FROST: Flexible Round-Optimized Schnorr Threshold Signatures

(draft under review)

Chelsea Komlo
University of Waterloo, Zcash Foundation

Ian Goldberg
University of Waterloo

January 20, 2020

Abstract

Unlike signatures in a single-party setting, threshold signatures require cooperation among a threshold number of signers each holding a share of a common private key. Consequently, generating signatures in a threshold setting imposes overhead due to network rounds among signers, proving costly when secret shares are stored on network-limited devices or when coordination occurs over unreliable networks. In this work, we present FROST, a Flexible Round-Optimized Schnorr Threshold signature scheme that improves upon the state of the art to reduce network overhead during signing operations. We introduce three variants of signing operations in FROST. We begin with two variants that are limited in concurrency but efficient in per-user computation; the first reduces the number of messages participants send and receive to two in total, and the second variant is a further optimization to a single-round signing protocol with a batched non-interactive preprocessing stage. We next present a third variant that does not restrict concurrency of signing operations but is more costly in per-signature computation. Across all variants, FROST achieves its efficiency improvements by allowing the protocol to abort in the presence of a misbehaving participant (who is then identified and excluded from future operations)—a reasonable model for practical deployment scenarios. We present two use cases of threshold signatures demonstrating the practicality of this tradeoff to real-world implementations, and prove FROST is as secure as Schnorr's signature scheme in a single-party setting.

1 Introduction

Threshold signature schemes are a cryptographic primitive to facilitate joint ownership over a private key by a set of participants, such that a threshold number of participants must cooperate to issue a signature that can be verified by a single public key. Thresh-

old signatures are useful across a range of settings that require a distributed root of trust among a set of equally trusted parties.

Similarly to signing operations in a single-party setting, some implementations of threshold signature schemes require performing signing operations at scale and under heavy load. For example, threshold signatures can be used by a set of signers to authenticate financial transactions in cryptocurrencies [10], or to sign a network consensus produced by a set of trusted authorities [13]. In both of these examples, as the number of signing parties or signing operations increases, the number of communication rounds between participants required to produce the joint signature becomes a performance bottleneck, in addition to the increased load experienced by each signer. This problem is further exacerbated when signers utilize network-limited devices or unreliable networks for transmission, or protocols that wish to allow signers to participate in signing operations asynchronously. As such, optimizing the network overhead of signing operations is highly beneficial to real-world applications of threshold signatures.

Today in the literature, the best threshold signature schemes are those that rely on pairing-based cryptography [3, 2], and can perform signing operations in a single round among participants. However, relying on pairing-based signature schemes is undesirable for some implementations in practice, such as those that do not wish to introduce a new cryptographic assumption, or that wish to maintain backwards compatibility with an existing signature scheme such as Schnorr signatures. Surprisingly, today's best non-pairing-based threshold schemes require multiple rounds of interaction during signing operations. The best threshold signature constructions that produce Schnorr signatures [20, 8] require at least three rounds of communication during signing operations: two rounds to generate a random nonce (where participants send values to every other participant in each round) and one round to publish and aggregate each participant's signature share. Notably, these Schnorr-based schemes have assumed *robustness* as a critical protocol feature, such that if any participant misbehaves, honest participants can detect this misbehaviour, disqualify the misbehaving participant, and continue the protocol to produce a signature, so long as at least the threshold number of honest parties remain.

In this work, we present FROST, a Flexible Round-Optimized Schnorr Threshold signature scheme[1] that addresses the need for efficient threshold signing operations in real-world settings. We present three variants of FROST, all of which improve upon the state of the art to reduce network rounds among participants. We begin by presenting two variants that are limited in concurrency but efficient in per-user computation; the first is a two-round variant where participants send and receive two messages in total, and the second is an optimization of the first, such that signing operations can be performed in a single (non-broadcast) round with a batched non-interactive pre-processing stage. We next present a third variant of FROST that requires more per-user computation but does not limit the parallelism of signing operations. In lieu of robustness, FROST achieves improved efficiency in the optimistic case that no participant misbehaves. However, in the case where a misbehaving participant contributes malformed values during the protocol, honest parties can identify and exclude the misbehaving

[1] Signatures generated using the FROST protocol can also be referred to as "FROSTy signatures."

participant, and re-run the protocol. We present two use cases of threshold signatures demonstrating the practicality of this tradeoff to real-world settings.

Contributions. In this work, we present the following contributions.

- We review related Schnorr-based threshold signature schemes and present a detailed analysis of their performance and designs.
- We discuss two use cases of threshold signatures to demonstrate practical tradeoffs between robustness and efficiency.
- We present FROST, a Flexible Round-Optimized Schnorr Threshold signature scheme, and define three variants of the signing protocol, all improving upon the state of the art to minimize network rounds during signing operations. The first two variants provide limited concurrency but minimize per-user computation during signing; the first variant is a two-round protocol where participants send and receive two messages in total, and the second is an optimization to a single-round variant with a batched non-interactive preprocessing stage. We introduce a third variant that does not restrict concurrency but requires additional per-user computation.
- We present proofs of security and correctness for FROST, building upon proofs of security for prior related schemes.

Organization. We present background information important to understanding our work in Section 2. In Section 3, we outline two use cases of threshold signatures demonstrating that the strongest notions of robustness are not always required in realistic deployments of threshold schemes. In Section 4 we give an overview of related threshold Schnorr signature constructions in the literature. In Section 5 we review notation and security assumptions maintained for our work. In Section 6 we introduce FROST and describe its protocols in detail. In Section 7 we give proofs of security and correctness for FROST. We discuss future research directions for FROST in Section 8, and conclude in Section 9.

2 Background

2.1 Threshold Schemes

Cryptographic protocols called (t, n)-*threshold schemes* allow a set of n participants to share a secret s, such that any t out of the n participants are required to cooperate in order to recover s, but any subset of fewer than t participants cannot recover any information about the secret.

Shamir Secret Sharing. Many threshold schemes build upon Shamir secret sharing [19], a (t, n)-threshold scheme that relies on Lagrange interpolation to recover a secret. In Shamir secret sharing, a trusted central dealer distributes a secret s to n participants in such a way that any cooperating subset of t participants can recover the secret. To distribute this secret, the dealer first selects $t-1$ coefficients $a_1, \ldots, a_{t-1}$ at random, and uses the randomly selected values as coefficients to define a polynomial $f(x) = s + \sum_{i=1}^{t-1} a_i x^i$ of degree $t-1$ where $f(0) = s$. The secret shares for each participant P_i are subsequently $(i, f(i))$, which the dealer is trusted to distribute honestly

to each participant $P_1, \ldots, P_n$. To reconstruct the secret, at least t participants perform Lagrange interpolation to reconstruct the polynomial and thus find the value $s = f(0)$. However, no group of fewer than t participants can reconstruct the secret, as at least t points are required to reconstruct a polynomial of degree $t - 1$.

Verifiable Secret Sharing. Feldman's Verifiable Secret Sharing (VSS) Scheme [6] builds upon Shamir secret sharing, adding a verification step to demonstrate the consistency of a participant's share with a public *commitment* that is assumed to be correctly visible to all participants. To validate that a share is well formed, each participant validates their share using this commitment. If the validation fails, the participant can issue a *complaint* against the dealer, and take actions such as broadcasting this complaint to all other participants. FROST similarly uses this technique as well.

The commitment produced in Feldman's scheme is as follows. As before in Shamir secret sharing, a dealer samples $t - 1$ random values $(a_1, \ldots, a_{t-1})$, and uses these values as coefficients to define a polynomial f_i of degree $t - 1$ such that $f(0) = s$. However, along with distributing the private share $(i, f(i))$ to each participant P_i, the dealer also distributes the public commitment

$$\vec{C} = \langle \phi_0, \ldots, \phi_{t-1} \rangle, \text{ where } \phi_0 = g^s \text{ and } \phi_j = g^{a_j}$$

Note that in a distributed setting, each participant P_i must be sure to have the same view of $\vec{C}$ as all other participants. In practice, implementations guarantee consistency of participants' views by using techniques such as posting commitments to a centralized server that is trusted to provide a single view to all participants, or adding another protocol round where participants compare their received commitment values to ensure they are identical.

2.2 Threshold Signature Schemes

Threshold signature schemes leverage the (t, n) security properties of threshold schemes, but allow participants to produce signatures over a message using their secret shares such that anyone can validate the integrity of the message, *without* ever reconstructing the secret. In threshold signature schemes, the secret key s is distributed among the n participants, while a single public key Y is used to represent the group. Signatures can be generated by a threshold of t cooperating signers.

For our work, we require the resulting signature produced by the threshold signature scheme to be valid under the Schnorr signature scheme [17]. We introduce Schnorr signatures in Section 2.4.

