

Blockstream



Blockstream RESEARCH

# ChillDKG: Distributed Key Generation for FROST

2024-09-18

Tim Ruffing & **Jonas Nick**

# FROST #138

Open **jesseposner** wants to merge 21 commits into `BlockstreamResearch:master` from `jesseposner:frost`

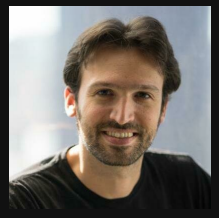
Conversation 141 | Commits 21 | Checks 107 | Files changed 25

**jesseposner** commented on Jul 21, 2021 • edited — Contributor

This PR implements a BIP-340 compatible threshold signature system based on FROST (Flexible Round-Optimized Schnorr Threshold Signatures).

**TODO**

- [x] Key generation APIs







- "I'm not sure"
- "but isn't it dangerous right now"
- "it's really hard to convince yourself that it works"
- "Couldn't it be a problem that there's no randomness?"
- "risks inventing complicated machinery that turns out to be broken"
- "Sorry, I entirely forgot what we're trying to do"
- "Sorry, I'm doing a lot of handwaving"
- "This can be mitigated by another communication round"

Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures.

**FROST KeyGen**

**Round 1**

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

2. Every $P_i$ computes a proof of knowledge to the corresponding secret $a_{i0}$ by calculating $\sigma_i = (R_i, \mu_i)$, such that $k \xleftarrow{\$} \mathbb{Z}_q$, $R_i = g^k$, $c_i = H(i, \Phi, g^{a_{i0}}, R_i)$, $\mu_i = k + a_{i0} \cdot c_i$, with $\Phi$ being a context string to prevent replay attacks.

3. Every participant $P_i$ computes a public commitment $\vec{C_i} = \langle \phi_{i0}, \ldots, \phi_{i(t-1)} \rangle$, where $\phi_{ij} = g^{a_{ij}}, 0 \leq j \leq t-1$

4. Every $P_i$ broadcasts $\vec{C_i}, \sigma_i$ to all other participants.

5. Upon receiving $\vec{C_\ell}, \sigma_\ell$ from participants $1 \leq \ell \leq n, \ell \neq i$, participant $P_i$ verifies $\sigma_\ell = (R_\ell, \mu_\ell)$, aborting on failure, by checking $R_\ell \stackrel{?}{=} g^{\mu_\ell} \cdot \phi_{\ell 0}^{-c_\ell}$, where $c_\ell = H(\ell, \Phi, \phi_{\ell 0}, R_\ell)$.

   Upon success, participants delete $\{\sigma_\ell : 1 \leq \ell \leq n\}$.

**Round 2**

1. Each $P_i$ securely sends to each other participant $P_\ell$ a secret share $(\ell, f_i(\ell))$, deleting $f_i$ and each share afterward except for $(i, f_i(i))$, which they keep for themselves.

2. Each $P_i$ verifies their shares by calculating: $g^{f_\ell(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} \phi_{\ell k}^{i^k \bmod q}$, aborting if the check fails.

3. Each $P_i$ calculates their long-lived private signing share by computing $s_i = \sum_{\ell=1}^{n} f_\ell(i)$, stores $s_i$ securely, and deletes each $f_\ell(i)$.

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

# What is distributed key generation in FROST?

- **Interactive** protocol between $n$ signers that takes $t$ as input
- **Outputs** for each signer $i$:
  - the threshold public key
  - the **secret share** signer $i$ will use for signing
- **Properties:**
  - $t$ out of $n$ signers can use their share to sign
  - No less than $t$ signers can produce a signature
    - In particular, this **excludes "trusted dealer"** that generates and distributes the shares

The FROST RFC famously does not specify a DKG. It relies on a trusted dealer.

We should write a detailed specification of the key generation protocol...







Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr Threshold Signatures Without the Algebraic Group Model.

- Replaces broadcast abstraction with Eq protocol
- ... and other minor changes

Interactive Algorithm $\mathsf{SimplPedPoP}(i)$

Signer $\mathcal{S}_i$ is connected to each other signer $\mathcal{S}_j$ via secure point-to-point channels, which guarantee authentication and confidentiality. This can, e.g., be realized with a public-key infrastructure (PKI).

