Multi-Party Signatures for Discrete-Log Based Cryptosystems

Chelsea Komlo, University of Waterloo, Zcash Foundation

University of Maryland, April 28, 2022

About Me

- Ph.D Candidate at the University of Waterloo, part-time at the Zcash Foundation.
- My work is largely focused on multi-party zero knowledge proofs of knowledge.

Overview

- Review of single-party discrete-log based signatures.
- Introduction to threshold signatures.
- How well do these schemes map to a threshold or multisignature setting?

Overview

- Review of single-party discrete-log based signatures.
- Introduction to threshold signatures.
- How well do these schemes map to a threshold or multisignature setting?

Overview

- Review of single-party discrete-log based signatures.
- Introduction to threshold signatures.
- How well do these schemes map to a threshold or multisignature setting?

Single-Party Signatures

Sigma Protocol

- A sigma proof of knowledge is a three-move protocol, where:

1. The prover is initialized with a witness k, and a challenger with the statement K.
2. The prover begins by issuing a public *commitment* R to another witness r.
3. The challenger then *sends* a challenge c.
4. The prover then *outputs* a response z.
5. The challenger then verifies that the prover indeed knows k corresponding to K, using (R, z, c).

Sigma Protocol

- A sigma proof of knowledge is a three-move protocol, where:

1. The prover is initialized with a witness k, and a challenger with the statement K.
2. The prover begins by issuing a public *commitment* R to another witness r.
3. The challenger then *sends* a challenge c.
4. The prover then *outputs* a response z.
5. The challenger then verifies that the prover indeed knows k corresponding to K, using (R, z, c).

Sigma Protocol

- A sigma proof of knowledge is a three-move protocol, where:

1. The prover is initialized with a witness k, and a challenger with the statement K.
2. The prover begins by issuing a public *commitment* R to another witness r.
3. The challenger then *sends* a challenge c.
4. The prover then *outputs* a response z.
5. The challenger then verifies that the prover indeed knows k corresponding to K, using (R, z, c).

Sigma Protocol

- A sigma proof of knowledge is a three-move protocol, where:

1. The prover is initialized with a witness k, and a challenger with the statement K.
2. The prover begins by issuing a public *commitment* R to another witness r.
3. The challenger then *sends* a challenge c.
4. The prover then *outputs* a response z.
5. The challenger then verifies that the prover indeed knows k corresponding to K, using (R, z, c).

Sigma Protocol

- A sigma proof of knowledge is a three-move protocol, where:

1. The prover is initialized with a witness k, and a challenger with the statement K.
2. The prover begins by issuing a public *commitment* R to another witness r.
3. The challenger then *sends* a challenge c.
4. The prover then *outputs* a response z.
5. The challenger then verifies that the prover indeed knows k corresponding to K, using (R, z, c).

Sigma Protocol

- A sigma proof of knowledge is a three-move protocol, where:

1. The prover is initialized with a witness k, and a challenger with the statement K.
2. The prover begins by issuing a public *commitment* R to another witness r.
3. The challenger then *sends* a challenge c.
4. The prover then *outputs* a response z.
5. The challenger then verifies that the prover indeed knows k corresponding to K, using (R, z, c).

Identification Protocol: Prove Knowledge of *sk*

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow\$ \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{R}$	
		$c \leftarrow\$ \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{z}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Identification Protocol: Prove Knowledge of sk

Prover | **Challenger**

$(sk, PK) \leftarrow KeyGen()$

$r \leftarrow\$ \mathbb{Z}_q$

$R \leftarrow g^r \in \mathbb{G}$

$\xrightarrow{R}$

$c \leftarrow\$ \mathbb{Z}_q$

$\xleftarrow{c}$

$z \leftarrow r + c \cdot sk$

$\xrightarrow{z}$

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Identification Protocol: Prove Knowledge of *sk*

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow\$ \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{R}$	
		$c \leftarrow\$ \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{z}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Identification Protocol: Prove Knowledge of *sk*

Prover

$(sk, PK) \leftarrow KeyGen()$

$r \leftarrow\!\!\$\ \mathbb{Z}_q$

$R \leftarrow g^r \in \mathbb{G}$

$\xrightarrow{\quad R \quad}$

Challenger

$c \leftarrow\!\!\$\ \mathbb{Z}_q$

$\xleftarrow{\quad c \quad}$

$z \leftarrow r + c \cdot sk$

$\xrightarrow{\quad z \quad}$

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Identification Protocol: Prove Knowledge of *sk*

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow\!\$\ \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{R}$	
		$c \leftarrow\!\$\ \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{z}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Identification Protocol: Prove Knowledge of sk

