Updates on FROST: Flexible Round-Optimized Schnorr Threshold Signatures

Chelsea Komlo[1,2] Ian Goldberg[1]

[1] University of Waterloo [2] Zcash Foundation

Zcon2 Lite, June 2021

CrySP
Cryptography, Security, and Privacy
— Research Group @ uWaterloo —

Threshold Secret Sharing

- Partitions a secret among a set of participants, such that recovering/using the secret requires cooperation among a threshold number of participants.
- Shamir secret sharing is the most well-known algorithm and what FROST builds upon.
- n represents the *total* number of allowed participants; t the threshold.

Threshold Secret Sharing

- Partitions a secret among a set of participants, such that recovering/using the secret requires cooperation among a threshold number of participants.
- Shamir secret sharing is the most well-known algorithm and what FROST builds upon.
- *n* represents the *total* number of allowed participants; *t* the threshold.

Threshold Secret Sharing

- Partitions a secret among a set of participants, such that recovering/using the secret requires cooperation among a threshold number of participants.
- Shamir secret sharing is the most well-known algorithm and what FROST builds upon.
- n represents the *total* number of allowed participants; t the threshold.

Why use Secret Sharing?

- Disaster recovery.
- Raise the bar for an adversary.
- Partition trust.

Why use Secret Sharing?

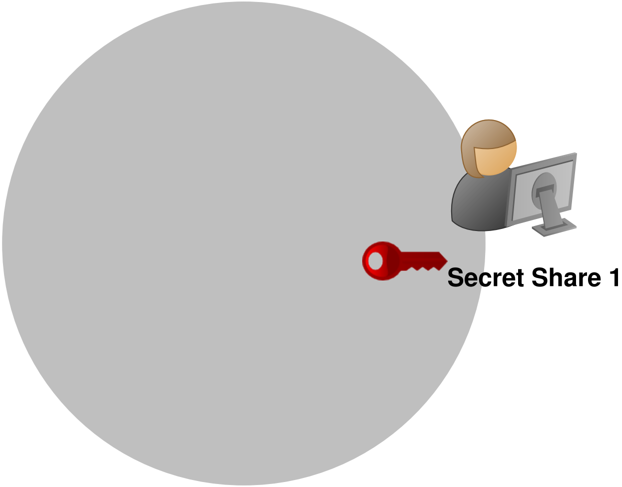
- Disaster recovery.
- Raise the bar for an adversary.
- Partition trust.

Why use Secret Sharing?

- Disaster recovery.
- Raise the bar for an adversary.
- Partition trust.

