Better than Advertised Security for Non-Interactive Threshold Signatures

Mihir Bellare[1][0000000287655573], Elizabeth Crites[2], Chelsea Komlo[3], Mary Maller[4], Stefano Tessaro[5], and Chenzhi Zhu[5]

[1] Department of Computer Science & Engineering, University of California San Diego, USA
[2] University of Edinburgh, UK
[3] University of Waterloo, Zcash Foundation
[4] Ethereum Foundation, UK
[5] Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, USA

mihir@eng.ucsd.edu, ecrites@ed.ac.uk, ckomlo@uwaterloo.ca, mary.maller@ethereum.org {tessaro,zhucz20}@cs.washington.edu

Abstract. We give a unified syntax, and a hierarchy of definitions of security of increasing strength, for non-interactive threshold signature schemes. These are schemes having a single-round signing protocol, possibly with one prior round of message-independent pre-processing. We fit FROST1 and BLS, which are leading practical schemes, into our hierarchy, in particular showing they meet stronger security definitions than they have been shown to meet so far. We also fit in our hierarchy a more efficient version FROST2 of FROST1 that we give. These definitions and results, for simplicity, all assume trusted key generation. Finally, we prove the security of FROST2 with key generation performed by an efficient distributed key generation protocol.

1 Introduction

Threshold signatures, which originated in the late 1980s [17, 18], are seeing renewed attention, driven in particular by an interest in using them to secure digital wallets in the cryptocurrencies ecosystem [22]. Parallel IETF [32] and NIST [35] standardization efforts are evidence as to the speed at which the area is moving into practice.

Whether securing a user's digital wallet, or being used by a CA to create a certificate, forgery of a digital signature is costly. The rising tide of system breaches and phishing attacks makes exposure of a signing key too plausible to ignore. The idea of a threshold signature scheme is to distribute the secret signing key across multiple parties who then interact to produce a signature, the intent being to retain security even in the face of compromise of up to a threshold number of these parties. Over the years, threshold versions of many schemes have been presented, including RSA [16, 26, 37], DSA/ECDSA [23, 25, 22, 9, 21, 34, 13], Schnorr signatures [39, 24, 30] and BLS signatures [8].

Today, we see interest converging on schemes that are non-interactive. The representative examples are BLS [12, 8], and FROST [30]. FROST is a partially non-interactive threshold signature scheme, consisting of a message-independent pre-processing round and one round of signing. Threshold BLS is fully non-interactive, i.e., consists of a single round, but it does require pairings.

Our Contributions. We advance the area of non-interactive threshold signature schemes via the following contributions.

1. Framework and stronger security. We contend that schemes like FROST and BLS are *better than advertised*, meeting definitions of security that are *stronger* than ones that have been previously defined, or that these schemes have been shown to meet in existing literature. Furthermore, these definitions capture natural strengths of the schemes that may be valuable for applications.

The classical development paradigm in theoretical cryptography is to ask what security we would like, define it, and then seek schemes that meet it. Yet if we look back, there has been another path alongside: canonical, reference schemes guided a choice of definitions that modeled them, and, once made, these definitions went on to be influential targets for future schemes. (The formal definition of trapdoor permutations [27], for example, was crafted to model RSA.) We are inspired by the latter path. BLS [11] yields a threshold scheme [8] so natural and simple that it is hard to not see it as canonical, and, within the space of Schnorr threshold schemes, FROST [30] has a similarly appealing minimality. Examining them, we see strengths not captured by current definitions or results. We step back to create corresponding abstractions, including a unified syntax and a hierarchy of definitions of security for non-interactive threshold signature schemes. We then return to ask where in this hierarchy we can fit the starting schemes, giving proofs that fit BLS and FROST as high as possible. The proofs this requires, and that we provide, turn out to be challenging and technically interesting.

Although inspired by specific schemes, our definitional development, once begun, unfolds in a logical way, and yields definitions that go beyond even what BLS and FROST achieve. These make intriguing new targets. We show how to achieve them, with minimal modifications to the existing schemes.

2. FROST2 and its security with DKG. We introduce FROST2, a variant of the original FROST scheme (we hereafter refer to the original as FROST1) that reduces the number of exponentiations required for signing and verification from *linear* in the number of signers to *constant*. We analyze the security of FROST2 in our above security framework, and highlight subtle differences between it and FROST1.

The above-discussed results are all in a setting with (ideal) trusted key generation. In practice however it is desirable that key generation itself be done via a threshold, distributed key generation protocol (DKG). Accordingly, we prove the security of FROST2 with a DKG, namely an efficient variant of Pedersen's DKG (PedPoP) introduced in conjunction with FROST1 [30]. Unlike prior proofs that modeled key generation using Pedersen's DKG [24], our security proof allows concurrent executions of the signing protocol once key generation has completed, and generalizes which honest parties are assumed to participate. We demonstrate that $\mathsf{FROST2}$ instantiated with PedPoP is secure in the random oracle model (ROM) [5] assuming extractable proofs of possession and the one-more discrete logarithm (OMDL) assumption of [2]. The assumption of extractable proofs of possession is required *only* for the simulation of PedPoP. Indeed, our proofs for $\mathsf{FROST1}$ and $\mathsf{FROST2}$ without ideal key generation only rely on the OMDL assumption, along with random oracles.

Our proofs here fill a gap towards demonstrating security with respect to well-understood assumptions. We have a complete implementation of our security proof in python[6] in which we see that our reduction accurately outputs a valid OMDL solution and that our simulated outputs pass verification.

NON-INTERACTIVE THRESHOLD SCHEMES. We consider schemes where the signing operations involve a leader and a set of ns nodes, which we refer to as servers, with server i holding a secret share sk_i of the secret signing key sk. Signing is done via an interactive protocol that begins with a leader request to some set of at least t number of servers and culminates with the leader holding the signature, where $t \leq \mathsf{ns}$, the threshold, is a protocol parameter.

In a *fully non-interactive* threshold signature scheme, this protocol is a simple, one-round one. The leader sends a leader request lr, which specifies a message M and possibly other things, to any server i and obtains in response a *partial signature*, $psig_i$, that i computes as a function of sk_i and M. The leader can request partial signatures asynchronously, at any time, and independently for each server, and there is no server-to-server communication. Once it has enough partial signatures, the leader aggregates them into a signature sig of M under the verification key vk corresponding to sk. The canonical example is the threshold BLS scheme [8, 12], where $sk, sk_1, \ldots, sk_{\mathsf{ns}} \in \mathbb{Z}_p$ for a public prime p, and $psig_i \leftarrow \mathsf{h}(M)^{sk_i}$ where $\mathsf{h} : \{0,1\}^* \rightarrow \mathbb{G}$ is a public hash function with range a group $\mathbb{G}$ of order p. Aggregation produces sig as a weighted product of the partial signatures.

A *partially non-interactive* threshold signature scheme adds to the above a message-independent pre-processing round in which, pinged by the leader at any point, a server i returns a pre-processing token pp_i. The leader's request for partial signatures will now depend on tokens it has received. The canonical example is FROST [30]. This understanding of a non-interactive scheme encompasses what FROST calls flexibility: obtaining $psig_i$ from *any* $\geq t$ servers allows reconstruction of the signature.

WHICH FORGERIES ARE NON-TRIVIAL? For a regular (non-threshold) signature scheme, the first and most basic notion of security is un-forgeability (UF) [27]. The adversary (given access to a signing oracle) outputs a forgery consisting of a message M and a valid signature for it. To win, the forgery must be *non-trivial*, meaning not obtained legitimately. This is naturally captured, in this context, as meaning that M was not a signing query.

[6] https://github.com/mmaller/multi_and_threshold_signature_reductions

Turning to define un-forgeability for a non-interactive threshold signature scheme, we assume the adversary has corrupted the leader and up to $t-1$ servers, where $1 \leq t \leq \mathsf{ns}$ is the threshold. Furthermore, it has access to the honest servers. Again, it outputs a forgery consisting of a message M and valid signature for it, and, to win, the forgery must be *non-trivial*, meaning not obtained legitimately. Deciding what "non-trivial" means, however, is now a good deal more delicate, and interesting, than it was for regular signatures.

In this regard, we suggest that many prior works have set a low bar, being more generous than necessary in declaring a forgery trivial, leading to definitions that are weaker than one can desire, and weaker even than what their own schemes seem to meet. The definitions we formulate rectify this by considering five non-triviality conditions of increasing stringency, yielding a corresponding hierarchy TS-UF-0 $\leftarrow$ TS-UF-1 $\leftarrow$ TS-UF-2 $\leftarrow$ TS-UF-3 $\leftarrow$ TS-UF-4 of notions of un-forgeability of increasing strength. (Here an arrow B $\leftarrow$ A means A implies B: any scheme that is A-secure is also B-secure.) TS-UF-0, the lowest in the hierarchy, is the notion currently in the literature.

Returning to regular (non-threshold) signature schemes, *strong* un-forgeability (SUF) has the same template as UF, but makes the non-triviality condition more strict, asking that there has been no signing query M that returned sig. We ask if SUF has any analogue in the threshold setting. For non-interactive schemes, we suggest it does and give a hierarchy of three definitions of strong unforgeability TS-SUF-2 $\leftarrow$ TS-SUF-3 $\leftarrow$ TS-SUF-4. The numbering reflects that TS-UF-i $\leftarrow$ TS-SUF-i for $i = 2, 3, 4$.

THE CASE OF BLS. Boldyreva's analysis of threshold BLS [8] adopts the formalism of Gennaro, Jarecki, Krawczyk, and Rabin [26, 23, 25]. The non-triviality condition here is that *no* server was asked to issue a partial signature on the forgery message M. This is TS-UF-0 in our hierarchy. But allowing asynchronous requests is a feature of this scheme and model. A corrupted leader could ask one honest server i for a partial signature. No other server would even be aware of this request, but the adversary would now have $psig_i$. Under TS-UF-0, the forgery is now trivial, and the adversary does not win. Yet (assuming a threshold $t \geq 2$), there is no reason possession of just $psig_i$ should allow creation of a signature, and indeed for threshold BLS there is no attack that seems able to create such a signature, indicating the scheme is achieving more than TS-UF-0. This leads to the next level of our hierarchy, TS-UF-1, where the non-triviality condition is that a partial signature of M was requested from at most $t-1-c$ honest servers, where c is the number of corrupted servers. Does threshold BLS achieve this TS-UF-1 definition? As we will see, proving this presents challenges, but we will succeed in showing that the answer is yes, under a variant of the computational Diffie-Hellman (CDH) assumption. (The proof is deferred to [7] for lack of space.) Yet, TS-UF-1 was not considered in the literature, and only TS-UF-0 is proved for many other non-interactive schemes [37, 29, 40, 10]. The only exceptions are the work of Libert, Joye, and Yung [33] and recent concurrent work by Groth [28], which comes to a similar conclusion/result on BLS. (We discuss the relation below.) We note that Shoup [37] implicitly tackles a similar technical challenge by dealing with differing corruption and reconstruction thresholds, but the resulting security notion is not TS-UF-1.

The distinction between TS-UF-1 and TS-UF-0 is not just academic. Implicit in applications of threshold signing in wallets is the fact that servers also perform well-formedness checks of what is being signed (typically, as part of a transaction). TS-UF-1 guarantees that every issued signature has been inspected by sufficiently many servers, but TS-UF-0 does not.

The case of FROST. Yet the hierarchy needs to go higher, and this becomes apparent when looking at partially non-interactive schemes like FROST1 [30], and its optimized version, FROST2, which we introduce. Here, the discussion becomes more subtle, and interesting.

In more detail, a FROST1 pre-processing token takes the form of a pair $pp_i = (g^{r_i}, g^{s_i})$ of group elements for one-time use. (A server will ensure that the pre-processing token in its name in the leader request is one it has previously sent, and will never use it again.) An honest request lr includes, along with the message M to be signed, a sufficiently large server set $lr.\mathsf{SS} \subseteq [1..\mathsf{ns}]$, and, for each i in this set, a pre-processing token pp_i that i previously sent. Each server $i \in lr.\mathsf{SS}$ will then generate a signature share $psig_i = (R, z_i)$, where R is a value which can be computed (publicly) from the tokens included in lr, whereas z_i depends on the discrete logarithms of the server's token and its own key share sk_i. The z_i's can then be aggregated into a value z such that (R, z) is a valid Schnorr signature for M.

In terms of our framework, we show that FROST1 achieves TS-SUF-3 security. This considers a signature trivial even if some of the honest servers in $lr.\mathsf{SS}$ do not respond to a (malicious) leader request, as long as the tokens associated with these servers are not honestly generated. In particular, the honest servers may not respond because they recognize these tokens as invalid, or because the malicious leader did not submit the request to them. We show that, while FROST2 fails to achieve TS-SUF-3, it achieves the next step down in our hierarchy, TS-SUF-2. This is still stronger than the notions lower in the hierarchy. Our proofs for FROST1 and FROST2 signing operations rely on the OMDL assumption and the ROM.

Stronger goals. A stronger security goal (TS-UF-4 in our hierarchy) is to expect that the *only* way to obtain a signature for a message M is to follow the above blueprint, i.e., to issue the same honest leader request lr to all servers in $lr.\mathsf{SS}$. In fact, we may even ask for more, in terms of *strong* unforgeability — the value R is uniquely defined by lr, and, along with the message M, it defines a *unique* signature (although not efficiently computable given the verification key alone). An ideal goal, which corresponds to our strongest security goal, is to ensure that the *only* way to generate the signature associated with lr is to obtain a signature share for lr from every honest server whose tokens are included in lr. This is a notion we refer to as TS-SUF-4.

We will however show that neither FROST1 nor FROST2 meet TS-SUF-4. To overcome this, we will show a general transformation which can boost the security of a TS-SUF-3-secure scheme like FROST1 to achieve TS-SUF-4. Our framework allows schemes more general than the FROST ones, and also leaves the question open of better and more efficient designs achieving the stronger notions. Moreover, we provide simple reference schemes for all of our notions, which, while inefficient, guide us in understanding the subtle differences among notions and baseline requirements. In particular, these schemes will enable us to separate the proposed notions.

A SUMMARY FOR OUR NOTIONS. In summary, our unforgeabilty notions declare a signature for a message M trivial in the following cases:

- TS-UF-0: A partial signature for the message M was generated by at least one honest server.
- TS-UF-1: A partial signature for the message M was generated by at least $t - c$ honest servers, where c is the number of corrupted servers.
- TS-UF-2: There exists a leader request lr for the message M which was answered by at least $t - c$ honest servers.
- TS-UF-3: There exists a leader request lr for the message M such that every honest server $i \in lr.\mathsf{SS}$ 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 such that every honest server $i \in lr.\mathsf{SS}$ answered lr.

Analogous notions of strong unforgeability are obtained by further associating a request lr to a (unique) signature, in addition to a message M.

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. In both FROST variants, for example, it is natural to expect that a signature can only be generated by a sufficient number of honest servers answering the *same* request, a property which we show is actually achieved. Further, one may also expect that all honest servers that generated these honest tokens need to be involved in the generation of a valid signature, but this stronger property is actually not achieved by either of the FROST variants.

FROST2 WITH DKG. Our syntax above assumes key generation is carried out by a trusted algorithm, which allows us to focus on the signing protocol. However, security in practice is enhanced when the key generation itself is a distributed threshold protocol, so that the key is never in the clear in any one location, even ephemerally. In this setting, we prove the security of FROST2 with the distributed key generation protocol (DKG) originally proposed in [30], which we refer to as PedPoP. Our proof for the combination of FROST2 and PedPoP relies on the ROM, the OMDL assumption, and a new knowledge-type assumption. However, we stress that the latter assumption is only necessary to handle PedPoP, as indeed we give *stronger* proofs of security without this assumption in a setting with ideal key generation.

WHAT WE DO NOT DO. Our framework does not handle adaptive corruptions, i.e., we demand instead that the adversary declares its corruption set initially. We could extend our definitions to adaptive corruptions rather easily, but our concrete bounds would be impacted. In particular, we would resort to a generic reduction guessing the corrupted set beforehand, with a multiplicative loss of 2^{ns}, which is acceptable for the smaller values of the number ns of parties that we consider common in practice.

Our framework cannot cover recent protocols, like that of Canetti et al. [13], which combine a *multi-round* message-independent pre-processing phase with a final, message-dependent, round. (Conversely, their UC security analysis does not give definitions which help our fine-grained framework.)

