Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices

PKC 2021

eprint 2020/1110

Ivan Damgård[1] Claudio Orlandi[1] Akira Takahashi[1] Mehdi Tibouchi[2]

[1]Aarhus University, Denmark

[2]NTT Corporation, Japan

Intro

Background & Motivation

- Two approaches to lattice-based signatures among the NIST PQC standardization finalists:
  - Hash-and-sign [GPV08]: Falcon
  - **Fiat–Shamir with aborts** [Lyu09]: Dilithium

- Renewed interest in multi-party signing: upcoming NIST standardization, Blockchain, etc.
  - Many existing works on round-efficient n-party signatures in the **discrete log** setting (cf. Drijvers et al. [DEF+19]).

- FSwA-style signature has a structure similar to **Schnorr**.

Can we construct a lattice-based, round-efficient multi-party signing protocol, by making the most of observations in the DL setting?

Background & Motivation

- Two approaches to lattice-based signatures among the NIST PQC standardization finalists:
  - Hash-and-sign [GPV08]: Falcon
  - **Fiat–Shamir with aborts** [Lyu09]: Dilithium
- Renewed interest in multi-party signing: upcoming NIST standardization, Blockchain, etc.
  - Many existing works on round-efficient n-party signatures in the **discrete log** setting (cf. Drijvers et al. [DEF$^+$19]).
- FSwA-style signature has a structure similar to **Schnorr**.

Can we construct a lattice-based, round-efficient multi-party signing protocol, by making the most of observations in the DL setting?

Background & Motivation

- Two approaches to lattice-based signatures among the NIST PQC standardization finalists:
  - Hash-and-sign [GPV08]: Falcon
  - **Fiat–Shamir with aborts** [Lyu09]: Dilithium
- Renewed interest in multi-party signing: upcoming NIST standardization, Blockchain, etc.
  - Many existing works on round-efficient n-party signatures in the **discrete log** setting (cf. Drijvers et al. [DEF+19]).
- FSwA-style signature has a structure similar to **Schnorr**.

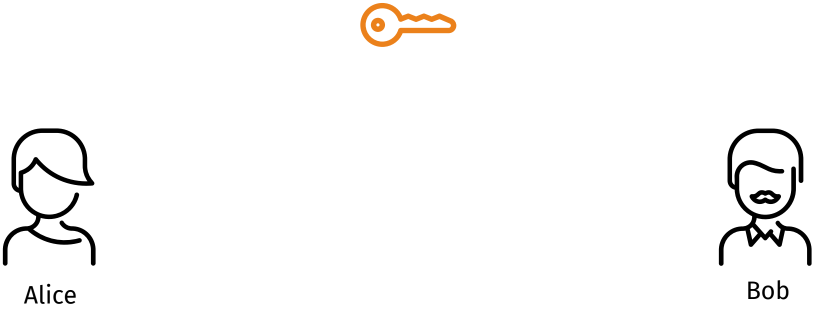
Can we construct a lattice-based, round-efficient multi-party signing protocol, by making the most of observations in the DL setting?

Background & Motivation

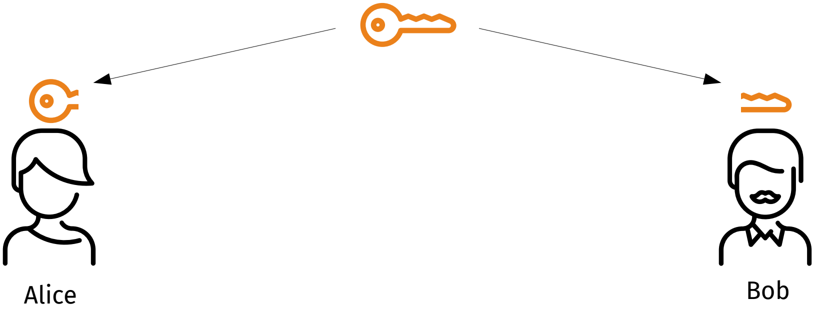
- Two approaches to lattice-based signatures among the NIST PQC standardization finalists:
  - Hash-and-sign [GPV08]: Falcon
  - **Fiat–Shamir with aborts** [Lyu09]: Dilithium
- Renewed interest in multi-party signing: upcoming NIST standardization, Blockchain, etc.
  - Many existing works on round-efficient n-party signatures in the **discrete log** setting (cf. Drijvers et al. [DEF$^+$19]).
- FSwA-style signature has a structure similar to **Schnorr**.