Because threshold signature schemes ensure that no participant (or indeed any group of fewer than t participants) ever learns the secret key s, the generation of s and distribution of shares $s_1, \ldots, s_n$ often require generating shares using a less-trusted method than relying on a central dealer. Instead, these schemes (including FROST) make use of a Distributed Key Generation (DKG) protocol, which we describe next.

2.3 Distributed Key Generation

While some threshold schemes such as Shamir secret sharing rely on a trusted dealer to generate and distribute secret shares to all participants, not all threat models can

allow for such a high degree of trust in a single individual. Distributed Key Generation (DKG) supports such threat models by enabling every participant to contribute equally to the generation of the shared secret. At the end of running the protocol, all participants share a joint public key Y, but each participant holds only a share s_i of the corresponding secret s such that no set of participants smaller than the threshold knows s.

Pedersen [15] presents a two-round DKG where each participant acts as the central dealer of Feldman's VSS [6] protocol, resulting in n parallel executions of the protocol. Consequently, this protocol requires two rounds of communication between all participants; after each participant selects a secret x_i, they first broadcast a commitment to x_i to all other participants, and then send all other participants a secret share of x_i.

Gennaro et al. [9] demonstrate a weakness of Pedersen's DKG [15] such that a misbehaving participant can bias the distribution of the resulting shared secret by issuing complaints against a participant *after* seeing the shares issued to them by this participant, thereby disqualifying them from contributing to the key generation. To address this issue, the authors define a modification to Pedersen's DKG to utilize both Feldman's VSS as well as a verifiable secret sharing scheme by Pedersen [16] resulting in a three-round protocol. To prevent adversaries from adaptively disqualifying participants based on their input, the authors add an additional "commitment round", such that the value of the resulting secret is determined after participants perform this commitment round (before having revealed their inputs).

In a later work, Gennaro et al. [8] prove that Pedersen's DKG as originally described [15] is *secure enough* in certain contexts, as the resulting secret is sufficiently random despite the chance for bias from a misbehaving participant adaptively selecting their input after seeing inputs from other participants. However, Pedersen's DKG requires larger security parameters to achieve the same level of security as the modified variant by Gennaro et al. [9] that requires the additional commitment round. In short, the two-round Pedersen's DKG [15] requires a larger group to be as secure as the three-round DKG presented by Gennaro et al. [9].

2.4 Schnorr Signatures

Often, it is desirable for signatures produced by threshold signing operations to be indistinguishable from signatures produced by a single participant, consequently remaining backwards compatible with existing systems, and also preventing a privacy leak of the identities of the individual signers. For our work, we require signatures produced by FROST signing operations to be indistinguishable from Schnorr signatures, and thus verifiable using the standard Schnorr verification operations. To this end, we now describe Schnorr signing and verification operations [17] in a single-signer setting.

Let $\mathbb{G}$ be a group with prime order q and generator g, and let H be a cryptographic hash function mapping to $\mathbb{Z}_q^*$. A Schnorr signature is generated over a message m by the following steps:

1. Sample a random nonce $k \in_R \mathbb{Z}_q$; compute the commitment $R \leftarrow g^k \in \mathbb{G}$
2. Compute the challenge $c = H(m, R)$
3. Using the secret key s, compute the response $z = k + s \cdot c \in \mathbb{Z}_q$

4. Define the signature over m to be $\sigma = (z, c)$

Validating the integrity of m using the public key $Y = g^s$ and the signature σ is performed as follows:

1. Parse σ as (z, c).
2. Compute $R' = g^z \cdot Y^{-c}$
3. Compute $z' = H(m, R')$
4. Output 1 if $z = z'$ to indicate success; otherwise, output 0.

Schnorr signatures are simply the standard Σ-protocol proof of knowledge of the discrete logarithm of Y, made non-interactive (and bound to the message m) with the Fiat-Shamir transform.

2.5 Additive Secret Sharing

Similarly to the single-party setting described above, FROST requires generating a random nonce k for each signing operation. However, in the threshold setting, k should be generated in such a way that each participant *contributes to* but *does not know* the resulting k (properties that performing a DKG as described in Section 2.3 also achieve). Key to our design of FROST is the observation that while an arbitrary t out of n entities are required to participate in a signing operation, a simpler t-out-of-t scheme will suffice to generate the random nonce k.

While Shamir secret sharing and derived constructions require shares to be points on a secret polynomial f where $f(0) = s$, an *additive secret sharing scheme* allows t participants to jointly compute a shared secret s by each participant P_i contributing a value s_i such that the resulting shared secret is $s = \sum_{i=1}^{t} s_i$, the summation of each participant's share. Consequently, this t-out-of-t secret sharing can be performed non-interactively; each participant directly chooses their own s_i.

Benaloh and Leichter [1] generalize this scheme to arbitrary access structures; the threshold t-out-of-n case (the "CNF" scheme) corresponds to, for each subset $A \subset \{1, \ldots, n\}$ of size $n-(t-1)$, selecting a random s_A, a copy of which is given to every participant P_i for $i \in A$. The secret is the sum of all the s_A, as before. Note that for general n and t, this scheme is inefficient because each participant holds $\binom{n-1}{t-1}$ shares. However, in the case $n = t$, this scheme reduces to exactly the simpler t-out-of-t additive scheme above.

Share Conversion. Cramer, Damgård, and Ishai [4] present a *non-interactive* mechanism for participants to locally convert additive shares generated via the above generalized additive secret sharing scheme to polynomial (Shamir) form. To perform share conversion using this technique, a secret polynomial f is constructed such that each participant P_i can evaluate f only at point i.

We start with the same setup as above: we consider subsets of $\{1, \ldots, n\}$ of size $n - (t - 1)$. Let U be the universe of all $\binom{n}{t-1}$ such subsets. For each $A \in U$ (so that A is a particular subset of size $n - (t - 1)$), there is a secret share s_A. Then for each $i \in A$, participant P_i holds a copy of s_A. The secret s is finally the sum $\sum_{A \in U} s_A$.

Cramer et al. [4] demonstrate how to non-interactively convert these t-out-of-n additive secret shares of s to t-out-of-n Shamir shares of the same s. For each $A \in U$,

define the polynomial $g_A(x) = \prod_{i \in \{1,\ldots,n\} \setminus A} \frac{i-x}{i}$ (this polynomial can be constructed from information that is entirely public to each participant). Note that for each A of size $n-(t-1)$, $g_A(x)$ is of degree $t-1$, satisfies $g(i) = 0$ for each $i \in \{1,\ldots,n\} \setminus A$, and $g(0) = 1$. Now define $f(x) = \sum_{A \in U} s_A g_A(x)$, which similarly is a degree $t-1$ polynomial. Each participant P_i can compute $f(i)$ using their knowledge of s_A for each A that contains i, but no other evaluation of f. Therefore, as $f(0) = \sum_{A \in U} s_A g_A(0) = \sum_{A \in U} s_A = s$, each $f(i)$ is indeed a t-out-of-n Shamir secret share of s.

In our work, we use the special case of this technique when $n = t$. In this case, each set A is of size 1; consequently, each participant P_i can simply choose their own $s_{\{i\}}$ *non-interactively*. The resulting $g_{\{i\}}(x)$ is a degree $t-1$ polynomial with $g_{\{i\}}(0) = 1$, $g_{\{i\}}(j) = 0$ for $j \in \{1,\ldots,t\} \setminus \{i\}$, and $g_{\{i\}}(i) = \prod_{j \in \{1,\ldots,t\} \setminus \{i\}} \frac{j-i}{j} = \frac{1}{\lambda_i}$, where λ_i is the i^{th} Langrange coefficient for interpolating on the set $\{1,\ldots,t\}$. Therefore, $f(i)$ is simply $\frac{s_{\{i\}}}{\lambda_i}$. The key observation is that if t participants each select s_i at random, then $\frac{s_i}{\lambda_i}$ is a t-out-of-t Shamir secret share of $s = \sum_i s_i$. Importantly, participants are not required to communicate *at all* when creating this Shamir secret sharing of a random value.[2]

In FROST, participants use this technique during signing operations to non-interactively generate a one-time secret nonce (as is required by Schnorr signatures, described in Section 2.4) that is Shamir secret shared among all t signing participants.

3 Motivation

Prior threshold signature constructions [20, 8] provide the property of *robustness*; if one participant misbehaves and provides malformed shares, the remaining honest participants can detect the misbehaviour, exclude the misbehaving participant, and complete the protocol, so long as the number of remaining honest participants is at least the threshold t. This kind of robust construction is appropriate in settings where signing participants might be arbitrary entities from the Internet, for example.

However, in settings where one can expect misbehaving participants to be rare, one can use a protocol that is more efficient in the "optimistic" case that all participants honestly follow the protocol, even if it means aborting and restarting the protocol (having kicked out the misbehaving participant) otherwise.