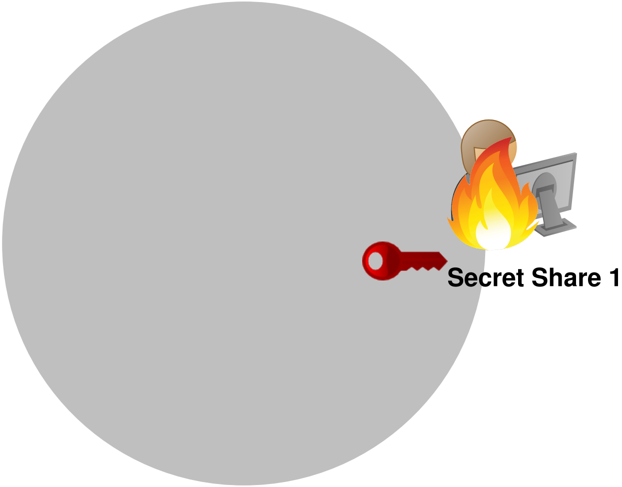
(t, n)-Threshold Secret Sharing

$(2, 3)$-Threshold Scheme Example



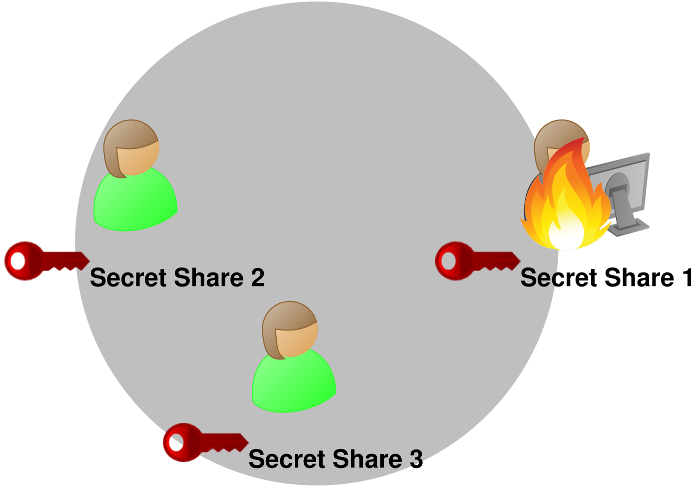
(t, n)-Threshold Secret Sharing

$(2, 3)$-Threshold Scheme Example



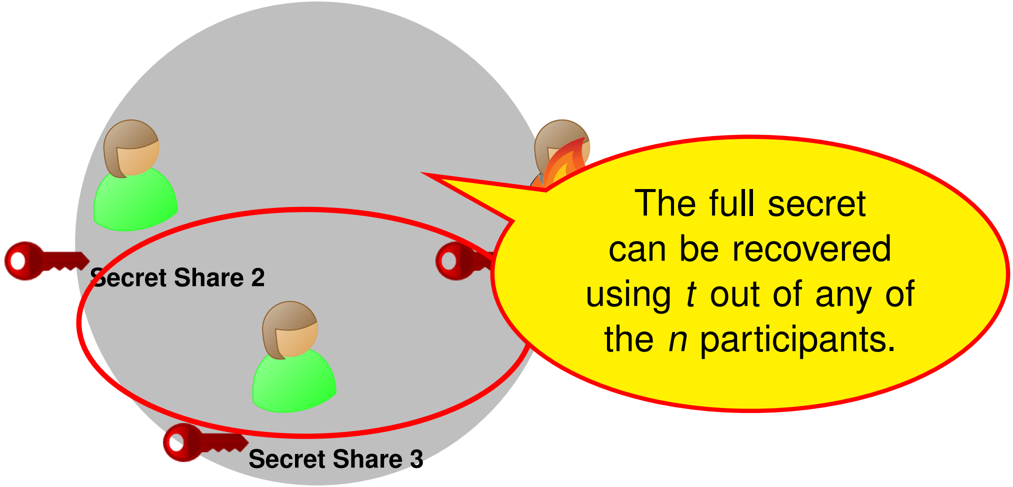
(t, n)-Threshold Secret Sharing

$(2, 3)$-Threshold Scheme Example



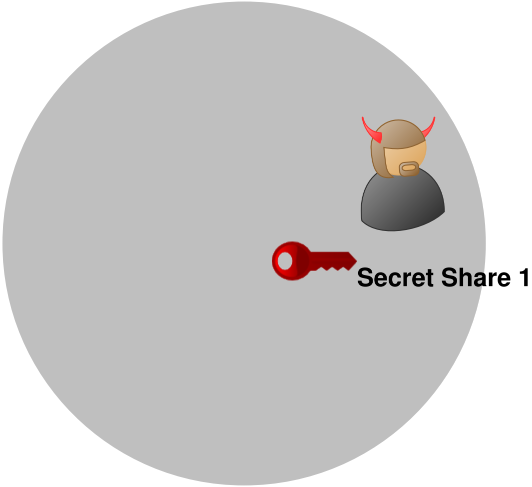
(t, n)-Threshold Secret Sharing

(2, 3)-Threshold Scheme Example



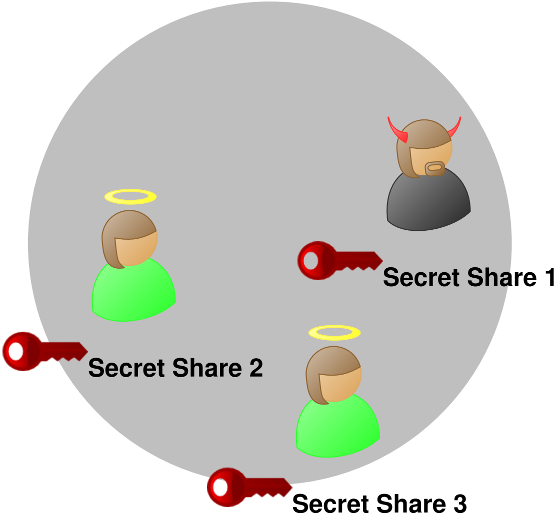
(t, n)-Threshold Secret Sharing

$(2, 3)$-Threshold Scheme Example



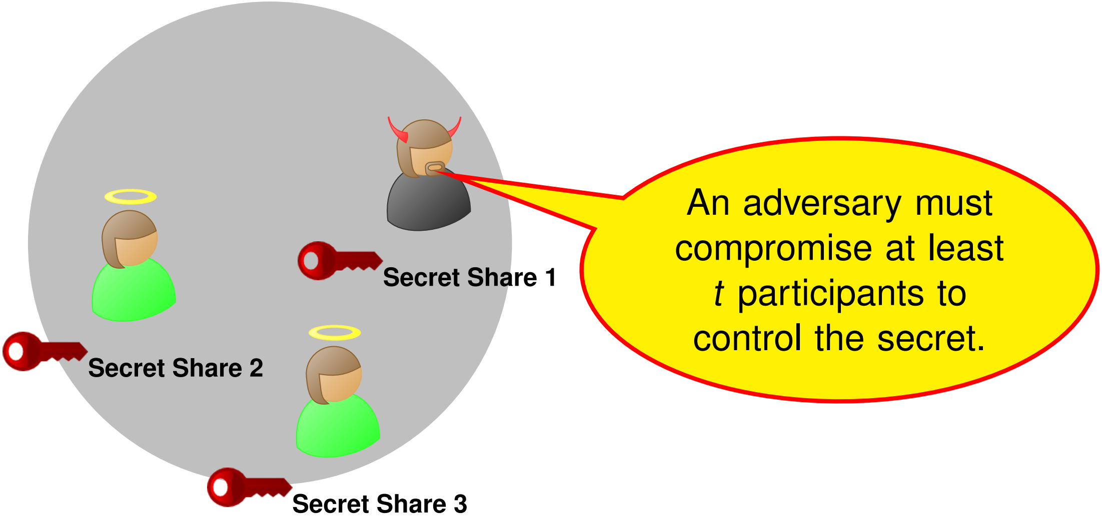
(t, n)-Threshold Secret Sharing

$(2, 3)$-Threshold Scheme Example



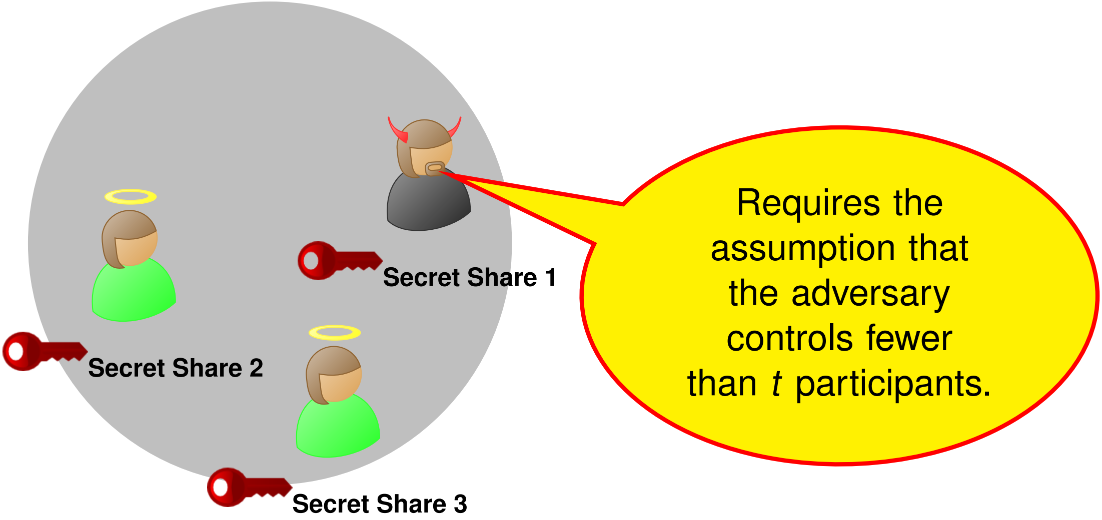
(t, n)-Threshold Secret Sharing

$(2, 3)$-Threshold Scheme Example

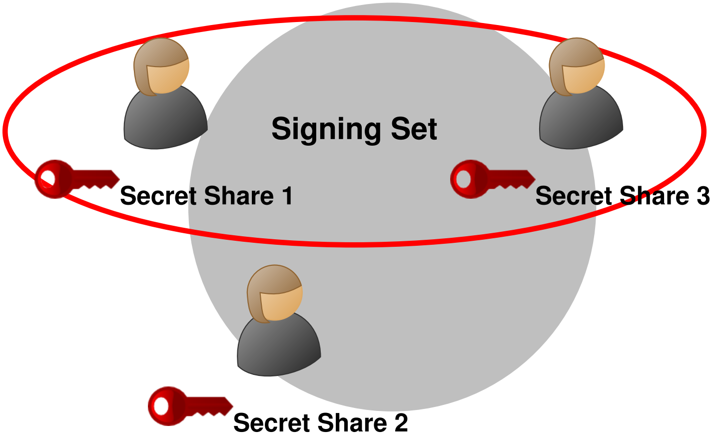


(t, n)-Threshold Secret Sharing

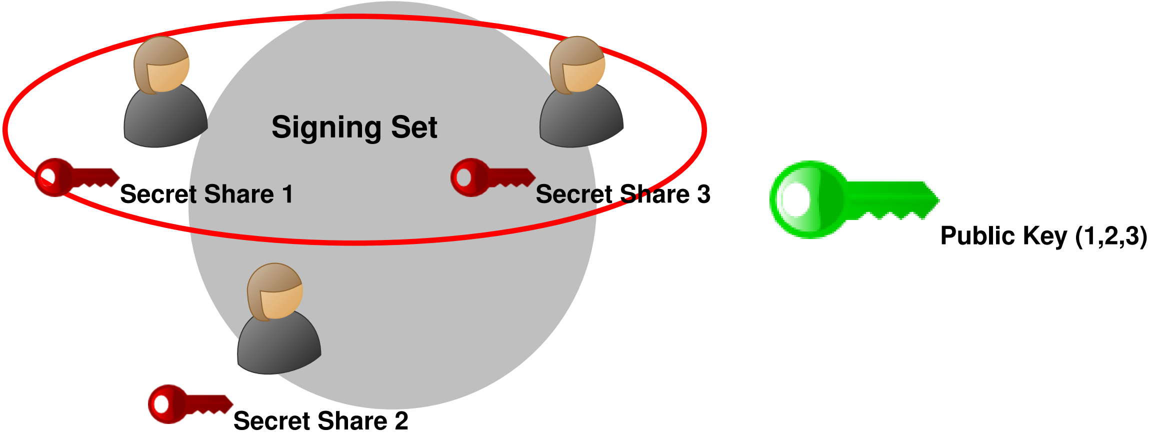
$(2, 3)$-Threshold Scheme Example



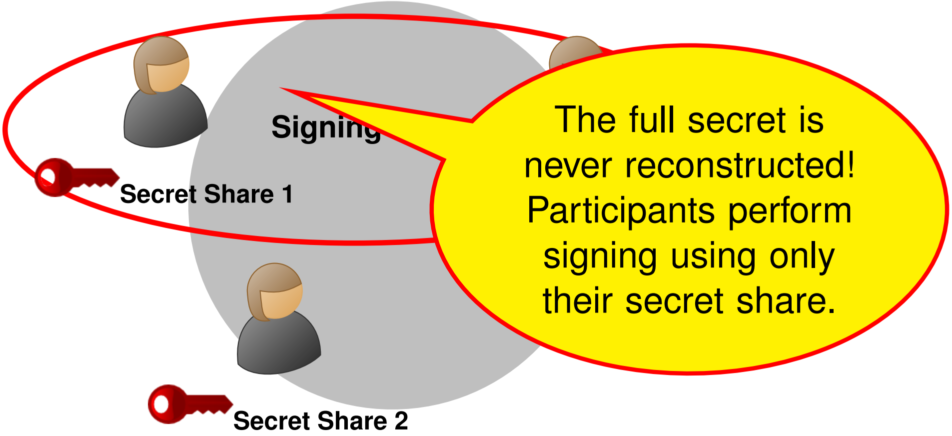
Threshold Signatures: Joint Public Key, Secret-Shared Private Key



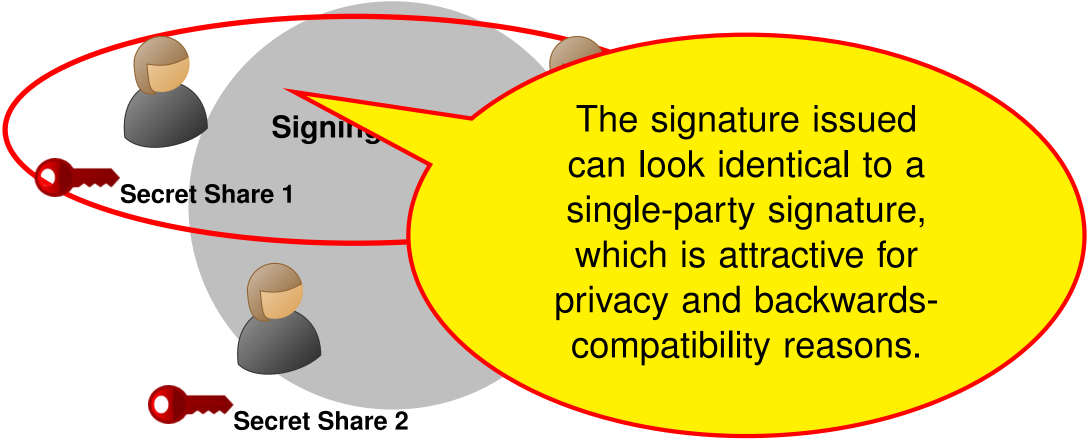
Threshold Signatures: Joint Public Key, Secret-Shared Private Key



Threshold Signatures: Joint Public Key, Secret-Shared Private Key



Threshold Signatures: Joint Public Key, Secret-Shared Private Key



Applications of Threshold Signing

- Issuance of certificates by certificate authorities.
- Distribution of Tor's consensus by directory authorities.
- Authorization of financial transactions.