Can we construct a lattice-based, round-efficient multi-party signing protocol, by making the most of observations in the DL setting?

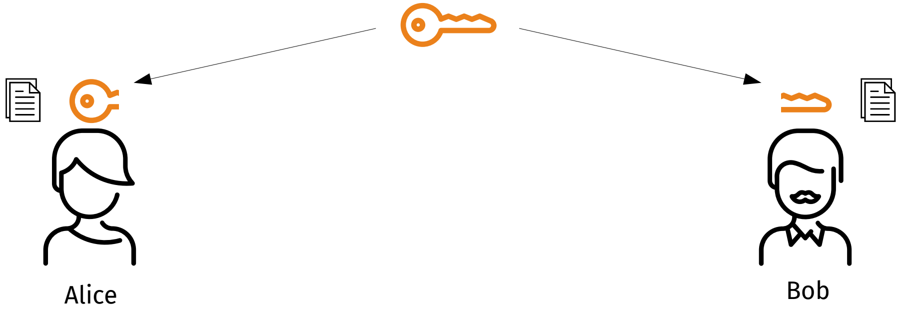
2-out-of-2 Signing



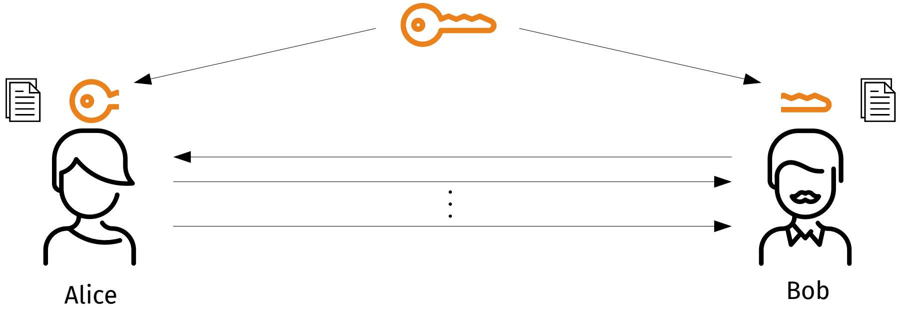
2-out-of-2 Signing



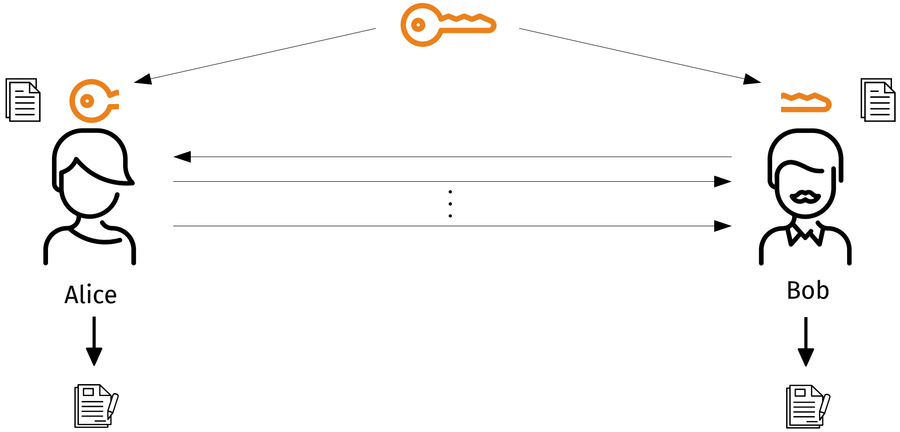
2-out-of-2 Signing



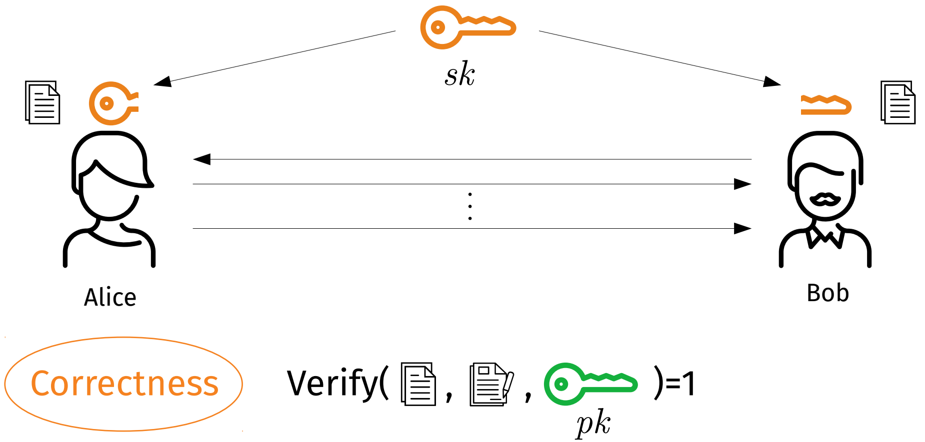
2-out-of-2 Signing



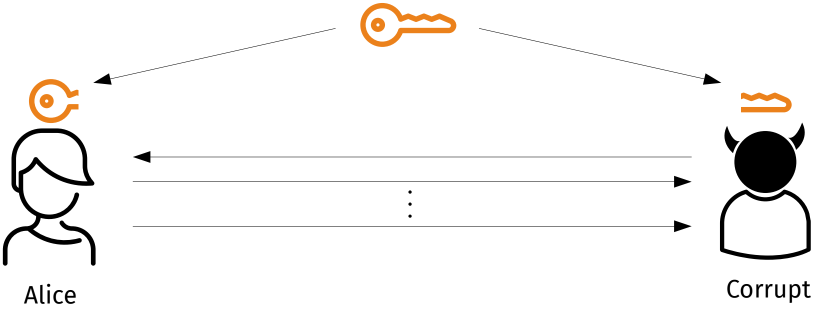
2-out-of-2 Signing



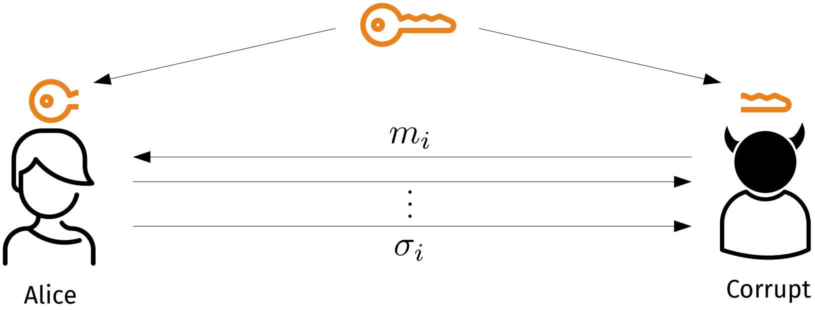
2-out-of-2 Signing



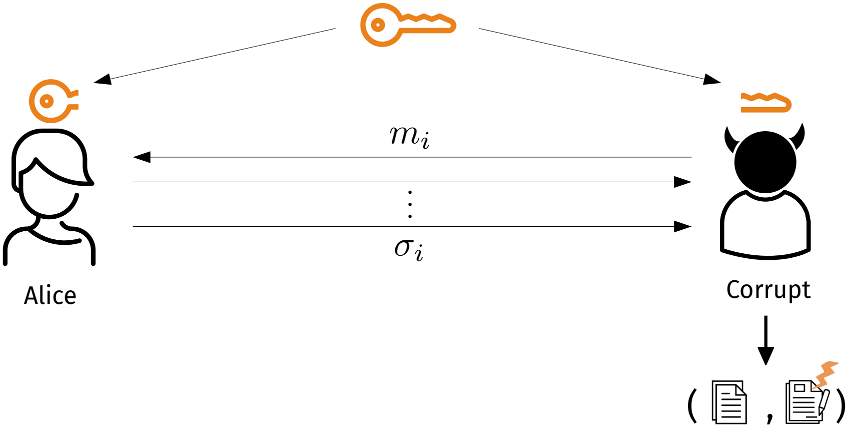
2-out-of-2 Signing : Game-based Security



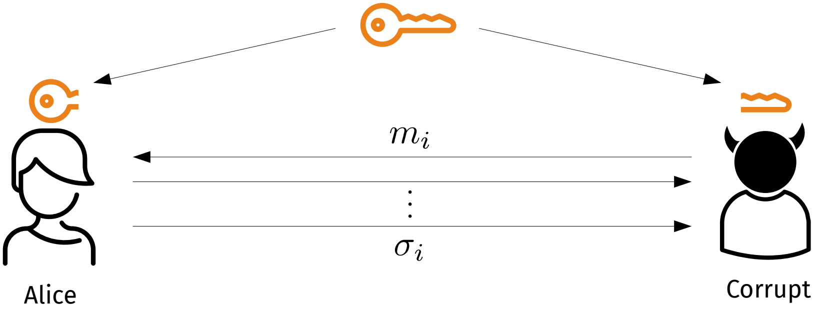