We now present two use cases of threshold signatures demonstrating when robustness as a security property for threshold signatures is not strictly required in real-world settings, so long as misbehaviour can be detected when it occurs and the misbehaving participant identified and excluded in the future.

[2] An interesting property of the Cramer et al. share conversion scheme that is tangential to our work but worth noting is how s can be updated in a distributed but non-interactive manner in the general t-out-of-n case. Once the copies of each s_A share have been distributed across the participants $P_{i \in A}$, the Shamir-shared secret $s = \sum_{A \in U} s_A$ can be updated in a forward-secret manner by having all participants modify their local copies of each s_A in a deterministic but one-way manner; for example, by hashing. The polynomial evaluations $f(i)$ are then recomputed (again, locally to each participant) with the new s_A values. In this way, a t-out-of-n Shamir shared secret s can be *non-interactively* updated in a *forward-secret* manner to a new random t-out-of-n Shamir shared secret.

Use Case One: Single Owner, Partitioned Secret. In a setting when a secret signing key s is partitioned among a set of devices *owned by the same entity*, robustness is not a strict requirement for signing operations. In this setting, s is partitioned to ensure redundancy in the case of device failure, while limiting the exposure of s to any single device. Notably, a device that issues malformed signatures during signing operations can be simply removed from the set of trusted devices, as the misbehaving device could either be broken or compromised. Because the set of devices is owned entirely by a single entity, simply aborting the protocol and replacing the malfunctioning device is sufficient for this setting.

Use Case Two: Required Abort on Misbehaviour. When s is divided among a set of n mutually untrusted participants, misbehaviour by one of the participants warrants immediate investigation in some settings. In other words, the misbehaviour by one participant *requires* immediate investigation and consequently aborting the protocol. One appropriate response after detecting misbehaviour in this setting can be to remove the participant from the set of trusted signers. In sum, while the act of misbehaving and the identity of the misbehaving participant should be *identifiable*, the protocol does not require robustness.

Observations. Importantly, both use cases underscore the practicality of favouring improved efficiency over robustness in the optimistic case that no participant misbehaves. However, if one participant does misbehave and contributes malformed shares, honest participants can identify the misbehaving participant and abort the protocol. The honest participants can then simply re-run the protocol amongst themselves, excluding the misbehaving participant. Consequently, we can leverage this insight to improve upon prior threshold signature constructions, trading off robustness in the protocol for improved efficiency.

4 Related Work

Stinson and Strobl [20] present a threshold signature scheme producing Schnorr signatures, using the modification of Pedersen's DKG presented by Gennaro et al. [9] to generate both the secret key s during key generation as well as the random nonce k during each signing operations as required by Schnorr signatures. In total, this construction requires at minimum four rounds for each signing operation (assuming no participant misbehaves): three rounds to perform the DKG to obtain k, and one round to distribute signature shares and compute the resulting group signature. Each round requires participants to send values to every other participant.

Gennaro et al. [8] present a threshold Schnorr signature protocol that uses Pedersen's DKG as presented originally [15] to generate both s during key generation and the random nonce k during signing operations. Recall from Section 2.3 that Pedersen's DKG requires two rounds to obtain the k value. In the setting that all participants maintain equal levels of trust, signing operations in this construction require three rounds of communication in total, where all participants send values to all other participants in each round. The authors also discuss an optimization that leverages a *signature aggregator* role, an entity trusted to gather signatures from each participant, perform validation, and publish the resulting signature, a role we also adopt in our work. In

their optimized variant, participants can perform Pedersen's DKG to generate multiple k values in a pre-processing stage independently of performing signing operations. In this variant, to compute ℓ number of signatures, signers first perform two rounds of ℓ parallel executions of Pedersen's DKG, thereby generating ℓ random nonces. The signers can then store these pre-processed values to later perform ℓ single-round signing operations.

Along with standard security notions of correctness and protection against adaptive chosen-message attacks, the schemes presented by Stinson and Strobl [20] and Gennaro et al. [8] are both *robust*; participants that contribute malformed values can be discarded and the protocol can complete, so long as at least t valid participants correctly follow the protocol.

Our work builds upon both of the above schemes; we adapt the proof of security presented by Stinson and Strobl [20] to prove the security of our scheme, and we use Pedersen's DKG for key generation with a requirement that in the case of misbehaviour, the protocol aborts and the cause investigated out of band. However, our work has one key difference. Notably, we *do not* perform a DKG during signing operations as is done in both of the above schemes, but instead make use of additive secret sharing and share conversion. Consequently, our scheme trades off robustness for more efficient signing operations, such that a misbehaving participant can cause the signing operation to abort. However, as described in Section 3, this tradeoff is practical to many real-world settings. Further, because FROST does not provide robustness, FROST is secure so long as the adversary controls fewer than the threshold t participants for any $t \leq n$; protocols that provide both secrecy and robustness can at best provide security for $t \leq n/2$ [9].

While FROST exchanges robustness for improved network round efficiency, other threshold scheme constructions have followed a similar trend. Gennaro and Goldfeder [7] present a threshold ECDSA scheme that similarly requires aborting the protocol in the case of participant misbehaviour. Their signing construction uses a two-round DKG to generate the nonce required for the ECDSA signature, leveraging additive-to-multiplicative share conversion, which has since been independently leveraged in a Schnorr threshold scheme context to generate the random nonce for signing operations [14].

4.1 Attack on Parallelized Schnorr Multisignatures

We next describe an attack recently introduced by Drijvers et al. [5] against some two-round Schnorr multisignature schemes and describe how this attack applies to a threshold setting. This attack can be performed when an adversary can open many (say ψ number of) parallel simultaneous signing operations such that the adversary can see the victim's commitment to their share of the signing nonce in each of the ψ parallel executions, *before* the adversary must choose their own shares for any of the executions. Successfully performing this attack requires finding a hash output $c^* = H(m^*, R^*)$ that is the sum of ψ other hash outputs $c^* = \sum_{j=1}^{\psi} H(m, R_j)$ (where c is the challenge, m the message, and R the commitment corresponding to a standard Schnorr signature as described in Section 2.4). However, Drijvers et al. use the k-tree algorithm

of Wagner [21] to find such hashes and perform the attack in time $O(\kappa \cdot b \cdot 2^{b/(1+\lg \kappa)})$, where $\kappa = \psi + 1$, and b is the bitlength of the order of the group.

Although this attack was proposed in a multisignature n-out-of-n setting, this attack applies similarly in a threshold t-out-of-n setting with the same parameters for an adversary that controls up to $t - 1$ participants. We note that the threshold scheme instantiated using Pedersen's DKG by Gennaro et al. [8] is likewise affected by this technique and so similarly has an upper bound to the amount of parallelism that can be safely allowed.

In Section 6.2 we analyze the effect of this attack on safe choices of the level of parallelism ψ that our first two variants of FROST can support. We also propose a third variant that avoids the attack at some extra computational cost by requiring each participant to select their shares of the signing nonces before seeing other participants' commitments (a similar technique to the secure two-round multisignature scheme presented also by Drijvers et al. in the same work [5]).

The authors also present a metareduction for the proofs of several Schnorr multisignature schemes in the literature that use a generalization of the forking lemma with rewinding, proving that the security demonstrated in a single-party setting does not extend when applying this proof technique to a multi-party setting; we show in Section 7 why this metareduction does not apply to our proof of security.

5 Preliminaries

Let $\mathbb{G}$ be a group of prime order q in which the Decisional Diffie-Hellman problem is hard, and let g be a generator of $\mathbb{G}$. Let H be a cryptographic hash function mapping to $\mathbb{Z}_q^*$.

Let n be the number of participants in the signature scheme, and t denote the threshold of the secret-sharing scheme. Let i denote the *participant identifier* for participant P_i where $1 \leq i \leq n$. Let s_i be the long-lived secret share for participant P_i. Let Y denote the long-lived public key shared by all participants in the threshold signature scheme, and let $Y_i = g^{s_i}$ be the public key share for the participant P_i. Finally, let m be the message to be signed.

For a fixed set $S = \{p_1, \ldots, p_t\}$ of t participant identifiers in the signing operation, let $\lambda_i = \prod_{j=1, j\neq i}^{t} \frac{p_j}{p_j - p_i}$ denote the i^{th} Lagrange coefficient for interpolating over S. Note that the information to derive these values depends on which t (out of n) participants are selected, and uses only the participant *identifiers*, and not their *shares*.[3]

Security Assumptions. We maintain the following assumptions, which implementations need to account for in practice.

- *Message Validation.* We assume every participant checks the validity of the message m to be signed before issuing its signature share. If the message is invalid, the participant should take actions to discard the message and report the misbehaviour to other participants.

[3]Note that if n is small, the λ_i for every possible S can be precomputed by each participant during the key generation phase of the protocol as a performance optimization to avoid re-computing these values for each signing operation.

