Workgroup: Crypto Forum Research Group
Internet-Draft: draft-komlo-frost-00
Published: 7 August 2020
Intended Status: Informational
Expires: 8 February 2021
Authors: C. Komlo
University of Waterloo, Zcash Foundation
I. Goldberg
University of Waterloo

FROST: Flexible Round-Optimized Schnorr Threshold Signatures

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. This draft describes FROST, a Flexible Round-Optimized Schnorr Threshold signature scheme that reduces network overhead during signing operations while employing a novel technique to protect against forgery attacks applicable to similar schemes in the literature. FROST improves upon the state of the art in Schnorr threshold signature protocols, as it can safely perform signing operations in a single round without limiting concurrency of signing operations, yet allows for true threshold signing, as only a threshold number of participants are required for signing operations. FROST can be used as either a two-round protocol where signers send and receive two messages in total, or optimized to a single-round signing protocol with a pre-processing stage. FROST achieves its efficiency improvements in part 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.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents

at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 February 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

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 [GGK_15], or to sign a network consensus produced by a set of trusted authorities [MOT_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 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 [BLS04] [BDN18], 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 signature constructions that produce Schnorr signatures with unlimited concurrency [SS01] [GJKR03] require at least three rounds of communication during signing operations, whereas constructions with fewer network rounds [GJKR03] must limit signing concurrency to protect against a forgery attack [DEF_19].

This draft describes FROST, a Flexible Round-Optimized Schnorr Threshold signature scheme that addresses the need for efficient threshold signing operations while improving upon the state of the art to ensure strong security properties *without* limiting the parallelism of signing operations. (Signatures generated using the FROST protocol can also be referred to as "FROSTy signatures".) FROST can be used as either a two-round protocol where signers send and receive two messages in total, or optimized to a (non-broadcast) single-round signing protocol with a pre-processing stage. 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 participant, and re-run the protocol.

The flexible design of FROST lends itself to supporting a number of practical use cases for threshold signing. Because the preprocessing round can be performed separately from the signing round, signing operations can be performed *asynchronously*; once the preprocessing round is complete, signers only need to receive and eventually reply with a single message to create a signature. Further, while some threshold schemes in the literature require all participants to be active during signing operations [GJKR03] [DJN 20], and refer to the threshold property of the protocol as merely a security property, FROST allows any threshold number of participants to produce valid signatures. Consequently, FROST can support use cases where a subset of participants (or participating devices) can remain offline, a property that is often desirable for security in practice.

Organization. We describe background information important to understanding FROST in Section 2, and in Section 3 we review notation and security assumptions. Section 4 describes the FROST protocol, and Section 5 reviews security considerations for FROST. In Section 6 we describe implementation considerations.

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 [Sha79], 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, ..., a_{t-1} at random, and uses the randomly selected values as coefficients to define a polynomial $f(x) = s + SUM(a_i\ x^i, i=1...t-1)$ 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, ..., 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 [Fel87] 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, ..., 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

C = < A_0, ..., A_{t-1} >, where A_0 = g^s and A_j = g^{a_j}

Note that in a distributed setting, each participant P_i must be sure to have the same view of 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 [Sch89]. 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, ..., 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.

FROST can use either Pedersen's DKG [Ped91] or Gennaro's DKG [GJKR07] to generate the shared long-lived secret key among participants during its key generation stage.

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 [Sch89] in a single-signer setting.

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

1. Sample a random nonce k <-$- Z_q; compute the commitment R = g^k in G

2. Compute the challenge c = H(m,R)

3. Using the secret key s, compute the response z = k + s * c in Z_q

4. Define the signature over m to be SIG = (z, c)

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

1. Parse SIG as (z, c).

2. Compute R' = g^z * Y^{-c}

3. Compute c' = H(m, R')

4. Output 1 if c = c' to indicate success; otherwise, output 0.

Schnorr signatures are simply the standard Sigma-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 the 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(s_i, i=1...t), 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 [BL88] generalize this scheme to arbitrary monotone access structures, and Cramer, Damgaerd, and Ishai [CDI05] present a *non-interactive* mechanism for participants to locally convert additive shares generated via the Benaloh and Leichter t-out-of-n additive secret sharing construction to polynomial (Shamir) form. In our work, we use the simplest t-out-of-t case of this transformation, in which, if s_i are *additive* secret shares of s, so that s is the sum of the s_i, then (s_i)/(L_i) are *Shamir* secret shares of the same s, where the L_i are Lagrange coefficients.