2-out-of-2 Signing : Game-based Security



2-out-of-2 Signing : Game-based Security



2-out-of-2 Signing : Game-based Security



Unforgeability Pr[Verify(, ,)=1 ∧ $\neq m_i$] is negl.

Fiat–Shamir with Aborts: Dilithium ID

$\boxed{P(\mathbf{A}, \mathbf{s})}$		$\boxed{V(\mathbf{A}, \mathbf{t} = \mathbf{A}\mathbf{s})}$
$\mathbf{y} \leftarrow\!\!\$\ D^{\ell+k} // D$ is Gaussian or uni.		
$\mathbf{w} := \mathbf{A}\mathbf{y}$	$\xrightarrow{\mathbf{w}}$	$c \leftarrow\!\!\$\ C \subset R$
$\mathbf{z} := c\mathbf{s} + \mathbf{y}$	$\xleftarrow{c}$	
If $\mathrm{RejSamp}(c\mathbf{s}, \mathbf{z}) = 0:$ restart	$\xrightarrow{\mathbf{z}}$	$\mathbf{A}\mathbf{z} - c\mathbf{t} = \mathbf{w} \wedge \|\mathbf{z}\|$ is small

- Operate on a vector of polynomials in a quotient ring $R_q = \mathbb{Z}_q[X]/(f(X))$.
- Secret key is a small $\mathbf{s} \in R_q^{\ell+k}$; public key consists of $\mathbf{A} = [\mathbf{A}'|\mathbf{I}]$ with random $\mathbf{A}' \in R_q^{k\times\ell}$ and $\mathbf{t} = \mathbf{A}\mathbf{s}$.
- $\mathbf{z} \in R_q^{\ell+k}$ has to be small $\leadsto$ c and $\mathbf{y}$ have to be small as well.
- RejSamp = rejection sampling: force $\mathbf{z}$ to be independent of $\mathbf{s}$ (**non-linear operation**)