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow\$ \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{R}$	
		$c \leftarrow\$ \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{z}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Identification Protocol: Prove Knowledge of *sk*

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow\$ \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{R}$	
		$c \leftarrow\$ \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{z}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Identification Protocol: Prove Knowledge of *sk*

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow\$ \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{R}$	
		$c \leftarrow\$ \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{z}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Identification Protocol: Prove Knowledge of *sk*

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow\$ \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{R}$	
		$c \leftarrow\$ \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{z}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Identification Protocol: Prove Knowledge of *sk*

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow \$ \, \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{R}$	
		$c \leftarrow \$ \, \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{z}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Identification Protocol: Prove Knowledge of sk

Prover		**Challenger**
$(sk, PK) \leftarrow KeyGen()$		
$r \leftarrow\$ \mathbb{Z}_q$		
$R \leftarrow g^r \in \mathbb{G}$		
	$\xrightarrow{\quad R \quad}$	
		$c \leftarrow\$ \mathbb{Z}_q$
	$\xleftarrow{\quad c \quad}$	
$z \leftarrow r + c \cdot sk$		
	$\xrightarrow{\quad z \quad}$	
		$R' = g^z \cdot PK^{-c'}$
		Output 1 if $R \stackrel{?}{=} R'$
		Otherwise, output 0

Non-Interactive Schnorr via Fiat-Shamir

- Fiat-Shamir allows a public-coin interactive proof system to be made non-interactive, by generating the challenge instead via a random oracle.

- For Schnorr signatures, the challenge is also bound to the message.

Non-Interactive Schnorr via Fiat-Shamir

- Fiat-Shamir allows a public-coin interactive proof system to be made non-interactive, by generating the challenge instead via a random oracle.
- For Schnorr signatures, the challenge is also bound to the message.

Schnorr Signature Scheme

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$r \leftarrow_{\$} \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$r \leftarrow_{\$} \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow_\$ \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow_\$ \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$r \leftarrow_{\$} \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

Verifier

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow_\$ \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow_{\$} \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow_{\$} \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer

Verifier

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$r \leftarrow_{\$} \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow_{\$} \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Schnorr Signature Scheme

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow_{\$} \mathbb{Z}_q$

$R \leftarrow g^r$

$c \leftarrow H(R, m)$

$z \leftarrow r + c \cdot sk$

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

$c \leftarrow H(R, m)$

$R' \leftarrow g^z \cdot PK^{-c}$

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Digital Signature Algorithm (DSA)

- Schnorr was patented until 2008.
- DSA is a variant on the Schnorr and ElGamal signature schemes, but not patented.
- Adopted by NIST in 1994.
- ECDSA is tailored specifically for elliptic curve groups.

Digital Signature Algorithm (DSA)

- Schnorr was patented until 2008.
- DSA is a variant on the Schnorr and ElGamal signature schemes, but not patented.
- Adopted by NIST in 1994.
- ECDSA is tailored specifically for elliptic curve groups.

Digital Signature Algorithm (DSA)

- Schnorr was patented until 2008.
- DSA is a variant on the Schnorr and ElGamal signature schemes, but not patented.
- Adopted by NIST in 1994.
- ECDSA is tailored specifically for elliptic curve groups.

Digital Signature Algorithm (DSA)

- Schnorr was patented until 2008.
- DSA is a variant on the Schnorr and ElGamal signature schemes, but not patented.
- Adopted by NIST in 1994.
- ECDSA is tailored specifically for elliptic curve groups.

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$k \leftarrow_\$ \mathbb{Z}_q;\ \ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the* x*-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ \ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

m

$\longleftarrow$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // Derive the x-coordinate

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

$(m, \sigma = (r, z))$

$\longrightarrow$

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$k \leftarrow_{\$} \mathbb{Z}_q;\; R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\; u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + \boxed{r \cdot k \cdot sk}$

This step is hard in a multi-party setting, where no single party knows k or sk.

$(m, \sigma = (r,$

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

ECDSA

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

ECDSA

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

Signer

$(sk, PK) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{\quad m \quad}$$

$k \leftarrow_{\$} \mathbb{Z}_q;\ R \leftarrow g^k \in \mathbb{G}$

$c \leftarrow H(m)$

$r \leftarrow R.x$ // *Derive the x-coordinate*

$z \leftarrow k \cdot c + r \cdot k \cdot sk$

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

$c \leftarrow H(m)$

$u_1 \leftarrow c \cdot z^{-1};\ u_2 \leftarrow r \cdot z^{-1}$

$R' \leftarrow g^{u_1} \cdot PK^{u_2}$

Output 1 if $r \stackrel{?}{=} R'.x$; otherwise 0

EdDSA

- Similar to single-party Schnorr, but with two distinctions.
- First, the challenge additionally hashes in the public key to prevent malleability.
- Second, nonce generation is deterministic with respect to *sk* and the message.