1. Signer $\mathcal{S}_i$ chooses a random polynomial $f_i(Z)$ over $\mathbb{Z}_p$ of degree $t-1$

$$f_i(Z) = a_{i,0} + a_{i,1}Z + \cdots + a_{i,t-1}Z^{t-1}$$

and computes $A_{i,k} = g^{a_{i,k}}$ for $k = 0, \ldots, t-1$. Denote $x_i = a_{i,0}$ and $X_i = A_{i,0}$. Signer $\mathcal{S}_i$ computes a proof of possession of $X_i$ as a Schnorr signature as follows. Signer $\mathcal{S}_i$ samples $\tilde{r}_i \leftarrow \mathbb{Z}_p$ and sets $\tilde{R}_i \leftarrow g^{\tilde{r}_i}$. Signer $\mathcal{S}_i$ computes $\tilde{c}_i \leftarrow \mathsf{H}_{\mathrm{reg}}(X_i, \tilde{R}_i, i)$ and sets $\tilde{s} \leftarrow \tilde{r} + \tilde{c}_i x_i$. Signer $\mathcal{S}_i$ then derives a commitment $(A_{i,0}, \ldots, A_{i,t-1})$ and sends $((\tilde{R}_i, \tilde{s}_i), (A_{i,0}, \ldots, A_{i,t-1}))$ to all signers $\mathcal{S}_j$ for $j \in \{1, \ldots, n\} \setminus \{i\}$.
Moreover, signer $\mathcal{S}_i$, for every $j \in \{1, \ldots, n\}$ (including $j = i$ itself), computes secret shares $\tilde{x}_{i,j} = f_i(j)$, and sends $\tilde{x}_{i,j}$ to signer $\mathcal{S}_j$.
2. Upon receiving proofs of possession, commitments and secret shares from all other signers, signer $\mathcal{S}_i$ verifies the Schnorr signatures by computing $\tilde{c}_j \leftarrow \mathsf{H}_{\mathrm{reg}}(X_i, \tilde{R}_i, i)$ and checking that

$$\tilde{R}_j A_{j,0}^{\tilde{c}_j} = g^{\tilde{s}_j} \text{ for } j \in \{1, \ldots, n\} \setminus \{i\}.$$

Moreover, signer $\mathcal{S}_i$ verifies the shares received from the other signers by checking

$$g^{\tilde{x}_{j,i}} = \prod_{k=0}^{t-1} A_{j,k}^{i^k}.$$

If any check fails, signer $\mathcal{S}_i$ aborts.
Otherwise, $\mathcal{S}_i$ runs interactive algorithm $\mathsf{Eq}(i, t_i)$ with all other signers $\mathcal{S}_j$ for $j \in \{1, \ldots, n\} \setminus \{i\}$ on local input

$$\eta_i \leftarrow \{(\tilde{R}_j, \tilde{s}_j), (A_{j,0}, \ldots, A_{j,t-1})\}_{j=1}^{n}.$$

3. When $\mathsf{Eq}(i, \eta_i)$ outputs $\mathsf{true}$ for $\mathcal{S}_i$, then $\mathcal{S}_i$ terminates the $\mathsf{SimplPedPoP}$ protocol successfully by outputting the joint public key $X \leftarrow \prod_{j=1}^{n} X_j$ and the local secret key $\tilde{x}_i \leftarrow \sum_{j=1}^{n} \tilde{x}_{j,i}$. When $\mathsf{Eq}(i, t_i)$ outputs $\mathsf{false}$, then $\mathcal{S}_i$ aborts.

**Fig. 3.** Interactive Algorithm $\mathsf{SimplPedPoP}$.

# SimplPedPop (simplified!)

1. Every signer generates $n$ shares and computes a **VSS commitment** to the shares. (VSS: verifiable secret sharing)
2. Every signer sends the $i$-th generated share and the VSS commitment to the $i$-th signer.
3. Every signer computes the threshold public key from the received VSS commitments.

How does the signer know that $t$ can sign for the threshold public key?