- *Reliable Message Delivery.* We assume messages are sent between participants using a reliable network channel.
- *Participant Identification.* In order to report misbehaving participants, we require that values submitted by participants to be identifiable within the signing group. Our protocols assume participants are not forging messages by other participants, but implementations can enforce this using a method of participant authentication within the signing group.[4]

6 FROST: Flexible Round-Optimized Schnorr Threshold signatures

We now present FROST, a Schnorr-based threshold signature scheme that addresses the network performance requirements of threshold signatures used in practice, including the use cases presented in Section 3. While prior constructions described in Section 2.3 use a multi-round DKG to generate shared random values during *both* key generation and signing operations, FROST leverages additive secret sharing (as described in Section 2.5) to *non-interactively* generate random values for signing operations. We present three variants of signing operations in FROST to demonstrate its flexibility in different settings. The first two variants require limiting the number of signing operations that any participant can perform in parallel, while the final variant does not restrict the amount of parallelism.

All three variants use the same key generation phase, which we now describe.

6.1 Key Generation

To generate long-lived key shares in our scheme's key generation protocol, FROST uses Pedersen's DKG for key generation. Similarly to Gennaro et al. [8], we refer to Pedersen's DKG as Ped-DKG for the remainder of this work, and present detailed protocol steps in Figure 1.

To begin the key generation protocol, a set of participants must be formed using some out-of-band mechanism decided upon by the implementation. After participating in the Ped-DKG protocol, each participant P_i holds a value (i, s_i) that is their long-lived secret signing share. Participant P_i's public key share $Y_i = g^{s_i}$ is used by other participants to verify the correctness of P_i's signature shares in the following signing phase, while the group public key Y can be used by parties external to the group to verify signatures issued by the group in the future.

View of Commitment Values. As required for *any* multi-party protocol using Feldman's VSS, the key generation stage in FROST similarly requires participants to maintain a consistent view of commitments issued during the execution of Ped-DKG. In this work, we assume participants broadcast the commitment values honestly (e.g., participants do not provide different commitment values to a subset of participants); recall Section 2.1 where we described techniques to achieve this guarantee in practice.

[4]For example, authentication tokens or TLS certificates could serve to authenticate participants to one another.

Distributed Key Generation (DKG) Protocol: Ped-DKG [15]

Round One

1. Every participant P_i samples t random values $(a_{i0}, \ldots, a_{i(t-1)}) \in_R \mathbb{Z}_q$, and uses these values as coefficients to define a polynomial $f_i(x) = \sum_{j=0}^{t-1} a_{ij} x^j$ of degree $t-1$ over $\mathbb{Z}_q$.

2. Every participant P_i computes a public commitment and broadcasts this commitment to all other participants.

$$\vec{C_i} = \langle \phi_{i0}, \ldots, \phi_{i(t-1)} \rangle, \text{ where } \phi_j = g^{a_{ij}}, 0 \leq j \leq t-1$$

Round Two

1. Each participant P_i securely sends to each other participant P_j a secret share $(j, f_i(j))$, and keeps $(i, f_i(i))$ for themselves.

2. Every participant P_i verifies the share they received from each other participant P_j, where $i \neq j$, by verifying:

$$g^{f_j(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} \phi_{jk}^{i^k \bmod q}$$

If the check fails, abort the protocol and investigate the participant that resulted in the failed check. Otherwise, continue to the next step.

3. Each participant P_i calculates their long-lived private signing share by computing the sum of their own and all their received shares $s_i = \sum_{j=1}^{n} f_j(i)$, and stores s_i securely.

4. Each participant then calculates their own public verification share $Y_i = g^{s_i}$, and the group's public key $Y = \prod_{j=1}^{n} \phi_{j0}$. Note that any participant can compute the public verification share of any other participant as $Y_i = \prod_{j=1}^{n} \prod_{k=0}^{t-1} \phi_{jk}^{i^k \bmod q}$.

Figure 1: **Ped-DKG.** A distributed key generation protocol introduced by Pedersen [15] where each of n participants executes Feldman's VSS as the dealer in parallel, and derives their secret share as the sum of the shares received from each of the n VSS executions. This variant requires *aborting* the protocol on misbehaviour.

Security tradeoffs. While Gennaro et al. [9] describe the "Stop, Kill, and Rewind" variant of Ped-DKG (where the protocol terminates and is re-run if misbehaviour is detected) as vulnerable to influence by the adversary, we note that in a real-world setting, good security practices typically require that the cause of misbehaviour is investigated once it has been detected; the protocol is not allowed to terminate and re-run continuously until the adversary finds a desirable output. However, implementations wishing for a robust DKG can use the construction presented by Gennaro et al. [9]. Note that the efficiency of the DKG for the key generation phase is not extremely critical, because this operation must be done only *once per key generation* for long-lived keys. For the per-signature operations, FROST optimizes the generation of random values *without* utilizing a DKG, as discussed next.

6.2 Signing

We now present three variants of the signing protocol for FROST. Note all variants leverage the use of a *signature aggregator*, which we now describe. FROST protocols can also be instantiated without a signature aggregator; we present this option in Appendix A.

Signature Aggregator Role. FROST assumes a semi-trusted role, which we call the signature aggregator $\mathcal{A}$. This role can be performed by *any* participant in the protocol, or even an external party, provided they know the participants' public-key shares Y_i. $\mathcal{A}$ is trusted to report misbehaving participants (we assume values submitted by participants can be authenticated, as discussed in Section 5) and to publish the group's signature at the end of the protocol. As we further describe in Section 7, if $\mathcal{A}$ deviates from the protocol, the protocol remains secure against adaptive chosen message attacks. A malicious $\mathcal{A}$ has the power to perform denial of service attacks and to falsely report misbehaviour by participants, but *cannot* learn the private key or cause improper messages to be signed. Note this signature aggregator role is also used by Gennaro et al. [8] to enable the optimized variant of their construction.

Limits on parallelism of two FROST variants. As noted in Section 4.1, the attack of Drijvers et al. [5] limits the number of concurrent signing operations a signing party may safely perform for some multi-party Schnorr signature schemes. Our first two variants of FROST are of the kind affected by the attack if the level of concurrency ψ is not limited (the third variant is not); we now concretely analyze the implications of their attack and give safe parameters for implementations.

As above, suppose an adversary in a t-out-of-n threshold signing scheme controls $t - 1$ of the t participants in the signing phase of the protocol. Further suppose the remaining (victim) participant can be invoked in such a way that they begin ψ parallel invocations of the signing protocol with the attacker. The victim will generate one nonce value d_{ij} and distribute one commitment D_{ij} for each of the ψ open signing invocations; importantly, they will do so before the adversary selects their own nonce values and reveals their commitments. Consequently, the adversary can create a forged signature from the group in time $O(\kappa \cdot b \cdot 2^{b/(1+\lg \kappa)})$, where $\kappa = \psi + 1$, and b is the bitlength of the order of the group.

In Table 1 we give the effective security level for protocols where this attack is possible (such as the first two variants of FROST) for two common group sizes and

Table 1: Effective security levels (in bits) for our first two variants of FROST at different numbers ψ of allowed concurrent signing operations.

Bitlength of	Effective security for $\psi =$			
group order	1	3	7	15
256	128	95	75	63
448	224	160	124	102

levels of parallelism, from $\psi = 1$ (no parallelism) to $\psi = 15$. We examine group orders of size approximately 256 bits, such as Curve25519, and of size approximately 448 bits, such as Ed448. We cap the security at $b/2$ bits as this is the cost of simply extracting the private key from the public key in a b-bit group.

As we can see, allowing up to $\psi = 3$ parallel signature invocations using a 256-bit elliptic curve group still allows for a quite reasonable 95 bits of security, while allowing up to $\psi = 7$ parallel signature invocations using a 448-bit elliptic curve group still allows for nearly 128-bit security.

Overview of FROST Signing Variants. We next describe our three variants of FROST. Our first is a basic two-round variant that requires each participant to send and receive only two messages in total. Our second variant reduces the online signing cost to a single round, but utilizes a non-interactive batch preprocessing phase. Both variants require imposing limitations on ψ, the number of concurrent signing operations, with security levels described on Table 1. The third variant *does not* restrict parallelism of signing operations, but instead requires participants to select their nonces *before* seeing the commitments to the nonces of other participants, but in turn requires either an additional round in the preprocessing phase or additional computation in the online signing phase.

6.3 Two-Round Interactive Signing Protocol

We present a two-round protocol in Figure 2; notably, this design requires participants to send and receive only two messages in total (one message in each round). The signature aggregator $\mathcal{A}$ performs coordination among all the participants and consequently sends and receives t messages in each round (or $t - 1$ messages if $\mathcal{A}$ is also a participant).