EdDSA

- Similar to single-party Schnorr, but with two distinctions.
- First, the challenge additionally hashes in the public key to prevent malleability.
- Second, nonce generation is deterministic with respect to *sk* and the message.

EdDSA

- Similar to single-party Schnorr, but with two distinctions.
- First, the challenge additionally hashes in the public key to prevent malleability.
- Second, nonce generation is deterministic with respect to *sk* and the message.

Single-Party EdDSA Signing and Verification

Signer

Verifier

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$r \leftarrow H(sk, m)$

// Deterministic to mitigate bad randomness

$R = g^r; \; c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Single-Party EdDSA Signing and Verification

Signer **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow H(sk, m)$

// *Deterministic to mitigate bad randomness*

$R = g^r;\ c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Single-Party EdDSA Signing and Verification

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow H(sk, m)$

// *Deterministic to mitigate bad randomness*

$R = g^r; \; c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Single-Party EdDSA Signing and Verification

Signer **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$r \leftarrow H(sk, m)$

// *Deterministic to mitigate bad randomness*

$R = g^r;\ c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Single-Party EdDSA Signing and Verification

Signer **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow H(sk, m)$

// *Deterministic to mitigate bad randomness*

$R = g^r;\ c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Single-Party EdDSA Signing and Verification

Signer **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow H(sk, m)$

// *Deterministic to mitigate bad randomness*

$R = g^r;\ c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Single-Party EdDSA Signing and Verification

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$\xleftarrow{\quad m \quad}$

$r \leftarrow H(sk, m)$

// Deterministic to mitigate bad randomness

$R = g^r;\; c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Single-Party EdDSA Signing and Verification

Signer **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow H(sk, m)$

// *Deterministic to mitigate bad randomness*

$R = g^r;\ c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Single-Party EdDSA Signing and Verification

Signer | **Verifier**

$(sk, PK) \leftarrow KeyGen()$

$$\xleftarrow{\quad m \quad}$$

$r \leftarrow H(sk, m)$

// *Deterministic to mitigate bad randomness*

$R = g^r;\ c = H(R, PK, m)$

$z = r + c \cdot sk$

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

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

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

Output 1 if $R \stackrel{?}{=} R'$

Otherwise, output 0

Threshold Signatures

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

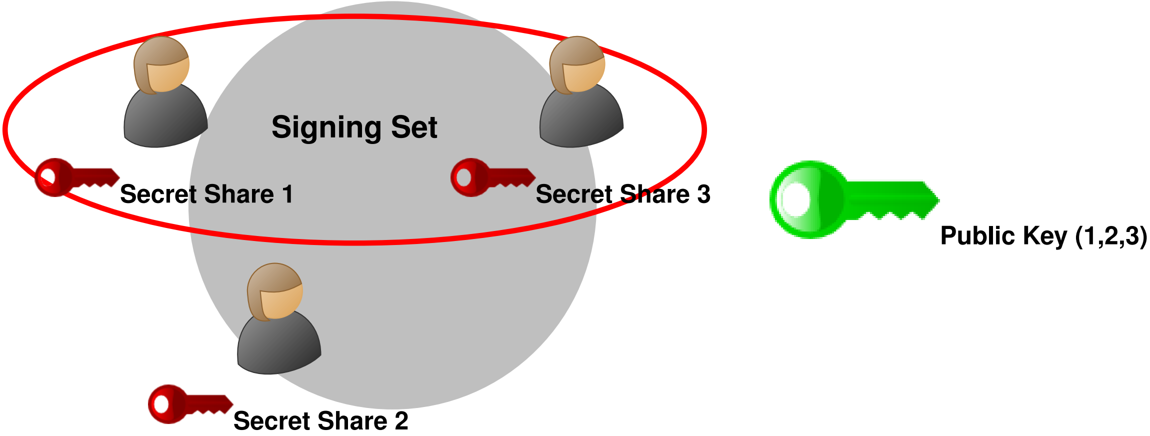
Shamir Secret Sharing

- Allows a *dealer* to share a secret α among n participants, where t participants must cooperate to recover α.
- f is the polynomial defined by the coefficients

