FROST: Flexible Round-Optimized Schnorr Threshold Signatures and Extensibility to EdDSA

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

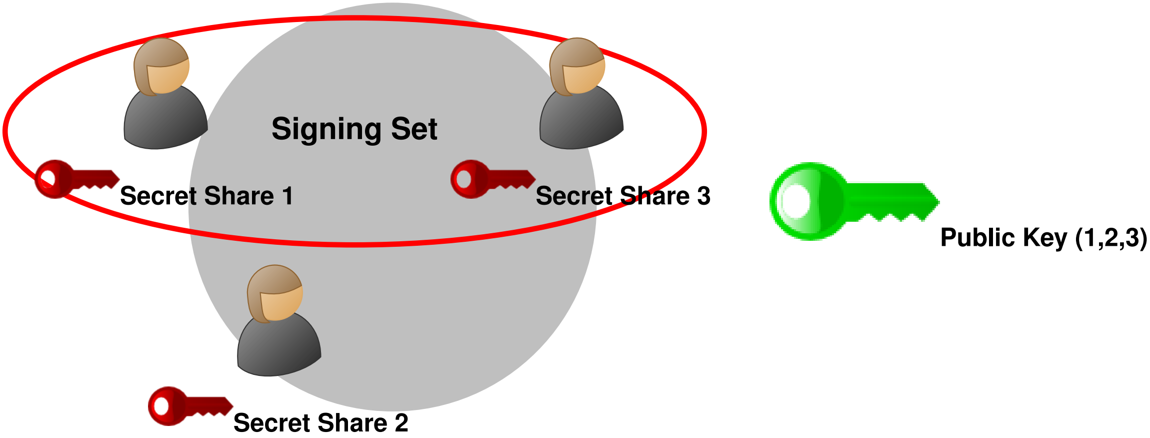
[1] University of Waterloo [2] Zcash Foundation

NIST Workshop on Multi-Party Threshold Schemes, November 2020

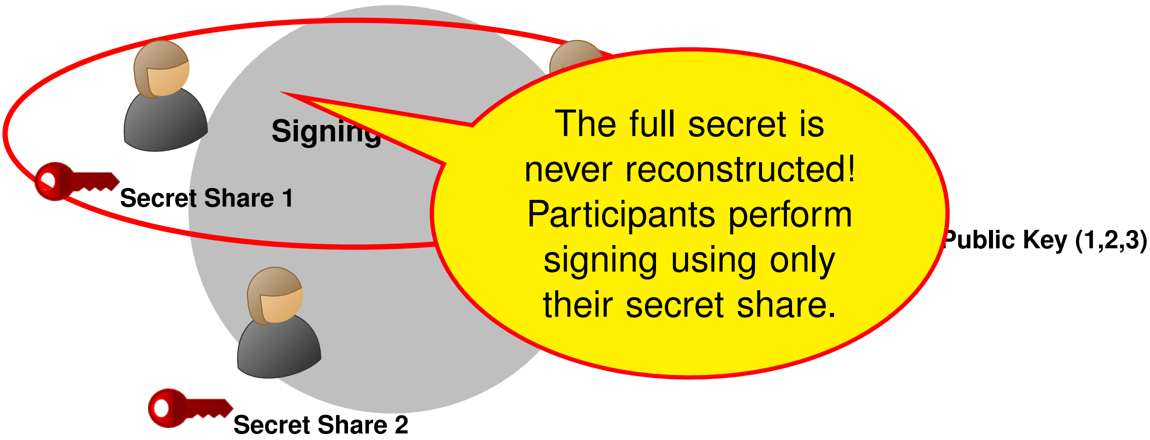
CrySP
Cryptography, Security, and Privacy
— Research Group @ uWaterloo —

ZF

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



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



Contributions of FROST

- Two-round Schnorr threshold signing protocol, or single-round with preprocessing
- Signing operations are secure when performed concurrently, improving upon prior similar schemes.
- 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 Schnorr threshold signing protocol, or single-round with preprocessing
- Signing operations are secure when performed concurrently, improving upon prior similar schemes.
- 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 Schnorr threshold signing protocol, or single-round with preprocessing
- Signing operations are secure when performed concurrently, improving upon prior similar schemes.
- 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 Schnorr threshold signing protocol, or single-round with preprocessing
- Signing operations are secure when performed concurrently, improving upon prior similar schemes.
- 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.

