FROST pseudocode for Coinbase

Daniel Zhou

May 2021

This document abstracts out the FROST pseudocode for implementation and research.

1 Common Notations

- G: EC base point of a group of prime order q in which DDH assumption is hard.
- H: hash functions mapping to $\mathbb{Z}_q^*$
- n: total number of participants in the protocol.
- t: threshold in the protocol.

2 Shamir Secret Sharing

We first recall Shamir secret sharing and Feldman Verifiable Secret Sharing that FROST uses and we have implemented them.

Algorithm 1 $[x_1, x_2, \cdots, x_n] \leftarrow \text{ShamirShare}(x, t, q, [p_1, \cdots, p_n])$

Input: $x, t, q, [p_1, \cdots, p_n]$

1. for $i \in [1, \cdots, t]$
2. Sample $a_i \in \mathbb{Z}_q$
3. for $i \in [1, \cdots, n]$
4. if $p_i = 0$, abort
5. $x_i = x + a_1 p_i + a_2 p_i^2 + \cdots + a_t p_t^t \mod q$
6. Return $[x_1, \cdots, x_n]$

Algorithm 2 $x \leftarrow \text{Reveal}(q, [p_1, x_1], \cdots, [p_{t+1}, x_{t+1}])$

Input: $q, [p_1, x_1], \cdots, [p_{t+1}, x_{t+1}]$

1. Set $x = 0$
2. For $i \in [1, \cdots, t+1]$
3. Set $\ell = x_i$
4. For $j \in [1, \cdots, t+1]$
5. If $i = j$, continue
6. Compute $\ell = \ell \times \frac{p_j}{p_j - p_i} \mod q$
7. Compute $x = x + \ell \mod q$
8. Return x.

3 Feldman VSS

Algorithm 3 $[v_0, \cdots, v_t], [x_1, \cdots, x_n] \leftarrow \text{FeldmanShare}(G, x, t, q, [p_1, \cdots, p_n])$

1. Compute $v_0 = x \cdot G$
2. For $i \in [1, \cdots, t]$
3. sample $a_i \leftarrow \mathbb{Z}_q$
4. $v_i = a_i \cdot G$
5. For $i \in [1, \cdots, n]$
6. If $p_i = 0$, abort
7. $x_i = x + a_1 p_i + a_2 p_i^2 + \cdots + a_t p_i^t \mod q$
8. Return $[v_0, \cdots, v_t], [x_1, \cdots, x_n]$.

Algorithm 4 $0/1 \leftarrow \text{FeldmanVerify}(G, q, x_i, p_i, [v_0, \cdots, v_t])$

1. Set $v = v_0$
2. For $j \in [1, \cdots, t]$
3. $c_j = p_i^j \mod q$
4. $v = v + c_j \cdot v_j$
5. If $v = x_i \cdot G$, return 1.
6. Else return 0

4 Schnorr Signature

FROST uses Schnorr signature as a proof of knowledge as a subroutine. We describe Schnorr signature here. Schnorr signature is simply the standard Sigma protocol proof of knowledge of the discrete log of verification key, made non-interactive with the Fiat-Shamir transform. In Schnorr signature, the secret key is $sk = s \in \mathbb{Z}_q$ and verification key $vk = s \cdot G$

Algorithm 5 $\sigma \leftarrow \text{SchnorrSign}(sk, m)$

1. Sample random nonce $k \leftarrow \mathbb{Z}_q$, compute $R = k \cdot G$
2. Compute challenge $c = H(m, R)$
3. Compute $z = k + s \cdot c \mod q \in \mathbb{Z}_q$
4. Output signature $\sigma = (z, c)$.

Algorithm 6 $0/1 \leftarrow \text{SchnorrVerify}(\sigma, m, vk)$

1. Parse $\sigma = (z, c)$
2. Compute $R' = z \cdot G + (-c) \cdot vk$
3. Compute $c' = H(m, R')$
4. Output 1 if $c = c'$, otherwise output 0.

5 FROST

FROST[1] 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. In the original technical report, it describes the protocol with a *signature aggregator (SA)* role. Including SA allows for improved efficiency in their description. In particular, with SA, the protocol can either finish in two rounds or in one-round with a preprocessing step However, we prefer a decentralized setting so we focus on an instantiation wihtout a SA. Fortunately, FROST also works without a SA. To do so, each participant simply performs a broadcast in place of SA performing coordination. We adapt and describe the FROST protocol without a SA below. The protocol has a 2-round DKG phase and a 3-round signing phase. The round complexity of signing can be reduced to 2 rounds by preprocessing the first round. That is, each participant pre-compute a fixed number, say Q, of commitments for further use so that we don't need to generate commitments every time.