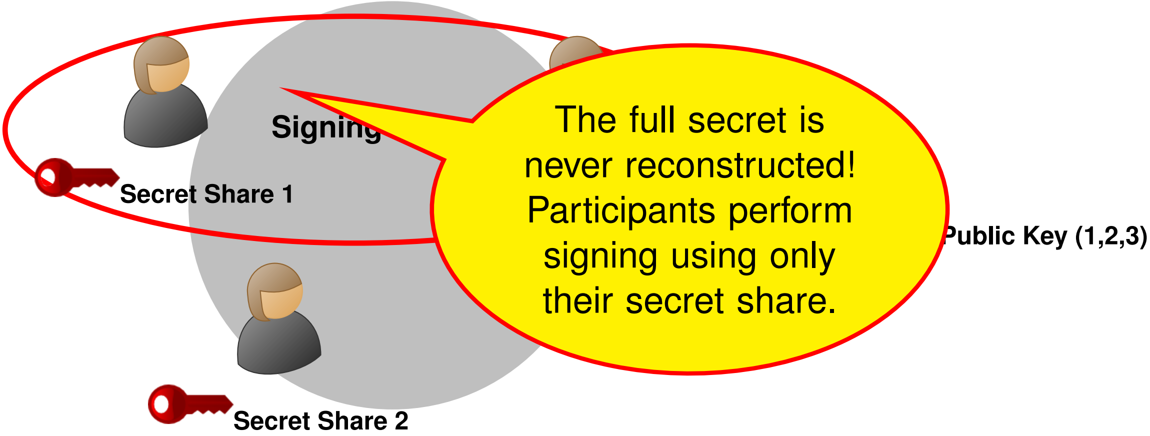
$$f = \alpha + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{t-1} x^{t-1}$$

- Each participant $i \in \{1, \ldots, n\}$ receives a share $w_i \leftarrow f(i)$.
- Recall that t points uniquely define a polynomial of degree $t - 1$!
- By polynomial interpolation, $\alpha = f(0) = \sum_{i=1}^{t} f(i)\lambda_i$.
- λ_i is $L_i(0)$, where L_i is the i^{th} Lagrange polynomial for the set $\{1, \ldots, t\}$.

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



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



Goals for Threshold Signatures

- Compatibile with single-party signing
- Low round efficiency
- Batching

Goals for Threshold Signatures

- Compatibile with single-party signing
- Low round efficiency
- Batching

Goals for Threshold Signatures

- Compatibile with single-party signing
- Low round efficiency
- Batching

ECDSA Threshold Signatures

GG18: Overview

- ”Fast Multiparty ECDSA with Fast Trustless Setup” by Gennaro and Goldfeder.
- Employs an additively homomorphic encryption scheme to perform $r \cdot (\sum k_i \cdot \sum sk_i)$ in a multi-party setting.
- Prior work used Pallier’s as this encryption scheme, but GG18 uses SPDZ (used for generic MPC).
- Seven rounds of broadcast, as well as two single-round pairwise multiplicative-to-additive share conversion steps.

GG18: Overview

- "Fast Multiparty ECDSA with Fast Trustless Setup" by Gennaro and Goldfeder.
- Employs an additively homomorphic encryption scheme to perform $r \cdot (\sum k_i \cdot \sum sk_i)$ in a multi-party setting.
- Prior work used Pallier's as this encryption scheme, but GG18 uses SPDZ (used for generic MPC).
- Seven rounds of broadcast, as well as two single-round pairwise multiplicative-to-additive share conversion steps.

GG18: Overview

- ”Fast Multiparty ECDSA with Fast Trustless Setup” by Gennaro and Goldfeder.
- Employs an additively homomorphic encryption scheme to perform $r \cdot (\sum k_i \cdot \sum sk_i)$ in a multi-party setting.
- Prior work used Pallier’s as this encryption scheme, but GG18 uses SPDZ (used for generic MPC).
- Seven rounds of broadcast, as well as two single-round pairwise multiplicative-to-additive share conversion steps.

GG18: Overview

- ”Fast Multiparty ECDSA with Fast Trustless Setup” by Gennaro and Goldfeder.
- Employs an additively homomorphic encryption scheme to perform $r \cdot (\sum k_i \cdot \sum sk_i)$ in a multi-party setting.
- Prior work used Pallier’s as this encryption scheme, but GG18 uses SPDZ (used for generic MPC).
- Seven rounds of broadcast, as well as two single-round pairwise multiplicative-to-additive share conversion steps.