- In general, distributed authentication that requires some minimum number of signers.

Applications of Threshold Signing

- Issuance of certificates by certificate authorities.
- Distribution of Tor's consensus by directory authorities.
- Authorization of financial transactions.
- In general, distributed authentication that requires some minimum number of signers.

Applications of Threshold Signing

- Issuance of certificates by certificate authorities.
- Distribution of Tor's consensus by directory authorities.
- Authorization of financial transactions.
- In general, distributed authentication that requires some minimum number of signers.

Applications of Threshold Signing

- Issuance of certificates by certificate authorities.
- Distribution of Tor's consensus by directory authorities.
- Authorization of financial transactions.
- In general, distributed authentication that requires some minimum number of signers.

Comparison to Multisignature Schemes

- A multisignature is a compact representation of n signatures over some message.
- (n, n) as opposed to (t, n).
- Each signer has their own public/private keypair (non-interactive key generation).

Comparison to Multisignature Schemes

- A multisignature is a compact representation of n signatures over some message.
- (n, n) as opposed to (t, n).
- Each signer has their own public/private keypair (non-interactive key generation).

Comparison to Multisignature Schemes

- A multisignature is a compact representation of n signatures over some message.
- (n, n) as opposed to (t, n).
- Each signer has their own public/private keypair (non-interactive key generation).

Tradeoffs Among Constructions

- **Number of Signing Rounds:** Required network rounds to generate one signature.
- **Robust:** Can the protocol complete when participants misbehave?
- **Parallel Secure:** Can signing operations be done in parallel without a reduction in security (Drijvers attack)?

Tradeoffs Among Constructions

- **Number of Signing Rounds:** Required network rounds to generate one signature.
- **Robust:** Can the protocol complete when participants misbehave?
- **Parallel Secure:** Can signing operations be done in parallel without a reduction in security (Drijvers attack)?

Tradeoffs Among Constructions

- **Number of Signing Rounds:** Required network rounds to generate one signature.
- **Robust:** Can the protocol complete when participants misbehave?
- **Parallel Secure:** Can signing operations be done in parallel without a reduction in security (Drijvers attack)?

Contributions of FROST

- Two-round threshold signing protocol, or single-round protocol with preprocessing
- Signing operations are secure when performed concurrently.
- Signing can be performed with a threshold t number of signers, where t can be less than the number of possible signers n.
- Secure against an adversary that controls up to $t - 1$ signers.