4. Every signer **verifies** their received shares against the received VSS commitments.

**Property of VSS**: If every signer received the same VSS commitments, then the signers can indeed sign!

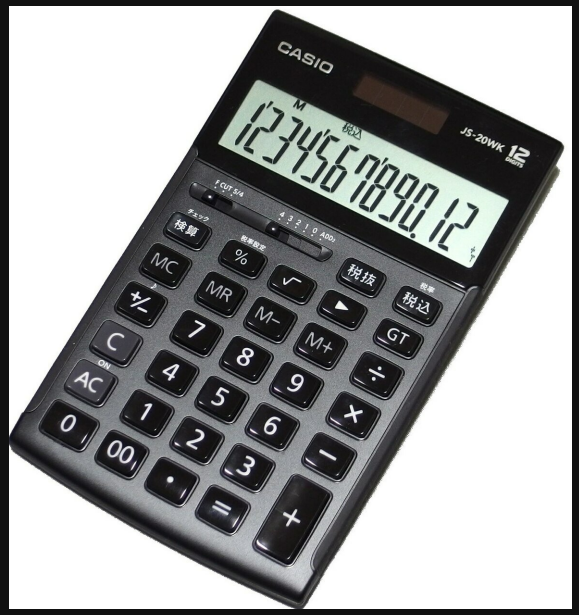
- Hence, signers need to ensure that **no** malicious signer sent a different commitment to signer $i$ than to signer $j \neq i$.
- That's what the equality (the broadcast) protocol is for.

# Interactive Protocol Eq($input$) outputs {true, false}

In SimplPedPop: $input$ contains the VSS commitments

**Integrity:** If some honest signers outputs true, all $input$ of honest signers are equal.

# A Simple Eq


CASIO
JS-20WK
12
検算
%
税抜
税込
MC
MR
M−
M+
GT
+/−
C
AC
ON
÷
×
−
+
=
0
00
.


CASIO
JS-20WK
12
検算
%
税抜
税込
MC
MR
M−
M+
GT
+/−
C
AC
ON
÷
×
−
+
=
0
00
.


CASIO
JS-20WK
12
検算
%
税抜
税込
MC
MR
M−
M+
GT
+/−
C
AC
ON
÷
×
−
+
=
0
00
.

# 2-of-3 Example



Eq *input*

valid Eq message



Eq returns true,

DKG finished,

let's send money to threshold $pk$!

Eq *input*

invalid Eq message



DKG not finished

Just 1 signer left, but we need 2!
Money gone!



Integrity is not enough



# Interactive Protocol Eq($input$) outputs {true, false}

In SimplPedPop: $input$ contains the VSS commitments

**Integrity:** If some honest signer outputs true, all input of honest signers are equal.

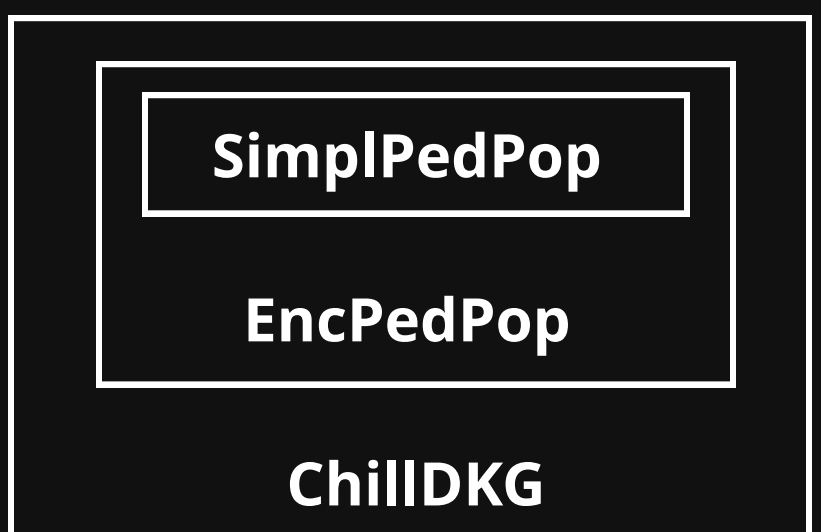
**Agreement:** If some honest signer outputs true, then eventually all honest signers will output true.

Agreement is often an overlooked requirement in the FROST world.

# Interim Summary

- We want to specify the SimplPedPop DKG
- SimplPedPop requires some Eq protocol and secure channels, we want to spec those as well