At the beginning of the signing protocol, $\mathcal{A}$ selects t participants (possibly including itself) to participate in the signing. Let S be the set of those t participants. In the first round, $\mathcal{A}$ asks each participant in S for a commitment share, which serves as a secret share to a random commitment for the group (corresponding to the commitment g^k to the nonce value k in step 1 of the single-party Schnorr signature scheme in Section 2.4). This technique is a t-out-of-t additive secret sharing; the resulting secret nonce is $k = \sum_{i \in S} d_i$, with each d_i selected by participant P_i, $i \in S$. Recall from Section 2.5, however, that if the d_i values are *additive* shares of k, then $\frac{d_i}{\lambda_i}$ are t-out-of-t *Shamir* shares of k. After generating their nonce value d_i locally and non-interactively, each participant sends a commitment share $D_i = g^{d_i}$ to $\mathcal{A}$.

In the second round, each participant in S receives from $\mathcal{A}$ the message m to be

Sign(m) $\rightarrow (m, \sigma)$

Let $\mathcal{A}$ denote the signature aggregator. Let S form the set of identifiers corresponding to the participants selected for the signing operation, where $|S| = t$. Note that $\mathcal{A}$ can themselves be one of the t participants.

Round 1

1. The signature aggregator $\mathcal{A}$ initializes a signing operation by sending a request for a commitment share to each participant $P_i : i \in S$.

2. Each P_i samples a fresh nonce $d_i \in_R \mathbb{Z}_q$.

3. Each P_i derives a corresponding single-use public commitment share $D_i = g^{d_i}$.

4. Each P_i returns D_i to $\mathcal{A}$, and stores (d_i, D_i) locally.

Round 2

1. The signature aggregator $\mathcal{A}$ computes the public commitment $R = \prod_{i \in S} D_i$ for the set of selected participants.

2. For $i \in S$, $\mathcal{A}$ sends P_i the tuple (m, R, S).

3. After receiving (m, R, S), each participant P_i for $i \in S$ first validates the message m, aborting if the check fails.

4. Each P_i computes the challenge $c = H(m, R)$.

5. Each P_i computes their response using their long-lived secret share s_i by computing $z_i = d_i + \lambda_i \cdot s_i \cdot c$, using S to determine λ_i.

6. Each P_i securely deletes (d_i, D_i), and then returns z_i to $\mathcal{A}$.

7. The signature aggregator $\mathcal{A}$ performs the following steps:

- **7.a** Verifies the validity of each response by checking $g^{z_i} \stackrel{?}{=} D_i \cdot Y_i^{c \cdot \lambda_i}$ for each signing share $z_1, \ldots, z_t$. If the equality does not hold, first identify and report the misbehaving participant, and then abort. Otherwise, continue.
- **7.b** Compute the group's response $z = \sum z_i$
- **7.c** Publish the signature $\sigma = (z, c)$ along with the message m.

Figure 2: FROST Two-Round Signing Protocol

signed and the group's public commitment R to k. Each participant checks that m is a message they are willing to sign. Then, as in single-party Schnorr signatures, each participant computes the challenge $c = H(m, R)$. The response to the challenge ($z = k + s \cdot c$ in the single-party case) can be computed using the long-term secret shares s_i, which are t-out-of-n (degree $t-1$) Shamir secret shares of the group's long-lived secret key s. Recalling that $\frac{d_i}{\lambda_i}$ are degree $t-1$ Shamir secret shares of k, we see that $\frac{d_i}{\lambda_i} + s_i \cdot c$ are degree $t-1$ Shamir secret shares of z. Using share conversion again, we get that $z_i = d_i + \lambda_i \cdot s_i \cdot c$ are t-out-of-t additive shares of z.

$\mathcal{A}$ finally checks the consistency of each participant's reported z_i with their commitment share D_i and their public key share Y_i and if every participant issued a correct z_i, then the sum of the z_i values, along with c, forms the Schnorr signature on m.

Implementing Concurrency. As written, the protocol only allows for a single (d_i, D_i) pair to be "outstanding"; that is, created in round 1 but not yet used in round 2. Implementers that wish to support parallel signature operations (multiple round 1 invocations can begin before the round 2 invocations are completed) will need $\mathcal{A}$ to also pass D_i in step 2 of round 2 to indicate to the signer which d_i to use in step 5. As described in Section 6.2, implementers should also take care to bound the levels of parallelism for this signing variant.

6.4 Single-Round Signing Protocol with Preprocessing

We now describe an optimization to the signing protocol presented in Figure 2. Instead of each participant generating one random nonce and commitment share at the time of each signing operation, participants instead generate a set of ψ of these values during an asynchronous and non-interactive batched pre-processing stage. This variant assumes the availability of a centralized location to store commitment shares which we term a *commitment server*, further described below. After performing this pre-processing step, this variant of FROST can support signing operations in a single (non-broadcast) round. We present the full protocol details of this variant in Figure 3.

One feature of the single-round variant that we wish to highlight is the ability to perform *asynchronous* signing operations. Specifically, signers only need to receive one message and reply eventually. Because the protocol occurs in one round, signers are not required to be online simultaneously and instead can process requests and respond with signature shares asynchronously; the final signature can be computed after the t^{th} signer provides their signature share. This computation can be done by $\mathcal{A}$, or even publicly (for example, on a blockchain), if the set of which particular t participants contributed to a given signature is not required to be secret.

Commitment Server Role. The commitment server role performs storage and management of participants' commitment shares, and consequently must be accessible to all participants. While the commitment server may be a separate entity, we note that the signature aggregator $\mathcal{A}$ can also provide this service in addition to its other duties. In this setting, the commitment server is trusted to provide the correct commitment shares upon request. If the commitment server chose to act maliciously, it could either prevent participants from performing the protocol by denial of service, or it could provide stale or malformed commitment values on behalf of honest participants, causing

Preprocess(ψ) $\rightarrow (i, \{D_{ij} : j \in \{1, \ldots, \psi\}\})$

Each participant P_i performs this stage independently as a prerequisite to participate in future signing operations. Let j be a counter denoting a specific nonce/commitment share pair, and let ψ be a constant indicating the number of such pairs generated in a single batch.

1. Create an empty list L_i.

2. For $j : (1, \ldots, \psi)$, perform the following steps:

2.a Sample $d_{ij} \leftarrow \mathbb{Z}_q$ as a single-use private nonce.

2.b Derive a corresponding single-use public commitment share $D_{ij} = g^{d_{ij}}$.

2.c Append D_{ij} to the list L_i, and store (d_{ij}, D_{ij}) locally.

3. Publish (i, L_i) to the commitment server, where i is the identifier for participant P_i.

Sign(m) $\rightarrow (m, \sigma)$

Let $\mathcal{A}$ denote the signature aggregator. Let S be the set of identifiers corresponding to the participants selected for the signing operation, where $|S| = t$.

1. The signature aggregator $\mathcal{A}$ fetches the *next unused* commitment share for each participant $(i, D_{ij}) : i \in S$ from the commitment server. The commitment server returns these values, and deletes them to prevent their use in future signing operations.

2. $\mathcal{A}$ computes the signing group's public commitment $R = \prod_{i \in S} D_{ij}$.

3. The signature aggregator $\mathcal{A}$ sends each selected participant the tuple (m, R, S, D_{ij}), including D_{ij} to indicate which nonce/commitment share the participant should use to perform signing.

4. After receiving (m, R, S, D_{ij}), each participant P_i checks to make sure that D_{ij} corresponds to a valid unused nonce d_{ij}. If not, the participant responds to $\mathcal{A}$ to indicate that the signing protocol must be re-run, and aborts.

5. Each participant validates the message m, aborting if m is invalid.

6. The signature aggregator and participants follow the remaining steps 4–7 in Round 2 of Figure 2.

Figure 3: FROST Single-Round Signing Protocol with a Non-Interactive and Batched Pre-Processing Stage

uncertainty as to whether the commitment server or the participant was the misbehaving entity. We note that if $\mathcal{A}$ assumes the commitment server role itself, this uncertainty is avoided; in addition, by performing the role of commitment server, $\mathcal{A}$ has a complete view of what participants have published and consequently can carry out their role to report misbehaviour when it occurs.

Preprocessing Stage. In the preprocessing stage, each participant begins by generating a set of *single-use* private nonces and corresponding public commitment shares $(d_{ij}, D_{ij}) : j \in \{1, \ldots, \psi\}$, where ψ, the batch size, reflects the number of commitments to nonces that are generated and published before any are used; it is effectively the same notion as the amount of parallelism for the first protocol variant above, and must be restricted to a small value in the same way. In this setting, j is a counter maintained by each participant locally to identify the next nonce/commitment share pair available to use for signing.

Each participant P_i then publishes this set of commitment shares and their own participant identifier $(i, \{D_{ij} : j \in \{1, \ldots, \psi\}\})$ to the commitment server. The commitment server stores this information for use in subsequent signing operations.