Many prior works also consider *robustness*, i.e., the guarantee that a signature is always produced. Here, we follow the same viewpoint as in FROST, and do not focus on robustness explicitly. This allows us to prevent imposing a small t (relative to ns) just for the sake of ensuring it. However, our schemes all implicitly give verification keys vk_i for each server, and it is not hard to verify individual partial signatures $psig_i$. Any t valid partial signatures will always aggregate into a valid signature.

Related and concurrent work. A recent preprint by Groth [28] presents a general definition for fully non-interactive schemes in a setting with a (non-interactive) DKG. His definition implies TS-UF-1, and he also provides a proof sketch that BLS (with his newly proposed non-interactive DKG) is secure under a variant of the OMCDH assumption, which is closely related to our variant of the CDH assumption and which we also show to be hard in the GGM. Groth's framework is not suitable for partially non-interactive schemes like FROST, which are the main focus of our work.

History of this paper. This paper is the result of a (hard) merge imposed by the Crypto 2022 PC on two submissions. CKM [14] introduces $\mathsf{FROST2}$. BTZ [7] introduces the framework and definitions for non-interactive schemes with trusted key generation and proofs for BLS, $\mathsf{FROST1}$ and $\mathsf{FROST2}$ in this framework. CKM [14] provides a proof of security for $\mathsf{FROST2}$ that includes distributed key generation. Most security proofs have been deferred to the respective full versions. We see each group of authors as responsible for the contribution relevant to their part of the work.

2 Preliminaries

Notation. If $b \geq a \geq 1$ are positive integers, then $\mathbb{Z}_a$ denotes the set $\{0, \ldots, a-1\}$ and $[a..b]$ denotes the set $\{a, \ldots, b\}$. If $\boldsymbol{x}$ is a vector then $|\boldsymbol{x}|$ is its length (the number of its coordinates), $\boldsymbol{x}[i]$ is its i-th coordinate and $[\boldsymbol{x}] = \{\, \boldsymbol{x}[i] : 1 \leq i \leq |\boldsymbol{x}| \,\}$ is the set of all its coordinates. A string is identified with a vector over $\{0,1\}$, so that if x is a string then $x[i]$ is its i-th bit and $|x|$ is its length. By ε we denote the empty vector or string. The size of a set S is denoted $|S|$. For sets D, R let $\mathrm{FNS}(D,R)$ denote the set of all functions $f : D \to R$.

Let S be a finite set. We let $x \leftarrow_{\$} S$ denote sampling an element uniformly at random from S and assigning it to x. We let $y \leftarrow A^{\mathrm{O}_1,\cdots}(x_1,\ldots;r)$ denote executing algorithm A on inputs $x_1,\ldots$ and coins r with access to oracles $\mathrm{O}_1,\ldots$ and letting y be the result. We let $y \leftarrow_{\$} A^{\mathrm{O}_1,\cdots}(x_1,\ldots)$ be the result of picking r

at random and letting $y \leftarrow A^{\mathrm{O}_1,\cdots}(x_1, \ldots; r)$. Algorithms are randomized unless otherwise indicated. Running time is worst case.

GAMES. We use the code-based game playing framework of [6]. (See Figure 2 for an example.) Games have procedures, also called oracles. Among the oracles are INIT (Initialize) and FIN (Finalize). In executing an adversary $\mathcal{A}$ with a game Gm, the adversary may query the oracles at will, with the restriction that its first query must be to INIT (if present), its last to FIN, and it can query these oracles at most once. The value returned by the FIN procedure is taken as the game output. By $\mathrm{Gm}(\mathcal{A}) \Rightarrow y$ we denote the event that the execution of game Gm with adversary $\mathcal{A}$ results in output y. We write $\Pr[\mathrm{Gm}(\mathcal{A})]$ as shorthand for $\Pr[\mathrm{Gm}(\mathcal{A}) \Rightarrow \mathsf{true}]$, the probability that the game returns true.

In writing game or adversary pseudocode, it is assumed that Boolean variables are initialized to false, integer variables are initialized to 0 and set-valued variables are initialized to the empty set $\emptyset$.

GROUPS. Let $\mathbb{G}$ be a group of order p. We will use multiplicative notation for the group operation, and we let $1_{\mathbb{G}}$ denote the identity element of $\mathbb{G}$. We let $\mathbb{G}^* = \mathbb{G} \setminus \{1_{\mathbb{G}}\}$ denote the set of non-identity elements, which is the set of generators of $\mathbb{G}$ if the latter has prime order. If $g \in \mathbb{G}^*$ is a generator and $X \in \mathbb{G}$, the discrete logarithm base g of X is denoted $\mathsf{DL}_{\mathbb{G},g}(X)$, and it is in the set $\mathbb{Z}_{|\mathbb{G}|}$.

3 A Framework for Non-Interactive Threshold Signatures

We present our hierarchy of definitions of security for non-interactive threshold schemes, formalizing both unforgeability (UF) and strong unforgeability (SUF) in several ways. We provide relations between all notions considered.

3.1 Syntax and Correctness

MAINTAINING STATE. Parties as implemented in protocols would maintain state. When activated with some inputs (which include messages from other parties), they would apply some algorithm Alg to these and their current state to get outputs (including outgoing messages) and an updated state. To model this, we do not change our definition of algorithms, but make the state an explicit input and output that will, in definitions, be maintained by the overlying game. Thus, we would write something like $(\cdots, \mathsf{st}) \leftarrow\!\!\$\ \mathsf{Alg}(\cdots, \mathsf{st})$.

SYNTAX. A non-interactive threshold signature scheme TS specifies a number $\mathsf{ns} \geq 1$ of servers, a reconstruction threshold t, a set HF of functions from which the random oracle is drawn, a key generation algorithm Kg, a server pre-processing algorithm SPP, a leader pre-processing algorithm LPP, a leader signing-request algorithm LR, a server partial-signature algorithm PS, a leader partial-signature aggregation algorithm Agg and a verification algorithm Vf. If disambiguation is needed, we write $\mathsf{TS.ns}, \mathsf{TS}.t, \mathsf{TS.HF}, \mathsf{TS.Kg}, \mathsf{TS.SPP}, \mathsf{TS.LPP}, \mathsf{TS.LR}, \mathsf{TS.PS}, \mathsf{TS.Agg}, \mathsf{TS.Vf}$, respectively. We now explain the operation and

```
Game $\mathbf{G}^{\text{ts-cor}}_{\mathsf{TS}}$

INIT:
 1 $\mathsf{h} \leftarrow_{\$} \mathsf{TS.HF}$ ; $sk_0 \leftarrow \bot$ ; $(vk, aux, sk_1, \ldots, sk_{\mathsf{ns}}) \leftarrow_{\$} \mathsf{Kg}[\mathsf{h}]$
 2 For $i = [0..\mathsf{ns}]$ do  // Initialize party states with keys
 3    $\mathsf{st}_i.\mathsf{sk} \leftarrow sk_i$ ; $\mathsf{st}_i.\mathsf{vk} \leftarrow vk$ ; $\mathsf{st}_i.\mathsf{aux} \leftarrow aux$
 4 Return $vk, aux, sk_1, \ldots, sk_{\mathsf{ns}}$

PPO($i$):  // $i \in [1..\mathsf{ns}]$
 5 $(pp, \mathsf{st}_i) \leftarrow_{\$} \mathsf{SPP}[\mathsf{h}](\mathsf{st}_i)$ ; $\mathsf{st}_0 \leftarrow \mathsf{LPP}[\mathsf{h}](pp, \mathsf{st}_0)$
 6 Require: $pp \neq \bot$
 7 Return $pp$

PPO($M, SS$):
 8 Require: $SS \subseteq [1..\mathsf{ns}]$ and $|SS| \geq t$  // Set of signers
 9 $(lr, \mathsf{st}_0) \leftarrow_{\$} \mathsf{LR}[\mathsf{h}](M, SS, \mathsf{st}_0)$
10 Require: $lr \neq \bot$  // Leader accepts request
11 If ($lr.\mathsf{msg} \neq M$ or $lr.\mathsf{SS} \neq SS$) then $\mathsf{win} \leftarrow \mathsf{true}$
12 For $i \in SS$ do
13    $(psig_i, \mathsf{st}_i) \leftarrow_{\$} \mathsf{PS}[\mathsf{h}](lr, i, \mathsf{st}_i)$  // Server partial signatures
14 $(sig, \mathsf{st}_0) \leftarrow_{\$} \mathsf{Agg}[\mathsf{h}](lr, \{psig_i\}_{i \in SS}, \mathsf{st}_0)$
15 If $\mathsf{Vf}[\mathsf{h}](vk, M, sig) = \mathsf{false}$ then $\mathsf{win} \leftarrow \mathsf{true}$

RO($x$):  // Random oracle
16 Return $\mathsf{h}(x)$

FIN:
17 Return $\mathsf{win}$
```

Fig. 1. Game used to define correctness of threshold signature scheme TS with threshold t.

use of these components, the understanding of which may be aided by already looking at the correctness game $\mathbf{G}^{\text{ts-cor}}_{\mathsf{TS}}$ of Figure 1.

Parties involved are a leader (numbered 0, implicit in some prior works, but made explicit here) and servers numbered $1, \ldots, \mathsf{ns}$, for a total of $\mathsf{ns} + 1$ parties. Algorithms have oracle access to a function h that is drawn at random from HF in games (line 1 Figure 1) and plays the role of the random oracle. Specifying HF as part of the scheme allows the domain and range of the random oracle to be scheme dependent.

The key generation algorithm Kg, run once at the beginning (line 1 of Figure 1), creates a public signature-verification key vk, associated public auxiliary information aux and an individual secret signing key sk_i for each server $i \in [1..\mathsf{ns}]$. (Usually, $sk_1, \ldots, sk_{\mathsf{ns}}$ will be shares of a global secret key sk, but the definitions do not need to make sk explicit. The leader does not hold any secrets associated to vk.) While key generation may in practice be performed by a distributed key generation protocol, our syntax assumes it done by a trusted algorithm to allow a modular treatment. Keys are held by parties in their state, encoded into dedicated fields of the latter as shown at line 3 of Figure 1. For specific scheme, we will typically use aux to model additional information that can be leaked by key generation step without violating security (e.g., the values g^{sk_i} in most cases).

The signing protocol can be seen as having two rounds, which we think as a pre-processing and online stage. In a pre-processing round, any server i can run $(pp, \mathsf{st}_i) \leftarrow\!\!\$\ \mathsf{SPP}[\mathsf{h}](\mathsf{st}_i)$ to get a *pre-processing token pp* which it sends to the leader. (Here st_i is the state of i.) Via $\mathsf{st}_0 \leftarrow \mathsf{LPP}[\mathsf{h}](pp, \mathsf{st}_0)$, the leader updates its state st_0 to incorporate token pp. (In Figure 1, this is reflected in lines 5–7.)

In a signing round the leader begins with a message and a choice of a signer set $SS \subseteq [1..\mathsf{ns}]$ of size at least t. Via $(lr, \mathsf{st}_0) \leftarrow\!\!\$\ \mathsf{LR}[\mathsf{h}](M, SS, \mathsf{st}_0)$ it generates a leader request lr that, through st_0, implicitly depends on a choice of pre-processing tokens. (Lines 8,9 of Figure 1.) The leader request is sent to each $i \in SS$, who, via $(psig_i, \mathsf{st}_i) \leftarrow\!\!\$\ \mathsf{PS}[\mathsf{h}](lr, \mathsf{st}_i)$, computes a partial signature $psig_i$ and returns it to the leader. Via $(sig, \mathsf{st}_0) \leftarrow\!\!\$\ \mathsf{Agg}[\mathsf{h}](lr, \{psig_i\}_{i \in SS}, \mathsf{st}_0)$, the leader aggregates the partial signatures into a signature sig of M, the desired output of the protocol. (Lines 12–14 of Figure 1.)

The verification algorithm, like in a standard signature scheme, takes vk, a message M and a candidate signature, and returns a boolean validity decision.

<u>Echo schemes.</u> We define a sub-class of non-interactive threshold schemes that we call *echo schemes*. Recall that a leader request lr 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. In an echo scheme, lr additionally specifies a function $lr.\mathsf{PP} : lr.\mathsf{SS} \to \{0,1\}^*$. If the leader is honest, $lr.\mathsf{PP}(i)$ is a token pp that i had previously sent to the leader. That is, the leader is echoing tokens back to the servers, whence the name. In considering security, of course, $lr.\mathsf{PP}(i)$ is picked by the adversary and may not be a prior token. As we will discuss in Section 4.1, FROST is a typical example of an echo scheme.

<u>Correctness of a TS scheme.</u> The game of Figure 1 defines correctness, and serves also to detail the above. Recall that TS specifies a threshold $t \in [1..\mathsf{ns}]$. The adversary will make the leader's pre-processing requests, via oracle PPO. It will likewise make signing requests via oracle PPO. If any condition listed under $\mathsf{Require}$: fails the adversary is understood as losing, the game automatically returning false. We let $\mathbf{Adv}^{\text{ts-corr}}_{\mathsf{TS}}(\mathcal{A}) = \Pr[\mathbf{G}^{\text{ts-cor}}_{\mathsf{TS}}(\mathcal{A})]$ be the advantage of an adversary $\mathcal{A}$. The default requirement is perfect correctness, which means that $\mathbf{Adv}^{\text{ts-corr}}_{\mathsf{TS}}(\mathcal{A}) = 0$ for all $\mathcal{A}$, regardless of computing time and number of oracle queries, but this can be relaxed, as may be necessary for lattice-based protocols.

The way in which we are supposed to interpret the correctness definition is that a request lr is associated with a set SS and a message M, and if such a request is issued successfully by the leader (i.e., $lr \neq \bot$), then the servers in SS would all accept lr producing partial signatures which aggregate into a

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

INIT($CS$):
1  Require: $CS \subseteq [1..\mathsf{ns}]$ and $|CS| < t$ // Set of corrupted parties
2  $\mathsf{h} \leftarrow\$ \mathsf{TS.HF}$ ; $(vk, aux, sk_1, \ldots, sk_{\mathsf{ns}}) \leftarrow\$ \mathsf{Kg}[\mathsf{h}]$
3  $HS \leftarrow [1..\mathsf{ns}] \setminus CS$ // Set of honest parties
4  For $i \in HS$ do
5      $\mathsf{st}_i.\mathsf{sk} \leftarrow sk_i$ ; $\mathsf{st}_i.\mathsf{vk} \leftarrow vk$ ; $\mathsf{st}_i.\mathsf{aux} \leftarrow aux$
6  Return $vk, aux, \{sk_i\}_{i \in CS}$

PPO($i$):
7  Require: $i \in HS$
8  $(pp, \mathsf{st}_i) \leftarrow\$ \mathsf{SPP}[\mathsf{h}](\mathsf{st}_i)$ ; $\mathrm{PP}_i \leftarrow \mathrm{PP}_i \cup \{pp\}$ ; Return $pp$

PSIGNO($i, lr$):
9  $M \leftarrow lr.\mathsf{msg}$
10 Require: $lr.\mathsf{SS} \subseteq [1..\mathsf{ns}]$ and $M \in \{0,1\}^*$ and $i \in HS$
11 $\mathrm{L} \leftarrow \mathrm{L} \cup \{lr\}$ ; $(psig, \mathsf{st}_i) \leftarrow\$ \mathsf{PS}[\mathsf{h}](lr, i, \mathsf{st}_i)$
12 If $(psig \neq \bot)$ then
13     $\mathrm{S}_1(M) \leftarrow \mathrm{S}_1(M) \cup \{i\}$ ; $\mathrm{S}_2(lr) \leftarrow \mathrm{S}_2(lr) \cup \{i\}$
14 Return $psig$