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.

2.6. Attacks on Parallelized Schnorr Multisignatures

Attack via Wagner's Algorithm. We next describe an attack recently introduced by Drijvers et al. [DEF_19] against some two-round Schnorr multisignature schemes and describe how this attack applies to a threshold setting. This attack can be performed when the

adversary has control over either choosing the message m to be signed, or the ability to adaptively choose its own individual commitments used to determine the group commitment R after seeing commitments from all other signing parties.

Successfully performing the Drijvers attack requires finding a hash output $c^* = H(m^*, R^*)$ that is the sum of T other hash outputs $c^* = SUM(H(m_j, R_j), j=1...T)$ (where c^* is the challenge, m_j the message, and R_j the commitment corresponding to a standard Schnorr signature as described in Section 2.4). To find T hash outputs that sum to c^*, the adversary can open many (say T number of) parallel simultaneous signing operations, varying in each of the T parallel executions either its individual commitment used to determine R_j or the message being signed m_j. Drijvers et al. use the k-tree algorithm of Wagner [Wag02] to find such hashes and perform the attack in time $O(K * b * 2^{b/(1+\lg K)})$, where K = T + 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. [GJKR03] 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 4.2 we discuss how FROST avoids the attack by ensuring that an attacker will not gain an advantage by adaptively choosing its own commitment (or that of any other of the signing participants) used to determine R_j, or adaptively selecting the message being signed.

Drijvers et al. [DEF_19] 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.

Attack via ROS Solver. Benhamouda et al. [BLOR20] recently present an efficient algorithm solving the ROS (Random inhomogeneities in a Overdetermined Solvable system of linear equations) problem. The authors demonstrate that threshold schemes using Gennaro et al.'s DKG [GJKR07] and multisignature schemes such as two-round MuSig [MPSW19] are not secure against their ROS-solving algorithm. However, the authors conclude that (the current version of) FROST is not affected by their ROS-solving algorithm.

3. Preliminaries

Let G be a group of prime order q in which the Decisional Diffie-Hellman problem is hard, and let g be a generator of G. Let H be a cryptographic hash function mapping to Z_q^*. We denote by x <-$- S that x is uniformly randomly selected from S.

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 <= i <= 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,...,p_t} of t participant identifiers in the signing operation, let L_i = PROD((p_j)/(p_j - p_i), j=1...t, j != i) denote the ith 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*. (Note that if n is small, the L_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.)

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. (For example, authentication tokens or TLS certificates could serve to authenticate participants to one another.)

4. FROST: Flexible Round-Optimized Schnorr Threshold Signatures

We now describe the FROST protocol, a Flexible Round-Optimized Schnorr Threshold signature scheme that minimizes the network overhead of producing Schnorr signatures in a threshold setting while allowing for unrestricted parallelism of signing operations and only a threshold number of signing participants.

Efficiency over Robustness. Prior threshold signature constructions [SS01] [GJKR03] 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, threshold signing protocols can be relaxed to be more efficient in the "optimistic" case that all participants honestly follow the protocol. In the case that a participant does misbehave, 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, FROST trades off robustness in the protocol for improved efficiency in this way.

Signature Aggregator Role. We instantiate FROST using a semi-trusted *signature aggregator* role, denoted as SA. Such a role is often practical in a real-world setting; we include this role as it also allows for improved efficiency. However, FROST can be instantiated without a signature aggregator. To do so, each participant simply performs a broadcast in place of SA performing coordination.

The signature aggregator 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. SA is trusted to report misbehaving participants (we assume participants can authenticate themselves to one another, as discussed in Section 3) and to publish the group's signature at the end of the protocol. If SA deviates from the protocol, the protocol remains secure against adaptive chosen message attacks, as SA is not given any more of a privileged view than the adversary we model. A malicious SA does have 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 in prior threshold signature constructions in the literature [GJKR03] as an optimization.

4.1. Key Generation

FROST KeyGen

Round 1

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

2. Every P_i computes a proof of knowledge to the corresponding secret a_{i0} by calculating a Schnorr signature $SIG_i = (w_i, c_i)$ using a_{i0} as the secret key, such that k <-$- Z_q, $R_i = g^k$, $c_i = H(i, CTX, g^{a_{i0}}, R_i)$, $w_i = k + a_{i0} * c_i$, with CTX being a context string to prevent replay attacks.