# Design



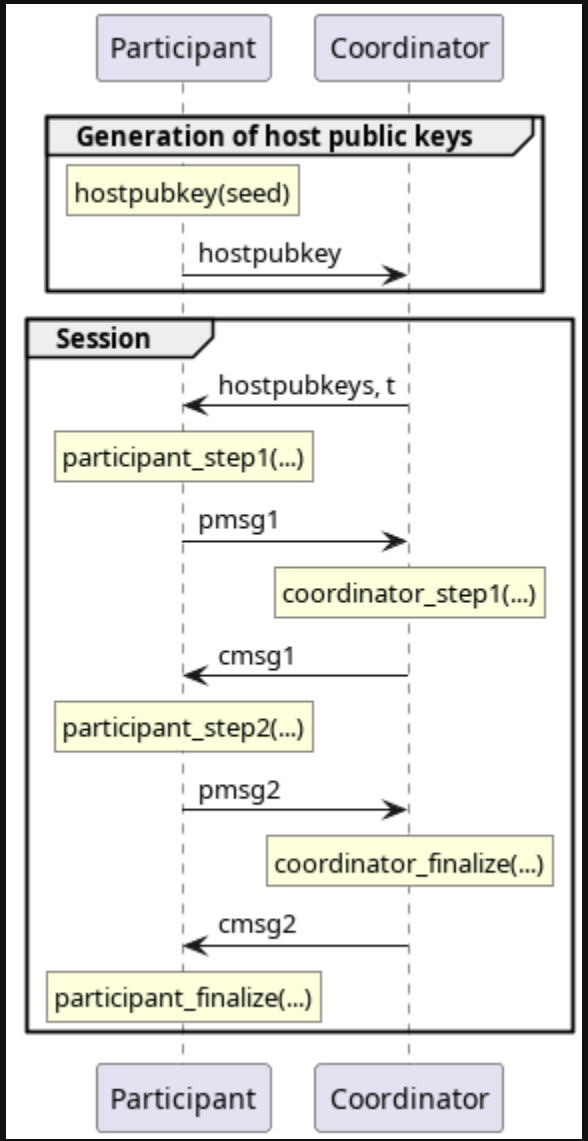
# EncPedPop

- Every signer has **long-term ECDH key pair**
$$(\mathsf{staticpub}, \mathsf{staticpriv})$$
- Assumption: everyone has a **correct copy** of every other signer's $\mathsf{staticpub}$.
- Encryption uses a one-time pad created through **ephemeral-static ECDH** key exchange between sender $i$ and receiver $j$.
$$\mathsf{share}_{i,j} + \mathsf{ECDH}(\mathsf{ephemeral}_i, \mathsf{static}_j)$$
- All signers' claimed $\mathsf{staticpub}$, $\mathsf{ephemeralpub}$ are **added** to Eq's input

# ChillDKG

- Every signer has long-term **"host" key pair**, derived from a seed
- Eq is instantiated with concrete protocol **"CertEq":**
  1. Everyone sends a **signature** of their Eq input to everyone else.
  2. Signers terminate successfully when they receive valid signatures from all $n$ participants (***"success certificate"***)
- Integrity: ✅
- Agreement: ✅ (can convince signer with success cert)

# ChillDKG


Participant
Coordinator
Generation of host public keys
hostpubkey(seed)
hostpubkey
Session
hostpubkeys, t
participant_step1(...)
pmsg1
coordinator_step1(...)
cmsg1
participant_step2(...)
pmsg2
coordinator_finalize(...)
cmsg2
participant_finalize(...)
Participant
Coordinator

# ChillDKG Backups

- In contrast to Schnorr sigs or MuSig, secret keys **cannot be derived** from the seed.
- Naive approach: backup **new secret data** per DKG session
- ChillDKG: backup seed once, and backup "**recovery data"** per DKG session. Recovery data is...
  - ...**self-authenticating** and contains secret data in **encrypted** form
    - can be stored with an **untrusted** third-party
  - ...the **same** for all participants
    - can be requested from **other** participants

# BIP ChillDKG

- Specification in the form of a Bitcoin Improvement Proposal (**BIP**)
- **Standalone** (fully specified), no external secure channels or consensus mechanism.
- Specification/reference implementation in **Python**
- Provides (conditional) **agreement**
- **Simpler backups**: recover from static seed and public recovery data
- Supports **any threshold** $t \leq n$
- Untrusted **coordinator** reduces communication overhead by aggregating some of the messages

# In-Progress Feature

- A **single** signer can cause ChillDKG not to succeed (e.g., by sending nothing, inconsistent VSS commitments, ...)
- In the setting we're considering, the signers **are not able to agree** on which signer is misbehaving
  - E.g., requires majority of signers to be **honest** or **synchronous** network
- However, we believe ChillDKG can be modified such that in case of failure, each honest signer can determine that **either a certain participant or the coordinator** are misbehaving.

# More TODOs

- Collect and address feedback
- Add test vectors



https://github.com/BlockstreamResearch/bip-frost-dkg