RO($x$): // Random oracle
15 Return $\mathsf{h}(x)$

FIN($M, sig$):
16 For all $lr \in \mathrm{L}$ do
17     $\mathrm{S}_3(lr) \leftarrow \{ i \in HS \cap lr.\mathsf{SS} : lr.\mathrm{PP}(i) \in \mathrm{PP}_i \}$ ; $\mathrm{S}_4(lr) \leftarrow HS \cap lr.\mathsf{SS}$
18 If (not $\mathsf{Vf}[\mathsf{h}](vk, M, sig)$) then return false
19 Return (not $\mathtt{tf}_i(M)$) // Game $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ for $i = 0, 1$
20 Return (not $\exists\, lr\, (\, lr.\mathsf{msg} = M$ and $\mathtt{tf}_i(lr)\,)$) // Game $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ for $i = 2, 3, 4$
21 Return (not $\exists\, lr\, (\, lr.\mathsf{msg} = M$ and $\mathtt{tsf}_i(lr, vk, sig)\,)$) // Game $\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}$
```

Fig. 2. Games used to define TS-UF-i and TS-SUF-i unforgeability of threshold signature scheme TS. Line 20 is included only in game $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ and line 21 only in game $\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}$. These lines refer to the trivial-forgery predicates $\mathtt{tf}_i(lr)$ and trivial-strong-forgery predicates $\mathtt{tsf}_i(lr, vk, sig)$ from Figure 3. In particular, the set $\mathrm{S}_3(lr)$ and, thus, TS-UF-3 and TS-SUF-3 unforgeability are defined only if TS is an echo scheme.

valid signature for M. We note that this definition assumes that we submit requests to all servers in the same order. One can give a stronger (but more complex) definition which ensures correctness even when servers process requests in different orders, but note that for all schemes we discuss below they will be equivalent, and we hence omit the more cumbersome game to define it.

3.2 Unforgeability and Strong Unfogeability

UNFORGEABILITY. Unforgeability as usual asks that the adversary be unable to produce a valid signature *sig* on some message M of its choice except in a trivial way. The question is what "trivial" means. For regular signatures, it means that the adversary did not obtain a signature of M from the signing oracle [27]. For threshold signatures, it is more subtle. We will give several definitions.

Figure 2 simultaneously describes several games, $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ for $i = 0, 1, 2, 3, 4$, where $\mathbf{G}^{\text{ts-uf-3}}_{\mathsf{TS}}$ is only defined if TS is an echo scheme. (We will get to the second set of games later.) They are almost the same, differing only at line 20. The corresponding advantages of an adversary $\mathcal{A}$ are $\mathbf{Adv}^{\text{ts-uf-i}}_{\mathsf{TS}}(\mathcal{A}) = \Pr[\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}(\mathcal{A})]$. The adversary calls INIT with a choice of a set of servers to corrupt. It is also viewed as having corrupted the leader. Playing the leader role, it can request pre-processing tokens via oracle PPO. It can provide a server with a leader-request *lr* of its choice to obtain a partial signature *psig*. At the end, it outputs to FIN its forgery message M and signature *sig*. If the signature is not valid, line 18 ensures that the adversary does not win. Now, to win, the signature must be non-trivial. It is in how this is defined that the games differ. Associated to i is a *trivial-forgery* predicate $\mathtt{tf}_i$ that is invoked at line 20. The choices for these predicates are shown in the table in Figure 3, and the notion corresponding to game $\mathtt{tf}_i$ is denoted TS-UF-i. When $i = 0$ we have the usual notion from the literature, used in particular in [8, 23, 25]. As i increases, we get more stringent (less generous) in declaring a forgery trivial, and the notion gets stronger.

Concretely, TS-UF-0 considers a signature for a message M trivial if a request *lr* with $lr.\mathsf{msg}$ was answered by server with a partial signature. Moving on, TS-UF-1 strengthens this by declaring a signature trivial only if at least $t - |CS|$ servers have responded to some request for message M, where these requests could have been different. In turn, TS-UF-2 strengthens this even further by requiring that there was a single prior request *lr* for M which was answered by $t - |CS|$ servers.

The notion TS-UF-3 only deals with echo schemes. Recall that for these schemes, a request *lr* contains a map $lr.\mathsf{PP} : lr.\mathsf{SS} \to \{0,1\}^*$, where $lr.\mathsf{PP}(i)$ is meant to be a token issued by server i. Here, we consider a signature for message M trivial if there exists a request *lr* for M which is answered by all honest servers i for which $lr.\mathsf{PP}(i)$ is a valid token previously output by i, and this set consists of at least $t - |CS|$ servers. Finally, our strongest notion, TS-UF-4 simply considers a signature trivial if there exists a request *lr* for M which is answered by all honest servers in $i \in lr.\mathsf{SS}$.

It is natural to expect TS-UF-3 and TS-UF-4 to be similar, but as we will see below, they are actually not equivalent. (Although we will give a transformation that boosts an TS-UF-3-secure scheme into an TS-UF-4-secure one.)

STRONG UNFORGEABILITY. For standard signatures, strong unforgeability asks, in addition to unforgeability, that the adversary be unable to produce a new signature on any message, where new means different from any obtained legitimately for that message. We ask, does this have any counterpart in threshold

$\mathtt{tf}_0(M)$	:	$\mathrm{S}_1(M) \neq \emptyset$
$\mathtt{tf}_1(M)$	:	$\lvert\mathrm{S}_1(M)\rvert \geq t - \lvert CS\rvert$
$\mathtt{tf}_2(lr)$	:	$\lvert\mathrm{S}_2(lr)\rvert \geq t - \lvert CS\rvert$
$\mathtt{tf}_3(lr)$	:	$\mathtt{tf}_2(lr)$ and $\mathrm{S}_2(lr) = \mathrm{S}_3(lr)$
$\mathtt{tf}_4(lr)$	:	$\mathtt{tf}_2(lr)$ and $\mathrm{S}_2(lr) = \mathrm{S}_4(lr)$
$\mathtt{tsf}_2(lr, vk, sig)$	:	$\mathtt{tf}_2(lr)$ and $\mathsf{SVf}[\mathsf{h}](vk, lr, sig)$
$\mathtt{tsf}_3(lr, vk, sig)$	:	$\mathtt{tf}_3(lr)$ and $\mathsf{SVf}[\mathsf{h}](vk, lr, sig)$
$\mathtt{tsf}_4(lr, vk, sig)$	:	$\mathtt{tf}_4(lr)$ and $\mathsf{SVf}[\mathsf{h}](vk, lr, sig)$

$$\begin{tikzcd}
\text{TS-UF-0} \arrow[r, bend right] & \text{TS-UF-1} \arrow[l] \arrow[r, dotted, bend right] & \text{TS-UF-2} \arrow[l] & \text{TS-UF-3} \arrow[l] \arrow[r, dashed, bend left] & \text{TS-UF-4} \arrow[l] \arrow[dll, dotted] \\
 & & \text{TS-SUF-2} \arrow[u] \arrow[ur, dotted] & \text{TS-SUF-3} \arrow[l] \arrow[u] \arrow[ur, dotted] \arrow[r, dashed, bend right] & \text{TS-SUF-4} \arrow[l] \arrow[u]
\end{tikzcd}$$

Fig. 3. Top: Trivial-forgery conditions $\mathtt{tf}_i(lr)$ $(i = 0, 1, 2, 3, 4)$ and trivial-strong-forgery conditions $\mathtt{tsf}_i(lr, vk, sig)$ $(i = 1, 2, 3, 4)$ used to define TS-SUF-i and TS-SUF-i security in games $\mathbf{G}^{\text{ts-uf-}i}_{\mathsf{TS}}$ and $\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}$, respectively. **Bottom:** Relations between notions of security.

signatures? In fact, FROST seems to have such a property. We now provide formalisms to capture such properties.

It turns out that giving a general definition of strong unforgeability is rather complex, and we will restrict ourselves to a natural sub-class of schemes (which includes FROST). Concretely, we ask that there is an algorithm SVf, called a *strong verification algorithm*, that takes a public key vk, a leader request lr, and a signature sig as inputs and outputs true or false. We require that for any vk, lr there exists at most one signature sig such that $\mathsf{SVf}(vk, lr, sig) = \mathsf{true}$. Also, TS is asked to satisfy a strong correctness property which is defined using the same game as $\mathbf{G}^{\text{ts-cor}}_{\mathsf{TS}}$ except the condition $\mathsf{Vf}[\mathsf{h}](vk, M, sig) = \mathsf{false}$ in line 15 is replaced with $\mathsf{SVf}[\mathsf{h}](vk, lr, sig) = \mathsf{false}$.

For a scheme TS with a strong verification algorithm, we consider the $\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}$ $(i = 2, 3, 4)$ games in Figure 2, where $\mathbf{G}^{\text{ts-suf-}3}_{\mathsf{TS}}$ is only defined if TS additionaly is an echo scheme. The differences (across the different values of i) are only in the trivial-strong forgery predicates $\mathtt{tsf}_i$ used at line 21, and the choices are again shown in the table in Figure 3. The corresponding advantage of an adversary $\mathcal{A}$ is $\mathbf{Adv}^{\text{ts-suf-i}}_{\mathsf{TS}}(\mathcal{A}) = \Pr[\mathbf{G}^{\text{ts-suf-}i}_{\mathsf{TS}}(\mathcal{A})]$. The ensuing notion is called TS-SUF-i.

3.3 Relations and Transformations

<u>Relations between notions.</u> Figure 3 shows relations between the notions of unforgeability and strong unforgeabilty that we have defined. A (non-dotted) arrow A $\rightarrow$ B is an implication, saying that A implies B: any scheme that is A-secure is also B-secure. Now see the nodes as forming a graph with edges the non-dotted arrows. The thin arrow from TS-UF-0 to TS-UF-1 indicates us that the implication only holds under a quantatively loose reduction. (We prove this in Theorem 1.) We claim that in this graph, if there is no path from a notion B to a notion A, they are separate or distinct: there exists a scheme that is B-secure but not A-secure. The dotted arrows are separations that we explicitly prove. These, together with the full arrows, prove the claim just made. The thick dotted arrows indicate the existence of a generic transformation lifting security of a scheme to achieve a stronger notion. (We establish this below as part of Theorem 2.)

<u>Reference schemes and proofs of relations.</u> In [7], we give a set of (fully) non-interactive threshold schemes that we call reference schemes. They represent simple, canonical ways to achieve the different notions. They may not be of practical interest, because they have key and signature sizes proportional to ns, but the point is to embody notions in a representative way. A few things emanate from these schemes. One is that we use them to establish the separations given by the dotted lines in Figure 3, thereby showing that any notions between which there is no path, in the graph given by the full arrows, are indeed separate. Second, we get a scheme that achieves our strongest notion, TS-SUF-4, which neither FROST nor BLS achieve. (Although we can get such a scheme by applying our transformation from Theorem 2 to $\mathsf{FROST1}$.) Finally, reference schemes, as canonical examples, are ways to understand the notions.

<u>From TS-UF-0 to TS-UF-1, loosely</u> The following theorem shows TS-UF-1 security is implied by TS-UF-0 security, although with an exponential loss in t, which is acceptable in settings where t is expected to be constant.

Theorem 1. *Let* TS *be a threshold signature scheme. For any TS-UF-1 adversary* $\mathcal{A}$ *there exists a TS-UF-0 adversary* $\mathcal{B}$ *such that* $\mathbf{Adv}^{\text{ts-uf-1}}_{\mathsf{TS}}(\mathcal{A}) \leq \binom{\mathsf{ns}}{t-1} \cdot \mathbf{Adv}^{\text{ts-uf-0}}_{\mathsf{TS}}(\mathcal{B})$ *. Moreover,* $\mathcal{B}$ *runs in time roughly equal that of* $\mathcal{A}$*, and the number of* $\mathcal{B}$*'s queries to each oracle is at most that of* $\mathcal{A}$*.*

If the adversary always corrupts $t-1$ parties, it is clear that TS-UF-0 and TS-UF-1 are equivalent. Otherwise, in general, for an adversary that breaks TS-UF-1 security and corrupts a subset CS of servers with size less than $t-1$, if the adversary wins the game $\mathbf{G}^{\text{ts-uf-1}}_{\mathsf{TS}}$ by outputting (M^*, sig^*), we know $|\mathrm{S}_1(M^*)| < t - |CS|$. Therefore, we can modify the adversary to initially guess a subset $ECS \subseteq [1..\mathsf{ns}] \setminus CS$ with size $t - |CS| - 1$ and corrupt all parties in ECS. If ECS happens to contain $\mathrm{S}_1(M^*)$, the adversary actually wins. It is not hard to see that the probability that this is true is $1/\binom{\mathsf{ns}-|CS|}{t-|CS|-1} \geq 1/\binom{\mathsf{ns}}{t-1}$. We give a formal proof in [7].

```
Protocol $\mathsf{ATS}[\mathsf{TS},\mathsf{DS}]$

$\mathsf{Kg}[\mathsf{h}]$:
1  $vk, taux, \{tsk_i\}_{i\in[1..\mathsf{ns}]} \leftarrow \mathsf{TS.Kg}$
2  For $i \in [1..\mathsf{ns}]$ do
3      $(svk_i, ssk_i) \leftarrow\!\!\$\ \mathsf{DS.Kg}$
4      $sk_i \leftarrow (tsk_i, ssk_i)$
5  $aux \leftarrow (taux, svk_1, \ldots, svk_{\mathsf{ns}})$
6  Return $vk, aux, \{sk_i\}_{i\in[1..\mathsf{ns}]}$

$\mathsf{SPP}[\mathsf{h}](\mathsf{st}_i)$:
7  $(tpp, \mathsf{st}_i) \leftarrow\!\!\$\ \mathsf{SPP}[\mathsf{h}](\mathsf{st}_i)$
8  $(tsk_i, ssk_i) \leftarrow \mathsf{st}_i.\mathsf{sk}$
9  $tsig \leftarrow\!\!\$\ \mathsf{DS.Sig}(ssk_i, tpp)$
10 Return $((tpp, tsig), \mathsf{st}_i)$

$\mathsf{LPP}[\mathsf{h}](i, pp, \mathsf{st}_0)$:
11 $(tpp, tsig) \leftarrow pp$
12 $\mathsf{st}_0.\mathsf{SigMap}(i, tpp) \leftarrow tsig$
13 Return $\mathsf{TS.LPP}[\mathsf{h}](i, tpp, \mathsf{st}_0)$

$\mathsf{OriginLR}(lr)$:
14 For $i \in lr.\mathsf{SS}$ do
15     $(tpp, tsig) \leftarrow lr.\mathsf{PP}(i)$
16     $lr.\mathsf{PP}(i) \leftarrow tpp$
17 Return $lr$

$\mathsf{LR}[\mathsf{h}](M, SS, \mathsf{st}_0)$:
18 $(lr, \mathsf{st}_0) \leftarrow \mathsf{TS.LR}[\mathsf{h}](M, SS, \mathsf{st}_0)$
19 For $i \in SS$ do
20     $tpp_i \leftarrow lr.\mathsf{PP}(i)$
21     $lr.\mathsf{PP}(i) \leftarrow (tpp_i, \mathsf{st}_0.\mathsf{SigMap}(i, tpp_i))$
22 Return $(lr, \mathsf{st}_0)$

$\mathsf{PS}[\mathsf{h}](lr, i, \mathsf{st}_i)$:
23 $(taux, svk_1, \ldots, svk_{\mathsf{ns}}) \leftarrow \mathsf{st}_i.\mathsf{aux}$
24 For $i \in lr.\mathsf{SS}$ do
25     $(tpp_i, tsig_i) \leftarrow lr.\mathsf{PP}(i)$
26     If $\mathsf{DS.Vf}(svk_i, tpp_i, tsig_i) = \mathsf{false}$ then
27         Return $\perp$
28 Return $\mathsf{TS.PS}[\mathsf{h}](\mathsf{OriginLR}(lr), i, \mathsf{st}_i)$

$\mathsf{Agg}[\mathsf{h}](\mathrm{PS}, \mathsf{st}_0)$:
29 Return $\mathsf{TS.Agg}[\mathsf{h}](\mathrm{PS}, \mathsf{st}_0)$

$\mathsf{Vf}[\mathsf{h}](vk, M, sig)$:
30 Return $\mathsf{TS.Vf}[\mathsf{h}](vk, M, sig)$

$\mathsf{SVf}[\mathsf{h}](vk, lr, sig)$:
31 Return $\mathsf{TS.SVf}[\mathsf{h}](vk, \mathsf{OriginLR}(lr), sig)$
```

Fig. 4. The threshold signature $\mathsf{ATS}[\mathsf{TS},\mathsf{DS}]$ constructed from an echo scheme TS and a digital signature scheme DS such that $\mathsf{ATS.ns} = \mathsf{TS.ns}$ and $\mathsf{ATS}.t = \mathsf{TS}.t$. The algorithm $\mathsf{OriginLR}$ transforms a well-formed leader request lr for ATS to a well-formed leader request in TS. $\mathsf{st}_0.\mathsf{SigMap}$ is a table that stores the signature corresponding to each token generated by honest servers, which is initially set to empty. PS denotes a set of partial signatures.

From TS-(S)UF-3 to TS-(S)UF-4. Figure 4 gives a general transformation from TS-(S)UF-3 security to TS-(S)UF-4 security. Concretely, we give a construction ATS from any TS-(S)UF-3-secure echo scheme TS and a digital signature scheme DS. The size of signatures produced by ATS and the verification algorithm Vf are exactly the same as TS. The main idea is to use signatures to authenticate each token contained in a leader request lr from TS, so that an honest server only answers the request if all the authentications are valid. The rest of the protocol remains the same.

In the game $\mathbf{G}^{\text{ts-(s)uf-4}}_{\mathsf{ATS}}$, we can show that as long as the adversary does not break the strong unforgeability of DS, for any leader request lr such that