3. Every participant P_i computes a public commitment $C_i = < A_{i0}, ..., A_{i(t-1)} >$, where $A_{ij} = g^{a_{ij}}$, 0 <= j <= t-1

4. Every P_i broadcasts C_i, SIG_i to all other participants.

5. Upon receiving C_p, SIG_p from participants 1 <= p <= n, p != i, participant P_i verifies $SIG_p = (w_p, c_p)$, aborting on failure, by checking: c_p =?= $H(p, CTX, A_{p0}, g^{w_p} * A_{p0}^{-c_p})$

Round 2

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

2. Each P_i verifies their shares by calculating: $g^{f_p(i)}$ =?= $PROD(A_{pk}^{(i^k \bmod q)}, k=0...t-1)$, aborting if the check fails.

3. Each P_i calculates their long-lived private signing share by computing $s_i = SUM(f_p(i), p=1...n)$, and stores s_i securely.

4. Each P_i calculates their public verification share $Y_i = g^{s_i}$, and the group's public key $Y = PROD(A_{j0}, j=1...n)$. Any participant can compute the public verification share of any other participant by calculating $Y_i = PROD((A_{jk})^{(i^k \bmod q)}, j=1...n, k=0...t-1)$

To generate long-lived key shares in our scheme's key generation protocol, FROST builds upon Pedersen's DKG for key generation; we detail these protocol steps in the above algorithm. Note that Pedersen's DKG is simply where each participant 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. In addition to the base Pedersen DKG protocol, FROST additionally requires each participant to demonstrate knowledge of their secret

a_{i0} by providing other participants with proof in zero knowledge, instantiated as a Schnorr signature, to protect against rogue-key attacks [BBS03] in the setting where t >= n/2.

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 C_i, 1 <= i <= n 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. [GJKR07] 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. Further, many protocols in practice do not prevent an adversary from aborting and re-executing key agreement at any point in the protocol; adversaries in protocols such as the widely used TLS protocol can skew the distribution of the resulting key simply by re-running the protocol.

However, implementations wishing for a robust DKG can adapt our key generation protocol to the robust construction presented by Gennaro et al. [GJKR07]. 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.

4.2. Threshold Signing with Unrestricted Parallelism

We now introduce the signing protocol for FROST. This operation builds upon known techniques in the literature [AAM20] [GJKR03] by employing additive secret sharing and share conversion in order to

non-interactively generate the nonce value for each signature. However, signing operations in FROST additionally leverage a binding technique to avoid known forgery attacks without limiting concurrency. We present FROST signing in two parts: a pre-processing phase and a single-round signing phase. However, these stages can be combined for a simple two-round protocol if desired.

As a reminder, the attack of Drijvers et al. [DEF_19] requires the adversary to either see the victim's T commitment values before selecting their own commitment, or to adaptively choose the message to be signed, so that the adversary can manipulate the resulting challenge c for the set of participants performing a group signing operation. To prevent this attack without limiting concurrency, FROST binds each participant's response to a specific message as well as the set of participants and their commitments used for that particular signing operation. In doing so, combining responses over different messages or participant/commitment pairs results in an invalid signature, thwarting attacks such as those of Drijvers et al.

Preprocess(Q) -> (i, D_{i1}, E_{i1}, ..., D_{iQ}, E_{iQ})

Each participant P_i, i in {1, ..., n} performs this stage prior to signing. Let j be a counter for a specific nonce/commitment share pair, and Q be the number of pairs generated at a time, such that Q signing operations can be performed before performing another preprocess step.

1. Create an empty list L_i. Then, for 1 <= j <= Q, perform the following:

   1.a Sample single-use nonces (d_{ij}, e_{ij}) <-$- Z_q^* x Z_q^*

   1.b Derive commitment shares (D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}}).

   1.c Append (D_{ij}, E_{ij}) to L_i. Store ((d_{ij}, D_{ij}), (e_{ij}, E_{ij})) for later use in signing operations.

2. Publish (i, L_i) to a predetermined location, as specified by the implementation.

Preprocessing Stage. We present above a preprocessing stage where participants generate and publish Q commitments at a time. In this setting, Q determines the number of nonces that are generated and their corresponding commitments that are published in a single preprocess step, and correspondingly the number of signing operations that can be performed before the participant must perform this preprocessing stage again. Note that implementations that do not wish to cache commitments can instead use a two-round protocol,

where participants publish a single commitment to each other in the first round.

