FROST: Flexible Round-Optimized Schnorr Threshold signatures [Extended Abstract]

Chelsea Komlo[1] and Ian Goldberg[2]

[1]University of Waterloo, Zcash Foundation
[2]University of Waterloo

January 7, 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 present two variants of signing operations in FROST, the first requiring participants to send and receive two messages in total, and an optimized single-round variant with a batched non-interactive preprocessing stage. FROST achieves its efficiency improvements by allowing the protocol to abort in the presence of a misbehaving party (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. Threshold 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 [9], or to sign a network consensus produced by a set of trusted authorities [11]. 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 signing party. 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 [2, 3], 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 [7, 16] 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 two variants of FROST; 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. In lieu of robustness, FROST achieves improved efficiency in the optimistic case that no party misbehaves. However, in the case where a misbehaving participant contributes malformed values during the protocol, honest parties can identify and exclude the misbehaving party, 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 extended abstract of our 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. We define two variants for performing signing operations, the first a

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

two-round protocol where participants send and receive two messages in total, and the second an optimization to a single-round variant with a batched non-interactive preprocessing stage.

- We give a high-level sketch of the security and correctness of FROST, building upon proofs of security for prior related schemes. We present our complete security proofs in the full version of this work.

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 intuition for our proofs of security, which we include in greater detail in the full version of this work, and conclude in Section 8.

2 Background

2.1 Threshold Schemes

Threshold schemes are cryptographic protocols allowing 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 [15], 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 [5] 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 $(a_1, \dots, a_{t-1})$, 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

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

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 Signatures 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 [14]. We introduce Schnorr signatures in Section 2.4.

Because threshold signature schemes ensure that no party (or indeed any group of fewer than t parties) ever learns the secret key s, the generation of s and distribution of shares $s_1, \dots, s_n$ 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 [12] presents a two-round DKG where each participant acts as the central dealer of Feldman's VSS [5] 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. [8] demonstrate a weakness of Pedersen's DKG [12] 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 [13] 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. [7] prove that Pedersen's DKG as originally described [12] 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. [8] that requires the additional commitment round. In short, the two-round Pedersen's DKG [12] requires a larger group to be as secure as the three-round DKG presented by Gennaro et al. [8].

2.4 Schnorr Signatures

Often, it is desirable for signatures produced by threshold signing operations to be indistinguishable from signatures produced by a single party, consequently being 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 [14] 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 secret. 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, such that the secret s is the sum $\sum_{A \in U} s_A$. Then for each $i \in A$, participant P_i holds a copy of s_A.

Cramer et al. [4] then show how to non-interactively convert these t-out-of-n additive secret shares of s to t-out-of-n Shamir shares of s. For each $A \in U$, define the polynomial $g_A(x) = \prod_{i \in \{1,\ldots,n\} \setminus A} \frac{(i-x)}{i}$. 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 will also be 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, since $f(0) = \sum_{A \in U} s_A g_A(0) = \sum_{A \in U} s_A = s$, the $f(i)$ are indeed t-out-of-n Shamir secret shares of s.

In our work, we will use the special case of this technique for when $n = t$. In that case, each set A is of size 1, and so each party P_i can simply choose their own $s_{\{i\}}$. Then $g_{\{i\}}(x)$ 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 parties each select s_i at random, then $\frac{s_i}{\lambda_i}$ are t-out-of-t Shamir secret shares of $s = \sum_i s_i$, and no communication was needed at all to create this Shamir secret sharing of a random value.

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 parties.

3 Motivation

Prior threshold signature constructions [7,16] 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. 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 party identified and excluded in the future.

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 parties, misbehaviour by one of the parties 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 party misbehaves. However, if one party does misbehave and contributes malformed shares, honest parties can identify the misbehaving party and abort the protocol. The honest parties 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 [16] present a threshold signature scheme producing Schnorr signatures, using the modification of Pedersen's DKG presented by Gennaro et al. [8] 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. [7] present a threshold Schnorr signature protocol that uses Pedersen's DKG as presented originally [12] 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 [16] and Gennaro et al. [7] 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 [16] 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 $t \leq n/2$ [8].

While FROST is the first Schnorr threshold signature scheme to exchange robustness for improved network round efficiency, other threshold scheme constructions have followed a similar trend. Gennaro and Goldfeder [6] 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.

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 $\lambda_i = \prod_{j=1, j\neq i}^{t} \frac{x_{p_j}}{x_{p_j} - x_{p_i}}$ 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.[2]

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.
- *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.[3]

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

[2]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.

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

the use cases presented in Section 3. While prior constructions described in Section 2.3 use Pedersen's DKG to generate shared random values during *both* key generation and signing operations, FROST leverages additive secret sharing (further discussed in Section 2.5) to *non-interactively* generate random values for signing operations. We present two variants of signing operations in FROST to demonstrate its flexibility in different settings.

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. [7], 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.

Security tradeoffs. While Gennaro et al. [8] 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. [8]. 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 two variants of the signing protocol for FROST. Note that both variants leverage the use of a *signature aggregator*, which we now describe.[4]

[4]FROST protocols can also be instantiated without a signature aggregator; we present this variant in the full version of this work.

Distributed Key Generation (DKG) Protocol: Ped-DKG [12]

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.

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

Round Two

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

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 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 [12] 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.

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 parties (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. [7] to enable the optimized variant of their construction.

6.2.1 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 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 party issued a correct z_i, then the sum of the z_i values, along with c, forms the Schnorr signature on m.