```
Game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{DS}}$

INIT:
 1 $(vk, sk) \leftarrow\$ \mathsf{DS.Kg}$
 2 Return $vk$

PPO$(M)$:
 3 $sig \leftarrow\$ \mathsf{DS.Sig}(sk, M)$
 4 $Q \leftarrow Q \cup \{(M, sig)\}$
 5 Return $sig$

FIN$(M, sig)$:
 6 If $\mathsf{DS.Vf}(vk, M, sig)$ and $(M, sig) \notin Q$ then
 7     Return $\mathsf{true}$
 8 Return $\mathsf{false}$
```

Fig. 5. The game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{DS}}$, where DS is a digital signature scheme.

$\mathrm{S}_2(lr) > 0$, it holds that $\mathrm{S}_3(lr) = \mathrm{S}_4(lr)$, which implies the conditions $\mathtt{tf}_3$ and $\mathtt{tf}_4$ are equivalent. Therefore, we can reduce TS-(S)UF-4 security of ATS to TS-(S)UF-3 security of TS and SUF-CMA security of DS. (The latter notion is formally defined via the game in Figure 5.) This is captured by the the following theorem. (The proof is in [7].)

Theorem 2. *Let XX $\in \{SUF, UF\}$. Let* TS *be an echo scheme and* DS *be a digital signature scheme. For any TS-XX-4 adversary $\mathcal{A}$ there exists a TS-XX-3 adversary $\mathcal{B}$ and a SUF-CMA adversary $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{ts-xx-4}}_{\mathsf{ATS[TS,DS]}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{ts-xx-3}}_{\mathsf{TS}}(\mathcal{B}) + \mathsf{ns} \cdot \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{DS}}(\mathcal{C}) \ .$$

Moreover, $\mathcal{B}$ and $\mathcal{C}$ run in time roughly equal that of $\mathcal{A}$. The number of $\mathcal{B}$'s queries to each oracle is at most that of $\mathcal{A}$. The number of $\mathcal{C}$'s PPO *queries is at most the number of* PPO *queries made by $\mathcal{A}$.*

4 The Security of FROST

4.1 The FROST1 and FROST2 Schemes

Scheme descriptions. This section revisits the security of FROST, first proposed in [31] by Komlo and Goldberg, as a (partially) non-interactive threshold signature scheme.

First, we consider the original scheme, which we refer to as $\mathsf{FROST1}$. We then present $\mathsf{FROST2}$, an optimized version that reduces the number of exponentiations required for signing and verification from $|lr.\mathsf{SS}|$ to one. We give a detailed description of both schemes in Figure 6. The leader state st_0 contains a set curPP_i for each server i representing the set of tokens generated by server i that has not yet been used in a signing request. The state st_i for server i contains a function mapPP that maps each token pp to the randomness that is used to generate pp and $\mathsf{st}_i.\text{mapPP}(pp) = \perp$ if pp is not generated by server i yet or has already been used in a signing request. The coefficient $\lambda_i^{lr.\mathsf{SS}}$ in line 39 is the

```
Protocol $\boxed{\mathsf{FROST1}}$, $\boxed{\mathsf{FROST2}}[\mathbb{G}]$

$\mathsf{Kg}[\mathsf{h}]$:
For $i \in [0..t-1]$ do
    $a_i \leftarrow\!\!\$\; \mathbb{Z}_p$
For $i \in [1..\mathsf{ns}]$ do
    $sk_i \leftarrow\!\!\$\; \sum_{j=0}^{t-1} i^j \cdot a_j$ ; $vk_i \leftarrow g^{sk_i}$
$vk \leftarrow g^{a_0}$
$aux \leftarrow (vk_1, \ldots, vk_{\mathsf{ns}})$
Return $vk, aux, \{sk_i\}_{i\in[1..\mathsf{ns}]}$

$\mathsf{SPP}[\mathsf{h}](\mathsf{st}_i)$:
$r \leftarrow \mathbb{Z}_p$ ; $s \leftarrow \mathbb{Z}_p$
$pp \leftarrow (g^r, g^s)$
$\mathsf{st}_i.\mathrm{mapPP}(pp) \leftarrow (r, s)$
Return $(pp, \mathsf{st}_i)$

$\mathsf{LPP}[\mathsf{h}](i, pp, \mathsf{st}_0)$:
$\mathsf{st}_0.\mathrm{curPP}_i \leftarrow \mathsf{st}_0.\mathrm{curPP}_i \cup \{pp\}$
Return $\mathsf{st}_0$

$\mathsf{LR}[\mathsf{h}](M, SS, \mathsf{st}_0)$:
If $\exists\, i \in SS : \mathsf{st}_0.\mathrm{curPP}_i = \emptyset$ then
    Return $\perp$
$lr.\mathsf{msg} \leftarrow M$ ; $lr.\mathsf{SS} \leftarrow SS$
For $i \in SS$ do
    Pick $pp_i$ from $\mathsf{st}_0.\mathrm{curPP}_i$
    $lr.\mathsf{PP}(i) \leftarrow pp_i$
    $\mathsf{st}_0.\mathrm{curPP}_i \leftarrow \mathsf{st}_0.\mathrm{curPP}_i \setminus \{pp_i\}$
Return $(lr, \mathsf{st}_0)$

$\mathsf{Vf}[\mathsf{h}](vk, M, sig)$:
$(R, z) \leftarrow sig$
$c \leftarrow \mathsf{h}_2(vk, M, R)$
Return $(g^z = R \cdot vk^c)$
```

```
$\mathsf{CompPar}[\mathsf{h}](vk, lr)$:
$M \leftarrow lr.\mathsf{msg}$
For $i \in lr.\mathsf{SS}$ do
    $\boxed{d_i \leftarrow \mathsf{h}_1(vk, lr, i)}$
    $\boxed{d_i \leftarrow \mathsf{h}_1(vk, lr)}$
    $(R_i, S_i) \leftarrow lr.\mathsf{PP}(i)$
$R \leftarrow \prod_{i\in lr.\mathsf{SS}} R_i S_i^{d_i}$
$c \leftarrow \mathsf{h}_2(vk, M, R)$
Return $(R, c, \{d_i\}_{i\in lr.\mathsf{SS}})$

$\mathsf{PS}[\mathsf{h}](lr, i, \mathsf{st}_i)$:
$pp_i \leftarrow lr.\mathsf{PP}(i)$
If $\mathsf{st}_i.\mathrm{mapPP}(pp_i) = \perp$ then
    Return $(\perp, \mathsf{st}_i)$
$(r_i, s_i) \leftarrow \mathsf{st}_i.\mathrm{mapPP}(pp_i)$
$\mathsf{st}_i.\mathrm{mapPP}(pp_i) \leftarrow \perp$
$(R, c, \{d_j\}_{j\in lr.\mathsf{SS}})$
    $\leftarrow \mathsf{CompPar}[\mathsf{h}](\mathsf{st}_i.\mathsf{vk}, lr)$
$z_i \leftarrow r_i + d_i \cdot s_i + c \cdot \lambda_i^{lr.\mathsf{SS}} \cdot \mathsf{st}_i.\mathsf{sk}$
Return $((R, z_i), \mathsf{st}_i)$

$\mathsf{Agg}[\mathsf{h}](\mathrm{PS}, \mathsf{st}_0)$:
$R \leftarrow \perp$ ; $z \leftarrow 0$
For $(R', z') \in \mathrm{PS}$ do
    If $R = \perp$ then $R \leftarrow R'$
    If $R \neq R'$ then return $(\perp, \mathsf{st}_0)$
    $z \leftarrow z + z'$
Return $((R, z), \mathsf{st}_0)$

$\mathsf{SVf}[\mathsf{h}](vk, lr, sig)$:
$(R^*, z^*) \leftarrow sig$
$(R, c, \{d_j\}_{j\in lr.\mathsf{SS}})$
    $\leftarrow \mathsf{CompPar}[\mathsf{h}](vk, lr)$
Return $(R = R^*) \wedge (g^{z^*} = R \cdot vk^c)$
```

Fig. 6. The protocol $\mathsf{FROST1}[\mathbb{G}]$ and $\mathsf{FROST2}[\mathbb{G}]$, where $\mathbb{G}$ is a cyclic group with prime order p and generator g. Further, ns is the number of parties, and t is the threshold of the schemes. We require $t \leq \mathsf{ns} \leq p-1$. The protocol $\mathsf{FROST1}$ contains all but the dashed box, and the protocol $\mathsf{FROST2}$ contains all but the solid box. The function $\mathsf{h}_i(\cdot)$ is computed as $\mathsf{h}(i, \cdot)$ for $i = 1, 2$. PS denotes a set of partial signatures.

Lagrange coefficient for the set $lr.\mathsf{SS}$, which is defined (for any set $S \subseteq [1..\mathsf{ns}]$)

```
Game $\mathbf{G}^{\mathrm{omdl}}_{\mathbb{G}}$

INIT:
1 cid ← 0; $\ell \leftarrow 0$; $T \leftarrow ()$

CHAL():
2 cid ← cid + 1; $x_{\mathrm{cid}} \leftarrow\!\!\$\ \mathbb{Z}_p$
3 Return $g^{x_{\mathrm{cid}}}$

DLOG($X$):
4 If $T(X) \neq \bot$ then return $T(X)$
5 $\ell \leftarrow \ell + 1$; $T(X) \leftarrow \mathsf{DL}_{\mathbb{G},g}(X)$
6 Return $T(X)$

FIN($\{y_i\}_{i \in [\mathrm{cid}]}$):
7 If $\ell \geq$ cid then return false
8 If $\forall\, i \in [\mathrm{cid}] : y_i = x_i$ then
9     Return true
10 Return false
```

Fig. 7. The OMDL game, where $\mathbb{G}$ is a cyclic group with prime order p and generator g.

as

$$\lambda_i^S := \prod_{j \in S, i \neq j} \frac{j}{j - i} \ .$$

The algorithm CompPar is a helper algorithm that computes the parameters $R, c, \{d_i\}_{i \in lr.\mathsf{SS}}$ used during signing. The difference between FROST1 and FROST2 is the way d_i is computed in CompPar. In FROST1, each d_i is a different hash value for each server i, while in FROST2, d_i's are the same hash value for all servers.

It is not hard to verify that both schemes satisfy perfect correctness.

OVERVIEW OF OUR RESULTS. We begin by showing that FROST2 is TS-SUF-2-secure (under OMDL) but not TS-UF-3-secure. We then show that FROST1 is TS-SUF-3-secure but not TS-UF-4-secure. Theoretically, our results imply the separations between TS-(S)UF-2 and TS-(S)UF-3 and between TS-(S)UF-3 and TS-(S)UF-4. Practically speaking, our results indicate a separation between the security of FROST1 and FROST2. To complete the picture, a TS-SUF-4-secure variant of FROST1 can be obtained via the general transformation from Theorem 2, although it is an interesting open question whether a more efficient variant exists.

4.2 TS-SUF-2 Security of FROST2

We first show that FROST2 is TS-SUF-2-secure in the ROM under the OMDL assumption, which is formally defined in Figure 7. Formally, we show the following theorem.

Theorem 3. *For any TS-SUF-2 adversary $\mathcal{A}$ making at most q_s queries to* PPO *and at most q_h queries to* RO*, there exists an OMDL adversary $\mathcal{B}$ making at most $2q_s + \mathsf{ns}$ queries to* CHAL *such that*

$$\mathbf{Adv}^{\mathrm{ts\text{-}suf\text{-}2}}_{\mathsf{FROST2}[\mathbb{G}]}(\mathcal{A}) \leq \sqrt{q \cdot (\mathbf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3q^2/p)} \ ,$$

where $q = q_s + q_h + 1$. Moreover, $\mathcal{B}$ runs in time roughly equal two times that of $\mathcal{A}$, plus the time to perform at most $(4\mathsf{ns} + 2) \cdot q + 2q_s + 2\mathsf{ns}^2$ exponentiations and group operations.

The core of the proof is a reduction from OMDL [2], which will need to use rewinding (via a variant of the Forking Lemma). The main challenge is to ensure that the reduction can simulate properly with a number of queries to Dlog which is smaller than the number of DL challenges. Further below, we are going to show that FROST2 is not TS-UF-3 secure, thus showing the above result is optimal with respect to our hierarchy.

Proof (of Theorem 3). Let $\mathcal{A}$ be an adversary as described in the theorem. Denote the output message-signature pair of $\mathcal{A}$ as $(M^*, sig^* = (R^*, z^*))$. Without loss of generality, we assume $\mathcal{A}$ always queries RO on $\mathsf{h}_2(vk, M^*, R^*)$ before $\mathcal{A}$ returns and always queries RO on $\mathsf{h}_1(vk, lr)$ prior to the query $\text{PSignO}(i, lr)$ for some i and lr. (This adds up to q_s additional RO queries, and we let $q = q_h + q_s + 1$.) Denote lr^* as the leader query such that $\mathsf{h}_1(vk, lr^*)$ is the first query prior to the query $\mathsf{h}_2(vk, M^*, R^*)$ satisfying $\mathsf{SVf}[\mathsf{h}](vk, lr^*, sig^*) = \mathsf{true}$. If such lr^* does not exists, lr^* is set to $\perp$. Denote the event E_1 as

$$\mathsf{Vf}[\mathsf{h}](vk, M^*, sig^*) \ \wedge \ (lr^* = \perp \ \vee \ \mathrm{S}_2(lr^*) < t - |CS|) \ .$$

It is clear that if $\mathcal{A}$ wins the game $\mathbf{G}^{\text{ts-suf-2}}_{\mathsf{FROST2}}$, then E_1 must occur, which implies $\Pr[E_1] \geq \mathbf{Adv}^{\text{ts-suf-2}}_{\mathsf{FROST2}[\mathbb{G}]}(\mathcal{A})$. Therefore, the theorem will follow from the following lemma. (We isolate this statement as its own lemma also because it will be helpful in the proof of Theorem 5 below.) $\square$

Lemma 4. *There exists an OMDL adversary $\mathcal{B}$ making at most $2q_s + t$ queries to* Chal *such that*

$$\Pr[E_1] \leq \sqrt{q \cdot (\mathbf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3q^2/p)} \ .$$

Moreover, $\mathcal{B}$ runs in time roughly twice that of $\mathcal{A}$, plus the time to perform at most $(4\mathsf{ns} + 2) \cdot q + 2q_s + 2\mathsf{ns}^2$ exponentiations and group operations.

The proof of Lemma 4 is in [7]. It uses a variant of the general Forking Lemma of [3], also given in 4, that allows us to get better bounds in our analysis.

4.3 TS-SUF-3 Security of FROST1

In this section, we show that FROST1 is TS-SUF-3-secure in the ROM under the OMDL assumption. Formally, we show the following theorem.

Theorem 5. *For any TS-SUF-3 adversary $\mathcal{A}$ making at most q_s queries to* PPO *and at most q_h queries to* RO*, there exists an OMDL adversary $\mathcal{B}$ making at most $2q_s + t$ queries to* Chal *such that*

$$\mathbf{Adv}^{\text{ts-suf-3}}_{\mathsf{FROST1}[\mathbb{G}]}(\mathcal{A}) \leq 4\mathsf{ns} \cdot q \cdot \sqrt{\mathbf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 6q/p} \ ,$$

where $q = q_s + q_h + 1$. Moreover, $\mathcal{B}$ runs in time roughly equal two times that of $\mathcal{A}$, plus the time to perform at most $6\mathsf{ns} \cdot q + 4q_s + 2\mathsf{ns}^2$ exponentiations and group operations.

The proof here follows a similar pattern than that of Theorem 3, but will be more complex. In particular, the lesser tight bound is due to the fact that we need to consider an additional bad event, which we upper bound via a different reduction from OMDL. As we explain in detail below, this reduction will make use of a looser Forking Lemma, which is a variant of the "Local Forking Lemma" [1], which only resamples a single random oracle output when rewinding. The extra looseness is due to needing to ensure an extra condition when rewinding.

Proof (of Theorem 5). Let $\mathcal{A}$ be the adversary described in the theorem. Denote the output message-signature pair of $\mathcal{A}$ as $(M^*, sig^* = (R^*, z^*))$. Without loss of generality, we assume $\mathcal{A}$ always queries RO on $\mathsf{h}_2(vk, M^*, R^*)$ before $\mathcal{A}$ returns and always queries RO on $\mathsf{h}_1(vk, lr, i)$ prior to the query $\text{PS}\textsc{ign}\text{O}(i, lr)$ for some i and lr. (This adds up to q_s additional RO queries, and we let $q = q_h + q_s + 1$.) Denote lr^* as the leader query such that $\mathsf{h}_1(vk, lr^*, i)$ is the first RO query prior to the $\mathsf{h}_2(vk, M^*, R^*)$ query for some i satisfying $\mathsf{SVf}[\mathsf{h}](vk, lr^*, sig^*) = \mathsf{true}$. If such lr^* does not exist, lr^* is set to $\perp$. Denote the event E_1 as

$$\mathsf{Vf}[\mathsf{h}](vk, M^*, sig^*) \ \wedge\ (lr^* = \perp \ \vee\ \mathrm{S}_2(lr^*) < t - |CS|) \ .$$

Denote the event E_2 as

$$\mathsf{Vf}[\mathsf{h}](vk, M^*, sig^*) \ \wedge\ lr^* \neq \perp \ \wedge\ \mathrm{S}_2(lr^*) \neq \mathrm{S}_3(lr^*) \ .$$

If $\mathcal{A}$ wins the game $\mathbf{G}^{\text{ts-suf-3}}_{\mathsf{FROST2}}$ and $lr^* \neq \perp$, we know either $\mathrm{S}_2(lr^*) < t - |CS|$ or $\mathrm{S}_2(lr^*) \neq \mathrm{S}_3(lr^*)$. Therefore, if $\mathcal{A}$ wins the game $\mathbf{G}^{\text{ts-suf-3}}_{\mathsf{FROST2}}$, then either E_1 or E_2 occurs, which implies

$$\mathbf{Adv}^{\text{ts-suf-3}}_{\mathsf{FROST1}[\mathbb{G}]}(\mathcal{A}) \leq \Pr[E_1] + \Pr[E_2] \leq 2\max\{\Pr[E_1], \Pr[E_2]\} \ .$$

Thus, we conclude the theorem with the following two lemmas.

Lemma 6. *There exists an OMDL adversary $\mathcal{B}$ making at most $2q_s + t$ queries to* $\textsc{Chal}$ *such that*

$$\Pr[E_1] \leq \sqrt{q \cdot (\mathbf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3q^2(\mathsf{ns}+1)^2/p)} \ ,$$

Moreover, $\mathcal{B}$ runs in time roughly equal two times that of $\mathcal{A}$, plus the time to perform at most $6\mathsf{ns} \cdot q + 4q_s + 2\mathsf{ns}^2$ exponentiations and group operations.

Lemma 7. *There exists an OMDL adversary $\mathcal{B}$ making at most $2q_s$ queries to* $\textsc{Chal}$ *such that*

$$\Pr[E_2] \leq \mathsf{ns} \cdot q\sqrt{2(\mathbf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 1/p)} \ .$$

Moreover, $\mathcal{B}$ runs in time roughly equal two times that of $\mathcal{A}$, plus the time to perform at most $6\mathsf{ns} \cdot q + 4q_s + 2\mathsf{ns}^2$ exponentiations and group operations.

The proof of Lemma 6 is almost the same as Lemma 4, so we omit the full proof. The only difference is that $\mathcal{C}$ takes as input $h_1, \ldots, h_{(\mathsf{ns}+1)q}$ in order to simulate all RO queries. For a RO query $\mathsf{h}_1(vk, lr, i)$, $\mathcal{C}$ first enumerates all $i' \in [\mathsf{ns}]$ and assigns $h_{(\mathrm{ctr}_h - 1)(\mathsf{ns}+1)+i'}$ to $\mathsf{h}_1(vk, lr, i')$. Then, $\mathcal{C}$ computes the nonce R for lr and assigns $h_{\mathrm{ctr}_h(\mathsf{ns}+1)}$ to $\mathsf{h}_2(vk, lr.\mathsf{msg}, R)$ if it is not assigned any value yet. Similarly, for a new RO query $\mathsf{h}_1(vk, M, R)$, its value is set to $h_{\mathrm{ctr}_h(\mathsf{ns}+1)}$. The rest follows by similar analysis.