Fiat–Shamir with Aborts: Dilithium ID vs. Schnorr ID

$$
\begin{array}{lcl}
\boxed{P(G, \mathbf{s})} & & \boxed{V(G, \mathbf{t} = [\mathbf{s}]G)} \\
\mathbf{y} \leftarrow_{\$} \mathbb{Z}_q & & \\
\mathbf{w} = [\mathbf{y}]G & \xrightarrow{\quad \mathbf{w} \quad} & c \leftarrow_{\$} C \subset R \\
\mathbf{z} := c\mathbf{s} + \mathbf{y} & \xleftarrow{\quad c \quad} & \\
 & \xrightarrow{\quad \mathbf{z} \quad} & [\mathbf{z}]G - c\mathbf{t} = \mathbf{w}
\end{array}
$$

- Operate on a vector of polynomials in a quotient ring $R_q = \mathbb{Z}_q[X]/(f(X))$.
- Secret key is a small $\mathbf{s} \in R_q^{\ell+k}$; public key consists of $\mathbf{A} = [\mathbf{A}'|\mathbf{I}]$ with random $\mathbf{A}' \in R_q^{k\times\ell}$ and $\mathbf{t} = \mathbf{A}\mathbf{s}$.
- $\mathbf{z} \in R_q^{\ell+k}$ has to be small $\rightsquigarrow$ c and $\mathbf{y}$ have to be small as well.
- RejSamp = rejection sampling: force $\mathbf{z}$ to be independent of $\mathbf{s}$ (**non-linear operation**)

Security of FSwA

- Soundness from **Module-SIS** and **Module-LWE**
  - Suppose $P^*(\mathbf{A}, \mathbf{t})$ can correctly answer c and c' for the same $\mathbf{w}$

$$\rightsquigarrow \mathbf{Az} - c\mathbf{t} = \mathbf{w} = \mathbf{Az}' - c'\mathbf{t}$$

  - $(\mathbf{A}, \mathbf{t} = \mathbf{As}) \approx^c (\mathbf{A}, \mathbf{t} \leftarrow_{\$} R_q^k)$ due to LWE.
  - Then using P^* find a non-zero solution to the SIS problem wrt $[\mathbf{A}|\mathbf{t}]$:

$$[\mathbf{A}|\mathbf{t}] \begin{bmatrix} \mathbf{z} - \mathbf{z}' \\ c' - c \end{bmatrix} = \mathbf{0}.$$

- Non-aborting statistical HVZK
  - If protocol doesn't abort: simulator outputs $(\mathbf{w} = \mathbf{Az} - c\mathbf{t}, c, \mathbf{z} \leftarrow_{\$} D^{\ell+k})$.

Security of FSwA

- Soundness from **Module-SIS** and **Module-LWE**
  - Suppose $P^*(\mathbf{A}, \mathbf{t})$ can correctly answer c and c' for the same $\mathbf{w}$

$$\rightsquigarrow \mathbf{Az} - c\mathbf{t} = \mathbf{w} = \mathbf{Az}' - c'\mathbf{t}$$

  - $(\mathbf{A}, \mathbf{t} = \mathbf{As}) \approx^c (\mathbf{A}, \mathbf{t} \leftarrow_\$ R_q^k)$ due to LWE.
  - Then using P^* find a non-zero solution to the SIS problem wrt $[\mathbf{A}|\mathbf{t}]$:

$$[\mathbf{A}|\mathbf{t}] \begin{bmatrix} \mathbf{z} - \mathbf{z}' \\ c' - c \end{bmatrix} = \mathbf{0}.$$

- Non-aborting statistical HVZK
  - If protocol doesn't abort: simulator outputs $(\mathbf{w} = \mathbf{Az} - c\mathbf{t}, c, \mathbf{z} \leftarrow_\$ D^{\ell+k})$.

