).
- TS-SUF-3 security does not imply TS-SUF-4 security [11, Proposation A.1].
- TS-SUF-4 security does not imply TS-SUF-3 security (see Theorem 4 and Theorem 5).

□

K Proof of Theorem 6

Proof. We split the theorem into the following two claims and prove them separately.

Claim 1: If TS is ϵ_{TS}-$(3{:}\,\mathtt{tLRhPP}, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1)$-$\mathtt{UF}_1$ secure, then TS$'$ is ϵ_{TS}-$(3{:}\,\mathtt{tLRhPP}, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1)$-$\mathtt{UF}_1$ secure.

Claim 2: If further DS is ϵ_{DS}-SUF-CMA secure, then TS$'$ is $\epsilon_{\mathsf{TS}'}$-$(4{:}\,\mathtt{aLRhPP}, \mathtt{Corr}_2, \mathtt{KGCh}_2, \mathtt{SiCh}_2)$-$\mathtt{UF}_2$ secure, where $\mathtt{Corr}_2 = \mathtt{Corr}_1$, $\mathtt{KGCh}_2 = \min(\mathtt{KGCh}_1, 1{:}\,\mathsf{KGaC})$, $\mathtt{SiCh}_2 = \mathtt{SiCh}_1$, $\mathtt{UF}_2 = \mathtt{UF}_1$, and $\epsilon_{\mathsf{TS}'} \leq n\epsilon_{\mathsf{DS}} + (n-t)\epsilon_{\mathsf{TS}}$.

K.1 Proof of Claim 1

We prove Claim 1 by reduction. If there exists an adversary $\mathcal{A}$ that can break the ϵ_{TS}-(3: $\mathsf{tLRhPP}, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1$)-$\mathtt{UF}_1$ security of TS', then we can construct an adversary $\mathcal{B}$ that breaks ϵ_{TS}-(3: $\mathsf{tLRhPP}, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1$)-$\mathtt{UF}_1$ of TS.

First, $\mathcal{B}$ initializes a sequence of states $\mathsf{st}_i^{\mathsf{ATS}}$ for all $i \in \{0, \ldots, n\}$, i.e., the leader and every signer. Moreover, $\mathcal{B}$ initializes counters rnd_i for $i \in [n]$ and $\mathsf{rnd}_{i,j}$ for $i \in [n]$ and $j \geq 1$ (whenever needed) that count the index of the next round for key generation or signing. Then, $\mathcal{B}$ invokes $\mathcal{A}$ and answers all queries to oracles from $\mathcal{A}$ as follows:

- OKGen(i, m): We consider the following four cases:

  1. If $u = 0$ and $\mathsf{rnd}_i = 0$, $\mathcal{B}$ samples a DS verification-signing key pair $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}})$ by running $\mathsf{DS.KGen}$ and stores them into the state $\mathsf{st}_i^{\mathsf{ATS}}$ locally. Then, $\mathcal{B}$ outputs a sequence of message $m'_{(i,i')} = vk^{\mathsf{DS}}$ for all $i' \in [n] \setminus \{i\}$ to $\mathcal{A}$.
  2. If $u = 0$ and $\mathsf{rnd}_i = 1$, $\mathcal{B}$ parses the input message to a sequences of DS verification key $vk_{i'}^{\mathsf{DS}}$ for all $i' \in [n] \setminus \{i\}$ and stores them into $\mathsf{st}_i^{\mathsf{ATS}}$. Then, $\mathcal{B}$ sends a query OKGen$(i, \mathbf{0})$ to its challenger and forwards the reply to $\mathcal{A}$.
  3. If $u \geq 1$ and $\mathsf{rnd}_i = 0$, $\mathcal{B}$ samples a DS verification-signing key pair $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}})$ by running $\mathsf{DS.KGen}$ and stores them into the state $\mathsf{st}_i^{\mathsf{ATS}}$ locally. Then, $\mathcal{B}$ sends a query OKGen$(i, \mathbf{0})$ to its challenger for a reply $\|_{i' \in [n] \setminus \{i\}}\, m'_{(i,i')}$. Finally, $\mathcal{B}$ appends vk^{DS} to every $m'_{(i,i')}$ and returns the appended $\|_{i' \in [n] \setminus \{i\}}\, m'_{(i,i')}$ to $\mathcal{A}$.
  4. If $u \geq 1$ and $\mathsf{rnd}_i \geq 1$, $\mathcal{A}$ extracts a sequence of DS verification key $vk_{i'}^{\mathsf{DS}}$ for $i' \in [n] \setminus \{i\}$ from the input message m and stores them into the state $\mathsf{st}_i^{\mathsf{ATS}}$. Then, $\mathcal{B}$ sends i and the rest of the input message, which removes $\{vk_{i'}^{\mathsf{DS}}\}_{i' \in [n] \setminus \{i\}}$ from m, to its OKGen oracle and forwards the reply to $\mathcal{A}$.