```
Adversary $\mathcal{A}^{\textsc{Init},\mathrm{PPO},\mathrm{PSignO},\mathrm{RO}}$:
$CS \leftarrow \{3,4\}$ ; $(vk, aux, \{sk_3, sk_4\}) \leftarrow\$ \textsc{Init}(CS)$
$(R_1, S_1) \leftarrow\$ \mathrm{PPO}(1)$ ; $(R_2, S_2) \leftarrow\$ \mathrm{PPO}(2)$ ; $\gamma \leftarrow \lambda_1^{\{1,3,4\}}/\lambda_1^{\{1,2,3\}}$
$lr.\mathsf{msg} \leftarrow M$ ; $lr.\mathsf{SS} \leftarrow \{1,2,3\}$
$lr.\mathsf{PP}(1) \leftarrow (R_1, S_1)$ ; $lr.\mathsf{PP}(2) \leftarrow (R_2, S_2)$
$lr.\mathsf{PP}(3) \leftarrow (R_1^{\gamma-1}R_2^{-1}, S_1^{\gamma-1}S_2^{-1})$
$z_1 \leftarrow \mathrm{PSignO}(1, lr)$
$d \leftarrow \mathrm{RO}(1, vk, lr)$ ; $R \leftarrow R_1^{\gamma}S_1^{\gamma\cdot d}$ ; $c \leftarrow \mathrm{RO}(2, vk, R, M)$
$z \leftarrow \gamma \cdot z_1 + c(\lambda_3^{\{1,3,4\}} \cdot sk_3 + \lambda_4^{\{1,3,4\}} \cdot sk_4)$
Return $(M, (R, z))$
```

Fig. 8. Adversary $\mathcal{A}$ that wins the game $\mathbf{G}^{\text{ts-uf-3}}_{\mathsf{FROST2}}$, where M is a fixed message.

```
Adversary $\mathcal{A}^{\textsc{Init},\mathrm{PPO},\mathrm{PSignO},\mathrm{RO}}$:
$CS \leftarrow \{5,10\}$ ; $(vk, aux, \{sk_5, sk_{10}\}) \leftarrow\$ \textsc{Init}(CS)$
$(R_1, S_1) \leftarrow\$ \mathrm{PPO}(11)$ ; $s_2, r_2, s_3, r_3 \leftarrow\$ \mathbb{Z}_p$
$lr.\mathsf{msg} \leftarrow M$ ; $lr.\mathsf{SS} \leftarrow \{11,15,20\}$
$lr.\mathsf{PP}(11) \leftarrow (R_1, S_1)$ ; $lr.\mathsf{PP}(15) \leftarrow (g^{r_2}, g^{s_2})$ ; $lr.\mathsf{PP}(20) \leftarrow (g^{r_3}, g^{s_3})$
$z_1 \leftarrow \mathrm{PSignO}(11, lr)$
For $i \in \{11,15,20\}$ do $d_i \leftarrow \mathrm{RO}(1, vk, lr, i)$
$R \leftarrow R_1 S_1^{d_{11}} g^{r_2+r_3+s_2\cdot d_{15}+s_3\cdot d_{20}}$ ; $c \leftarrow \mathrm{RO}(2, vk, R, M)$
$z \leftarrow z_1 + r_2 + r_3 + s_2 \cdot d_{15} + s_3 \cdot d_{20} + c(\lambda_5^{\{5,10,11\}} \cdot sk_5 + \lambda_{10}^{\{5,10,11\}} \cdot sk_{10})$
Return $(M, (R, z))$
```

Fig. 9. Adversary $\mathcal{A}$ that wins the game $\mathbf{G}^{\text{ts-uf-4}}_{\mathsf{FROST1}}$, where M is a fixed message.

To prove Lemma 7, we need a variant of the Local Forking Lemma of [1], which is given in [7] along with the proof of Lemma 7 itself.

4.4 Attacks for FROST1 and FROST2

<u>FROST2 IS NOT TS-UF-3 SECURE</u> Consider the setting where $\mathsf{ns} = 4$ and $t = 3$ and the adversary $\mathcal{A}$ for the game $\mathbf{G}^{\text{ts-uf-3}}_{\mathsf{FROST2}}$ described in Figure 8. We now show that $\mathbf{Adv}^{\text{ts-uf-3}}_{\mathsf{FROST2}}(\mathcal{A}) = 1$. From the execution of PSignO, we know $g^{z_1} = R_1 S_1^d vk_1^{\lambda_1^{\{1,2,3\}}\cdot c}$. Therefore,

$$g^z = R_1^{\gamma} S_1^{d\cdot\gamma} vk_1^{\gamma\cdot\lambda_1^{\{1,2,3\}}\cdot c} vk_3^{\lambda_3^{\{1,3,4\}}\cdot c} vk_4^{\lambda_4^{\{1,3,4\}}\cdot c}$$
$$= Rg^{c\cdot\sum_{i\in\{1,3,4\}}\lambda_i^{\{1,3,4\}}\cdot sk_i} = R \cdot vk^c ,$$

which implies $(M, (R, z))$ is valid for vk. Also, it is clear that $\mathrm{S}_2(lr) = \{1\}$ and $\mathrm{S}_3(lr) = \{1,2\}$, which implies the condition $\mathtt{tf}_3(lr)$ does not hold. Therefore, $\mathcal{A}$ wins the game $\mathbf{G}^{\text{ts-uf-3}}_{\mathsf{FROST2}}$ with probability 1.

<u>FROST1 is not TS-UF-4 secure</u> Consider the setting where $\mathsf{ns} = 20$ and $t = 3$ and the adversary $\mathcal{A}$ for the game $\mathbf{G}^{\text{ts-uf-4}}_{\mathsf{FROST1}}$ described in Figure 9. We now show that $\mathbf{Adv}^{\text{ts-uf-4}}_{\mathsf{FROST1}}(\mathcal{A}) = 1$. From the execution of PSignO, we know $g^{z_1} = R_1 S_1^{d_{11}} vk_{11}^{\lambda_{11}^{\{11,15,20\}}\cdot c}$. The key observation here is that $\lambda_{11}^{\{11,15,20\}} = \frac{15\cdot 20}{(15-11)(20-11)} = \frac{25}{3} = \frac{5\cdot 10}{(5-11)(10-11)} = \lambda_{11}^{\{5,10,11\}}$. Therefore,

$$
\begin{aligned}
g^z &= R_1 S_1^{d_{11}} g^{r_2+r_3+s_2\cdot d_{15}+s_3\cdot d_{20}} vk_{11}^{\lambda_{11}^{\{11,15,20\}}\cdot c} vk_5^{\lambda_5^{\{5,10,11\}}\cdot c} vk_{10}^{\lambda_{10}^{\{5,10,11\}}\cdot c} \\
&= Rg^{c\cdot\sum_{i\in\{5,10,11\}}\lambda_i^{\{5,10,11\}}\cdot sk_i} = R\cdot vk^c \ ,
\end{aligned}
$$

which implies $(M,(R,z))$ is valid for vk. Also, it is clear that $\mathrm{S}_2(lr) = \{11\}$ and $\mathrm{S}_4(lr) = \{11,15,20\}$, which implies the condition $\mathtt{tf}_4(lr)$ does not hold. Therefore, $\mathcal{A}$ wins the game $\mathbf{G}^{\text{ts-uf-4}}_{\mathsf{FROST1}}$ with probability 1.

The reason why the attack is possible for FROST1 is because the honest server 11 replies to the leader request lr with tokens $lr.\mathsf{PP}(15)$ and $lr.\mathsf{PP}(20)$ not generated by the honest servers 15 and 20 but by the adversary instead. Therefore, the attack is prevented by the general transformation from TS-SUF-3 security to TS-SUF-4 security described in Figure 4 since after the transformation, an honest server replies to a leader request only when all the tokens within the request are authenticated by the corresponding servers, and thus the adversary cannot generate tokens on behalf of honest servers anymore.

5 FROST2 with Distributed Key Generation

In this section, we prove the security of FROST2 together with distributed key generation (DKG). In particular, we prove the security of FROST2 with the variant of the Pedersen DKG protocol [24] with proofs of possession originally proposed in combination with FROST1 [30]. We call this protocol PedPoP and provide a description in Figure 10.

Throughout this section, we denote public keys by X, instead of vk, and corresponding secret keys by x, instead of sk. We also denote the joint public key by $\tilde{X}$ and aggregated nonce by $\tilde{R}$. Hash function $\mathsf{h}_i(\cdot)$ is computed as $\mathsf{h}_i(\cdot)$ for $i = 0, 1, 2$.

<u>Efficient distributed key generation.</u> The Pedersen DKG can be viewed as ns parallel instantiations of Feldman verifiable secret sharing (VSS) [19], which itself is derived from Shamir secret sharing [36] but additionally requires each participant to provide a vector commitment $\vec{C}$ to ensure their received share is consistent with all other participants' shares. In addition, PedPoP requires each participant to provide a Schnorr proof of knowledge of the secret corresponding to the first term of their commitment. This is to ensure that unforgeability (but not liveness) holds even if more than half of the participants are dishonest.

<u>Schnorr Knowledge of Exponent Assumption.</u> We introduce the Schnorr knowledge of exponent assumption (Schnorr-KoE), which we show is true under the discrete logarithm (DL) assumption in the algebraic group model (AGM)

$\mathsf{PedPoP.KeyGen}(t, \mathsf{ns})$

1. Each party P_i chooses a random polynomial $f_i(Z)$ over $\mathbb{Z}_p$ of degree $t-1$
$$f_i(Z) = a_{i,0} + a_{i,1}Z + \cdots + a_{i,t-1}Z^{t-1}$$
and computes $A_{i,k} = g^{a_{i,k}}$ for $k =\in [0..t-1]$. Denote $x_i = a_{i,0}$ and $X_{i,0} = A_{i,0}$. Each P_i computes a proof of possession of $X_{i,0}$ as a Schnorr signature on $X_{i,0}$ as follows. They sample $\bar{r}_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and set $\bar{R}_i \leftarrow g^{\bar{r}_i}$. They compute $\bar{c}_i \leftarrow \mathsf{h}_0(X_{i,0}, X_{i,0}, \bar{R}_i)$ and set $\bar{z}_i \leftarrow \bar{r}_i + \bar{c}_i \cdot x_i$. They then derive a commitment $\vec{C}_i = (A_{i,0}, ..., A_{i,t-1})$ and broadcast $((\bar{R}_i, \bar{z}_i), \vec{C}_i)$.
2. After receiving commitments from all other parties, each participant verifies the Schnorr signatures by computing $\bar{c}_j \leftarrow \mathsf{h}_0(A_{j,0}, A_{j,0}, \bar{R}_j)$ and checking that
$$g^{\bar{z}_j} = \bar{R}_j {A_{j,0}}^{\bar{c}_j} \text{ for } j \in [1..\mathsf{ns}]$$
If any checks fail, they disqualify the corresponding participant.
3. Each P_i computes secret shares $\bar{x}_{i,j} = f_i(\mathsf{id}_j)$ for $j \in [1..\mathsf{ns}]$, where id_j is the participant identifier, and sends $\bar{x}_{i,j}$ secretly to party P_j.
4. Each party P_j verifies the shares they received from the other parties by checking that
$$g^{\bar{x}_{i,j}} = \prod_{k=0}^{t-1} A_{i,k}^{\mathsf{id}_j^k}$$
If the check fails for an index i, P_j broadcasts a complaint against P_i.
5. For each of the complaining parties P_j against P_i, P_i broadcasts the share $\bar{x}_{i,j}$. If any of the revealed shares fails to satisfy the equation, or should P_i not broadcast anything for a complaining player, then P_i is disqualified. The share of a disqualified party P_i is set to 0.
6. The secret share for each P_j is $\bar{x}_j = \sum_{i=1}^{\mathsf{ns}} \bar{x}_{i,j}$.
7. If $X_{i,0} = X_{j,0}$ for any $i \neq j$, then abort. Else, the output is the joint public key $\tilde{X} = \prod_{i=1}^{\mathsf{ns}} X_{i,0}$.

Fig. 10. PedPoP: The Pedersen distributed key generation protocol with proofs of possession.

without any tightness loss. The purpose of the Schnorr-KoE assumption is to ensure that the Pedersen DKG can be run in the honest minority setting, where we assume the existence of at least a single honest party and up to $t-1$ corrupt parties. The Schnorr-KoE assumption can be avoided if we assume an honest majority in the DKG. However, we prefer to allow more corruptions with the tradeoff of a stronger assumption.