Each participant P_i begins by generating a list of *single-use* private nonce pairs and corresponding public commitment shares $((d_{ij}, D_{ij} = g^{d_{ij}}), (e_{ij}, E_{ij} = g^{e_{ij}}))$ for j=1,...,Q, where j is a counter that identifies the next nonce/commitment share pair available to use for signing. Each P_i then publishes (i, L_i), where L_i is their list of commitment shares $L_i = <(D_{ij}, E_{ij})$ for $j=1,...,Q>$. The location where participants publish these values can depend on the implementation; options include broadcasting to all other participants or publishing to a centralized location that all participants can access (we discuss these options further in Section 6). The set of (i, L_i) tuples are then stored by any entity that might perform the signature aggregator role during signing.

Sign(m) -> (m, SIG)

Let SA denote the signature aggregator (who themselves can be one of the t signing participants). Let S be the set of participants selected for use for this signing operation. Let $B = < (i, D_{ij}, E_{ij})$ for i in S> denote the ordered list of participant indices corresponding to each participant P_i, and L_i be the set of available commitment values for P_i that were published during the Preprocess stage. Each identifier i is coupled with the jth commitments (D_{ij}, E_{ij}) published by P_i that will be used for this particular signing operation. Let H_1, H_2 be hash functions whose outputs are in Z_q^*.

1. SA begins by fetching the next available commitment for each participant P_i in S from L_i and constructs B.

2. For each i in S, SA sends P_i the tuple (m, B).

3. After receiving (m, B), each P_i first validates the message m, and then checks $D_{p\ j}$, $E_{p\ j}$ in G^* for each commitment in B, aborting if either check fails.

4. Each P_i then computes the set of binding values $r_p = H_1(p, m, B)$, p in S. Each P_i then derives the group commitment $R = PROD(D_{pj} * (E_{pj})^{r_p}, p \text{ in } S)$, and the challenge $c = H_2(m, R)$.

5. Each P_i computes their response using their long-lived secret share s_i by computing $z_i = d_{ij} + (e_{ij} * r_i) + L_i * s_i * c$, using S to determine L_i.

6. Each P_i securely deletes $((d_{ij}, D_{ij}), (e_{ij}, E_{ij}))$ from their local storage, and then returns z_i to SA.

7. The signature aggregator SA performs the following steps:

7.a Derive $r_i = H_1(i,m,B)$ and $R_i = D_{ij} * (E_{ij})^{r_i}$ for i in S, and subsequently $R=PROD(R_i, i \text{ in } S)$ and $c = H_2(m,R)$.

7.b Verify the validity of each response by checking $g^{z_i} =?= R_i * \{Y_i\}^{c * L_i}$ for each signing share z_i, i in S. If the equality does not hold, first identify and report the misbehaving participant, and then abort. Otherwise, continue.

7.c Compute the group's response $z = SUM(z_i, i \text{ in } S)$

7.d Publish the signature SIG = (z, c) along with the message m.

Signing Protocol. At the beginning of the signing protocol above, SA selects t participants (possibly including itself) to participate in the signing. Let S be the set of those t participants. SA then selects the next available commitment (D_{ij}, E_{ij}) for each participant in S, which are later used to generate a secret share to a random commitment R for the signing group. (Each participant contributes to the group commitment R, which corresponds to the commitment g^k to the nonce 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 $k = SUM(k_i, i \text{ in } S)$, where each $k_i = d_{ij} + e_{ij} * r_i$ (we next describe how participants calculate r_i), and (d_{ij}, e_{ij}) correspond to the $(D_{ij} = g^{d_{ij}}, E_{ij}=g^{e_{ij}})$ values published during the Preprocess stage. Recall from Section 2.5 that if the k_i are *additive* shares of k, then each $(k_i)/(L_i)$ are t-out-of-t *Shamir* shares of k.

After these steps, SA then creates the set B, where B is the ordered list of tuples $<(i, D_{ij}, E_{ij}) \text{ for } i \text{ in } S>$. SA then sends (m, B) to every P_i, i in S.

After receiving (m, B) from SA to initialize a signing operation, each participant checks that m is a message they are willing to sign. Then, using m and B, all participants derive the "binding values" r_i, i in S such that $r_i = H_1(i, m, B)$, where H_1 is a hash function whose outputs are in Z_q^*.

Each participant can then compute the commitment R_i for each participant in S by deriving $R_i = D_{ij} * (E_{ij})^{r_i}$. Doing so binds the message, the set of signing participants, and each participant's commitment to each signature share, such that signature shares on one message cannot be used for another, assuming that (d_{ij}, e_{ij}) remain secret and are used only once. This binding technique thwarts the attack of Drijvers et al. described in Section 2.6 as attackers cannot combine signature shares across disjoint signing operations or permute the set of signers or published commitments for each signer.