- OCorrupt(i): $\mathcal{B}$ forwards the query to its challenger. If $\mathcal{B}$ receives a signer state S_i that is non-$\perp$, $\mathcal{B}$ returns both S_i and $\mathsf{st}_i^{\mathsf{ATS}}$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ returns nothing.
- OPP(i): $\mathcal{B}$ first checks whether $v = 1$. If $v = 0$, $\mathcal{B}$ exits. Otherwise, $\mathcal{B}$ forwards the query to its challenger for a pre-processing token pp. Then, $\mathcal{B}$ use DS along with the signing key $\mathsf{st}_i^{\mathsf{ATS}}.sk$ to sign pp for a signature σ^{pp}. Finally, $\mathcal{B}$ forwards both (pp, σ^{pp}) and the index i to $\mathcal{A}$.
- OSign(i, j, m): We consider the following two cases:

  1. If $\mathsf{rnd}_{i,j} = 0$ and $v = 1$, $\mathcal{B}$ parses the input message m as a leader request lr. Next, for each $i' \in lr.\mathsf{SS}$, $\mathcal{B}$ parses a pre-processing token $pp_{i'}$ and an associated signature $\sigma_{i'}^{pp}$ from $lr.\mathsf{PP}(i')$. $\mathcal{B}$ verifies whether $\sigma_{i'}^{pp}$ is a valid DS signature of pre-processing token $pp_{i'}$ by using the DS verification key $\mathsf{st}_i^{\mathsf{ATS}}.VK[i']$. If the check fails, then $\mathcal{B}$ immediately exists and returns $\perp$ to $\mathcal{A}$. If the check passes, $\mathcal{B}$ forwards (i, j, lr') to its oracle OSign, where lr' is identical to lr except for removing all signature $\sigma_{i'}^{pp}$ from $lr.\mathsf{PP}(i')$ for all $i' \in lr.\mathsf{SS}$. Finally, $\mathcal{B}$ forwards the reply from its challenger to $\mathcal{A}$.
  2. Otherwise, $\mathcal{B}$ forwards the query to its challenger and forwards the reply from its challenger to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a challenge message-signature pair $(m^\star, \sigma^\star)$ and wins if (1) $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$, (2) $|\mathcal{L}_{\mathsf{MS}}| < t$, and (3) $\neg\big(\exists lr$ with $lr.m = m^\star$: $\mathtt{tf}_{3:\mathsf{tLRhPP}}(lr)\big)$ (if $\mathtt{UF} = \mathsf{EUF\text{-}CMA}$) or $\neg\big(\exists lr$ with $lr.m = m^\star$: $\mathtt{tsf}_{3:\mathsf{tLRhPP}}(lr)\big)$ (if $\mathtt{UF} = \mathsf{SUF\text{-}CMA}$).

$\mathcal{B}$ forwards the challenge message-signature pair $(m^\star, \sigma^\star)$ to its challenger and wins if (1) $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$, (2) $|\mathcal{L}_{\mathsf{MS}}| < t$, and (3) $\neg\big(\exists lr$ with $lr.m = m^\star$: $\mathtt{tf}_{3:\mathsf{tLRhPP}}(lr)\big)$ (if $\mathtt{UF} = \mathsf{EUF\text{-}CMA}$) or $\neg\big(\exists lr$ with $lr.m = m^\star$: $\mathtt{tsf}_{3:\mathsf{tLRhPP}}(lr)\big)$ (if $\mathtt{UF} = \mathsf{SUF\text{-}CMA}$).

Note that $\mathcal{B}$ perfectly simulates the $\mathrm{Game}_{\mathsf{TS}'}^{\mathtt{UF}}$ to $\mathcal{A}$, and wins if and only if $\mathcal{A}$ wins. Thus, if $\mathcal{A}$ can win with probability ϵ_{TS}, then so can $\mathcal{B}$.

K.2 Proof of Claim 2

We prove Claim 2 by reduction. If there exists an adversary $\mathcal{A}$ that can break the (4: $\mathsf{aLRhPP}, \mathtt{Corr}_2$, $\mathtt{KGCh}_2, \mathtt{SiCh}_2$)-$\mathtt{UF}_2$ security of TS', where $\mathtt{Corr}_2 = \mathtt{Corr}_1$, $\mathtt{KGCh}_2 = \min(\mathtt{KGCh}_1, 1\text{: }\mathsf{KGaC})$, $\mathtt{SiCh}_2 = \mathtt{SiCh}_1$, $\mathtt{UF}_2 = \mathtt{UF}_1$, then we can construct an adversary $\mathcal{B}$ that breaks ϵ_{DS}-SUF-CMA security of DS.

First, $\mathcal{B}$ samples an index $i^\star \in [n]$ uniformly at random. Next, $\mathcal{B}$ initializes a sequence of long-term states $\mathsf{st}_i^{\mathsf{TS}}$ and $\mathsf{st}_i^{\mathsf{ATS}}$ for all $i \in \{0, \ldots, n\}$, i.e., the leader and every signer, and a sequence of session states π_i^j for all $j \geq 1$ (whenever needed). Moreover, $\mathcal{B}$ initializes counters rnd_i for $i \in [n]$ and $\mathsf{rnd}_{i,j}$ for $i \in [n]$ and $j \geq 1$ (whenever needed) that count the index of the next round for key generation or signing. $\mathcal{B}$ receives a DS challenge public verification key $vk^\star$ from its challenger. Then, $\mathcal{B}$ invokes $\mathcal{A}$ and honestly simulates the honest execution of the corresponding oracle by itself, because $\mathcal{B}$ owns all states that are needed:

- $\text{OKGen}(i, m)$ with $i \in [n] \setminus \{i^\star\}$
- $\text{OCorrupt}(i)$ with $i \in [n] \setminus \{i^\star\}$
- $\text{OPP}(i)$ with $i \in [n] \setminus \{i^\star\}$
- $\text{OSign}(i, j, m)$ with $i \in [n]$

For all other queries, $\mathcal{B}$ answers in the following ways:

- $\text{OKGen}(i^\star, m)$: We consider the following four cases:
  1. If $u = 0$ and $\mathsf{rnd}_{i^\star} = 0$, $\mathcal{B}$ stores $vk^\star$ into the state $\mathsf{st}_{i^\star}^{\mathsf{ATS}}.VK[i^\star]$ locally. Then, $\mathcal{B}$ outputs a sequence of message $m'_{(i,i')} = vk^\star$ for all $i' \in [n] \setminus \{i\}$ to $\mathcal{A}$.
  2. If $u = 0$ and $\mathsf{rnd}_{i^\star} = 1$, $\mathcal{B}$ parses the input message to a sequences of DS verification key $vk_{i'}^{\mathsf{DS}}$ for all $i' \in [n] \setminus \{i\}$ and stores them into $\mathsf{st}_{i^\star}^{\mathsf{ATS}}.VK[i']$. Then, $\mathcal{B}$ sends a query $\text{OKGen}(i^\star, \mathbf{0})$ to its challenger and forwards the reply to $\mathcal{A}$.
  3. If $u \geq 1$ and $\mathsf{rnd}_{i^\star} = 0$, $\mathcal{B}$ stores $vk^\star$ into the state $\mathsf{st}_{i^\star}^{\mathsf{ATS}}.VK[i^\star]$ locally. Then, $\mathcal{B}$ sends a query $\text{OKGen}(i, \mathbf{0})$ to its challenger for a reply $\|_{i' \in [n] \setminus \{i\}} m'_{(i,i')}$. Finally, $\mathcal{B}$ appends $vk^\star$ to every $m'_{(i,i')}$ and returns the appended $\|_{i' \in [n] \setminus \{i\}} m'_{(i,i')}$ to $\mathcal{A}$.
  4. If $u \geq 1$ and $\mathsf{rnd}_{i^\star} \geq 1$, $\mathcal{A}$ extracts a sequence of DS verification key $vk_{i'}^{\mathsf{DS}}$ for $i' \in [n] \setminus \{i^\star\}$ from the input message m and stores them into the state $\mathsf{st}_{i^\star}^{\mathsf{ATS}}.VK[i']$. Then, $\mathcal{B}$ sends $i^\star$ and the rest of the input message, which removes $\{vk_{i'}^{\mathsf{DS}}\}_{i' \in [n] \setminus \{i^\star\}}$ from m, to its OKGen oracle and forwards the reply to $\mathcal{A}$.
- $\text{OCorrupt}(i^\star)$: $\mathcal{B}$ aborts and loses. Note that $\mathcal{A}$ can query OCorrupt at most $t-1$ times, and $i^\star$ is sampled uniformly at random. The probability that $\mathcal{A}$ sends such queries happens with probability at most $\frac{t}{n}$.
- $\text{OPP}(i^\star)$: $\mathcal{B}$ first checks whether $v = 1$. If $v = 0$, $\mathcal{B}$ exits. Otherwise, $\mathcal{B}$ forwards the query to its challenger for a pre-processing token pp. Then, $\mathcal{B}$ forwards pp to its challenger for a signature σ^{pp}. Finally, $\mathcal{B}$ forwards both (pp, σ^{pp}) and the index i to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a challenge message-signature pair $(m^\star, \sigma^\star)$ and wins if

- $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$,
- $|\mathcal{L}_{\mathsf{MS}}| < t$, and
- $\neg\big(\exists lr \text{ with } lr.m = m^\star\colon \mathtt{tf}_{4:\mathsf{aLRhPP}}(lr)\big)$ (if $\mathtt{UF} = \mathsf{EUF\text{-}CMA}$) or $\neg\big(\exists lr \text{ with } lr.m = m^\star\colon \mathtt{tsf}_{4:\mathsf{aLRhPP}}(lr, gvk, \sigma^\star)\big)$ (if $\mathtt{UF} = \mathsf{SUF\text{-}CMA}$).

$\mathcal{B}$ first checks whether $\mathsf{Vrfy}(gvk, m^\star, \sigma^\star) = \mathsf{true}$ and whether $|\mathcal{L}_{\mathsf{MS}}| < t$. If any check fails, then both $\mathcal{A}$ and $\mathcal{B}$ lose. $\mathcal{A}$ can win if and only if $\neg\big(\exists lr \text{ with } lr.m = m^\star\colon \mathtt{tf}_{4:\mathsf{aLRhPP}}(lr)\big)$ (if $\mathtt{UF} = \mathsf{EUF\text{-}CMA}$) or $\neg\big(\exists lr \text{ with } lr.m = m^\star\colon \mathtt{tsf}_{4:\mathsf{aLRhPP}}(lr, gvk, \sigma^\star)\big)$ (if $\mathtt{UF} = \mathsf{SUF\text{-}CMA}$). Note that $\mathtt{tsf}_{4:\mathsf{aLRhPP}}(lr, gvk, \sigma^\star) = \mathtt{tf}_{4:\mathsf{aLRhPP}}(lr)$ and $\mathsf{SVf}(gvk, lr, \sigma)$. We know that $\mathcal{A}$ wins only if

$$\begin{aligned}
&\neg\big(\exists lr \text{ with } lr.m = m^\star : \mathtt{tf}_{4:\mathsf{aLRhPP}}(lr)\big)\\
\Leftrightarrow&\forall lr \text{ with } lr.m = m^\star : \neg\mathtt{tf}_{4:\mathsf{aLRhPP}}(lr)\\
\Leftrightarrow&\forall lr \text{ with } lr.m = m^\star : \neg\big(\mathtt{tf}_{2:\mathsf{tLR}}(lr) \text{ and } \mathcal{D}_2[lr] = \mathcal{D}_3[lr] = \mathcal{D}_4[lr]\big)\\
\Leftrightarrow&\forall lr \text{ with } lr.m = m^\star : \neg\mathtt{tf}_{2:\mathsf{tLR}}(lr) \text{ or } \neg\big(\mathcal{D}_2[lr] = \mathcal{D}_3[lr] = \mathcal{D}_4[lr]\big)
\end{aligned}$$