EdDSA Threshold Signatures

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \leftarrow_\$ \mathbb{Z}_q^*;\ R_i \leftarrow g^{r_i}$

$\xrightarrow{R_i}$

Signature Aggregator

$B = ((1, R_1), \ldots, (t, R_t))$

$\xleftarrow{(m, B)}$

$$R = \prod_{\ell \in S} R_\ell$$

$$c = H_1(R, PK, m)$$

$$z_i = r_i + \lambda_i \cdot sk_i \cdot c$$

$\xrightarrow{z_i}$

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \leftarrow_\$ \mathbb{Z}_q^*;\ R_i \leftarrow g^{r_i}$

Signature Aggregator

$$\xrightarrow{\quad R_i \quad}$$

$B = ((1, R_1), \ldots, (t, R_t))$

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

$$R = \prod_{\ell \in S} R_\ell$$

$c = H_1(R, PK, m)$

$z_i = r_i + \lambda_i \cdot sk_i \cdot c$

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

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \leftarrow\!\$\, \mathbb{Z}_q^*;\ R_i \leftarrow g^{r_i}$

$\xrightarrow{R_i}$

Signature Aggregator

$B = ((1, R_1), \ldots, (t, R_t))$

$\xleftarrow{(m, B)}$

$R = \prod_{\ell \in S} R_\ell$

$c = H_1(R, PK, m)$

$z_i = r_i + \lambda_i \cdot sk_i \cdot c$

$\xrightarrow{z_i}$

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \leftarrow_\$ \mathbb{Z}_q^*;\ R_i \leftarrow g^{r_i}$

$$\xrightarrow{\quad R_i \quad}$$

Signature Aggregator

$B = ((1, R_1), \dots, (t, R_t))$

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

$$R = \prod_{\ell \in S} R_\ell$$

$c = H_1(R, PK, m)$

$z_i = r_i + \lambda_i \cdot sk_i \cdot c$

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

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \leftarrow_{\$} \mathbb{Z}_q^*;\ R_i \leftarrow g^{r_i}$

$$\xrightarrow{\quad R_i \quad}$$

Signature Aggregator

$B = ((1, R_1), \ldots, (t, R_t))$

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

$$R = \prod_{\ell \in S} R_\ell$$

$c = H_1(R, PK, m)$

$z_i = r_i + \lambda_i \cdot sk_i \cdot c$

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

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \xleftarrow{\$} \mathbb{Z}_q^*; \; R_i \leftarrow g^{r_i}$

$$\xrightarrow{\quad R_i \quad}$$

Signature Aggregator

$B = ((1, R_1), \ldots, (t, R_t))$

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

$$R = \prod_{\ell \in S} R_\ell$$

$$c = H_1(R, PK, m)$$

$$z_i = r_i + \lambda_i \cdot sk_i \cdot c$$

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

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \leftarrow\!\!\$\ \mathbb{Z}_q^*;\ \ R_i \leftarrow g^{r_i}$

$$\xrightarrow{\quad R_i \quad}$$

Signature Aggregator

$B = ((1, R_1), \ldots, (t, R_t))$

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

$$R = \prod_{\ell \in S} R_\ell$$

$$c = H_1(R, PK, m)$$

$$z_i = r_i + \lambda_i \cdot sk_i \cdot c$$

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

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i **Signature Aggregator**

$r_i \leftarrow\!\!\$\ \mathbb{Z}_q^*;\ R_i \leftarrow g^{r_i}$

$\xrightarrow{R_i}$

$\xleftarrow{(m}$

$$R = \prod_{\ell \in S} R_\ell$$

$$c = H_1(R, PK, m)$$

$$z_i = r_i + \boxed{\lambda_i \cdot sk_i \cdot c}$$

$\xrightarrow{z_i}$

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

Each sk_i is a Shamir secret share (i, sk_i) of a secret sk.

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i | **Signature Aggregator**

$r_i \leftarrow\!\$\, \mathbb{Z}_q^*;\ R_i \leftarrow g^{r_i}$

$\xrightarrow{R_i}$

$\xleftarrow{(m}$

$sk = \sum sk_i \lambda_i,$
for $i \in S$
$S \in \{1, \ldots n\}, |S| \geq t.$

$$R = \prod_{\ell \in S} R_\ell$$

$$c = H_1(R, PK, m)$$

$$z_i = r_i + \boxed{\lambda_i \cdot sk_i \cdot c}$$

$\xrightarrow{z_i}$

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \leftarrow\!\!\$\; \mathbb{Z}_q^*;\;\; R_i \leftarrow g^{r_i}$