We present the complete set of steps for this preprocessing stage in Figure 3. Each participant must separately perform this operation at least once per each ψ signing operations.

Signing Protocol. The signing protocol in this variant remains largely the same as in Section 6.3, as demonstrated in Figure 3. However, instead of the signature aggregator $\mathcal{A}$ requesting D_{ij} from each participant, $\mathcal{A}$ instead fetches these values from the commitment server, which deletes each one (or marks it as used) after serving it to $\mathcal{A}$. Upon receiving the request from $\mathcal{A}$ to begin the signing protocol that includes the commitment share D_{ij} indicating which nonce to use in the signing operation, each participant first checks to ensure that D_{ij} corresponds to a valid nonce and commitment share (e.g., the participant has not yet deleted d_{ij} after using it in an earlier signing round).

Because each nonce and commitment share generated during the preprocessing stage described in Figure 3 must be used *at most once*, participants delete these values after using them in a signing operation, as indicated in Step 5 in Figure 2. An accidentally reused d_{ij} can lead to exposure of the participant's long-term secret s_i, so participants must securely delete them, and defend against snapshot rollback attacks as in any implementation of Schnorr signatures.

On the other hand, if the commitment server serves an already-used commitment share D_{ij} to $\mathcal{A}$ during the signing protocol (e.g, the commitment server fails to delete or mark as used D_{ij} after its use in a previous signing operation), the commitment server will cause a denial of service on the signing protocol. Further, as above, settings where the commitment server is not run by $\mathcal{A}$ itself, the commitment server may cause ambiguity as to whether it was the commitment server (serving a stale D_{ij}) or the participant (refusing to respond to a valid D_{ij}) that is misbehaving. However, private keys of participants will never be revealed as a result of misbehaviour by the commitment server, and the security of the scheme remains the same as single-party Schnorr, as described in Section 7.

6.5 Signing with Unrestricted Concurrency

Some deployments of FROST may find the restriction of ψ, the number of allowable parallel invocations of the protocol of Figure 2, or the batch size of the protocol of Figure 3, to be undesirable. To that end, we propose alterations to our single-round signing protocol to safely allow an *unrestricted* number of parallel signing operations.

As a reminder, the limitation on concurrent signing operations in the first two variants of FROST is a safeguard against the attack of Drijvers et al., which requires the adversary to see the victim's ψ values of D_{ij} before selecting their own commitment. To prevent this attack without limiting concurrency, our third variant requires participants to select their d_{ij} nonces (and so their commitment shares $D_{ij} = g^{d_{ij}}$) *in advance of* seeing other participants' commitment shares. We now describe possible instantiations of such a technique.

Commitment via Additional Round. A simple solution to bind participants to their commitment values is to require an additional round in the *preprocessing* phase of the protocol in Figure 3. In step 2 of the preprocessing phase, in addition to computing $D_{ij} = g^{d_{ij}}$, participant P_i also computes π_{ij}, a noninteractive zero-knowledge proof of knowledge of d_{ij}. (Note that this is just a regular single-party Schnorr signature, using d_{ij} as the *private key*, over the message (i, j).) The pair (D_{ij}, π_{ij}) is appended to the list L_i instead of just D_{ij}. Then between steps 2 and 3 of the preprocessing phase, each participant P_i computes a Merkle tree whose leaves are the (D_{ij}, π_{ij}) pairs. Let ρ_i be the root of the Merkle tree. P_i publishes (i, ρ_i) to the commitment server. Each participant then reads and stores each other participant's ρ_i value before proceeding to the existing step 3, to publish its list of ψ number of (D_{ij}, π_{ij}) pairs to the commitment server.

When performing signing operations, in step 3 of the signing phase, instead of sending (m, R, S, D_{ij}) to each of the t participants P_i in the signing protocol, $\mathcal{A}$ instead sends $(m, \langle (i, D_{ij}, \pi_{ij}, \mu_{ij}) \rangle_{i \in S})$ to all t participants, where μ_{ij} is the Merkle proof that (D_{ij}, π_{ij}) was in the Merkle tree rooted at ρ_i. Each signer can then check the signatures π_{ij}, the Merkle proofs μ_{ij}, compute R to be the product of the D_{ij} values, and proceed with step 4 as before.

Note that we use a Merkle root so that each participant is required to locally store only a constant amount of data about each other participant, independent of ψ.

Commitment without Additional Round. For implementations that do not wish to add an extra network round even during the preprocessing phase, we now describe an alternative approach that has higher computational cost per signer but no additional rounds. In this approach, we use two additional one-way functions h_1 and h_2, each mapping $\mathbb{Z}_q$ to $\mathbb{Z}_q$.

In the *key-generation* phase of the protocol, each participant P_i selects a random $r_{i0} \in_R \mathbb{Z}_q$, and publishes $D_{i0} = g^{h_1(r_{i0})}$, in parallel with the rest of the key generation protocol. Then during the preprocessing phase of Figure 3, instead of participant P_i choosing their d_{ij} values at random, they are computed from r_{i0} by ratcheting:

$$r_{ij} = h_2(r_{i(j-1)}), d_{ij} = h_1(r_{ij}), D_{ij} = g^{d_{ij}}.$$

For forward secrecy, the r_{ij} values must be discarded as soon as d_{ij} and $r_{i(j+1)}$ have been computed. Then the list L_i will contain (D_{ij}, π_{ij}) pairs, where π_{ij} here

is a non-interactive zero-knowledge proof of knowledge of $r = r_{i(j-1)}$ such that $D_{i(j-1)} = g^{h_1(r)}$ and $D_{ij} = g^{h_1(h_2(r))}$. $\mathcal{A}$ will then provide $(m, \langle (i, D_{ij}, \pi_{ij}) \rangle_{i \in S})$ to all t participants in the signing protocol, as above. ($\mathcal{A}$ will also need to provide the D_{ij} and π_{ij} values that were "skipped" to any of the t participants that did not participate in a previous signing round. Each participant just keeps track of the last (j, D_{ij}) value they saw for each other participant P_i.)

As a proof of concept, we implemented this zero-knowledge proof using libsnark [18]. As our one-way functions we defined $h_1(r)$ as the x-coordinate of the point rH_1 on a 256-bit elliptic curve, and similarly for h_2, where H_1 and H_2 are constant points with unknown discrete logarithm. We computed the commitments $D_{ij} = g^{d_{ij}}$ also on that 256-bit elliptic curve.[5] The complete zkSNARK required 3840 R1CS constraints. In 100 runs of this implementation, zero-knowledge proof generation took an average of 240 ± 2 ms, and proof verification took an average of 1.2 ± 0.1 ms. The proofs are a constant 137 bytes in size. Other zero-knowledge proof systems could of course be used instead of zkSNARKs.

7 Security

We now present proofs of correctness and security of FROST by employing proof techniques first presented by Stinson and Strobl [20] but adapted for our work.

7.1 Correctness

As in the proof of correctness by Stinson and Strobl, signatures in FROST are also constructed from two degree $t-1$ polynomials; the first polynomial $F_1(x)$ defining the secret sharing of the private signing key s and the second polynomial $F_2(x)$ defining the secret sharing of the nonce k. During the key generation phase described in Figure 1, the first polynomial $F_1(x) = \sum_{j=1}^{n} f_j(x)$ is generated such that the secret key shares are $s_i = F_1(i)$ and the secret key is $s = F_1(0)$. During the signature phase (Figure 2), each of the t participants whose identifiers are in the set S select a d_i, and using share conversion, define a degree $t-1$ polynomial $F_2(x)$ interpolating the values $(i, \frac{d_i}{\lambda_i})$, such that $F_2(0) = \sum_{i \in S} d_i$.

Then let $F_3(x) = F_2(x) + c \cdot F_1(x)$, where $c = H(m, R)$. Now z_i in Figure 2 is $d_i + \lambda_i \cdot s_i \cdot c = \lambda_i (F_2(i) + cF_1(i)) = \lambda_i F_3(i)$, so $z = \sum_{i \in S} z_i$ is simply the Lagrange interpolation of $F_3(0) = (\sum_{i \in S} d_i) + c \cdot s$. Because $R = g^{\sum_{i \in S} d_i}$ and $c = H(m, R)$, (z, c) is a correct Schnorr signature on m.

7.2 Security Against Chosen Message Attacks

To demonstrate that signatures in FROST are unforgeable under adaptively chosen message attack in the random oracle model so long as the adversary controls fewer than the threshold t participants for $t \leq n$, we demonstrate that an adversary A_{Thresh}

[5] This curve is specifically chosen to be efficient for the zkSNARK computation; using a "standard" curve for the final commitment computation would be more expensive. We leave that exploration past our proof-of-concept implementation for future work.

working against FROST can be reduced to a adversary A_{Norm} in Schnorr's signature scheme, and vice versa. We follow the proof strategy used by Stinson and Strobl [20] to demonstrate the security of their scheme in which a DKG is used to generate the shared random value in both key generation and signing phases. We adapt this proof strategy to our setting, which uses additive secret sharing and share conversion when generating a shared-but-secret random nonce during signing operations.