```
Game $\mathbf{G}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{A})$

INIT:
  $\omega \leftarrow\!\!\$ \{0,1\}^{\mathsf{rl}_{\mathcal{A}}}$  // Coins given to $\mathcal{A}$
  Return $\omega$

FSIGNO:
  $x, \bar{r} \leftarrow\!\!\$ \mathbb{Z}_p$
  $X \leftarrow g^x$ ; $\bar{R} \leftarrow g^{\bar{r}}$
  $\bar{c} \leftarrow \tilde{\mathsf{h}}_0(X, X, \bar{R})$
  $\bar{z} \leftarrow \bar{r} + \bar{c} \cdot x$
  $Q_{\text{FSIGNO}} \leftarrow Q_{\text{FSIGNO}} \cup \{(X, \bar{R}, \bar{z})\}$
  Return $(X, \bar{R}, \bar{z})$

CHAL$(X, \bar{R}, \bar{z})$:
  $\bar{c} \leftarrow \tilde{\mathsf{h}}_0(X, X, \bar{R})$
  If $(X, \bar{R}, \bar{z}) \in Q_{\text{FSIGNO}}$ or $g^{\bar{z}} \neq \bar{R}X^{\bar{c}}$
    Return $\bot$
  $\alpha \leftarrow\!\!\$ \mathsf{Ext}(\mathbb{G}, \omega, Q_{\text{FSIGNO}}, \mathsf{Q}_{\tilde{\mathsf{h}}_0})$
  If $g^{\alpha} \neq X$ then $\mathsf{win} \leftarrow \mathsf{true}$
  Return $\alpha$

RO$(\theta)$: // Random oracle
  If $\tilde{\mathsf{h}}(\theta) = \bot$ then $\tilde{\mathsf{h}}(\theta) \leftarrow\!\!\$ \mathbb{Z}_p$
  Return $\tilde{\mathsf{h}}(\theta)$

FIN$(\{0,1\}^*)$: // $\mathcal{A}$ outputs a bit string
  Return $\mathsf{win}$
```

Fig. 11. Game used to define the Schnorr knowledge of exponent (Schnorr-KoE) assumption, where $\mathbb{G}$ is a cyclic group of order p with generator g. By $\mathsf{rl}_{\mathcal{A}}$ we denote the randomness length of $\mathcal{A}$. $\tilde{\mathsf{h}}$ is initialized to be an empty table.

The Schnorr-KoE assumption allows us to prove the security of multi-party signatures in the setting where each participant is required to provide a proof of possession of their secret key during a key generation and registration phase. By formatting our desired security property directly as an assumption, we avoid the complexity of rewinding adversaries, which is required when proving security of Schnorr signatures in the ROM only, and which may result in a loss of tightness exponential in the number of parties that the adversary controls [38]. The Schnorr-KoE assumption implies that if an adversary can forge a Schnorr signature for some public key, then it must know the corresponding secret key. It is a non-falsifiable assumption.

Our proof for Schnorr-KoE extends a result by Fuchsbauer et al. [20], which showed that the security of Schnorr signatures can be tightly reduced to the DL assumption in the AGM. We improve on their result by considering extraction rather than forgeability and by allowing extraction even when the adversary chooses their own public key. While new to the setting of multi-party signatures, Schnorr-KoE is reminiscent of prior knowledge of exponent assumptions [15, 4] employed to prove the security of Succinct NIZK arguments (SNARKs).

For the definition, consider the game in Figure 11 associated to group $\mathbb{G}$, adversary $\mathcal{A}$, and an algorithm Ext, called an extractor. The adversary $\mathcal{A}$ is run with coins ω. $\mathcal{A}$ has access to a signing oracle FSIGNO that outputs a Schnorr signature under a randomly sampled key X on the message X. (The name, Full Sign Oracle, reflects that the oracle samples a fresh public key with each invocation.) It can call its challenge oracle CHAL with a triple $(X, \bar{R}, \bar{z})$. If this is not a triple returned by the full signing oracle, yet verifies as a Schnorr signature under public key X, the extractor is asked to find the discrete logarithm α of X, and the adversary wins (the game sets win to true) if the extractor fails. The inputs to the extractor are the coins of the adversary, the description of the group $\mathbb{G}$, the set Q_{FSIGNO} and a list $\mathsf{Q}_{\tilde{\mathsf{h}}_0}$. The latter, for every query $(X, X, \bar{R})$ that $\mathcal{A}$ made to random oracle $\tilde{\mathsf{h}}_0$, stores the response of the oracle. (The length of the list is thus the number of $\tilde{\mathsf{h}}_0$ queries made by $\mathcal{A}$.) Note that multiple queries to CHAL are allowed, so that this captures the ability to perform multiple extractions.

Asymptotically, we would say that the Schnorr-KoE assumption holds with respect to $\mathbb{G}$ if for all PPT adversaries $\mathcal{A}$, there exists a PPT extractor Ext such that $\mathbf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{A})$, which would now be a function of the security parameter, is negligible.

The proof of the following can be found in [14]. For convenience, the statement is asymptotic. Note that the random oracle model is implicit through $\tilde{\mathsf{h}}_0$ being a random oracle in the game of Figure 11.

Theorem 8 (DL $\Rightarrow$ Schnorr-KoE)**.** *The* Schnorr-KoE *assumption with respect to the group* $\mathbb{G}$ *is implied by the DL assumption with respect to* $\mathbb{G}$ *in the AGM.*

TS-UF-0 SECURITY. In terms of our framework, our proof of FROST2 + PedPoP considers a single honest player and so aligns with the notion of TS-UF-0 security defined in Figure 2. Since we now consider distributed key generation instead of trusted key generation, the initialization oracle INIT is replaced with a single execution of PedPoP.KeyGen as defined in Figure 10. The proofs of possession required by PedPoP.KeyGen ensure that the simulator in our security reduction is able to extract sufficient information (via the Schnorr-KoE assumption) to simulate signing as in the TS-UF-0 definition, in which all secret key material is generated by the simulator directly. The signing oracles PPO and PSIGNO remain identical to Figure 2. We have not currently investigated whether TS-UF-2 security holds.

5.1 Security of FROST2 + PedPoP

We now prove the security of FROST2 with distributed key generation protocol PedPoP under the Schnorr-KoE assumption and OMDL assumption in the ROM.

Theorem 9 (FROST2 + PedPoP)**.** FROST2 *with distributed key generation protocol* PedPoP *is* TS-UF-0 *secure under the* Schnorr-KoE *assumption and the OMDL assumption in the ROM.*

We make use of an intermediary assumption, the binonce Schnorr computational (Bischnorr) assumption, which we define and prove secure under the OMDL assumption in the ROM.

Equipped with this assumption, our proof proceeds as follows. Let $\mathcal{A}$ be a PPT adversary attempting to break the TS-UF-0 security of FROST2. We construct a PPT adversary $\mathcal{B}_1$ playing game $\mathbf{G}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_1)$ and thence, from the

```
Game $\mathbf{G}^{\text{bi-sch}}_{\mathbb{G}}$:

Init:
    $\dot{x} \leftarrow\!\$\ \mathbb{Z}_p$
    $\dot{X} \leftarrow g^{\dot{x}}$
    Return $\dot{X}$

BinonceO():
    $r, s \leftarrow\!\$\ \mathbb{Z}_p$
    $(R, S) \leftarrow (g^r, g^s)$
    $Q_{\text{Bin}} \leftarrow Q_{\text{Bin}} \cup \{(R, S, r, s)\}$
    Return $(R, S)$

BisignO($k, M, \{(\gamma_i, R_i, S_i)\}_{i \in SS}$):
    If $(R_k, S_k, r_k, s_k) \notin Q_{\text{Bin}}$ or $(R_k, S_k) \in Q_{\text{Used}}$
        Return false
    $Q_{\text{Used}} \leftarrow Q_{\text{Used}} \cup \{(R_k, S_k)\}$
    $d \leftarrow \hat{\mathsf{h}}_1(\dot{X}, M, \{(\gamma_i, R_i, S_i)\}_{i \in SS})$
    $\tilde{R} \leftarrow \prod_{i \in SS} R_i S_i^d$
    $c \leftarrow \hat{\mathsf{h}}_2(\dot{X}, M, \tilde{R})$
    $z_k \leftarrow r_k + d \cdot s_k + c \cdot \gamma_k \cdot \dot{x}$
    $Q_{\text{Bis}} \leftarrow Q_{\text{Bis}} \cup \{(M, \tilde{R})\}$
    Return $z_k$

RO($\theta$): // Random oracle
    If $\hat{\mathsf{h}}(\theta) = \perp$ then $\hat{\mathsf{h}}(\theta) \leftarrow\!\$\ \mathbb{Z}_p$
    Return $\hat{\mathsf{h}}(\theta)$