Recall from Claim 1 that TS' is ϵ_{TS}-$(3\colon \mathtt{tLRhPP}, \mathtt{Corr}_1, \mathtt{KGCh}_1, \mathtt{SiCh}_1)$-$\mathtt{UF}_1$ secure, which means that it holds with probability at least $(1 - \epsilon_{\mathsf{TS}})$ that

$$\begin{aligned}
&\exists lr \text{ with } lr.m = m^\star : \mathtt{tf}_{3:\mathsf{tLRhPP}}(lr)\\
\Leftrightarrow&\exists lr \text{ with } lr.m = m^\star : \mathtt{tf}_{2:\mathsf{tLR}}(lr) \text{ and } \mathcal{D}_2[lr] = \mathcal{D}_3[lr]
\end{aligned}$$

This means that $\mathcal{A}$ wins only if it holds for all above lr with (1) $lr.m = m^\star$, (2) $\mathtt{tf}_{2:\mathsf{tLR}}(lr)$, and (3) $\mathcal{D}_2[lr] = \mathcal{D}_3[lr]$ that

$$\mathcal{D}_3[lr] \neq \mathcal{D}_4[lr]$$

Recall that $\mathcal{D}_3[lr] \subseteq \mathcal{D}_4[lr]$ by definition. Therefore, $\mathcal{A}$ can only win if

$$\mathcal{D}_4[lr] \not\subseteq \mathcal{D}_3[lr]$$

which means that there exists a signer $i \in \mathcal{D}_4[lr]$ but $i \notin \mathcal{D}_3[lr]$. This further means that there exists an honest signer $i \in lr.\mathsf{SS}$ such that $lr.\mathsf{PP}(i) = (pp_i, \sigma_i^{pp})$ is not honestly generated.

If such $i = i^\star$ exists, $\mathcal{B}$ returns (pp_i, σ_i^{pp}) to its challenger. In this case, the adversary $\mathcal{B}$ wins the SUF-CMA security model by definition. Recall that $i^\star$ is chosen uniformly at random at the beginning of the reduction, the probability that $i = i^\star$ holds with $\frac{1}{n-t}$ (under the condition that the reduction does not abort). Otherwise, $\mathcal{B}$ immediately loses. In summary, it holds that

$$\epsilon_{\mathsf{TS}'} \leq n\epsilon_{\mathsf{DS}} + (n-t)\epsilon_{\mathsf{TS}}$$

□

L Our $\mathsf{ATS}^{\mathsf{3:tLRhPP}}_{\mathsf{4:aLRhPP}}$ Transformation

We depict our $\mathsf{ATS}^{\mathsf{3:tLRhPP}}_{\mathsf{4:aLRhPP}}[\mathsf{TS}, \mathsf{DS}]$ transformation in Figure 8. It transforms a (t, n)-threshold signature scheme TS with (u, v, w) rounds to a new (t, n)-threshold signature scheme $\mathsf{ATS}^{\mathsf{3:tLRhPP}}_{\mathsf{4:aLRhPP}}[\mathsf{TS}, \mathsf{DS}]$ with $(\max(1, u), v, w)$ rounds by using a digital signature DS. For every $i \in \{0, \ldots, n\}$, the state st_i includes two sub-states: $\mathsf{st}_i^{\mathsf{TS}}$ is the state of the TS and $\mathsf{st}_i^{\mathsf{ATS}}$ is an additional state in our transformation. Below, we explain each algorithm in our $\mathsf{ATS}^{\mathsf{3:tLRhPP}}_{\mathsf{4:aLRhPP}}$ transformation.

Key Generation. The key generation algorithm KGen is divided into the following two cases:

1. If the original TS has zero key generation rounds among signers ($u = 0$), the $\mathsf{ATS}^{\mathsf{3:tLRhPP}}_{\mathsf{4:aLRhPP}}$ transformation has to increment the number of rounds by 1. In the $\mathsf{KGen}^{(0)}(\mathsf{st}_i)$ algorithm, every signer i first parses st_i into two sub-states $\mathsf{st}_i^{\mathsf{TS}}$ and $\mathsf{st}_i^{\mathsf{ATS}}$. Next, signer i generates a DS verification-signing key pair and stores them into $\mathsf{st}_i^{\mathsf{ATS}}$. Finally, signer i sends the DS verification key to every other signer. In the $\mathsf{KGen}^{(1)}(\mathsf{st}_i, m)$ algorithm, signer i first parses every other signers' DS verification key from the input message m and stores them in the state $\mathsf{st}_i^{\mathsf{ATS}}$. Finally, signer i runs the $m' \xleftarrow{\$} \mathsf{KGen}^{(0)}(\mathsf{st}_i)$ algorithm of the original TS protocol and outputs m'.
2. If the original TS has more than zero key generation rounds among signers ($u \geq 1$), the KGen algorithm is almost identical to the one of the original TS, with one minor exception: The $\mathsf{ATS}^{\mathsf{3:tLRhPP}}_{\mathsf{4:aLRhPP}}$ transformation first samples a DS verification-signing key pair, stores them into $\mathsf{st}_i^{\mathsf{ATS}}$, and distributes the DS verification key in $\mathsf{KGen}^{(0)}$. Finally, it stores other signers' DS verification keys in the beginning of $\mathsf{KGen}^{(1)}$.