7.2.1 Notion of Security

We begin by highlighting two important features of the Stinson and Strobl proof strategy. First, the adversary is playing the EUF-CMA (existentially unforgeable against a chosen message attack) game: the adversary is given a signing oracle to which it can pass arbitrary messages, and the adversary wins if it can produce a valid signature on a message not passed to that oracle. In the single-party setting, that signing oracle simply produces the signature, but the multi-party setting, that oracle also produces the *view* of the adversary: any private values chosen by the $t-1$ compromised participants, and the public values sent by the honest participants.

However, one important feature of this model is that the adversary does not get to manipulate the inside of the oracle. In particular, it cannot choose its private inputs in an oracle call based on the outputs of an honest participant. This is indeed where the Drijvers et al. attack fits, and why we limit the parallelism ψ in our first two variants to defeat it. On the other hand, in our third variant, by forcing the adversary to commit to its inputs before the signing oracle is invoked, we restore the fidelity of the model and we no longer need to limit ψ.

The second important feature of the Stinson and Strobl proof strategy is that the proof is a reduction for the end-to-end protocol, including both key generation as well as signing steps. In particular, the adversary is not passed a public key as input, but rather the adversary participates in the key generation protocol to *output* a public key. The proof, as we will see below, then proves that for any adversary A_{Thresh} against FROST that outputs a forged signature with a certain probability, given that the key generation algorithm outputted the public key Y, there is (the usual) adversary A_{Norm} against single-party Schnorr signatures that inputs the same public key Y, and outputs a forged signature with the same probability. It is the group public key, and not an honest participant's public key, that matches the public key for the single-party case. In this way, Stinson and Strobl's proof strategy avoids the metareduction of Drijvers et al. [5].

We therefore aim to prove that an adversary A_{Thresh} against FROST can be simulated by an adversary A_{Norm} against single-party Schnorr, and vice versa. We assume A_{Thresh} can compromise up to $t-1$ participants. To simulate the role of honest participants during the key generation phase, we use a simulator SIM as described by Gennaro et al. [8] to simulate the honest participants.

7.2.2 Adversary View

We now describe the view of A_{Thresh} during the execution of key generation and signing operations in FROST.

During key generation, the view of A_{Thresh} includes the following:

1. For the $t-1$ corrupted participants:

- **1.a** The coefficients $\alpha_{i0}, \ldots, \alpha_{i(t-1)}$ defining the secret polynomials $f_i(x)$.
- **1.b** The secret shares $j, f_i(j)$.

2. For all t participants:

- **2.a** The public commitments $\vec{C_i}$ for $1 \leq i \leq n$.
- **2.b** The long-lived public key Y_i for each participant, as well as the group public key Y.

During signing operations, the view of A_{Thresh} includes the following:

1. For the $t-1$ corrupted participants:

- **1.a** The private nonce values d_i for the $t-1$ corrupted participants.

2. For all t participants:

- **2.a** The message m, and the set S comprising the identifiers for all participants selected for the signing operation.
- **2.b** The public commitment shares D_i, $i \in S$.
- **2.c** The group's commitment value R, and the challenge $c = H(m, R)$.
- **2.d** Each participant's response $z_i, i \in S$, and the signature $\sigma = (\sum z_i, c)$.

7.2.3 Unforgeability

Again following the proof strategy of Stinson and Strobl [20], we demonstrate a reduction between an adversary in the threshold setting that can produce a chosen message attack and a adversary in a single-party setting producing a valid forgery for normal Schnorr. By demonstrating the reduction between these two adversaries, we demonstrate that FROST is as secure as single-party Schnorr in its security against a chosen message attack in the EUF-CMA model.

We now provide formal definitions of A_{Thresh} and A_{Norm}, as similarly defined by Stinson and Strobl [20].

Definition 1. Let A_{Thresh} be a probabilistic polynomial time adversary who has the power to corrupt up to $t-1$ participants, and let B denote the set of identifiers for these $t-1$ corrupted participants. Consequently, A_{Thresh} can view both the private and public values for $P_i, i \in B$ during protocol execution. This adversary also has access to t participants (including the corrupted ones) who perform signatures on a message m and public key Y, assuming the key generation protocol outputs Y. Let $A_{Thresh}(\mathbb{G}, q, \mathcal{V}|Y)$ denote the random variable that takes the value $(m_1, \ldots, m_r, (\bar{m}, \bar{\sigma}))$ with the probability that the adversary A_{Thresh}, given the group parameters and the view $\mathcal{V}$ from the key generation phase, will query the signing oracle with the messages $(m_1, \ldots, m_r)$ and output a valid signature $\bar{\sigma}$ on an unqueried message $\bar{m}$, conditioned on the group public key being Y.

Definition 2. Let A_{Norm} be a probabilistic polynomial time adversary who can query a signing oracle with a message and public key (m, Y) and receive in return a signature σ that is valid under Y. Let $A_{Norm}(G, q, Y)$ be the random variable that takes the value $(m_1, \ldots, m_r, (\bar{m}, \bar{\sigma}))$ with the probability that the adversary A_{Norm}, given the group parameters and the public key Y, will query the signing oracle with the messages $(m_1, \ldots, m_r)$ and output a valid signature $\bar{\sigma}$ on an unqueried message $\bar{m}$. Note that in the single-party setting, the adversary does not have a view into the key generation phase.

Again similarly to Stinson and Strobl [20], we prove that any instantiation of a single-party adversary A_{Norm} can be simulated by our threshold adversary A_{Thresh} and vice versa. We start with the easy direction.

Theorem 7.1. *For any adversary A_{Norm} against single-party Schnorr, there exists an adversary A_{Thresh} against FROST such that*

$$Pr[A_{Thresh}(G, q, \mathcal{V}|Y) = (m_1, \ldots, m_r, (\bar{m}, \bar{\sigma}))] = Pr[A_{Norm}(G, q, Y) = (m_1, \ldots, m_r, (\bar{m}, \bar{\sigma}))]$$

Proof. We demonstrate how to construct A_{Thresh} given A_{Norm}. A_{Thresh} starts by executing the key generation phase of FROST honestly. Let Y be the resulting public key.

A_{Thresh} begins by invoking A_{Norm} on (G, q, Y). To simulate the single-party signing oracle to A_{Norm} for a message m_i, A_{Thresh} asks t participants to issue a signature over m_i and returns the output m_i, σ_i. Having been supplied its initial parameters and the signing oracle by A_{Thresh}, A_{Norm} can subsequently perform its chosen message attack. Consequently, A_{Thresh} will output $\bar{m}, \bar{\sigma}$ when A_{Norm} outputs $\bar{m}, \bar{\sigma}$. □

We follow a similar approach to Stinson and Strobl [20] to prove the relation in the opposite direction: that any threshold adversary A_{Thresh} can be simulated by a single-party adversary A_{Norm}. However, while the proof by Stinson and Strobl required use of the simulator SIM to simulate the behavior of honest participants during the signing phase (as their construction makes use of a DKG), our proof allows A_{Norm} to fully simulate the view for A_{Thresh}, including the public values for the honest participant, simply by using its access to the signing oracle and information obtained from the $t-1$ corrupted participants.

Theorem 7.2. *For any adversary A_{Thresh} against FROST in a threshold setting, there exists an adversary A_{Norm} in the single-party Schnorr setting such that*

$$Pr[A_{Norm}(G, q, Y) = (m_1, \ldots, m_r, (\bar{m}, \bar{\sigma}))] = Pr[A_{Thresh}(G, q, \mathcal{V}|Y) = (m_1, \ldots, m_r, (\bar{m}, \bar{\sigma}))]$$

Proof. We demonstrate how A_{Norm} can be constructed given A_{Thresh}. A_{Norm} will simulate the honest participants to A_{Thresh} and provide access to the single-party signing oracle.

A_{Norm} begins by invoking A_{Thresh}, sending G, q. A_{Thresh} performs the key generation stage, interacting with the simulator SIM resulting in the group's long-lived public key Y.

A_{Thresh} then begins its chosen message attack using $t-1$ corrupted participants and one honest participant. When A_{Thresh} queries its t participants to obtain a signature over m_i, A_{Norm} must be able to produce a correct transcript (D_h and z_h) for the honest participant. A_{Norm} first queries its signing oracle with the input (m_i, Y) and obtains σ. To compute the honest participant's public commitment share D_h, A_{Norm} first computes $R = g^z \cdot Y^{-c}$, where $\sigma = (z, c)$. Next, A_{Norm} computes D_h by computing:

$$D_h = \frac{R}{g^{\sum_{j \in B} d_j}}$$

Note that this step is where our first two FROST variants require the adversary to be bound within the EUF-CMA game's signing oracle model, and consequently assume the adversary supplies its inputs, including the d_j values for the compromised participants, to the signing oracle before seeing the honest participant's D_h. As discussed above, our third FROST variant allows for an adversary not bound to this model, and remains secure by requiring zero-knowledge proofs to bind all participants' choices of d_j in advance of seeing other participants' D_j values. In this latter setting, the honest party's zero-knowledge proofs are simulated by A_{Norm}.