Fin($M^*, R^*, z^*$):
    If $g^{z^*} = R^* \dot{X}^{\hat{\mathsf{h}}_2(\dot{X}, M^*, R^*)}$ and $(M^*, R^*) \notin Q_{\text{Bis}}$
        Return true
    Else return false
```

Fig. 12. Game used to define the binonce Schnorr computational (Bischnorr) assumption, where $\mathbb{G}$ is a cyclic group of order p with generator g. $\hat{\mathsf{h}}$ is initialized to be an empty table.

Schnorr-KoE assumption, obtain an extractor Ext for it. We construct a PPT adversary $\mathcal{B}_2$ playing game $\mathbf{G}^{\text{bi-sch}}_{\mathbb{G}}(\mathcal{B}_2)$ such that whenever $\mathcal{A}$ outputs a valid forgery, either $\mathcal{B}_1$ breaks the Schnorr-KoE assumption or $\mathcal{B}_2$ breaks the Bischnorr assumption. Formally, for security parameter κ, we have

$$\mathbf{Adv}^{\text{ts-uf-0}}_{\mathsf{FROST2}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_1) + \mathbf{Adv}^{\text{bi-sch}}_{\mathbb{G}}(\mathcal{B}_2) + \mathsf{negl}(\kappa)$$

Binonce Schnorr assumption. The Bischnorr assumption equips an adversary with two oracles, BinonceO and BisignO, and two hash functions, $\hat{\mathsf{h}}_1$ and $\hat{\mathsf{h}}_2$, and asks it to forge a new Schnorr signature with respect to a challenge public key $\dot{X}$. The BinonceO oracle takes no input and responds with two random nonces (R, S). The BisignO oracle takes as input an index k, a message M, and a set of nonces and scalars $\{(\gamma_i, R_i, S_i)\}_{i \in SS}$. It checks that (R_k, S_k) is a BinonceO response and that it has not been queried on (R_k, S_k) before. It returns an error if not. It then computes an aggregated randomized nonce $\tilde{R} = \prod_{i \in SS} R_i S_i^d$, where $d = \hat{\mathsf{h}}_1(\dot{X}, M, \{(\gamma_i, R_i, S_i)\}_{i \in SS})$. BisignO then returns z such that $(\tilde{R}, z)$ is a valid Schnorr signature with respect to $\hat{\mathsf{h}}_2$. The adversary wins if it can output a verifying (M^*, R^*, z^*) that was not output by BisignO.

The oracle BisignO can only be queried once for each pair of nonces (R, S) output by BinonceO. The index k denotes which (γ_k, R_k, S_k) out of the list $\{(\gamma_i, R_i, S_i)\}_{i \in SS}$ is being queried; the remaining scalars and nonces appear only to inform BINONCEO what to include as input to $\hat{\mathsf{h}}_1$. The scalar γ_k allows the response z_k to be given as $z_k = r_k + d \cdot s_k + c \cdot \gamma_k \cdot \dot{x}$, as opposed to $r_k + d \cdot s_k + c \cdot \dot{x}$. This is useful for threshold signatures, where γ_k corresponds to the Lagrange coefficient. Note that $\{\gamma_i\}_{i \in SS}$ (in addition to the nonces) must be included as input to $\hat{\mathsf{h}}_1$ or else there is an attack.

Asymptotically, we would say that the Bischnorr assumption holds with respect to $\mathbb{G}$ if for all PPT adversaries $\mathcal{A}$, we have that $\mathbf{Adv}^{\text{bi-sch}}_{\mathbb{G}}(\mathcal{A})$, which would now be a function of the security parameter, is negligible.

The proof of the following can be found in [14]. For convenience, the statement is asymptotic.

Lemma 10 (OMDL $\Rightarrow$ Bischnorr)**.** *Let* $\hat{\mathsf{h}}_1, \hat{\mathsf{h}}_2$ *be random oracles. The* Bischnorr *assumption is implied by the OMDL assumption with respect to the group* $\mathbb{G}$ *and* $\hat{\mathsf{h}}_1, \hat{\mathsf{h}}_2$*.*

Equipped with this assumption, we are now ready to prove Theorem 9.

Proof (of Theorem 9). Let $\mathcal{A}$ be a PPT adversary attempting to break the TS-UF-0 security of $\mathsf{FROST2}$. We construct a PPT adversary $\mathcal{B}_1$ playing game $\mathbf{G}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_1)$ and thence, from the Schnorr-KoE assumption, obtain an extractor Ext for it. We construct a PPT adversary $\mathcal{B}_2$ playing game $\mathbf{G}^{\text{bi-sch}}_{\mathbb{G}}(\mathcal{B}_2)$ such that whenever $\mathcal{A}$ outputs a valid forgery, either $\mathcal{B}_1$ breaks the Schnorr-KoE assumption or $\mathcal{B}_2$ breaks the Bischnorr assumption. Formally, for security parameter κ, we have

$$\mathbf{Adv}^{\text{ts-uf-0}}_{\mathsf{FROST2}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_1) + \mathbf{Adv}^{\text{bi-sch}}_{\mathbb{G}}(\mathcal{B}_2) + \mathsf{negl}(\kappa)$$

The Reduction $\mathcal{B}_1$: We first define the reduction $\mathcal{B}_1$ against Schnorr-KoE. Let $CS = \{\mathsf{id}_j\}$ be the set of corrupt parties, and let $HS = \{\mathsf{id}_k\}$ be the set of honest parties. Assume that $|CS| = t - 1$ and $|HS| = \mathsf{ns} - (t - 1)$. We will show that when PedPoP outputs public key share $\tilde{X}_k = g^{\bar{x}_k}$ for each honest party $\mathsf{id}_k \in HS$, $\mathcal{B}_1$ returns (α_k, β_k) such that $\tilde{X}_k = \dot{X}^{\alpha_k} g^{\beta_k}$. $\mathcal{B}_1$ is responsible for simulating honest parties in PedPoP (Figure 10) and queries to h_0, h_1, and h_2. $\mathcal{B}_1$ receives as input a group $\mathbb{G}$ and random coins ω. It can query the random oracle RO from Schnorr-KoE. It can also query FSIGNO to receive signatures under $\tilde{\mathsf{h}}_0$ and CHAL on inputs $(X, \bar{R}, \bar{z})$ to challenge the extractor Ext to output a discrete logarithm α for X.

Initialization. $\mathcal{B}_1$ may program $\mathsf{h}_0, \mathsf{h}_1$, and h_2, but not $\tilde{\mathsf{h}}_0$ (because it is part of $\mathcal{B}_1$'s challenge). Let $\mathsf{Q}_{\mathsf{h}_0}$ be the set of h_0 queries and their responses. $\mathcal{B}_1$ first queries FSIGNO and receives $(\dot{X}, \bar{R}, \bar{z})$. $\mathcal{B}_1$ computes α_k for each honest party $\mathsf{id}_k \in HS$ as follows. First, $\mathcal{B}_1$ computes the t Lagrange polynomials $\{L'_k(Z), \{L'_j(Z)\}_{\mathsf{id}_j \in CS}\}$ relating to the set $\mathsf{id}_k \cup CS$. Then, $\mathcal{B}_1$ sets $\alpha_k \leftarrow L'_k(0)^{-1}$. (It will become clear why α_k is computed this way.)

Hash Queries. $\mathcal{B}_1$ handles $\mathcal{A}$'s hash queries throughout the DKG protocol as follows.

h_0: When $\mathcal{A}$ queries h_0 on $(X, X, \bar{R})$, $\mathcal{B}_1$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{h}_0}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_1$ queries $\bar{c} \leftarrow \tilde{\mathsf{h}}_0(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{h}_0}$, and returns $\bar{c}$.

$\underline{\mathsf{h}_1}$: When $\mathcal{A}$ queries h_1 on $(X, lr) = (X, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS})$, $\mathcal{B}_1$ queries $\hat{d} \leftarrow \tilde{\mathsf{h}}_1(X, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS})$ and returns $\hat{d}$.

$\underline{\mathsf{h}_2}$: When $\mathcal{A}$ queries h_2 on (X, M, R), $\mathcal{B}_1$ queries $\hat{c} \leftarrow \tilde{\mathsf{h}}_2(X, M, R)$ and returns $\hat{c}$.

Simulating the DKG. $\mathcal{B}_1$ runs $\mathcal{A}$ on input coins ω and simulates PedPoP as follows. $\mathcal{B}_1$ embeds $\dot{X}$ as the public key of the honest party that the adversary queries first. Let this first honest party be id_τ. $\mathcal{B}_1$ simulates the public view of id_τ but follows the PedPoP protocol for all other honest parties $\{\mathsf{id}_k\}_{k \neq \tau}$ as prescribed. Note that $\mathcal{A}$ can choose the order in which it interacts with honest parties, so $\mathcal{B}_1$ must be able to simulate any of them.

Honest Party id_τ. $\mathcal{B}_1$ is required to output

$$(\bar{R}_\tau, \bar{z}_\tau), \vec{C}_\tau = (A_{\tau,0} = X_{\tau,0}, A_{\tau,1}, ..., A_{\tau,t-1})$$

that are indistinguishable from valid outputs as well as $t-1$ shares $f_\tau(\mathsf{id}_j) = \bar{x}_{\tau,j}$, one to be sent to each corrupt party $\mathsf{id}_j \in CS$. Here, $(\bar{R}_\tau, \bar{z}_\tau)$ is a Schnorr signature proving knowledge of the discrete logarithm of $X_{\tau,0}$, and $\vec{C}_\tau$ is a commitment to the coefficients that represent f_τ. $\mathcal{B}_1$ simulates honest party id_τ as follows.

1. $\mathcal{B}_1$ sets the public key $X_{\tau,0} \leftarrow \dot{X}$.
2. $\mathcal{B}_1$ simulates a verifiable Shamir secret sharing of the discrete logarithm of $\dot{X}$ by performing the following steps.
   (a) $\mathcal{B}_1$ samples $t-1$ random values $\bar{x}_{\tau,j} \leftarrow\!\!\$\ \mathbb{Z}_p$ for $\mathsf{id}_j \in CS$.
   (b) Let f_τ be the polynomial whose constant term is the challenge $f_\tau(0) = \dot{x}$ and for which $f_\tau(\mathsf{id}_j) = \bar{x}_{\tau,j}$ for all $\mathsf{id}_j \in CS$. $\mathcal{B}_1$ computes the t Lagrange polynomials $\{L'_0(Z), \{L'_j(Z)\}_{\mathsf{id}_j \in CS}\}$ relating to the set $0 \cup CS$.
   (c) For $1 \leq i \leq t-1$, $\mathcal{B}_1$ computes
   $$A_{\tau,i} = \dot{X}^{L'_{0,i}} \prod_{\mathsf{id}_j \in CS} g^{\bar{x}_{\tau,j} \cdot L'_{j,i}} \tag{1}$$
   where $L'_{j,i}$ is the i^{th} coefficient of $L'_j(Z) = L'_{j,0} + L'_{j,1}Z + \cdots + L'_{j,t-1}Z^{t-1}$.
   (d) $\mathcal{B}_1$ outputs $(\bar{R}_\tau, \bar{z}_\tau), \vec{C}_\tau = (A_{\tau,0} = X_{\tau,0}, A_{\tau,1}, ..., A_{\tau,t-1})$ for the broadcast round, and then sends shares $\bar{x}_{\tau,j}$ for each $\mathsf{id}_j \in CS$.
3. $\mathcal{B}_1$ simulates private shares $f_\tau(\mathsf{id}_k) = \bar{x}_{\tau,k}$ for honest parties $\mathsf{id}_k \in HS$ by computing α'_k, β'_k such that $g^{\bar{x}_{\tau,k}} = \dot{X}^{\alpha'_k} g^{\beta'_k}$. First, $\mathcal{B}_1$ computes the t Lagrange polynomials $\{L'_k(Z), \{L'_j(Z)\}_{\mathsf{id}_j \in CS}\}$ relating to the set $\mathsf{id}_k \cup CS$. Then, implicitly,
   $$f_\tau(0) = \dot{x} = \bar{x}_{\tau,k} \cdot L'_k(0) + \sum_{\mathsf{id}_j \in CS} \bar{x}_{\tau,j} \cdot L'_j(0)$$
   Solving for $\bar{x}_{\tau,k}$, $\mathcal{B}_1$ sets $\alpha'_k = L'_k(0)^{-1}$ and $\beta'_k = -\alpha'_k \sum_{\mathsf{id}_j \in CS} \bar{x}_{\tau,j} \cdot L'_j(0)$.

All Other Honest Parties. For all other honest parties $\mathsf{id}_k \in HS, k \neq \tau$, $\mathcal{B}_1$ follows the protocol. $\mathcal{B}_1$ samples $f_k(Z) = a_{k,0} + a_{k,1}Z + ... + a_{k,t-1}Z^{t-1} \leftarrow\!\!\$\ \mathbb{Z}_p[Z]$ and sets $A_{k,i} \leftarrow g^{a_{k,i}}$ for all $i \in [0..t-1]$. $\mathcal{B}_1$ provides a proof of possession $(\bar{R}_k, \bar{z}_k)$ of the public key $X_{k,0} = A_{k,0}$ and computes the private shares $\bar{x}_{k,i} = f_k(\mathsf{id}_i)$.

Adversarial Contributions. When $\mathcal{A}$ returns a contribution

$$(\bar{R}_j, \bar{z}_j), \vec{C}_j = (A_{j,0} = X_{j,0}, A_{j,1}, ..., A_{j,t-1})$$

if $(X_{j,0}, \bar{R}_j, \bar{z}_j)$ verifies (i.e., $g^{\bar{z}_j} = \bar{R}_j X_{j,0}^{\tilde{\mathsf{h}}_0(X_{j,0},X_{j,0},\bar{R}_j)}$) and $X_{j,0} \neq \dot{X}$, then $\mathcal{B}_1$ queries $\textsc{Chal}(X_{j,0}, \bar{R}_j, \bar{z}_j)$ from the Schnorr-KoE game.

Complaints. If $\mathcal{A}$ broadcasts a complaint, $\mathcal{B}_1$ reveals the relevant $\bar{x}_{k,j}$. If $\mathcal{A}$ does not send verifying $\bar{x}_{j,k}$ to party $\mathsf{id}_k \in HS$, then $\mathcal{B}_1$ broadcasts a complaint. If $\bar{x}_{j,k}$ fails to satisfy the equation, or should $\mathcal{A}$ not broadcast a share at all, then id_j is disqualified.

DKG Termination. When PedPoP terminates, the output is the joint public key $\tilde{X} = \prod_{i=0}^{\mathsf{ns}} X_{i,0}$. $\mathcal{B}_1$ simulates private shares $\bar{x}_k$ for honest parties $\mathsf{id}_k \in HS$ by computing α_k, β_k such that $\tilde{X}_k = g^{\bar{x}_k} = \dot{X}^{\alpha_k} g^{\beta_k}$. Implicitly, $\bar{x}_k = \bar{x}_{\tau,k} + \sum_{i=1,i\neq\tau}^{\mathsf{ns}} \bar{x}_{i,k}$ and $\bar{x}_{\tau,k} = \dot{x} \cdot \alpha'_k + \beta'_k$ from Step 3 above, so $\alpha_k = \alpha'_k$ and $\beta_k = \beta'_k + \sum_{i=1,i\neq\tau}^{\mathsf{ns}} \bar{x}_{i,k}$. $\mathcal{B}_1$ returns $\{a_k\}_{\mathsf{id}_k \in HS, k\neq\tau}, \{(\alpha_k, \beta_k)\}_{\mathsf{id}_k \in HS}$.

We now argue that: (1) $\mathcal{A}$ cannot distinguish between a real run of the DKG protocol and its interaction with $\mathcal{B}_1$; and (2) $\mathsf{Ext}(\mathbb{G}, \omega, Q_{\text{FSignO}}, \mathsf{Q}_{\tilde{\mathsf{h}}_0})$ outputs $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ whenever $\mathcal{B}_1$ queries $\textsc{Chal}(X_{j,0}, \bar{R}_j, \bar{z}_j)$.

(1) See that $\mathcal{B}_1$'s simulation of PedPoP is perfect, as performing validation of each player's share (Step 4 in Figure 10) holds, and by Equation 1, interpolation in the exponent correctly evaluates to the challenge $\dot{X}$.

(2) See that $\mathsf{h}_0(X_{j,0}, X_{j,0}, \bar{R}_j) = \tilde{\mathsf{h}}_0(X_{j,0}, X_{j,0}, \bar{R}_j)$ unless $(X_{j,0}, \bar{R}_j) = (\dot{X}, \bar{R}_\tau)$. The latter happens only if $X_{j,0} = X_{\tau,0}$, but in this case PedPoP will not terminate. We thus have that $(X_{j,0}, \bar{R}_j, \bar{z}_j)$ is a verifying signature under $\tilde{\mathsf{h}}_0$ and either Ext succeeds, or $\mathcal{B}_1$ breaks the Schnorr-KoE assumption. Therefore, the probability of the event occurring where Ext fails to outputs $a_{j,0}$ is bounded by $\mathbf{Adv}^{\mathsf{sch\text{-}koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_1)$.

<u>**The Reduction $\mathcal{B}_2$:**</u> We next define the reduction $\mathcal{B}_2$ against Bischnorr. We will show that when PedPoP outputs the joint public key $\tilde{X}$, $\mathcal{B}_2$ returns y such that $\tilde{X} = \dot{X} g^y$. Together with the (α_k, β_k) returned by $\mathcal{B}_1$ such that $\tilde{X}_k = \dot{X}^{\alpha_k} g^{\beta_k}$, this representation allows $\mathcal{B}_2$ to simulate $\mathsf{FROST2}$ signing under each $\tilde{X}_k$. $\mathcal{B}_2$ is responsible for simulating honest parties during signing and queries to h_0, h_1, and h_2. $\mathcal{B}_2$ receives as input a group $\mathbb{G}$ and a challenge public key $\dot{X}$. It can query RO, $\textsc{BinonceO}$, $\textsc{BisignO}$ from the Bischnorr game.

<u>**Initialization.**</u> $\mathcal{B}_2$ may program $\mathsf{h}_0, \mathsf{h}_1$, and h_2, but not $\hat{\mathsf{h}}_1$ or $\hat{\mathsf{h}}_2$ (because they are part of $\mathcal{B}_2$'s challenge). Let Q_{PPO} be the set of PPO queries and responses in the pre-processing round, and let Q_{PSignO} be the set of PSignO queries and responses in the signing round.

<u>**DKG Extraction.**</u> $\mathcal{B}_2$ first simulates a Schnorr proof of possession of $\dot{X}$ as follows. $\mathcal{B}_2$ samples $\bar{c}_\tau, \bar{z}_\tau \leftarrow\!\!\$\ \mathbb{Z}_p$, computes $\bar{R}_\tau \leftarrow g^{\bar{z}_\tau} \dot{X}^{-\bar{c}_\tau}$, and appends $(\dot{X}, \dot{X}, \bar{R}_\tau, \bar{c}_\tau)$ to $\mathsf{Q}_{\mathsf{h}_0}$. Then, $\mathcal{B}_2$ runs

$$\{a_{k,0}\}_{\mathsf{id}_k \in HS, k\neq\tau}, \{(\alpha_k, \beta_k)\}_{\mathsf{id}_k \in HS} \leftarrow \mathcal{B}_1(\mathbb{G}; \omega)$$

on coins ω. $\mathcal{B}_2$ handles $\mathcal{B}_1$'s queries as follows. When $\mathcal{B}_1$ queries $\tilde{\mathsf{h}}_0$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\tilde{\mathsf{h}}_0}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{h}}_0(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\tilde{\mathsf{h}}_0}$, and returns $\bar{c}$. When $\mathcal{B}_1$ queries $\tilde{\mathsf{h}}_1, \tilde{\mathsf{h}}_2$, $\mathcal{B}_2$ handles them the same way it handles $\mathcal{A}$'s $\mathsf{h}_1, \mathsf{h}_2$ queries, described below. The first time $\mathcal{B}_1$ queries its FSignO oracle, $\mathcal{B}_2$ returns $(\dot{X}, \bar{R}_\tau, \bar{z}_\tau)$. When

$\mathcal{B}_1$ queries $\text{CHAL}(X_{j,0}, \bar{R}_j, \bar{z}_j)$, $\mathcal{B}_2$ runs $a_{j,0} \leftarrow \mathsf{Ext}(\mathbb{G}, \omega, Q_{\text{FSIGNO}}, \mathsf{Q}_{\tilde{\mathsf{h}}_0})$ to obtain $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ and aborts otherwise. Then $y = \sum_{i=1, i \neq \tau}^{\mathsf{ns}} a_{i,0}$ such that $\tilde{X} = \dot{X} g^y$.

$\mathcal{A}$'s Hash Queries. $\mathcal{B}_2$ handles $\mathcal{A}$'s hash queries throughout the signing protocol as follows.