Verification Key Aggregation. The verification key aggregation algorithm VkAgg invokes and outputs the VkAgg of the original TS protocol using $\mathsf{st}_0^{\mathsf{TS}}$.

Signer Pre-Processing. The signer pre-processing algorithm SPP is only invoked if $v = 1$. First, signer i invokes the SPP algorithm of the original TS protocol to receive a pre-processing token pp. Then, signer i signs pp using the DS signing key stored in the state $\mathsf{st}_i^{\mathsf{ATS}}$ to produce a pre-processing signature σ^{pp}. Finally, signer i outputs the index i, the pre-processing token pp, and the associated signature σ^{pp}.

Leader Pre-Processing. The leader pro-processing algorithm LPP is only invoked if $v = 1$. First, the leader parses the input message into a tuple containing a signer index i, a pre-processing token pp, and an associated signature σ^{pp}. Then, the leader runs the LPP algorithm of the original TS protocol using the corresponding state $\mathsf{st}_0^{\mathsf{TS}}$ and the token pp. Finally, the leader stores (pp, σ^{pp}) in the local pre-processing dictionary $\mathcal{D}_{\mathsf{PP}}[i]$ in the state $\mathsf{st}_0^{\mathsf{ATS}}$.

Leader Signing-Request. The leader first produces a leader request lr using the LR algorithm of the original TS protocol. Then, if $v = 1$, the leader replaces the pre-processing token $lr.\mathsf{PP}(i)$ with the corresponding tuple that was stored in $\mathsf{st}_0^{\mathsf{ATS}}.\mathcal{D}_{\mathsf{PP}}[i]$ for all $i \in lr.\mathsf{SS}$. If such a tuple does not exist for any

```
$\underline{\mathsf{KGen}^{(0)}(\mathsf{st}_i)}$ // if $u = 0$:
  $(\mathsf{st}_i^{\mathsf{TS}}, \mathsf{st}_i^{\mathsf{ATS}}) \leftarrow \mathsf{st}_i$
  $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}}) \xleftarrow{\$} \mathsf{DS.KGen}$
  $\mathsf{st}_i^{\mathsf{ATS}}.sk \leftarrow sk^{\mathsf{DS}}$; $\mathsf{st}_i^{\mathsf{ATS}}.VK[i] \leftarrow vk^{\mathsf{DS}}$
  foreach $i \in [n] \setminus \{i\}$ do
    $m'_{(i,i')} \leftarrow vk^{\mathsf{DS}}$
  return $\|_{i' \in [n] \setminus \{i\}}\, m'_{(i,i')}$

$\underline{\mathsf{KGen}^{(1)}(\mathsf{st}_i, m)}$ // if $u = 0$
  $(\mathsf{st}_i^{\mathsf{TS}}, \mathsf{st}_i^{\mathsf{ATS}}) \leftarrow \mathsf{st}_i$
  parse $\|_{i' \in [n] \setminus \{i\}}\, vk_{i'}^{\mathsf{DS}} \leftarrow m$
  foreach $i' \in [n] \setminus \{i\}$ do
    $\mathsf{st}_i^{\mathsf{ATS}}.VK[i'] \leftarrow vk_{i'}^{\mathsf{DS}}$
  $m' \leftarrow \mathsf{TS.KGen}^{(0)}(\mathsf{st}_i^{\mathsf{TS}})$
  return $m'$

$\underline{\mathsf{KGen}^{(0)}(\mathsf{st}_i)}$ // if $u \geq 1$
  $(\mathsf{st}_i^{\mathsf{TS}}, \mathsf{st}_i^{\mathsf{ATS}}) \leftarrow \mathsf{st}_i$
  $(vk^{\mathsf{DS}}, sk^{\mathsf{DS}}) \xleftarrow{\$} \mathsf{DS.KGen}$
  $\mathsf{st}_i^{\mathsf{ATS}}.sk \leftarrow sk^{\mathsf{DS}}$; $\mathsf{st}_i^{\mathsf{ATS}}.VK[i] \leftarrow vk^{\mathsf{DS}}$
  $\|_{i' \in [n] \setminus \{i\}}\, m'_{(i,i')} \leftarrow \mathsf{TS.KGen}^{(0)}(\mathsf{st}_i^{\mathsf{TS}})$
  foreach $i \in [n] \setminus \{i\}$ do
    $m'_{(i,i')} \overset{\|}{\leftarrow} vk^{\mathsf{DS}}$
  return $\|_{i' \in [n] \setminus \{i\}}\, m'_{(i,i')}$

$\underline{\mathsf{KGen}^{(\mathsf{rnd})}(\mathsf{st}_i, m)}$ // if $u \geq 1$ and $\mathsf{rnd} \geq 1$
  $(\mathsf{st}_i^{\mathsf{TS}}, \mathsf{st}_i^{\mathsf{ATS}}) \leftarrow \mathsf{st}_i$
  if $\mathsf{rnd} = 1$ then