The commitment for the set of signers is then simply $R = PROD(R_i, i \text{ in } S)$. As in single-party Schnorr signatures, each participant computes the challenge $c = H_2(m,R)$.

Each participant's response z_i to the challenge can be computed using the single-use nonces (d_{ij}, e_{ij}) and 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 $(k_i)/(L_i)$ are degree t-1 Shamir secret shares of k, we see that $(k_i)/(L_i) + s_i * c$ are degree t-1 Shamir secret shares of the correct response $z = k + s * c$ for a plain (single-party) Schorr signature. Using share conversion again, and that $k_i = d_{ij} + (e_{ij} * r_i)$, we get that $z_i = d_{ij} + (e_{ij} * r_i) + L_i * s_i * c$ are t-out-of-t additive shares of z.

SA finally checks the consistency of each participant's reported z_i with their commitment share (D_{ij}, E_{ij}) and their public key share Y_i. 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. This signature will verify properly to a verifier unaware that FROST was used to generate the signature, and who checks it with the standard single-party Schnorr verification equation with Y as the public key (Section 2.4).

Handling Ephemeral Outstanding Shares. Because each nonce and commitment share generated during the preprocessing stage described in the Preprocess algorithm must be used *at most once*, participants delete these values after using them in a signing operation, as indicated in Step 5 in the Sign algorithm. An accidentally reused (d_{ij}, e_{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.

However, if SA chooses to re-use a commitment set (D_{ij}, E_{ij}) during the signing protocol, doing so simply results in the participant P_i aborting the protocol, and consequently does not increase the power of SA.

5. Security Considerations

5.1. Proof of Security Properties

We present proofs and arguments of security in our technical report [KG20] to show that FROST is secure against chosen-message attacks, assuming the discrete logarithm problem is hard and the adversary controls fewer participants than the threshold. The strategy is as follows. We first define an intermediate protocol called FROST-Interactive that has one extra round of communication in each of the Preprocess and Sign phases, and prove the security of

FROST-Interactive in the random oracle model. We then give a heuristic argument that the differences between FROST-Interactive and FROST itself do not adversely affect its security.

5.2. 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. Consequently, FROST requires participants to abort once they have detected misbehaviour.

If one of the signing participants provides an incorrect signature share, SA will detect that and abort the protocol, if SA is itself behaving correctly. The protocol can then be rerun with the misbehaving party removed. If SA is itself misbehaving, and even if up to t-1 participants are corrupted, SA still cannot produce a valid signature on a message not approved by at least one honest participant.

6. Operational Considerations

6.1. Publishing Commitments to a Commitment Server

The preprocessing step for FROST in Section 4.2 requires some agreed-upon location for participants to publish their commitments to. We now discuss choices for such a location for implementations, and possible security implications.

While participants could simply broadcast commitments to each other, this approach requires memory overhead and possibly coordination effort. Alternatively, implementations may wish to employ a commitment server specifically tasked with performing and managing of participants' commitment shares. While the commitment server may be a separate entity, we note that the signature aggregator SA can also provide this service in addition to its other duties. In this setting, the commitment server is trusted to provide the correct (i.e, valid and unused) 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. However, simply having access to the set of a participant's *public* published commitments does not grant any additional powers, and a misbehaving commitment server (or SA) that provides old commitment values for a signing operation simply results in either a denial of service or an invalid signature. If SA assumes the commitment server role itself, any uncertainty as to who is the cause of misbehaviour

can be avoided, and allows SA to carry out their role to report misbehaviour when it occurs.

6.2. Adaptively Choosing the Set of Signing Participants

While FROST requires exactly t signers due to the structure of non-interactively generating the nonce k (more specifically, so participants can determine L_i during signing), implementations can still adaptively choose signing participants based on their availability if the implementation does not wish to assume which t signers are online and available when beginning a FROST signing operation.

How implementations should determine the availability of participants, and select which t participants will perform signing, falls outside FROST, and will depend on the implementation details of the communications among the participants. In the worst case, however, implementations can simply add an additional round before performing the FROST signing protocol, during which participants can demonstrate their availability and coordinate how available signers are selected to perform the signing round (such as using some simple tie-breaking exercise or ordering rule).