6.2.2 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

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

of each signing operation, participants instead generate a set of these values during an asynchronous and non-interactive batched pre-processing stage. We present this variant assuming 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.

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, 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 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 uncertainly 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 $(d_{ij}, D_{ij}) : j \in \{1, \ldots, \zeta\}$, such that $\zeta \in \mathbb{Z}$ is a constant term indicating the number of nonce/commitment shares generated by each participant in a single batch. 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 $(i, \{D_{ij} : j \in \{1, \ldots, \zeta\}\})$ 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 ζ signing operations.

Signing Protocol. The signing protocol in this variant remains largely the same as in Section 6.2.1, 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

Preprocess(ζ) $\rightarrow (i, \{D_{ij} : j \in \{1, \dots, \zeta\}\})$

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 ζ be a constant indicating the number of such pairs generated in a single batch.

1. Create an empty list ρ.
2. For $j : (1, \dots, \zeta)$, 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 ρ, and store (d_{ij}, D_{ij}) locally.
3. Publish (i, ρ) 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

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.

7 Security

In the full version of this work, we present detailed proofs of our protocol's security and correctness by employing proof techniques first presented by Stinson and Strobl [16] but adapted for our work. We now review high-level proof sketches for the intuition of these proofs.

7.1 Correctness

As in the proof of correctness by Stinson and Strobl, we can similarly prove the correctness for FROST as signatures in FROST are also constructed using secrets from two separate secret polynomials. The first polynomial $f_1(\cdot)$ is defined in the key generation stage via Ped-DKG. The second polynomial $f_2(\cdot)$ is defined during the signing phase by the random nonces selected by the t participants participating in signing, after each participant converts their random nonce (serving as an additive secret share) to a Shamir share using share conversion (as described in Sections 2.5 and 6.2.1).

Finally, the composition of f_1 and f_2 results in the third secret polynomial:

$$f_3(\cdot) = f_2(\cdot) + H(m, R) \cdot f_1(\cdot)$$

As such, the correctness of FROST stems from the fact that the signature $\sigma = (c, z)$ is defined by $c = H(m, R)$ and $z = f_3(0) = f_2(0) + c \cdot f_1(0)$ via additive operations.

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 $A_{ThreshSchnorr}$ working against FROST can be reduced to a adversary $A_{NormSchnorr}$ in Schnorr's signature scheme, and vice versa. This proof strategy is first presented by Stinson and Strobl [16] 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 adopt this proof strategy to demonstrate that the use of additive secret sharing and share conversion is similarly secure when generating a shared-but-secret random value during signing operations. Here, we provide a high-level intuition for this proof but present the complete proof in our full work.

In the proof presented by Stinson and Strobl, honest party behaviours during the DKG are emulated by a simulator, which the proof demonstrates to be indistinguishable from honest parties in the view of $A_{ThreshSchnorr}$. $A_{NormSchnorr}$ thereby employs this simulator when invoking $A_{ThreshSchnorr}$ to emulate honest participants during the DKG phase. To simulate the signing of the honest participants, $A_{NormSchnorr}$ feeds valid partial signatures to $A_{ThreshSchnorr}$ when requested. To compute these partial signatures without actually compromising an honest party, the proof leverages the structure of threshold signatures such that if the $A_{ThreshSchnorr}$ controls $t-1$ parties, $A_{NormSchnorr}$ can produce the partial signature for the t^{th} (honest) participant (and thus simulate them to $A_{ThreshSchnorr}$), even without knowing the long-lived secret share s_i held by that participant.

To prove the security of FROST against chosen message attacks, we employ this same proof strategy. However, because each participant produces their single-use secret non-interactively in FROST, our proof does not require a multi-party simulator as is required by Stinson and Strobl. Instead, $A_{NormSchnorr}$ can directly produce the commitment shares and the signature shares for the t^{th} (honest) participant *without simulating rounds of interaction*, using only the knowledge of the public and private values for the $t-1$ corrupted participants. Consequently, by feeding these values to $A_{ThreshSchnorr}$ when requested, $A_{NormSchnorr}$ can simulate the complete view to $A_{ThreshSchnorr}$ in the threshold setting.

7.3 Aborting Due to Malformed Inputs

The FROST signing protocol will abort when a participant submits a malformed partial signature, assuming the signature aggregator $\mathcal{A}$ correctly follows the protocol; otherwise, the resulting signature will be invalid. As demonstrated in both signing variants in Figures 2 and 3, because the public group commitment R is included in the signature's challenge $c = H(m, R = \prod_{i \in X} D_i)$, if any participant's partial signature z_i fails the validity check performed by $\mathcal{A}$ (specified in step 7.a in Figure 3), an honest $\mathcal{A}$ will abort the protocol. Even if $\mathcal{A}$ acts maliciously and does not report a failed validity check, the proof established in Section 7.2 demonstrates that the difficulty to produce a valid signature that requires the cooperation of at least one honest party (where the adversary controls up to $t-1$ signing parties) is as difficult to an adversary as in the single-party Schnorr case.

8 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 extended abstract, 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 two signing protocol variants; the first is a two-round protocol where participants send and receive only two messages in total, and the second is an optimization of the first, to a single-round signing operation with a batched non-interactive pre-processing stage. 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 high-level security proof sketches for FROST building on prior proofs of security demonstrating its security relative to Schnorr's signature scheme in a single-party setting; we expand upon these proofs in the full version of this work.

Acknowledgments

We thank George Tankersley, Henry DeValence, and Deirdre Connolly for their helpful feedback and discussions about real-world applications of threshold signatures.

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 [10, §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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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.