Two-party Signing from FSwA

Our results

- Two-round multi-party FSwA signing with full security proof in CROM
- Two instantiations: n**-out-of-**n signatures and **multi-signatures**.
- This talk: focused on $n = 2$, but the approach can be generalized to $n > 2$.

Comparison with previous lattice-based multi-party signing

	Functionality	# Rounds	Type	Security	Building blocks
[BGG+18]	t-out-of-n	1	FSwA	Lyubashevsky ’12	Threshold FHE
[BKP13]	t-out-of-n	1	H&S	GPV ’08	Honest-majority MPC
Our DS_3	n-out-of-n	3	FSwA	MLWE	Homomorphic COM
Our DS_2	n-out-of-n	2	FSwA	MLWE & MSIS	Homomorphic TDCOM
[BS16]	Multisig	3	FSwA	DCK	—
[FH20]	Multisig	3	FSwA	Heuristic assumption / QROM	—
Our MS_2	Multisig	2	FSwA	MLWE & MSIS	Homomorphic TDCOM

Bare-bone 2-party signing: Schnorr vs Dilithium

$P_1(\mathbf{s}_1, pk = [\mathbf{s}_1 + \mathbf{s}_2]G)$		$P_2(\mathbf{s}_2, pk)$
$\mathbf{y}_1 \leftarrow\$ \mathbb{Z}_q; \mathbf{w}_1 = [\mathbf{y}_1]G$	$\xrightarrow{\quad \mathbf{w}_1 \quad}$	
$c \leftarrow \mathsf{H}(\mathbf{w}_1 + \mathbf{w}_2, m, pk)$	$\xleftarrow{\quad \mathbf{w}_2 \quad}$	
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
	$\xrightarrow{\quad \mathbf{z}_1 \quad}$	
	$\xleftarrow{\quad \mathbf{z}_2 \quad}$	
Output $((\mathbf{w}_1 + \mathbf{w}_2, \mathbf{z}_1 + \mathbf{z}_2), m)$		

- Round 1: Exchange “commitments” $\mathbf{w}_i$ and locally derive a joint challenge c
- Round 2: Compute signature shares $\mathbf{z}_i$ and exchange them

Bare-bone 2-party signing: Schnorr vs Dilithium

$\boxed{P_1(\mathbf{s}_1, pk = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2))}$ $\boxed{P_2(\mathbf{s}_2, pk)}$

$\mathbf{y}_1 \leftarrow\!\!\$\; D; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$ $\xrightarrow{\mathbf{w}_1}$

$c \leftarrow \mathsf{H}(\mathbf{w}_1 + \mathbf{w}_2, m, pk)$ $\xleftarrow{\mathbf{w}_2}$

$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$

If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : \mathbf{z}_1 := \perp$ $\xrightarrow{\mathbf{z}_1}$

If $\mathbf{z}_i = \perp$: restart $\xleftarrow{\mathbf{z}_2}$

Output $((\mathbf{w}_1 + \mathbf{w}_2, \mathbf{z}_1 + \mathbf{z}_2), m)$

- Round 1: Exchange “commitments” $\mathbf{w}_i$ and locally derive a joint challenge c
- Round 2: Compute signature shares $\mathbf{z}_i$ and exchange them **only if they pass the rejection sampling**

Bare-bone 2-party signing: Schnorr vs Dilithium

$\boxed{P_1(\mathbf{s}_1, pk = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2))}$ $\qquad\qquad$ $\boxed{P_2(\mathbf{s}_2, pk)}$

$\mathbf{y}_1 \leftarrow\!\!\$\ D; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$ $\qquad \xrightarrow{\quad\mathbf{w}_1\quad}$

$c \leftarrow \mathsf{H}(\mathbf{w}_1 + \mathbf{w}_2, m, pk)$ $\qquad \xleftarrow{\quad\mathbf{w}_2\quad}$

$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$

If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : \mathbf{z}_1 := \perp$ $\qquad \xrightarrow{\quad\mathbf{z}_1\quad}$