Contributions of FROST

- Two-round threshold signing protocol, or single-round protocol with preprocessing
- Signing operations are secure when performed concurrently.
- Signing can be performed with a threshold t number of signers, where t can be less than the number of possible signers n.
- Secure against an adversary that controls up to $t - 1$ signers.

Contributions of FROST

- Two-round threshold signing protocol, or single-round protocol with preprocessing
- Signing operations are secure when performed concurrently.
- Signing can be performed with a threshold t number of signers, where t can be less than the number of possible signers n.
- Secure against an adversary that controls up to $t - 1$ signers.

Contributions of FROST

- Two-round threshold signing protocol, or single-round protocol with preprocessing
- Signing operations are secure when performed concurrently.
- Signing can be performed with a threshold t number of signers, where t can be less than the number of possible signers n.
- Secure against an adversary that controls up to $t - 1$ signers.

Single-Party Schnorr Signing and Verification

Signer

$(x, Y) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad (m, Y) \quad}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer

$(x, Y) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{(m, Y)}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{(m, \sigma = (R, z))}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer

$(x, Y) \leftarrow KeyGen()$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$\xleftarrow{(m, Y)}$

$\xrightarrow{(m, \sigma = (R, z))}$

Verifier

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer

$(x, Y) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad (m, Y) \quad}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer **Verifier**

$(x, Y) \leftarrow KeyGen()$

$\xleftarrow{(m, Y)}$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$\xrightarrow{(m, \sigma = (R, z))}$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer | **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{(m, Y)}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{(m, \sigma = (R, z))}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer | **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{\quad (m, Y) \quad}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer		**Verifier**
$(x, Y) \leftarrow KeyGen()$		
	$\xleftarrow{(m, Y)}$	
$k \xleftarrow{\$} \mathbb{Z}_q$		
$R = g^k \in \mathbb{G}$		
$c = H(R, Y, m)$		
$z = k + c \cdot x$		
	$\xrightarrow{(m, \sigma = (R, z))}$	
		$c = H(R, Y, m)$
		$R' = g^z \cdot Y^{-c}$
		Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{\quad (m, Y) \quad}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer		**Verifier**
$(x, Y) \leftarrow KeyGen()$		
	$\xleftarrow{\quad (m, Y) \quad}$	
$k \xleftarrow{\$} \mathbb{Z}_q$ $R = g^k \in \mathbb{G}$ $c = H(R, Y, m)$ $z = k + c \cdot x$		
	$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$	
		$c = H(R, Y, m)$ $R' = g^z \cdot Y^{-c}$ Output $R \stackrel{?}{=} R'$

FROST Keygen

- Can be performed by either a trusted dealer or a Distributed Key Generation (DKG) Protocol
- Trusted dealer is simply a (t, n) Shamir secret sharing with a randomly generated secret s, where the public key is $Y = g^s$.
- The DKG is an n-wise Shamir Secret Sharing protocol, with each participant acting as a dealer, so that no party learns s.
- After KeyGen, each participant holds secret share s_i and public key Y_i (used for verification during signing) with joint public key Y.

FROST Keygen

- Can be performed by either a trusted dealer or a Distributed Key Generation (DKG) Protocol
- Trusted dealer is simply a (t, n) Shamir secret sharing with a randomly generated secret s, where the public key is $Y = g^s$.
- The DKG is an n-wise Shamir Secret Sharing protocol, with each participant acting as a dealer, so that no party learns s.
- After KeyGen, each participant holds secret share s_i and public key Y_i (used for verification during signing) with joint public key Y.

FROST Keygen

- Can be performed by either a trusted dealer or a Distributed Key Generation (DKG) Protocol
- Trusted dealer is simply a (t, n) Shamir secret sharing with a randomly generated secret s, where the public key is $Y = g^s$.
- The DKG is an n-wise Shamir Secret Sharing protocol, with each participant acting as a dealer, so that no party learns s.
- After KeyGen, each participant holds secret share s_i and public key Y_i (used for verification during signing) with joint public key Y.

FROST Keygen

- Can be performed by either a trusted dealer or a Distributed Key Generation (DKG) Protocol
- Trusted dealer is simply a (t, n) Shamir secret sharing with a randomly generated secret s, where the public key is $Y = g^s$.
- The DKG is an n-wise Shamir Secret Sharing protocol, with each participant acting as a dealer, so that no party learns s.
- After KeyGen, each participant holds secret share s_i and public key Y_i (used for verification during signing) with joint public key Y.