5.1 FROST Key Generation Round 1

Algorithm 7 $(C_i, w_i, c_i, \{x_{i,j}\}_{j\in[n]}) \leftarrow \text{KeyGenRound1}(g, q, G, t, n)$

Takes input g, q, G, t, n, each participant P_i does the following steps.

1. Sample secret $s = a_{i,0} \leftarrow \mathbb{Z}_q$ and run Feldman Share

$$(A_{i,0}, \cdots, A_{i,t}), (x_{i,1}, \cdots, x_{i,n}) \leftarrow \text{FeldmanShare}(s)$$

   Set $C_i = (A_{i,0}, \cdots, A_{i,t})$

2. Sample $k_i \leftarrow \mathbb{Z}_q$.
3. Compute $R_i = k_i \cdot G$
4. Compute $c_i = H(i, CTX, s \cdot G, R_i)$, where CTX is a fixed context string.
5. Compute $w_i = k_i + s \cdot c_i \mod q$
6. Broadcast (C_i, w_i, c_i) to other participants
7. P2PSend $(j, x_{i,j})$ to each participant P_j and keep $(i, x_{i,i})$ for himself.

5.2 FROST Key Generation Round 2

Algorithm 8 $(vk, sk_i, vk_i) \leftarrow \text{KeyGenRound2}(CTX, (C_j, w_j, c_j)_{j\in[n]}, \{x_{j,i}\}_{j\in[n]})$

1. Parse $C_j = (A_{j,0}, \cdots, A_{j,t})$ for each $j \in [n]$
2. For $j \in [n]$
3. if $j == i$, continue
4. Check equation $c_j = H(j, CTX, A_{j,0}, w_j \cdot G + (-c_j) \cdot A_{j,0})$, abort if check fails.
5. FeldmanVerify$(g, q, x_{j,i}, A_{j,0}, \cdots, A_{j,t})$. Abort if check fails.
6. Compute signing key share

$$sk_i = \sum_{j=1}^{n} x_{j,i}$$

   and store it locally.

7. Compute verification key share $vk_i = sk_i \cdot G$.
8. Compute verification key $vk = \sum_{j=1}^{n} A_{j,0}$
9. Broadcast (vk, vk_i) and store sk_i locally.

5.3 FROST Signing Round 1

Algorithm 9 $(d_i, e_i, D_i, E_i) \leftarrow \text{SignRound1}(g, q, G, t, n)$

Each participant P_i does the following

1. Sample $(d_i, e_i) \leftarrow \mathbb{Z}_q^* \times \mathbb{Z}_q^*$
2. Compute $(D_i, E_i) \leftarrow (d_i \cdot G, e_i \cdot G)$
3. Broadcast (i, D_i, E_i) and store (d_i, D_i, e_i, E_i) locally.

5.4 FROST Signing Round 2

Algorithm 10 $(z_i, vk_i) \leftarrow \text{SignRound2}(\{j, D_j, E_j\}_{j\in[1,\cdots,t]}, t, m)$

The signing member P_i does the following

1. Check message m is valid, abort if check fails.
2. Check $D_j, E_j \in G$ for each j are valid, abort if check fails. Store $\{D_j, E_j\}_{j\in[t]}$
3. For $j \in [1, \cdots, t]$
4. Compute $r_j = H(j, m, \{D_j, E_j\}_{j\in[t]})$
5. Compute $R_j = D_j + r_j \cdot E_j$
6. $R = R + R_j$
7. Compute $c = H(m, R)$.
8. Store c, R and all R_j
9. Compute $z_i = d_i + e_i \cdot r_i + L_i \cdot sk_i \cdot c$, where L_i is Lagrange coefficient

$$L_i = \prod_{j=1,\cdots,t,j\neq i} \frac{j}{j-i}$$

10. Broadcast z_i, vk_i to other participants.

5.5 FROST Signing Round 3

Algorithm 11 $\sigma \leftarrow \text{SignRound3}(\{z_j, vk_j\}_{j\in[1,\cdots,t]}, t, n)$

Each participant P_i does the following

1. For $j \in [1, \cdots, t]$
2. Verify equation $z_j \cdot G = R_j + c \cdot L_j \cdot vk_j$, abort if check fails.
3. Compute $z = z + z_j$
4. Self-verify the signature $\sigma = (z, c)$:
5. $R' = z \cdot G + (-c) \cdot vk$
6. $c' = H(m, R')$
7. Output 1 if $c = c'$, otherwise output 0.
8. Output the signature $\sigma = (z, c)$ along with message m.

References

[1] Chelsea Komlo and Ian Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. Internet-Draft draft-komlo-frost-00, Internet Engineering Task Force, August 2020. Work in Progress.