    parse $\left(m, \|_{i' \in [n] \setminus \{i\}}\, vk_{i'}^{\mathsf{DS}}\right) \leftarrow m$
    foreach $i' \in [n] \setminus \{i\}$ do
      $\mathsf{st}_i^{\mathsf{ATS}}.VK[i'] \leftarrow vk_{i'}^{\mathsf{DS}}$
  $m' \leftarrow \mathsf{TS.KGen}^{(\mathsf{rnd})}(\mathsf{st}_i^{\mathsf{TS}}, m)$
  return $m'$

$\underline{\mathsf{VkAgg}(\mathsf{st}_0, \{m_i\}_{i \in [n]})}$
  $(\mathsf{st}_0^{\mathsf{TS}}, \mathsf{st}_0^{\mathsf{ATS}}) \leftarrow \mathsf{st}_0$
  return $\mathsf{TS.VkAgg}(\mathsf{st}_0^{\mathsf{TS}}, \{m_i\}_{i \in [n]})$
```

```
$\underline{\mathsf{SPP}(\mathsf{st}_i)}$
  req $v = 1$
  $(\mathsf{st}_i^{\mathsf{TS}}, \mathsf{st}_i^{\mathsf{ATS}}) \leftarrow \mathsf{st}_i$
  $pp \xleftarrow{\$} \mathsf{TS.SPP}(\mathsf{st}_i^{\mathsf{TS}})$; $\sigma^{pp} \xleftarrow{\$} \mathsf{DS.Sign}(\mathsf{st}_i^{\mathsf{ATS}}.sk, pp)$
  return $(i, pp, \sigma^{pp})$

$\underline{\mathsf{LPP}(\mathsf{st}_0, m)}$
  req $v = 1$
  $(\mathsf{st}_0^{\mathsf{TS}}, \mathsf{st}_0^{\mathsf{ATS}}) \leftarrow \mathsf{st}_0$
  parse $(i, pp, \sigma^{pp}) \leftarrow m$
  $\mathsf{TS.LPP}(\mathsf{st}_0^{\mathsf{TS}}, pp)$; $\mathsf{st}_0^{\mathsf{ATS}}.\mathcal{D}_{\mathsf{PP}}[i] \leftarrow (pp, \sigma^{pp})$
  return

$\underline{\mathsf{LR}(\mathsf{st}_0, SS, m)}$
  $(\mathsf{st}_0^{\mathsf{TS}}, \mathsf{st}_0^{\mathsf{ATS}}) \leftarrow \mathsf{st}_0$; $lr \leftarrow \mathsf{TS.LR}(\mathsf{st}_0^{\mathsf{TS}}, SS, m)$
  if $v = 1$ do
    foreach $i \in lr.\mathsf{SS}$ do
      $pp \leftarrow lr.\mathsf{PP}(i)$
      req $\exists \sigma^{pp}$ s.t. $(pp, \sigma^{pp}) \in \mathsf{st}_0^{\mathsf{ATS}}.\mathcal{D}_{\mathsf{PP}}[i]$
      $lr.\mathsf{PP}(i) \leftarrow (pp, \sigma^{pp})$
  return $lr$

$\underline{\mathsf{Sign}^{(\mathsf{rnd})}(S_i, m)}$
  $(\mathsf{st}_i^{\mathsf{TS}}, \mathsf{st}_i^{\mathsf{ATS}}) \leftarrow \mathsf{st}_i$
  if $\mathsf{rnd} = 0$ and $v = 1$ then
    parse $lr \leftarrow m$
    foreach $i' \in lr.\mathsf{SS}$ do
      parse $(pp_{i'}, \sigma_{i'}^{pp}) \leftarrow lr.\mathsf{PP}(i')$
      req $\mathsf{DS.Vrfy}\left(\mathsf{st}_i^{\mathsf{ATS}}.VK[i'], pp_{i'}, \sigma_{i'}^{pp}\right)$
      $lr.\mathsf{PP}(i') \leftarrow pp_{i'}$
    $m \leftarrow lr$
  $S_i^{\mathsf{TS}} \leftarrow S_i \setminus \{\mathsf{st}_i\} \cup \{\mathsf{st}_i^{\mathsf{TS}}\}$
  $m' \xleftarrow{\$} \mathsf{TS.Sign}^{(\mathsf{rnd})}(S_i^{\mathsf{TS}}, m)$
  return $m'$

$\underline{\mathsf{SigAgg}(\mathsf{st}_0, lr, \{\varsigma_i\}_{i \in lr.SS})}$
  $(\mathsf{st}_0^{\mathsf{TS}}, \mathsf{st}_0^{\mathsf{ATS}}) \leftarrow \mathsf{st}_0$
  return $\mathsf{TS.SigAgg}(\mathsf{st}_0^{\mathsf{TS}}, lr, \{\varsigma_i\}_{i \in lr.SS})$

$\underline{\mathsf{Vrfy}(gvk, m, \sigma)}$
  return $\mathsf{TS.Vrfy}(gvk, m, \sigma)$

$\underline{\mathsf{SVrfy}(gvk, lr, \sigma)}$
  if $v = 1$ then
    foreach $i \in lr.\mathsf{SS}$ do
      $(pp, \sigma^{pp}) \leftarrow lr.\mathsf{PP}(i)$; $lr.\mathsf{PP}(i) \leftarrow pp$
  return $\mathsf{TS.SVrfy}(gvk, lr, \sigma)$
```

Figure 8: Our $\mathsf{ATS}^{3:\mathsf{tLRhPP}}_{4:\mathsf{aLRhPP}}[\mathsf{TS}, \mathsf{DS}]$ transformation that transforms a (t, n)-threshold signature scheme TS with (u, v, w) rounds to a new (t, n)-threshold signature scheme $\mathsf{ATS}^{3:\mathsf{tLRhPP}}_{4:\mathsf{aLRhPP}}[\mathsf{TS}, \mathsf{DS}]$ with $(\max(1, u), v, w)$ rounds by using a digital signature DS.