$\xrightarrow{\quad R_i \quad}$

Signature Aggregator

$B = ((1, R_1), \ldots, (t, R_t))$

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

$$R = \prod_{\ell \in S} R_\ell$$

$c = H_1(R, PK, m)$

$z_i = r_i + \lambda_i \cdot sk_i \cdot c$

$\xrightarrow{\quad z_i \quad}$

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i

$r_i \leftarrow_{\$} \mathbb{Z}_q^*;\ R_i \leftarrow g^{r_i}$

Signature Aggregator

$\xrightarrow{R_i}$

$B = ((1, R_1), \ldots, (t, R_t))$

$\xleftarrow{(m, B)}$

$$R = \prod_{\ell \in S} R_\ell$$

$$c = H_1(R, PK, m)$$

$$z_i = r_i + \lambda_i \cdot sk_i \cdot c$$

$\xrightarrow{z_i}$

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

A Trivial (and Broken!) EdDSA Threshold Signature

Signer i | **Signature Aggregator**

$r_i \leftarrow\!\!\$\ \mathbb{Z}_q^*;\ \ R_i \leftarrow g^{r_i}$

$$\xrightarrow{\quad R_i \quad}$$

$))$

$$\xleftarrow{\quad (m \quad}$$

This structure allows for ROS-style attacks!

$$R = \prod_{\ell \in S} R_\ell$$

$$c = H_1(R, PK, m)$$

$$z_i = r_i + \boxed{\lambda_i \cdot sk_i \cdot c}$$

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

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

ROS Attack

- Random inhomogeneities in a Overdetermined Solvable system of linear equations
- Birthday paradox: Among a set of $\sqrt{p}$ random elements from a set of size p, two elements will collide with high probability.
- However, given a random oracle $H : \{0,1\}^n \rightarrow \mathbb{Z}_q$, the **ROS problem** asks to find a set $\langle c_1, \ldots, c_\ell \rangle$ where $c_i \leftarrow H(\cdot)$, a vector $\gamma_1, \ldots, \gamma_\ell \in \mathbb{Z}^\ell$, and c^* such that

$$\gamma_i \sum_{i=1}^{\ell} c_i = c^*$$

- "On the (in)security of ROS": Benhamouda et al. showed that the ROS problem can be solved in polynomial time.

ROS Attack

- Random inhomogeneities in a Overdetermined Solvable system of linear equations
- Birthday paradox: Among a set of $\sqrt{p}$ random elements from a set of size p, two elements will collide with high probability.
- However, given a random oracle $H : \{0,1\}^n \rightarrow \mathbb{Z}_q$, the **ROS problem** asks to find a set $\langle c_1, \ldots, c_\ell \rangle$ where $c_i \leftarrow H(\cdot)$, a vector $\gamma_1, \ldots, \gamma_\ell \in \mathbb{Z}^\ell$, and c^* such that

$$\gamma_i \sum_{i=1}^{\ell} c_i = c^*$$

- "On the (in)security of ROS": Benhamouda et al. showed that the ROS problem can be solved in polynomial time.

ROS Attack

- Random inhomogeneities in a Overdetermined Solvable system of linear equations
- Birthday paradox: Among a set of $\sqrt{p}$ random elements from a set of size p, two elements will collide with high probability.
- However, given a random oracle $H : \{0,1\}^n \rightarrow \mathbb{Z}_q$, the **ROS problem** asks to find a set $\langle c_1, \ldots, c_\ell \rangle$ where $c_i \leftarrow H(\cdot)$, a vector $\gamma_1, \ldots, \gamma_\ell \in \mathbb{Z}^\ell$, and c^* such that

$$\gamma_i \sum_{i=1}^{\ell} c_i = c^*$$

- "On the (in)security of ROS": Benhamouda et al. showed that the ROS problem can be solved in polynomial time.

ROS Attack

- Random inhomogeneities in a Overdetermined Solvable system of linear equations
- Birthday paradox: Among a set of $\sqrt{p}$ random elements from a set of size p, two elements will collide with high probability.
- However, given a random oracle $H : \{0,1\}^n \rightarrow \mathbb{Z}_q$, the **ROS problem** asks to find a set $\langle c_1, \ldots, c_\ell \rangle$ where $c_i \leftarrow H(\cdot)$, a vector $\gamma_1, \ldots, \gamma_\ell \in \mathbb{Z}^\ell$, and c^* such that

$$\gamma_i \sum_{i=1}^{\ell} c_i = c^*$$

- "On the (in)security of ROS": Benhamouda et al. showed that the ROS problem can be solved in polynomial time.