FROST Sign

- Can be performed in two rounds, or optimized to single round with preprocessing
- We show here with a signature aggregator, but can be performed without centralized roles

FROST Sign

- Can be performed in two rounds, or optimized to single round with preprocessing
- We show here with a signature aggregator, but can be performed without centralized roles

FROST Preprocess

Participant i | **Commitment Server**

$((d_{ij}, e_{ij}), \dots) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$

Store $((d_{ij}, D_{ij}), (e_{ij}, E_{ij}), \dots)$

$\xrightarrow{((D_{ij}, E_{ij}), \dots)}$

Store $((D_{ij}, E_{ij}), \dots)$

FROST Preprocess

Participant i **Commitment Server**

$((d_{ij}, e_{ij}), \ldots) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$

Store $((d_{ij}, D_{ij}), (e_{ij}, E_{ij}), \ldots)$

$\xrightarrow{((D_{ij}, E_{ij}), \ldots)}$

Store $((D_{ij}, E_{ij}), \ldots)$

FROST Preprocess

Participant i | **Commitment Server**

$((d_{ij}, e_{ij}), \ldots) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$

Store $((d_{ij}, D_{ij}), (e_{ij}, E_{ij}), \ldots)$

$\xrightarrow{((D_{ij}, E_{ij}), \ldots)}$

Store $((D_{ij}, E_{ij}), \ldots)$

FROST Preprocess

Participant i | **Commitment Server**

$((d_{ij}, e_{ij}), \ldots) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$

Store $((d_{ij}, D_{ij}), (e_{ij}, E_{ij}), \ldots)$

$$\xrightarrow{((D_{ij}, E_{ij}), \ldots)}$$

Store $((D_{ij}, E_{ij}), \ldots)$

FROST Preprocess

Participant i | **Commitment Server**

$((d_{ij}, e_{ij}), \dots) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$

Store $((d_{ij}, D_{ij}), (e_{ij}, E_{ij}), \dots)$

$\xrightarrow{((D_{ij}, E_{ij}), \dots)}$

Store $((D_{ij}, E_{ij}), \dots)$

FROST Preprocess

Participant i **Commitment Server**

$((d_{ij}, e_{ij}), \ldots) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$

Store $((d_{ij}, D_{ij}), (e_{ij}, E_{ij}), \ldots$

In the two-round variant, this step is performed immediately before signing with only one commitment.

Store $((D_{ij}, E_{ij}), \ldots)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \dots, (t, D_t, E_t))$

(m, B)

$\xleftarrow{\hspace{3cm}}$

$\rho_\ell = H_1(\ell, m, B), \ell \in S$

$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$

$c = H_2(R, Y, m)$

$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$

z_i

$\xrightarrow{\hspace{3cm}}$

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

$$\xleftarrow{(m, B)}$$

$\rho_\ell = H_1(\ell, m, B), \ell \in S$

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$$

$c = H_2(R, Y, m)$

$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$

$$\xrightarrow{z_i}$$

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

(m, B) ←

$\rho_\ell = H_1(\ell, m, B), \ell \in S$

$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$

$c = H_2(R, Y, m)$

$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$

“binding value” to bind signing shares to ℓ, m, and B

$\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

$$\xleftarrow{(m, B)}$$

$\rho_\ell = H_1(\ell, m, B), \ell \in S$

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$$

$c = H_2(R, Y, m)$

$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$

$$\xrightarrow{z_i}$$

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

$$\xleftarrow{(m, B)}$$

$\rho_\ell = H_1(\ell, m, B), \ell \in S$

$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$

$c = H_2(R, Y, m)$

$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$

$$\xrightarrow{z_i}$$

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

(m, B)

$\longleftarrow$

$$\rho_\ell = H_1(\ell, m, B), \ell \in S$$

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$$

$$c = H_2(R, Y, m)$$

$$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$$

z_i

$\longrightarrow$

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \dots, (t, D_t, E_t))$

$\xleftarrow{(m, B)}$

$\rho_\ell = H_1(\ell, m, B), \ell \in S$

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$$

$c = H_2(R, Y, m)$

$z_i = d_i + (e_i \cdot \rho_i) \boxed{+ \lambda_i \cdot s_i \cdot c}$

Here, $s = \sum \lambda_i \cdot s_i$, where λ_i is a Lagrange coefficient.

$\xrightarrow{z_i}$

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

(m, B)

$\rho_\ell = H_1(\ell, m, B), \ell \in S$

$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$

$c = H_2(R, Y, m)$

$z_i = \boxed{d_i + (e_i \cdot \rho_i)} + \lambda_i \cdot s_i \cdot c$

This step cannot be inverted by anyone who does not know (d_i, e_i).

z_i

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

$$\xleftarrow{(m, B)}$$

$$\rho_\ell = H_1(\ell, m, B), \ell \in S$$

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$$

$$c = H_2(R, Y, m)$$

$$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$$

$$\xrightarrow{z_i}$$

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

$$\xleftarrow{\quad (m, B) \quad}$$

$$\rho_\ell = H_1(\ell, m, B), \ell \in S$$

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$$

$$c = H_2(R, Y, m)$$

$$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$$

$$\xrightarrow{\quad z_i \quad}$$

Publish $\sigma = (R, z = \sum z_i)$

FROST Sign

Signer i

Signature Aggregator

$B = ((1, D_1, E_1), \ldots, (t, D_t, E_t))$

$\xleftarrow{(m, B)}$

$$\rho_\ell = H_1(\ell, m, B), \ell \in S$$

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$$

$$c = H_2(R, Y, m)$$

$$z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$$

$\xrightarrow{z_i}$

Signature format and verification are identical to single-party Schnorr.