Tradeoffs Among Constructions

- **Number of Signing Rounds:** Required network rounds to generate one signature.
- **Robust:** Can the protocol complete when participants misbehave?
- **Required Number of Signers:** Can a signature be created by just t participants, or are all n needed?
- **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?
- **Required Number of Signers:** Can a signature be created by just t participants, or are all n needed?
- **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?
- **Required Number of Signers:** Can a signature be created by just t participants, or are all n needed?
- **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?
- **Required Number of Signers:** Can a signature be created by just t participants, or are all n needed?
- **Parallel Secure:** Can signing operations be done in parallel without a reduction in security (Drijvers attack)?

Tradeoffs Among Constructions

	Num. Rounds	**Robust**	**Num. Signers**	**Parallel Secure**
Stinson Strobl	4	Yes	t	Yes
Gennaro et al.	1 w/ preprocessing	No	n	No
FROST	1 w/ preprocessing	No	t	Yes

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

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

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

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

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

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

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

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

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**

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

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

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

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

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

$z = k + c \cdot x$

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

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

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

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

FROST Keygen

- Can be performed by either a trusted dealer or a Distributed Key Generation (DKG) Protocol
- The DKG is an n-wise Shamir Secret Sharing protocol, with each participant acting as a dealer
- 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
- The DKG is an *n*-wise Shamir Secret Sharing protocol, with each participant acting as a dealer
- 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
- The DKG is an n-wise Shamir Secret Sharing protocol, with each participant acting as a dealer
- 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
- Centralized roles are used for coordination and don't have access to privilaged information; trusted to not perform a denial-of-service.

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
- Centralized roles are used for coordination and don't have access to privilaged information; trusted to not perform a denial-of-service.

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
- Centralized roles are used for coordination and don't have access to privilaged information; trusted to not perform a denial-of-service.

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

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

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

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

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

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

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

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

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

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

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

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

FROST Sign

Signer i

Signature Aggregator

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

(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

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

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

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

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

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

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

$$\xrightarrow{z_i}$$

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

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

$\xleftarrow{(m, B)}$

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

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

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

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

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

$\xrightarrow{z_i}$

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

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

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

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

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

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

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

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

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

Protocol Complexity

- Per-signer bandwidth overhead for signing scales linearly relative to the number of signers (because of B).
- Total bandwidth overhead scales quadratically
- Network round complexity remains constant, assuming centralized commitment storage and signature aggregation

Protocol Complexity

- Per-signer bandwidth overhead for signing scales linearly relative to the number of signers (because of B).
- Total bandwidth overhead scales quadratically
- Network round complexity remains constant, assuming centralized commitment storage and signature aggregation

Protocol Complexity

- Per-signer bandwidth overhead for signing scales linearly relative to the number of signers (because of B).
- Total bandwidth overhead scales quadratically
- Network round complexity remains constant, assuming centralized commitment storage and signature aggregation

FROST Compatibility with EdDSA

- **Signature Verification:** FROST can produce non-deterministic signatures comptabible with EdDSA verification.
- **Deterministic Signatures:** Deriving the nonce via a hash of the secret key and message is *not* secure for schemes with non-interactive nonce generation (FROST, Gennaro et al., MuSig, etc).

FROST Compatibility with EdDSA

- **Signature Verification:** FROST can produce non-deterministic signatures comptabible with EdDSA verification.
- **Deterministic Signatures:** Deriving the nonce via a hash of the secret key and message is *not* secure for schemes with non-interactive nonce generation (FROST, Gennaro et al., MuSig, etc).

EdDSA-Style Determinism is not Straightforward in a Threshold Setting

- **Complexity:** To safely ensure determinism, additional factors beyond each participant's secret and the message would be required (such as a counter), but increases complexity.

- **Statefulness is Required, Regardless:** Even in a setting where determinism is possible, state must be maintained by signers between rounds.

EdDSA-Style Determinism is not Straightforward in a Threshold Setting

- **Complexity:** To safely ensure determinism, additional factors beyond each participant's secret and the message would be required (such as a counter), but increases complexity.

- **Statefulness is Required, Regardless:** Even in a setting where determinism is possible, state must be maintained by signers between rounds.

FROST Network Requirements

- KeyGen requires a trusted, authenticated channel for distributing secret shares.
- Signing can be performed over a trustless public channel.
- We assume a reliable enough network connection to successfully complete the protocol with at least t signers.

FROST Network Requirements

- KeyGen requires a trusted, authenticated channel for distributing secret shares.
- Signing can be performed over a trustless public channel.
- We assume a reliable enough network connection to successfully complete the protocol with at least t signers.