$i \in lr.\mathsf{SS}$, the leader undoes all execution within this algorithm and exits. If no error occurs, the leader outputs the (possibly modified) leader request lr.

Signing. The signing algorithm Sign is mostly identical to the one of the original TS protocol, except for round 0: If $v = 1$, signer i extracts a pre-processing token $pp_{i'}$ and an associated signature $\sigma_{i'}^{pp}$ from the leader request lr for every $i' \in lr.\mathsf{SS}$. Then, signer i checks whether the token $pp_{i'}$ can be verified by the signature $\sigma_{i'}^{pp}$ under the DS verification key that was stored in $\mathsf{st}_i^{\mathsf{ATS}}.VK[i']$. If any verification fails, signer i undoes all execution within this algorithm and exits.

Signature Aggregation. The signature aggregation algorithm SigAgg invokes and outputs the SigAgg algorithm of the original TS protocol by using state $\mathsf{st}_0^{\mathsf{TS}}$.

Verification. The verification algorithm Vrfy invokes and outputs the verification algorithm of the original TS protocol.

Strong Verification. The strong verification algorithm SVrfy first removes the pre-processing signature σ_i^{pp} from $lr.\mathsf{PP}(i)$ for all $i \in lr.\mathsf{SS}$ if $v = 1$. Then (or in case $v = 0$), it invokes and outputs the SVrfy algorithm of the original TS protocol.

M Security Guarantees for SiGu

Table 3: Security guarantees for each level of signer guarantees in our hierarchy. Let HS be the set of honest signers that are willing to sign a message, CS the set of compromised signers, and SS a subset of all signers. We assume that the adversary has corrupted the leader and some of the signers, s.t. $|CS| \leq t-1$.

SiGu	Security guarantees
0: eM	An adversary cannot forge a signature with access to less than t partial signatures, s.t. $|HS| = 0$ and $|CS| \leq t-1$.
1: tM	An adversary cannot forge a signature with access to less than t partial signatures, s.t. $|HS| + |CS| \leq t-1$.
2: tLR	An adversary cannot forge a signature for a given leader request lr with access to less than t partial signatures, s.t. $|HS_{lr}| + |CS_{lr}| \leq t-1$, where $HS_{lr} = HS \cap lr.SS$ is the subset of honest signers in $lr.SS$ and $CS_{lr} = CS \cap lr.SS$ is the subset of compromised signers in $lr.SS$.
3: tLRhPP	An adversary cannot forge a signature for a given leader request lr with access to less than t partial signatures, s.t. $|HS_{\mathsf{PP}}| + |CS_{lr}| \leq t-1$, where $HS_{\mathsf{PP}} = HS \cap lr.SS$ is the subset of honest signers in $lr.SS$ that received a valid pre-processing token in the request and $CS_{lr} = CS \cap lr.SS$ is the subset of compromised signers in $lr.SS$.
4: aLRhPP	An adversary cannot forge a signature for a given leader request lr without all honest signers in lr receiving valid pre-processing tokens and providing a partial signature, i.e., $|HS_{\mathsf{PP}}| = |lr.SS|$, where $HS_{\mathsf{PP}} = HS \cap lr.SS$ is the subset of honest signers in $lr.SS$ that received a valid pre-processing token in the request.

We give an alternative characterization of the Signer Guarantee levels in Table 3.

N Distributed Key Generation

Pedersen's distributed key generation (DKG) protocol [46] consists of three steps:

1. Share distribution. Each signer n_i:

- Generates a random polynomial $f_i(x) = \sum_{i=0}^{t-1} a_i x^i$ and a corresponding commitment polynomial $F_i(x) = \prod_i g^{a_i x^i}$
- Broadcasts a list of commitments corresponding to the commitment polynomial.
  - $[A_{i,0}, \ldots, A_{i,t-1}]$ is the list of commitments for n_i.
- Generates a random secret s_i, splits it into n shares, and distributes them to the other signers.

    - n_j receives share $s_{i,j} = (j, f_i(j))$ from n_i.

2. **Share verification**.
   - When n_j receives $(j, s_{i,j})$, it verifies that the share matches the corresponding commitments:
     - $g^{s_{i,j}} = \prod_{k=0}^{t-1} (A_{i,k})^{j^k}$.
   - If a signer receives an invalid share, it broadcasts a complaint.
     - If the signer responsible of the invalid share fails to falsify the complaint by revealing the (valid) share, it is considered invalid and removed from the group.

3. **Share finalization**.
   - The final (secret) share of n_i is:
     - $s_i = \sum_{j=1}^{n} s_{j,i}$ for all valid signers j
   - The collective public key associated with the valid shares is:
     - $S = \sum_{j=1}^{n} A_{j,0}$ for all valid signers j

N.1 Rogue-Key Attack

Since Pedersen's protocol does not require any previous knowledge of the other participants, a group of malicious signers can perform a *rogue-key attack*. In a group of n signers, the adversary first receives commitments $A_{1,0}, \ldots, A_{n-1,0}$ from the other signers, then chooses $X = Sk_A$, and claims that:

$$A_{n,0} = g^{X - \sum_{i=1}^{n-1} a_{i,0}}, \text{ i.e., } s_{n,0} = X - \sum_{i=1}^{n-1} a_{i,0}$$

When the signers calculate their shared key, it becomes:

$$Sk_G = (X - \sum_{i=1}^{n-1} a_{i,0}) + \sum_{i=1}^{n-1} a_{i,0} = X$$

This allows the adversary to sign messages with Sk_A on behalf of the group.