h_0: When $\mathcal{A}$ queries h_0 on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{h}_0}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{h}}_0(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{h}_0}$, and returns $\bar{c}$. Note that $\mathcal{B}_1$ and $\mathcal{B}_2$ share the state of $\mathsf{Q}_{\mathsf{h}_0}$.

h_1: When $\mathcal{A}$ queries h_1 on $(X, lr) = (X, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS})$, $\mathcal{B}_2$ checks whether $(X, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS}, \hat{M}, \hat{d}) \in \mathsf{Q}_{\mathsf{h}_1}$ and, if so, returns $\hat{d}$. Else, $\mathcal{B}_2$ checks whether there exists some $k' \in SS$ such that $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\text{PPO}}$. If not, $\mathcal{B}_2$ samples a random message $\hat{M}$ and a random value $\hat{d}$, appends $(X, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS}, \hat{M}, \hat{d})$ to $\mathsf{Q}_{\mathsf{h}_1}$, and returns $\hat{d}$.

If there does exist some $k' \in SS$ such that $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\text{PPO}}$, $\mathcal{B}_2$ computes the Lagrange coefficients $\{\lambda_i\}_{i \in SS}$, where $\lambda_i = L_i(0)$ and $\{L_i(Z)\}_{i \in SS}$ are the Lagrange polynomials relating to the set $\{\mathsf{id}_i\}_{i \in SS}$. $\mathcal{B}_2$ sets $\gamma_k = \lambda_k \cdot \alpha_k$ for all $\mathsf{id}_k \in HS$ and $\gamma_j = \lambda_j$ for all $\mathsf{id}_j \in CS$ in the set SS. $\mathcal{B}_2$ then samples a random message $\hat{M}$ (to prevent trivial collisions), queries $\hat{d} \leftarrow \hat{\mathsf{h}}_1(\dot{X}, \hat{M}, \{(\gamma_i, R_i, S_i)\}_{i \in SS})$, and appends $(X, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS}, \hat{M}, \hat{d})$ to $\mathsf{Q}_{\mathsf{h}_1}$. $\mathcal{B}_2$ computes $\hat{R} = \prod_{i \in SS} R_i S_i^{\hat{d}}$ and checks if there exists a record $(X, M, \hat{R}, \hat{M}, \hat{c}) \in \mathsf{Q}_{\mathsf{h}_2}$. If so, $\mathcal{B}_2$ aborts. Else, $\mathcal{B}_2$ queries $\hat{c} \leftarrow \hat{\mathsf{h}}_2(\dot{X}, \hat{M}, \hat{R})$ and appends $(\tilde{X}, M, \hat{R}, \hat{M}, \hat{c})$ to $\mathsf{Q}_{\mathsf{h}_2}$. Finally, $\mathcal{B}_2$ returns $\hat{d}$.

h_2: When $\mathcal{A}$ queries h_2 on (X, M, R), $\mathcal{B}_2$ checks whether $(X, M, R, \hat{M}, \hat{c}) \in \mathsf{Q}_{\mathsf{h}_2}$ and, if so, returns $\hat{c}$. Else, $\mathcal{B}_2$ samples a random message $\hat{M}$, queries $\hat{c} \leftarrow \hat{\mathsf{h}}_2(\dot{X}, \hat{M}, R)$, appends $(X, M, R, \hat{M}, \hat{c})$ to $\mathsf{Q}_{\mathsf{h}_2}$, and returns $\hat{c}$.

Simulating FROST2 Signing. After $\mathcal{B}_1$ completes the simulation of PedPoP, $\mathcal{B}_2$ then simulates honest parties in the FROST2 signing protocol.

Pre-Processing Round. When $\mathcal{A}$ queries PPO on $\mathsf{id}_k \in HS$, $\mathcal{B}_2$ queries BINONCEO to get (R_k, S_k), appends $(\mathsf{id}_k, R_k, S_k)$ to Q_{PPO}, and returns (R_k, S_k).

Signing Round. When $\mathcal{A}$ queries PSIGNO on $(k', lr) = (k', M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS})$, $\mathcal{B}_2$ first checks whether $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\text{PPO}}$ and, if not, returns $\perp$. Then, $\mathcal{B}$ checks whether $(R_{k'}, S_{k'}) \in Q_{\text{PSIGNO}}$ and, if so, returns $\perp$.

If all checks pass, $\mathcal{B}_2$ internally queries $\hat{\mathsf{h}}_1$ on $(\tilde{X}, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS})$ to get $\hat{d}'$ and looks up $\hat{M}'$ such that $(\tilde{X}, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS}), \hat{M}', \hat{d}') \in \mathsf{Q}_{\mathsf{h}_1}$. $\mathcal{B}_2$ computes $\hat{R}' = \prod_{i \in SS} R_i S_i^{\hat{d}'}$ and internally queries $\hat{\mathsf{h}}_2$ on $(\tilde{X}, M, \hat{R}')$ to get $\hat{c}'$.

Next, $\mathcal{B}_2$ computes the Lagrange coefficients $\{\lambda_i\}_{i \in SS}$, where $\lambda_i = L_i(0)$ and $\{L_i(Z)\}_{i \in SS}$ are the Lagrange polynomials relating to the set $\{\mathsf{id}_i\}_{i \in SS}$. $\mathcal{B}_2$ sets $\gamma_k = \lambda_k \cdot \alpha_k$ for all $\mathsf{id}_k \in HS$ and $\gamma_j = \lambda_j$ for all $\mathsf{id}_j \in CS$ in the set SS. Then, $\mathcal{B}_2$ queries BISIGNO on $(k', \hat{M}', \{(\gamma_i, R_i, S_i)\}_{i \in SS})$ to get $z_{k'}$. Finally, $\mathcal{B}_2$ computes

$$\tilde{z}_{k'} = z_{k'} + \hat{c}' \cdot \lambda_{k'} \cdot \beta_{k'} \tag{2}$$

For $\mathcal{A}$'s query to PSIGNO, $\mathcal{B}_2$ returns $\tilde{z}_{k'}$.

Output. When $\mathcal{A}$ returns $(\tilde{X}, M^*, \mathit{sig}^*)$ such that $\mathit{sig}^* = (\tilde{R}^*, z^*)$ and $\mathsf{Vf}(\tilde{X}, M^*, \mathit{sig}^*) = 1$, $\mathcal{B}_2$ computes its output as follows. $\mathcal{B}_2$ looks up $\hat{M}^*$ such that $(\tilde{X}, M^*, \tilde{R}^*, \hat{M}^*, \hat{c}^*) \in \mathsf{Q}_{\mathsf{h}_2}$ and outputs $(\hat{M}^*, \tilde{R}^*, z^* - \hat{c}^* \cdot y)$.

To complete the proof, we must argue that: (1) $\mathcal{B}_2$ only aborts with negligible probability; (2) $\mathcal{A}$ cannot distinguish between a real run of the protocol and its interaction with $\mathcal{B}_2$; and (3) whenever $\mathcal{A}$ succeeds, $\mathcal{B}_2$ succeeds.

(1) $\mathcal{B}_2$ aborts if Ext fails to return $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ for some j. This happens with maximum probability $\mathbf{Adv}^{\mathsf{sch\text{-}koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_1)$.

$\mathcal{B}_2$ aborts if $\mathcal{A}$ queries h_2 on $(\tilde{X}, M, \prod_{i \in SS} R_i S_i^{\hat{d}})$ before having first queried h_1 on $(\tilde{X}, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS})$. This requires $\mathcal{A}$ to have guessed $\hat{d}$ ahead of time, which occurs with negligible probability q_H/p.

(2) As long as $\mathcal{B}_2$ does not abort, $\mathcal{B}_2$ is able to simulate the appropriate responses to $\mathcal{A}$'s oracle queries so that $\mathcal{A}$ cannot distinguish between a real run of the protocol and its interaction with $\mathcal{B}_2$.

Indeed, $\mathcal{B}_1$'s simulation of PedPoP is perfect.

When $\mathcal{A}$ queries h_2 on (X, M, R), $\mathcal{B}_2$ queries $\hat{c} \leftarrow \hat{\mathsf{h}}_2(\dot{X}, \hat{M}, R)$ on a random message $\hat{M}$. The random message prevents trivial collisions; for example, if $\mathcal{A}$ were to query h_2 on (X, M, R) and (X', M, R), where $X' \neq X$, $\mathcal{A}$ would receive the same value $c \leftarrow \hat{\mathsf{h}}_2(\dot{X}, M, R)$ for both and would know it was operating inside a reduction. Random messages ensure that the outputs are random, so $\mathcal{A}$'s view is correct. $\mathcal{B}_2$ also ensures that $\mathcal{A}$ receives h_1 values that are consistent with h_2 queries.

After the signing rounds have been completed, $\mathcal{A}$ may verify the signature share $\tilde{z}_{k'}$ on M as follows. $\mathcal{A}$ checks if

$$g^{\tilde{z}_{k'}} = R_{k'} S_{k'}^{\mathsf{h}_1(\tilde{X}, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS})} \tilde{X}_{k'}^{\lambda_{k'} \mathsf{h}_2(\tilde{X}, M, \prod_{i \in SS} R_i S_i^{\mathsf{h}_1(\tilde{X}, M, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in SS})})} \tag{3}$$

When $\mathcal{B}_2$ queried BISIGNO on $(k', \hat{M}', \{(\gamma_i, R_i, S_i)\}_{i \in SS})$ in the Signing Round, the signature share $z_{k'}$ was computed such that

$$g^{z_{k'}} = R_{k'} S_{k'}^{\hat{\mathsf{h}}_1(\dot{X}, \hat{M}', \{(\gamma_i, R_i, S_i)\}_{i \in SS})} \dot{X}^{\gamma_{k'} \hat{\mathsf{h}}_2(\dot{X}, \hat{M}', \prod_{i \in SS} R_i S_i^{\hat{\mathsf{h}}_1(\dot{X}, \hat{M}', \{(\gamma_i, R_i, S_i)\}_{i \in SS})})}$$

$\mathcal{B}$ computed the signature share $\tilde{z}_{k'}$ (Equation 2) as

$$\begin{aligned} \tilde{z}_{k'} = z_{k'} + \hat{c}' \cdot \lambda_{k'} \cdot \beta_{k'} &= r_{k'} + d \cdot s_{k'} + \hat{c}' \cdot \gamma_{k'} \cdot \dot{x} + \hat{c}' \cdot \lambda_{k'} \cdot \beta_{k'} \\ &= r_{k'} + d \cdot s_{k'} + \hat{c}' \cdot \lambda_{k'} (\alpha_{k'} \cdot \dot{x} + \beta_{k'}) \end{aligned}$$

where $\hat{c}' = \hat{\mathsf{h}}_2(\dot{X}, \hat{M}', \prod_{i \in SS} R_i S_i^{\hat{\mathsf{h}}_1(\dot{X}, \hat{M}', \{(\gamma_i, R_i, S_i)\}_{i \in SS})})$. Thus, $\tilde{z}_{k'}$ satisfies

$$g^{\tilde{z}_{k'}} = R_{k'} S_{k'}^{\hat{\mathsf{h}}_1(\dot{X}, \hat{M}', \{(\gamma_i, R_i, S_i)\}_{i \in SS})} \tilde{X}_{k'}^{\lambda_{k'} \hat{\mathsf{h}}_2(\dot{X}, \hat{M}', \prod_{i \in SS} R_i S_i^{\hat{\mathsf{h}}_1(\dot{X}, \hat{M}', \{(\gamma_i, R_i, S_i)\}_{i \in SS})})} \tag{4}$$

$\mathcal{B}_2$ has programmed the hash values in Equations 3 and 4 to be equal and therefore simulates $\tilde{z}_{k'}$ correctly.

(3) $\mathcal{A}$'s forgery satisfies $\mathsf{Vf}(\tilde{X}, M^*, \mathit{sig}^*) = 1$, which implies:

$$\begin{aligned} g^{z^*} &= \tilde{R}^* (\tilde{X})^{\mathsf{h}_2(\tilde{X}, M^*, \tilde{R}^*)} = \tilde{R}^* (\dot{X} g^y)^{\mathsf{h}_2(\tilde{X}, M^*, \tilde{R}^*)} \\ g^{z^* - \mathsf{h}_2(\tilde{X}, M^*, \tilde{R}^*) \cdot y} &= \tilde{R}^* \dot{X}^{\mathsf{h}_2(\tilde{X}, M^*, \tilde{R}^*)} \end{aligned}$$

At some point, $\mathcal{A}$ queried h_2 on $(\tilde{X}, M^*, \tilde{R}^*)$ and received one of two values: (1) $\hat{c}^* \leftarrow \hat{\mathsf{h}}_2(\dot{X}, \hat{M}^*, \prod_{i \in SS^*} R_i^*(S_i^*)^{\hat{d}^*})$ related to a query $\mathcal{A}$ made to h_1 on $(M^*, \{(\mathsf{id}_i^*, R_i^*, S_i^*)\}_{i \in SS^*})$, where it received $\hat{d}^* \leftarrow \hat{\mathsf{h}}_1(\dot{X}, \hat{M}^*, (\gamma_i^*, R_i^*, S_i^*)_{i \in SS^*})$, or (2) $\hat{c}^* \leftarrow \hat{\mathsf{h}}_2(\dot{X}, \hat{M}^*, \tilde{R}^*)$ without having queried h_1 first. In either case, $\mathcal{B}_2$ has a record $(\tilde{X}, M^*, \tilde{R}^*, \hat{M}^*, \hat{c}^*) \in \mathsf{Q}_{\mathsf{h}_2}$ such that $\hat{c}^* \leftarrow \hat{\mathsf{h}}_2(\dot{X}, \hat{M}^*, \tilde{R}^*)$. (Note that $\mathcal{B}_2$ can check which case occurred by looking for $\hat{M}^*$ in its $\mathsf{Q}_{\mathsf{h}_1}$ records.) Thus, $\mathcal{A}$'s forgery satisfies

$$g^{z^* - \hat{\mathsf{h}}_2(\dot{X}, \hat{M}^*, \tilde{R}^*) \cdot y} = \tilde{R}^* \dot{X}^{\hat{\mathsf{h}}_2(\dot{X}, \hat{M}^*, \tilde{R}^*)}$$

and $\mathcal{B}_2$'s output $(\hat{M}^*, \tilde{R}^*, z^* - \hat{c}^* \cdot y)$ under $\dot{X}$ is correct.

Acknowledgements

Bellare was supported in part by NSF grant CNS-2154272 and a gift from Microsoft. Elizabeth Crites was supported by Input Output through their funding of the Edinburgh Blockchain Technology Lab. Tessaro and Zhu are supported in part by NSF grants CNS-1930117 (CAREER), CNS-2026774, CNS-2154174, a JP Morgan Faculty Award, a CISCO Faculty Award, and a gift from Microsoft.

References

1. M. Bellare, W. Dai, and L. Li. The local forking lemma and its application to deterministic encryption. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 607–636. Springer, Heidelberg, Dec. 2019. 20, 21
2. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003. 3, 19
3. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, and S. De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, Oct. / Nov. 2006. 19
4. M. Bellare and A. Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 273–289. Springer, Heidelberg, Aug. 2004. 24
5. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993. 3
6. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. 8
7. M. Bellare, S. Tessaro, and C. Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Report 2022/833, 2022. https://eprint.iacr.org/2022/833. 4, 7, 14, 16, 19, 21
8. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, Jan. 2003. 1, 2, 3, 4, 12

9. D. Boneh, R. Gennaro, and S. Goldfeder. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In T. Lange and O. Dunkelman, editors, *LATINCRYPT 2017*, volume 11368 of *LNCS*, pages 352–377. Springer, Heidelberg, Sept. 2017. 1
10. D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, Aug. 2018. 4
11. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, Dec. 2001. 2
12. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, Sept. 2004. 2, 3
13. R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, Nov. 2020. 1, 7
14. E. Crites, C. Komlo, and M. Maller. How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375, 2021. https://eprint.iacr.org/2021/1375. 7, 25, 27
15. I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In J. Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Heidelberg, Aug. 1992. 24
16. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994. 1
17. Y. Desmedt. Society and group oriented cryptography: A new concept. In C. Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Heidelberg, Aug. 1988. 1
18. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, Aug. 1990. 1
19. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437. IEEE Computer Society Press, Oct. 1987. 22
20. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, Aug. 2018. 24
21. R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 1179–1194. ACM Press, Oct. 2018. 1
22. R. Gennaro, S. Goldfeder, and A. Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In M. Manulis, A.-R. Sadeghi, and S. Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 156–174. Springer, Heidelberg, June 2016. 1
23. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In U. M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 354–371. Springer, Heidelberg, May 1996. 1, 4, 12
24. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure applications of Pedersen's distributed key generation protocol. In M. Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 373–390. Springer, Heidelberg, Apr. 2003. 1, 2, 22

25. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, Jan. 2007. 1, 4, 12
26. R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. Robust and efficient sharing of RSA functions. *Journal of Cryptology*, 13(2):273–300, Mar. 2000. 1, 4
27. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, Apr. 1988. 2, 3, 12
28. J. Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. https://eprint.iacr.org/2021/339. 4, 7
29. J. Katz and M. Yung. Threshold cryptosystems based on factoring. In Y. Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 192–205. Springer, Heidelberg, Dec. 2002. 4
30. C. Komlo and I. Goldberg. Frost: flexible round-optimized schnorr threshold signatures. In *International Conference on Selected Areas in Cryptography*, pages 34–65. Springer, 2020. 1, 2, 3, 5, 6, 22
31. C. Komlo and I. Goldberg. FROST: Flexible round-optimized schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852, 2020. https://eprint.iacr.org/2020/852. 16
32. C. Komlo, I. Goldberg, and T. Wilson-Brown. Two-Round Threshold Signatures with FROST. Internet-Draft draft-irtf-cfrg-frost-01, Internet Engineering Task Force, Aug. 2021. Work in Progress. 1
33. B. Libert, M. Joye, and M. Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In M. M. Halldórsson and S. Dolev, editors, *33rd ACM PODC*, pages 303–312. ACM, July 2014. 4
34. Y. Lindell, A. Nof, and S. Ranellucci. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. Cryptology ePrint Archive, Report 2018/987, 2018. https://eprint.iacr.org/2018/987. 1
35. National Institute of Standards and Technology. Multi-Party Threshold Cryptography, 2018–Present. https://csrc.nist.gov/Projects/threshold-cryptography. 1
36. A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979. 22
37. V. Shoup. Practical threshold signatures. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000. 1, 4
38. V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In K. Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 1–16. Springer, Heidelberg, May / June 1998. 24
39. D. R. Stinson and R. Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In V. Varadharajan and Y. Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Heidelberg, July 2001. 1
40. H. Wee. Threshold and revocation cryptosystems via extractable hash proofs. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 589–609. Springer, Heidelberg, May 2011. 4