ROS Attack: Applied to Trivial Scheme

- A forgery for the trivial construction can be produced using this ROS solver, by finding

$$c^* = H(R^*, PK, m^*) = \sum_{j=1}^{k} H(R_j, PK', m_j) = \sum c_j \text{ for some } (R_j, m_j), \ldots$$

Allowing for the forgery for participant i:

$$z_i^* = \sum r_{ij} + \lambda_i \cdot sk_i \cdot \sum c_j = \sum r_{ij} + \lambda_i \cdot sk_i \cdot c^*$$

ROS Attack: Applied to Trivial Scheme

- A forgery for the trivial construction can be produced using this ROS solver, by finding

$$c^* = H(R^*, PK, m^*) = \sum_{j=1}^{k} H(R_j, PK`, m_j) = \sum c_j \text{ for some } (R_j, m_j), \ldots$$

Allowing for the forgery for participant i:

$$z_i^* = \sum r_{ij} + \lambda_i \cdot sk_i \cdot \sum c_j = \sum r_{ij} + \lambda_i \cdot sk_i \cdot c^*$$

ROS Attack: Applied to Trivial Scheme

- A forgery for the trivial construction can be produced using this ROS solver, by finding

$$c^* = H(R^*, PK, m^*) = \sum_{j=1}^{k} H(R_j, PK`, m_j) = \sum c_j \text{ for some } (R_j, m_j), \ldots$$

Allowing for the forgery for participant i:

$$z_i^* = \sum r_{ij} + \lambda_i \cdot sk_i \cdot \sum c_j = \sum r_{ij} + \lambda_i \cdot sk_i \cdot c^*$$

ROS Attack: Applied to Trivial Scheme

- A forgery for the trivial construction can be produced using this ROS solver, by finding

$$c^* = H(R^*, PK, m^*) = \sum_{j=1}^{k} H(R_j, PK`, m_j) = \sum c_j \text{ for some } (R_j, m_j), \ldots$$

Allowing for the forgery for participant i:

$$z_i^* = \sum r_{ij} + \lambda_i \cdot sk_i \cdot \sum c_j = \sum r_{ij} + \lambda_i \cdot sk_i \cdot c^*$$

FROST: Flexible Round-Optimized Schnorr Threshold Signatures

- Two-round threshold signing protocol, or single-round with preprocessing.
- Secure against an adversary that controls up to $t - 1$ signers, in OMDL/ROM.

FROST: Flexible Round-Optimized Schnorr Threshold Signatures

- Two-round threshold signing protocol, or single-round with preprocessing.
- Secure against an adversary that controls up to $t - 1$ signers, in OMDL/ROM.

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 PK_i (used for verification during signing) with joint public key PK.

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 PK_i (used for verification during signing) with joint public key PK.

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 PK_i (used for verification during signing) with joint public key PK.

FROST Sign

- We show here with a signature aggregator, but can be performed without centralized roles

FROST Sign

Signer i

$(d_i, e_i) \leftarrow\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$(D_i = g^{d_i}, E_i = g^{e_i})$ $\longrightarrow$

Signature Aggregator

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

$\longleftarrow$ (m, B)

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

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

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

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

z_i $\longrightarrow$

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

FROST Sign

Signer i

$(d_i, e_i) \leftarrow\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

Signature Aggregator

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

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

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

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

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

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

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

$$\xrightarrow{z_i}$$

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

FROST Sign

Signer i

$(d_i, e_i) \leftarrow\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

Signature Aggregator

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

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

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

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

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

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

$$\xrightarrow{z_i}$$

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

FROST Sign

Signer i

$(d_i, e_i) \leftarrow\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

Signature Aggregator

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

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

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

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

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

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

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

$$\xrightarrow{z_i}$$

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

FROST Sign

Signer i

$(d_i, e_i) \leftarrow\!\$\, \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

Signature Aggregator

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

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

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

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

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

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

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

$$\xrightarrow{z_i}$$

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

FROST Sign

Signer i

$(d_i, e_i) \leftarrow\!\$\; \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

Signature Aggregator

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

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

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

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

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

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

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

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

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

FROST Sign

Signer i

$(d_i, e_i) \leftarrow\!\$\, \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$(D_i = g^{d_i}, E_i = g^{e_i})$ $\longrightarrow$

Signature Aggregator

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

$\longleftarrow$ (m, B)