N.2 Proposed Mitigation Methods

A common way to analyze key distribution schemes is to require knowledge of the secret key (KOSK) [47]. Since the commitment the adversary sends to the honest signers is derived from their commitments, the adversary does not know the corresponding secret. By requiring proof of knowing the secret key (e.g., with a zero-knowledge proof) we can prevent the adversary from choosing a fraudulent commitment.

Proof of Possession (POP). Proof of possession requires each signer to prove that they have access to the secret corresponding to a commitment. For example, in FROST [42], each signer computes a zero-knowledge proof of the value $a_{i,0}$:

$\sigma_i = (R_i, \mu_i)$, where

- k is a random value,
- $R_i = g^k$,
- Φ is a context string to prevent replay attacks,
- $c_i = H(i, \Phi, g^{a_{i,0}}, R_i)$, and
- $\mu_i = k + a_{i,0} \cdot c_i$

This can then be checked by the receiver by verifying that

$$R_i = g^{\mu_i} \cdot \phi_{i,0}^{-c_i}, \text{ where } c_i = H(i, \Phi, \phi_{i,0}, R_i)$$

The algorithm is referred to as PedPoP in later publications.

Key Commitment. Key commitment is a multi-round solution that requires each party to commit to their secret before sending or receiving anything. For example, each node can be asked to send a hash of their commitments and only proceeding once everyone has done so.

O Practical Applications for Threshold Signatures

Threshold signature schemes have in recent years re-gained popularity as an alternative to traditional signatures and have been adapted for applications, such as blockchains [53], timelock encryption [34], and cryptocurrency wallets [58]. Next, we briefly introduce examples of each use case and discuss potential security risks of implementing schemes of lower hierarchy levels.

Blockchain Interoperability. Blockchain interoperability is an open problem that has received significant attention through the emergence and recent popularity of cryptocurrencies. Users wishing to transfer or convert their coins between two (non-interoperable) blockchains have traditionally been forced to rely on centralized exchanges, violating the goal of decentralization. Furthermore, giving a single entity access to all funds on multiple blockchains poses a severe security risk.

Threshold signatures have been proposed as a solution to this problem (e.g., [3, 17, 52, 57]): by distributing control of funds over multiple independent signers, the single point of failure can be eliminated. Each signer is incentivized to validate transactions over the exchange and provide partial signatures to accept them. If sufficiently many validators sign a given transaction, it gets executed.

Since each signer is expected to actively validate transaction and is penalized for failure to participate, the subset of signers that is included in producing any given group signature does not matter. In other words, the hierarchy level of the chosen scheme has no practical implications, as long as it requires at least t signers to produce a valid signature.

However, choosing a scheme vulnerable to rogue-key attacks gives malicious signers a non-negligible advantage. In many of these implementations, the number of signers is relatively high (100+) and the threshold t is often around $2/3 \times n$. Recall that a rogue-key attack allows a malicious signer to control the group's secret key by compromising $n - t + 1$ signers. For example, in a scheme with $n = 150$ and $t = 100$, this would mean that the adversary only needs to compromise approximately a third of the signers for the attack to work. The requirement to become a signer is typically a sufficiently large proof-of-stake.

Distributed Randomness Beacons. Threshold signatures can also be used to create pseudorandom values. drand [31] is a distributed randomness beacon that produces verifiable random values at fixed intervals. A drand network consists of a set of nodes that broadcast (partial) signatures of shared messages at periodic intervals. Once a node has collected a threshold of *t-out-of-n* signatures, it combines them into a BLS signature that is verifiable with the groups' public key. The signature is then hashed using SHA-256 and published as a random value.

Threshold BLS provides $\texttt{SiGu} = \texttt{1:tM}$ security [11]. This implies that an adversary cannot forge a signature with access to less than t partial signatures, s.t. $|HS| = 0$ and $|CS| \leq t - 1$. Since BLS signatures do not specify the subset of signers that provided partial signatures (i.e., no leader requests), an adversary would have to either control $> t$ signers, or influence the key derivation process to gain control of the private key.

drand implements Pedersen's DKG scheme, which, as explained in Section 6.3, is vulnerable to a rogue-key attack. However, unlike in the previous application example, we assume a high trust relation between nodes. In the case of drand, each new node goes through a selection process to ensure a adequate level of trust and availability. In practice, this makes it extremely unlikely for a malicious node to gain a sufficiently large subset of compromised nodes to execute the attack.

Multi-User Wallets. A multi-user wallet (e.g., [51, 56]) uses threshold signatures to split control over multiple users. Rather than a single signature controlling the funds, we can require that at least t group members agree on any given operation. The functionality is similar to the validators discussed in the case of blockchain interoperability, without the expectation to validate transactions.

As in the case of the randomness beacon, we can assume a high level of trust between the signers. All signing keys could, for example, be controlled by a single party that distributes them over multiple devices.

Other Applications. Other applications include e.g., multi-factor authentication [43, 55] and DNSSEC [26]. Many more use cases are also envisioned for the future [50].