7. Acknowledgments

We thank Douglas Stebila for his helpful observations on the proof of security and deriving security bounds. We thank Richard Barnes for his helpful discussion on practical constraints and for identifying significant optimizations to a prior version of FROST, which our final version of FROST builds upon.

We thank George Tankersley, Henry DeValence, Deirdre Connolly, and Ian Miers for their feedback and discussions about real-world applications of threshold signatures. We thank Omer Shlomovits and Elichai Turkel for pointing out the case of rogue-key attacks in plain Ped-DKG and the suggestion to use a proof of knowledge for a_{i0} as a prevention mechanism.

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 [Lue17],S.2.5.2.

8. Informative References

[AAM20] Abidin, A., Aly, A., and M.A. Mustafa, "Collaborative Authentication Using Threshold Cryptography", Emerging Technologies for Authorization and Authentication , 2020.

[BBS03] Bellare, M., Boldyreva, A., and J. Staddon, "Randomness Re-use in Multi-recipient Encryption Schemeas", Public Key Cryptography , 2003.

[BDN18] Boneh, D., Drijvers, M., and G. Neven, "Compact Multi-signatures for Smaller Blockchains", ASIACRYPT , 2018.

[BL88] Benaloh, J. and J. Leichter, "Generalized Secret Sharing and Monotone Functions", CRYPTO , 1988.

[BLOR20] Benhamouda, F., Lepoint, T., Orrù, M., and M. Raykova, "On the (in)security of ROS", 2020, <https://eprint.iacr.org/2020/945>.

[BLS04] Boneh, D., Lynn, B., and H. Shacham, "Short Signatures from the Weil Pairing", Journal of Cryptology , 2004.

[CDI05] Cramer, R., Damgård, I., and Y. Ishai, "Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation", Theory of Cryptography , 2005.

[DEF_19] Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., and I. Stepanovs, "On the Security of Two-Round Multi-Signatures", 2019 IEEE Symposium on Security and Privacy (SP) , 2019.

[DJN_20] Damgård, I., Jakobsen, T.P., Nielsen, J.B., Pagter, J.I., and M.B. Østergård, "Fast Threshold ECDSA with Honest Majority", 2020, <https://eprint.iacr.org/2020/501>.

[Fel87] Feldman, P., "A Practical Scheme for Non-interactive Verifiable Secret Sharing", Proceedings of the 28th Annual Symposium on Foundations of Computer Science , 1987.

[GGK_15] Goldfeder, S., Gennaro, R., Kalodner, H., Bonneau, J., Kroll, J.A., Felten, E.W., and A. Narayanan, "Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme", 2015.

[GJKR03] Gennaro, R., Jarecki, S., Krawczyk, H., and T. Rabin, "Secure Applications of Pedersen's Distributed Key Generation Protocol", Topics in Cryptology - CT-RSA 2003 , 2003.

[GJKR07] Gennaro, R., Jarecki, S., Krawczyk, H., and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems", Journal of Cryptology , 2007.

[KG20] Komlo, C. and I. Goldberg, "FROST: Flexible Round-Optimized Schnorr Threshold Signatures", 2020, <https://eprint.iacr.org/2020/852>.

[Lue17] Lueks, W., "Security and Privacy via Cryptography -- Having your cake and eating it too", 2017.

[MOT_11] Mittal, P., Olumofin, F., Troncoso, C., Borisov, N., and I. Goldberg, "PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval", 20th USENIX Security Symposium , 2011.

[MPSW19] Maxwell, G., Poelstra, A., Seurin, Y., and P. Wuille, "Simple Schnorr multi-signatures with applications to Bitcoin", Designs, Codes and Cryptography , 2019.

[Ped91] Pedersen, T.P., "A Threshold Cryptosystem without a Trusted Party (Extended Abstract)", EUROCRYPT '91 , 1991.

[SS01] Stinson, D.R. and R. Strobl, "Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates", Proceedings of the 6th Australasian Conference on Information Security and Privacy , 2001.

[Sch89] Schnorr, C., "Efficient Identification and Signatures for Smart Cards", CRYPTO , 1989.

[Sha79] Shamir, A., "How to share a secret", Communications of the ACM , 1979.

[Wag02] Wagner, D., "A Generalized Birthday Problem", CRYPTO , 2002.

Authors' Addresses

Chelsea Komlo
University of Waterloo, Zcash Foundation

Email: ckomlo@uwaterloo.ca

Ian Goldberg
University of Waterloo

Email: iang@uwaterloo.ca