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

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

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

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

z_i $\longrightarrow$

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

FROST Sign

Signer i

Signature Aggregator

$(d_i, e_i) \leftarrow_\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

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

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

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

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

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

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

$$\xrightarrow{z_i}$$

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

FROST Sign

Signer i

$(d_i, e_i) \leftarrow\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

Signature Aggregator

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

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

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

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

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

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

$$\xrightarrow{z_i}$$

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

FROST Sign

Signer i

Signature Aggregator

$(d_i, e_i) \leftarrow_\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

$D_t, E_t))$

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

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

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

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

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

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**

$(d_i, e_i) \leftarrow\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

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

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

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

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

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

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

$$\xrightarrow{z_i}$$

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

FROST Sign

Signer i

$(d_i, e_i) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$$

Signature Aggregator

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

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

Signer i

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

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

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

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

$$\xrightarrow{z_i}$$

Signature Aggregator

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

FROST Sign

Signer i

$(d_i, e_i) \leftarrow_\$ \mathbb{Z}_q^* \times \mathbb{Z}_q^*$

$\xrightarrow{(D_i = g^{d_i}, E_i = g^{e_i})}$

Signature Aggregator

$D_t, E_t))$

$\xleftarrow{(m, B)}$

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

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

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

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

$\xrightarrow{z_i}$

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

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

FROST Security Against ROS

- An adversary can still find some $c^* = \sum_{j=1}^{k} H(R_j, PK, m_j)$
- But it is difficult to do so while also finding some ρ^* such that

$$\rho^* = H(B^*, PK, m^*) = \sum_{j=1}^{k} H(B_j, PK, m_j)$$

Where B_j completely defines R_j, such that

$$z_i^* = \sum_{j=1}^{k} z_{ij} = \sum d_{ij} + e_{ij} \cdot \rho_j + \lambda_i \cdot sk_i \cdot \sum c_j$$

$$= \sum d_{ij} + e_{ij} \cdot \rho^* + \lambda_i \cdot sk_i \cdot c^*$$

FROST Security Against ROS

- An adversary can still find some $c^* = \sum_{j=1}^{k} H(R_j, PK, m_j)$
- But it is difficult to do so while also finding some ρ^* such that

$$\rho^* = H(B^*, PK, m^*) = \sum_{j=1}^{k} H(B_j, PK, m_j)$$

Where B_j completely defines R_j, such that

$$z_i^* = \sum_{j=1}^{k} z_{ij} = \sum d_{ij} + e_{ij} \cdot \rho_j + \lambda_i \cdot sk_i \cdot \sum c_j$$

$$= \sum d_{ij} + e_{ij} \cdot \rho^* + \lambda_i \cdot sk_i \cdot c^*$$

FROST Security Against ROS

- An adversary can still find some $c^* = \sum_{j=1}^{k} H(R_j, PK, m_j)$
- But it is difficult to do so while also finding some ρ^* such that

$$\rho^* = H(B^*, PK, m^*) = \sum_{j=1}^{k} H(B_j, PK, m_j)$$

Where B_j completely defines R_j, such that

$$z_i^* = \sum_{j=1}^{k} z_{ij} = \sum d_{ij} + e_{ij} \cdot \rho_j + \lambda_i \cdot sk_i \cdot \sum c_j$$

$$= \sum d_{ij} + e_{ij} \cdot \rho^* + \lambda_i \cdot sk_i \cdot c^*$$

FROST Security Against ROS

- An adversary can still find some $c^* = \sum_{j=1}^{k} H(R_j, PK, m_j)$
- But it is difficult to do so while also finding some ρ^* such that

$$\rho^* = H(B^*, PK, m^*) = \sum_{j=1}^{k} H(B_j, PK, m_j)$$

Where B_j completely defines R_j, such that

$$z_i^* = \sum_{j=1}^{k} z_{ij} = \sum d_{ij} + e_{ij} \cdot \rho_j + \lambda_i \cdot sk_i \cdot \sum c_j$$

$$= \sum d_{ij} + e_{ij} \cdot \rho^* + \lambda_i \cdot sk_i \cdot c^*$$

Takeaways

- Schnorr signatures are simple to thresholdize, but ROS attacks are viable.
- ECDSA signatures are harder to thresholdize due to their structure and lead to more complicated constructions.
- In general, patents in cryptography lead to convulted workarounds that persist long beyond the patent expires!

Takeaways

- Schnorr signatures are simple to thresholdize, but ROS attacks are viable.
- ECDSA signatures are harder to thresholdize due to their structure and lead to more complicated constructions.
- In general, patents in cryptography lead to convulted workarounds that persist long beyond the patent expires!

Takeaways

- Schnorr signatures are simple to thresholdize, but ROS attacks are viable.
- ECDSA signatures are harder to thresholdize due to their structure and lead to more complicated constructions.
- In general, patents in cryptography lead to convulted workarounds that persist long beyond the patent expires!