FROST Network Requirements

- KeyGen requires a trusted, authenticated channel for distributing secret shares.
- Signing can be performed over a trustless public channel.
- We assume a reliable enough network connection to successfully complete the protocol with at least t signers.

Real-World Applications

- Use in cryptocurrency (Zcash) protocols for signing transactions
- Consideration for standardization by CFRG.

Real-World Applications

- Use in cryptocurrency (Zcash) protocols for signing transactions
- Consideration for standardization by CFRG.

Takeaways

- FROST improves upon prior schemes by defining a single-round threshold signing protocol (with preprocessing) that is secure in a parallelized setting.
- The simplicity and flexibility of FROST makes it attractive to real-world applications.
- Determinism should be a recommendation, not a requirement for threshold signatures, as it requires statefulness and increased complexity.

Find our paper and artifact at https://crysp.uwaterloo.ca/software/frost.

Takeaways

- FROST improves upon prior schemes by defining a single-round threshold signing protocol (with preprocessing) that is secure in a parallelized setting.
- The simplicity and flexibility of FROST makes it attractive to real-world applications.
- Determinism should be a recommendation, not a requirement for threshold signatures, as it requires statefulness and increased complexity.

Find our paper and artifact at https://crysp.uwaterloo.ca/software/frost.

Takeaways

- FROST improves upon prior schemes by defining a single-round threshold signing protocol (with preprocessing) that is secure in a parallelized setting.
- The simplicity and flexibility of FROST makes it attractive to real-world applications.
- Determinism should be a recommendation, not a requirement for threshold signatures, as it requires statefulness and increased complexity.

Find our paper and artifact at https://crysp.uwaterloo.ca/software/frost.

Extras: 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: $R^* \neq g^z \cdot Y^{-c}$

Extras: 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: $R^* \neq g^z \cdot Y^{-c}$

Extras: 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: $R^* \neq g^z \cdot Y^{-c}$

Extras: 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: $R^* \neq g^z \cdot Y^{-c}$

Extras: 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: $R^* \neq g^z \cdot Y^{-c}$

Extras: 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: $R^* \neq g^z \cdot Y^{-c}$

Extras: 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: $R^* \neq g^z \cdot Y^{-c}$

Extras: Provable Security

- We prove the EUF-CMA security of an interactive variant of FROST, then extend to plain FROST.

- FROST-Interactive generates the binding value ρ_i via a one-time VRF to allow for parallelism in our simulator.

- Recall that plain (non-interactive) FROST generates ρ_i via a hash function.

Extras: Provable Security

- We prove the EUF-CMA security of an interactive variant of FROST, then extend to plain FROST.
- FROST-Interactive generates the binding value ρ_i via a one-time VRF to allow for parallelism in our simulator.
- Recall that plain (non-interactive) FROST generates ρ_i via a hash function.

Extras: Provable Security

- We prove the EUF-CMA security of an interactive variant of FROST, then extend to plain FROST.
- FROST-Interactive generates the binding value ρ_i via a one-time VRF to allow for parallelism in our simulator.
- Recall that plain (non-interactive) FROST generates ρ_i via a hash function.

Sign(m) $\rightarrow (m, \sigma)$

1. For each $i \in S$, $\mathcal{SA}$ sends P_i (m, B).
2. Each P_i validates m, and then checks $D_\ell, E_\ell \in \mathbb{G}^*, \forall (D_\ell, E_\ell) \in B$.
4. Each P_i computes $\rho_\ell = H_1(\ell, m, B), \ell \in S$, and derives $R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell}$, and $c = H_2(R, Y, m)$.
5. Each P_i computes $z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$.
6. Each P_i securely deletes $((d_i, D_i), (e_i, E_i))$ and returns z_i to $\mathcal{SA}$.

7.a $\mathcal{SA}$ re-derives $\rho_i = H_1(i, m, B)$ and $R_i = D_{ij} \cdot (E_{ij})^{\rho_i}$ for $i \in S$, and subsequently $R = \prod_{i \in S} R_i$ and $c = H_2(R, Y, m)$.

7.b $\mathcal{SA}$ verifies each response by checking $g^{z_i} \stackrel{?}{=} R_i \cdot Y_i{}^{c \cdot \lambda_i}$ for each signing share z_i, $i \in S$, aborting/reporting if the equality does not hold. If the equality does not hold, identify and report the misbehaving

7.c $\mathcal{SA}$ computes $z = \sum z_i$ and publishes $\sigma = (R, z)$ along with m.