Publish $\sigma = (R, z = \sum z_i)$

Security against Drijvers

Without $\rho_\ell = H_1(\ell, m, B)$, an adversary could produce a c^* such that:

$$c^* = H(R^*, Y, m^*) = \sum_{i=1}^{k} H(R_i, Y, m_i) = \sum c_i \text{ for some } (R_i, m_i), \ldots$$

,

After sending receiving the victim's z_i for each (R_i, m_i), the adversary can produce a valid forgery $\sigma^* = (R^*, z)$, as

$$z = \sum d_i + e_i + \lambda_t \cdot s_t \cdot \sum c_i = \sum d_i + e_i + \lambda_t \cdot s_t \cdot c^*$$

The binding factor in FROST makes each z_i strongly tied to (m_i, R_i).

$$z = \sum d_i + (e_i * \rho_i) + \lambda_t \cdot s_t \cdot \sum c_i$$

Resulting in an invalid signature: $R^* \neq g^z \cdot Y^{-c}$

Security against Drijvers

Without $\rho_\ell = H_1(\ell, m, B)$, an adversary could produce a c^* such that:

$$c^* = H(R^*, Y, m^*) = \sum_{i=1}^{k} H(R_i, Y, m_i) = \sum c_i \text{ for some } (R_i, m_i), \ldots$$

,

After sending receiving the victim's z_i for each (R_i, m_i), the adversary can produce a valid forgery $\sigma^* = (R^*, z)$, as

$$z = \sum d_i + e_i + \lambda_t \cdot s_t \cdot \sum c_i = \sum d_i + e_i + \lambda_t \cdot s_t \cdot c^*$$

The binding factor in FROST makes each z_i strongly tied to (m_i, R_i).

$$z = \sum d_i + (e_i * \rho_i) + \lambda_t \cdot s_t \cdot \sum c_i$$

Resulting in an invalid signature: $R^* \neq g^z \cdot Y^{-c}$

Security against Drijvers

Without $\rho_\ell = H_1(\ell, m, B)$, an adversary could produce a c^* such that:

$$c^* = H(R^*, Y, m^*) = \sum_{i=1}^{k} H(R_i, Y, m_i) = \sum c_i \text{ for some } (R_i, m_i), \ldots$$

,

After sending receiving the victim's z_i for each (R_i, m_i), the adversary can produce a valid forgery $\sigma^* = (R^*, z)$, as

$$z = \sum d_i + e_i + \lambda_t \cdot s_t \cdot \sum c_i = \sum d_i + e_i + \lambda_t \cdot s_t \cdot c^*$$

The binding factor in FROST makes each z_i strongly tied to (m_i, R_i).

$$z = \sum d_i + (e_i * \rho_i) + \lambda_t \cdot s_t \cdot \sum c_i$$

Resulting in an invalid signature: $R^* \neq g^z \cdot Y^{-c}$

Security against Drijvers

Without $\rho_\ell = H_1(\ell, m, B)$, an adversary could produce a c^* such that:

$$c^* = H(R^*, Y, m^*) = \sum_{i=1}^{k} H(R_i, Y, m_i) = \sum c_i \text{ for some } (R_i, m_i), \ldots$$

,

After sending receiving the victim's z_i for each (R_i, m_i), the adversary can produce a valid forgery $\sigma^* = (R^*, z)$, as

$$z = \sum d_i + e_i + \lambda_t \cdot s_t \cdot \sum c_i = \sum d_i + e_i + \lambda_t \cdot s_t \cdot c^*$$

The binding factor in FROST makes each z_i strongly tied to (m_i, R_i).

$$z = \sum d_i + (e_i * \rho_i) + \lambda_t \cdot s_t \cdot \sum c_i$$

Resulting in an invalid signature: $R^* \neq g^z \cdot Y^{-c}$

Security against Drijvers

Without $\rho_\ell = H_1(\ell, m, B)$, an adversary could produce a c^* such that:

$$c^* = H(R^*, Y, m^*) = \sum_{i=1}^{k} H(R_i, Y, m_i) = \sum c_i \text{ for some } (R_i, m_i), \ldots$$

,

After sending receiving the victim's z_i for each (R_i, m_i), the adversary can produce a valid forgery $\sigma^* = (R^*, z)$, as

$$z = \sum d_i + e_i + \lambda_t \cdot s_t \cdot \sum c_i = \sum d_i + e_i + \lambda_t \cdot s_t \cdot c^*$$

The binding factor in FROST makes each z_i strongly tied to (m_i, R_i).

$$z = \sum d_i + (e_i * \rho_i) + \lambda_t \cdot s_t \cdot \sum c_i$$

Resulting in an invalid signature: $R^* \neq g^z \cdot Y^{-c}$

Security against Drijvers

Without $\rho_\ell = H_1(\ell, m, B)$, an adversary could produce a c^* such that:

$$c^* = H(R^*, Y, m^*) = \sum_{i=1}^{k} H(R_i, Y, m_i) = \sum c_i \text{ for some } (R_i, m_i), \ldots$$

,

After sending receiving the victim's z_i for each (R_i, m_i), the adversary can produce a valid forgery $\sigma^* = (R^*, z)$, as

$$z = \sum d_i + e_i + \lambda_t \cdot s_t \cdot \sum c_i = \sum d_i + e_i + \lambda_t \cdot s_t \cdot c^*$$

The binding factor in FROST makes each z_i strongly tied to (m_i, R_i).