A_{Norm} computes the response z_h for the honest participant using similar techniques (recall that $z = \sum z_i$):

$$z_h = z - \textstyle\sum_{j \in B} (d_j + \lambda_j \cdot s_j \cdot c)$$

With these values, A_{Norm} can provide the complete view to A_{Thresh} to perform its chosen message attack. □

7.3 Aborting on Misbehaviour

As discussed above, the goal of FROST is to save communication rounds (particularly at signing time), at the cost of sacrificing robustness. If one of the signing participants provides an incorrect signature share, $\mathcal{A}$ will detect that and abort the protocol, if $\mathcal{A}$ is itself behaving correctly. The protocol can then be rerun with the misbehaving party removed. If $\mathcal{A}$ is itself misbehaving, and even if up to $t-1$ participants are corrupted, $\mathcal{A}$ still cannot produce a valid signature on a message not approved by at least one honest participant.

8 Future Work

Alternative Commitment Binding Technique. As described in Section 6.5, implementations of FROST can be unlimited in concurrent signing operations so long as all participants can be bound to their commitment shares before viewing those of other participants. While we describe two techniques to achieve this binding, we now discuss a future research direction for an alternative technique.

Ideally, from a single public value D_{i0} output by participant P_i during the key generation phase, it would be extremely useful for the other participants to be able to simply *compute* the subsequent D_{ij} values, given each j. The difficulty in this approach is ensuring forward secrecy: D_{ij} should be computable from D_{i0} and j (and possibly

m as well), and d_{ij} (such that $D_{ij} = g^{d_{ij}}$) should be computable from some private state (held by P_i) and j (and possibly m), but after d_{ij} is used, it should *no longer* be computable from P_i's private state. We suspect a technique akin to puncturable encryption [11] could be used here, but we leave this to future work.

Formalization of Security Models. Because of the flexibility yet distributed nature of multisignature schemes, the setting and threat model of implementations will impact the acceptable security model for the scheme. While clearly erring on the side of the strongest possible security model is generally the best approach, not all implementations of secret-sharing schemes require the level of security as is appropriate for "strangers on the internet"; in fact, such settings may benefit from a relaxation in threat model for improvements in efficiency or usability. Formalizing the security expectations of such divergent threat models can help protocol designers focus on ensuring necessary security guarantees while preserving other tradeoffs such as performance or protocol composability.

9 Conclusion

While threshold signatures provide a unique cryptographic functionality that is applicable across a range of settings, implementations incur network overhead costs when performing signing operations under heavy load. As such, minimizing the number of network rounds when generating signatures in threshold signature schemes will reduce the cost of network overhead, benefiting implementations such as those with network-limited devices, where network transmission is costly, or where signers can go offline but wish to perform a signing operation asynchronously.

In this work, we introduce FROST, a flexible Schnorr-based threshold signature scheme that trades protocol robustness in exchange for optimizing the number of rounds required for signing. We present three signing protocol variants. The first two variants are limited in concurrency but efficient in per-user computation; the first reduces the number of messages participants send and receive, and the second variant is a further optimization to a single-round signing operation with a batched non-interactive preprocessing stage. The third variant does not restrict concurrency of signing operations, but is more costly in per-signature operations. We present two use cases demonstrating examples when trading protocol robustness for improved efficiency is desirable for implementations of threshold signatures, and discuss how aborting the signing protocol is practical in these settings, so long as misbehaviour is detected and the misbehaving participant is identified. We present proofs of security and correctness for FROST, building on prior proofs of security demonstrating its security relative to Schnorr's signature scheme in a single-party setting.

Acknowledgments

We thank George Tankersley, Henry DeValence, and Deirdre Connolly for their helpful feedback and discussions about real-world applications of threshold signatures. We thank Ian Miers for his observation that often participants in multisignature schemes

are already semi-trusted and not strangers on the internet.

We acknowledge the helpful description of additive secret sharing and share conversion as a useful technique to non-interactively generate secrets for Shamir secret-sharing schemes by Lueks [12, §2.5.2].

We thank the Royal Bank of Canada and NSERC grant CRDPJ-534381 for funding this work. This research was undertaken, in part, thanks to funding from the Canada Research Chairs program.

References

[1] Josh Benaloh and Jerry Leichter. Generalized Secret Sharing and Monotone Functions. In *CRYPTO'88*, 1988.

[2] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact Multi-signatures for Smaller Blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 435–464, Cham, 2018. Springer International Publishing.

[3] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, Sep 2004.

[4] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *Theory of Cryptography*, pages 342–362. Springer, 2005.

[5] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1084–1101, 2019.

[6] Paul Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 427–438, Washington, DC, USA, 1987. IEEE Computer Society.

[7] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1179–1194, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of pedersen's distributed key generation protocol. In Marc Joye, editor, *Topics in Cryptology — CT-RSA 2003*, pages 373–390. Springer, 2003.

[9] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology*, 20:51–83, 2007.

[10] Steven Goldfeder, Rosario Gennaro, Harry Kalodner, Joseph Bonneau, Joshua A. Kroll, Edward W. Felten, and Arvind Narayanan. Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme. http://stevengoldfeder.com/papers/threshold_sigs.pdf, 2015. Accessed Dec 2019.

[11] Matthew D. Green and Ian Miers. Forward Secure Asynchronous Messaging from Puncturable Encryption. In *36th IEEE Symposium on Security and Privacy*, 2015.

[12] Wouter Lueks. Security and Privacy via Cryptography — Having your cake and eating it too. https://wouterlueks.nl/assets/docs/thesis_lueks_def.pdf, 2017.

[13] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, Berkeley, CA, USA, 2011. USENIX Association.

[14] KZen Networks. Multi Party Schnorr Signatures. https://github.com/KZen-networks/multi-party-schnorr, 2019. Accessed Jan 2020.

[15] Torben P. Pedersen. A Threshold Cryptosystem without a Trusted Party (Extended Abstract). In *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer, 1991.

[16] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140. Springer-Verlag, 1991.

[17] Claus-Peter Schnorr. Efficient Identification and Signatures for Smart Cards. In *CRYPTO*, 1989.

[18] SCIPR Lab and contributors. libsnark: a C++ library for zkSNARK proofs. https://github.com/scipr-lab/libsnark, 2019.

[19] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[20] Douglas R. Stinson and Reto Strobl. Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates. In *Proceedings of the 6th Australasian Conference on Information Security and Privacy*, ACISP '01, pages 417–434, London, UK, 2001. Springer-Verlag.

[21] David Wagner. A Generalized Birthday Problem. In *CRYPTO '02*, 2002.

A Two-Round Signing Protocol Without a Signature Aggregator

Not every implementation may wish to utilize a signature aggregator role. Below, we present a variant of the two-round signing protocol where all participants are trusted equally.

Let S form the set of identifiers corresponding to the participants selected for the signing operation, where $|S| = t$. We assume participants have utilized an out-of-band channel to obtain the message m which the signature is to be computed for, as well as the set S.

Round 1

2. Each P_i samples a fresh nonce $d_i \in_R \mathbb{Z}_q$.
3. Each P_i derives a corresponding single-use public commitment share $D_i = g^{d_i}$.
4. Each P_i broadcasts D_i to all other participants, and stores (d_i, D_i) locally.

Round 2

1. Using the received public values D_j where $j \in S, j \neq i$ and their own public value D_i, each participant computes the public commitment $R = \prod_{i \in S} D_i$ for the set of selected participants.
3. Each P_i for $i \in S$ first validates the message m, aborting if the check fails.
4. Each P_i computes the challenge $c = H(m, R)$.
5. Each P_i computes their response using their long-lived secret share s_i by computing $z_i = d_i + \lambda_i \cdot s_i \cdot c$, using S to determine λ_i.
6. Each P_i securely deletes (d_i, D_i), and then broadcasts z_i to all other participants.
7. Each P_i performs the following steps:

   7.a Verifies the validity of each response by checking $g^{z_i} \stackrel{?}{=} D_i \cdot Y_i^{c \cdot \lambda_i}$ for each signing share $z_1, \ldots, z_t$. If the equality does not hold, first identify and report the misbehaving participant, and then abort. Otherwise, continue.

   7.b Compute the group's response $z = \sum z_i$

   7.c Publish the signature $\sigma = (z, c)$ along with the message m.