$$z = \sum d_i + (e_i * \rho_i) + \lambda_t \cdot s_t \cdot \sum c_i$$

Resulting in an invalid signature: $R^* \neq g^z \cdot Y^{-c}$

Security against Drijvers

Without $\rho_\ell = H_1(\ell, m, B)$, an adversary could produce a c^* such that:

$$c^* = H(R^*, Y, m^*) = \sum_{i=1}^{k} H(R_i, Y, m_i) = \sum c_i \text{ for some } (R_i, m_i), \ldots$$

,

After sending receiving the victim's z_i for each (R_i, m_i), the adversary can produce a valid forgery $\sigma^* = (R^*, z)$, as

$$z = \sum d_i + e_i + \lambda_t \cdot s_t \cdot \sum c_i = \sum d_i + e_i + \lambda_t \cdot s_t \cdot c^*$$

The binding factor in FROST makes each z_i strongly tied to (m_i, R_i).

$$z = \sum d_i + (e_i * \rho_i) + \lambda_t \cdot s_t \cdot \sum c_i$$

Resulting in an invalid signature: $R^* \neq g^z \cdot Y^{-c}$

RedJubjub FROST

- Finished an implementation of RedJubjub FROST!
- Audited by the Taurus Group, no major issues found; we have fixed all minor issues.
- Just finished work to serialize/deserialize FROST messages for network transmission/saving key material to disk.
- The entire Zcash Foundation team has contributed here, so big thanks to them.

RedJubjub FROST

- Finished an implementation of RedJubjub FROST!
- Audited by the Taurus Group, no major issues found; we have fixed all minor issues.
- Just finished work to serialize/deserialize FROST messages for network transmission/saving key material to disk.
- The entire Zcash Foundation team has contributed here, so big thanks to them.

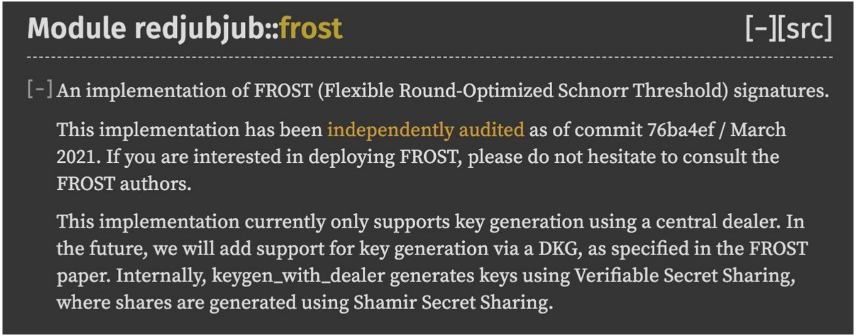
RedJubjub FROST

- Finished an implementation of RedJubjub FROST!
- Audited by the Taurus Group, no major issues found; we have fixed all minor issues.
- Just finished work to serialize/deserialize FROST messages for network transmission/saving key material to disk.
- The entire Zcash Foundation team has contributed here, so big thanks to them.

RedJubjub FROST

- Finished an implementation of RedJubjub FROST!
- Audited by the Taurus Group, no major issues found; we have fixed all minor issues.
- Just finished work to serialize/deserialize FROST messages for network transmission/saving key material to disk.
- The entire Zcash Foundation team has contributed here, so big thanks to them.

RedJubjub FROST



Module redjubjub::frost [-][src]

[-] An implementation of FROST (Flexible Round-Optimized Schnorr Threshold) signatures.

This implementation has been independently audited as of commit 76ba4ef / March 2021. If you are interested in deploying FROST, please do not hesitate to consult the FROST authors.

This implementation currently only supports key generation using a central dealer. In the future, we will add support for key generation via a DKG, as specified in the FROST paper. Internally, keygen_with_dealer generates keys using Verifiable Secret Sharing, where shares are generated using Shamir Secret Sharing.

See crates.io/crates/redjubjub/0.3.0

Next Steps: Unlinkable FROST Signatures

- Currently, the Zcash protocol achieves transaction unlinkability via *rerandomizing* signatures.
- With collaboration with Daira, Deirdre, Jack and Sean, we have a “trusted prover” variant.
- Later this year we will release a deterministic variant that does not require a trusted, distinct prover.

Next Steps: Unlinkable FROST Signatures

- Currently, the Zcash protocol achieves transaction unlinkability via *rerandomizing* signatures.
- With collaboration with Daira, Deirdre, Jack and Sean, we have a "trusted prover" variant.
- Later this year we will release a deterministic variant that does not require a trusted, distinct prover.

Next Steps: Unlinkable FROST Signatures

- Currently, the Zcash protocol achieves transaction unlinkability via *rerandomizing* signatures.
- With collaboration with Daira, Deirdre, Jack and Sean, we have a “trusted prover” variant.
- Later this year we will release a deterministic variant that does not require a trusted, distinct prover.

Takeaways

- FROST defines a threshold signing protocol that is secure even when signing is performed concurrently.
- RedJubjub-FROST is ready for use.
- Unlinkable RedJubjub-FROST coming soon!

Takeaways

- FROST defines a threshold signing protocol that is secure even when signing is performed concurrently.
- RedJubjub-FROST is ready for use.
- Unlinkable RedJubjub-FROST coming soon!

Takeaways

- FROST defines a threshold signing protocol that is secure even when signing is performed concurrently.
- RedJubjub-FROST is ready for use.
- Unlinkable RedJubjub-FROST coming soon!