If $\mathbf{z}_i = \perp :$ restart $\qquad \xleftarrow{\quad\mathbf{z}_2\quad}$

Output $((\mathbf{w}_1 + \mathbf{w}_2, \mathbf{z}_1 + \mathbf{z}_2), m)$

- Round 1: Exchange “commitments” $\mathbf{w}_i$ and locally derive a joint challenge c
- Round 2: Compute signature shares $\mathbf{z}_i$ and exchange them **only if they pass the rejection sampling**

Bare-bone 2-party signing: Schnorr vs Dilithium

$P_1(\mathbf{s}_1, pk = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2))$		$P_2(\mathbf{s}_2, pk)$
$\mathbf{y}_1 \leftarrow\!\$\; D; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$	$\xrightarrow{\mathbf{w}_1}$	
$c \leftarrow \mathsf{H}(\mathbf{w}_1 + \mathbf{w}_2, m, pk)$	$\xleftarrow{\mathbf{w}_2}$	
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : \mathbf{z}_1 := \perp$	$\xrightarrow{\mathbf{z}_1}$	
If $\mathbf{z}_i = \perp :$ restart	$\xleftarrow{\mathbf{z}_2}$	
Output $((\mathbf{w}_1 + \mathbf{w}_2, \mathbf{z}_1 + \mathbf{z}_2), m)$		

- Round 1: Exchange "commitments" $\mathbf{w}_i$ and locally derive a joint challenge c
- Round 2: Compute signature shares $\mathbf{z}_i$ and exchange them **only if they pass the rejection sampling**

Two issues of bare-bone protocol

1. Simulation of rejected $(\mathbf{w}_i, c, \perp)$
    - Not a problem for single-user signing or NIZK
    - Problematic in **interactive** FSwA protocols
    - Just sending Commit($\mathbf{w}_i$) is not enough: need $\mathbf{w}_1 + \mathbf{w}_2$ **before** computing challenge
2. Malicious P_2 can choose the first message depending on P_1's output!
    - Naive: **extra round for "committing to commitment"** to construct an honest party simulator
    - Potential **concurrent attack** (variant of Drijvers et al. [DEF+19] against Schnorr multisigs)

Two issues of bare-bone protocol

1. Simulation of rejected $(\mathbf{w}_i, c, \perp)$
    - Not a problem for single-user signing or NIZK
    - Problematic in **interactive** FSwA protocols
    - Just sending Commit($\mathbf{w}_i$) is not enough: need $\mathbf{w}_1 + \mathbf{w}_2$ **before** computing challenge
2. Malicious P_2 can choose the first message depending on P_1's output!
    - Naive: **extra round for "committing to commitment"** to construct an honest party simulator
    - Potential **concurrent attack** (variant of Drijvers et al. [DEF+19] against Schnorr multisigs)

Our solutions

1. Simulation of rejected $(\mathbf{w}_i, c, \perp)$
   - Send homomorphic $\mathsf{Commit}(\mathbf{w}_i)$
   - Hide $\mathbf{w}_i$ until the rejection sampling succeeds while computing $\mathbf{w}_1 + \mathbf{w}_2$ earlier.
2. Malicious P_2 could choose $\mathbf{w}_2$ depending on $\mathbf{w}_1$!
   - Use trapdoor homomorphic commitment to avoid an extra round

First step: Three-round protocol from "double" commitments

$P_1(\mathbf{s}_1, \mathbf{t} = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2), ck)$		$P_2(\mathbf{s}_2, \mathbf{t}, ck)$
$\mathbf{y}_1 \leftarrow\$ D^{\ell+k}; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$	$\xrightarrow{h_1 = \mathsf{H}(com_1)}$	
$com_1 \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_1; r_1)$	$\xleftarrow{h_2 = \mathsf{H}(com_2)}$	
	$\xrightarrow{com_1}$	
Check $\mathsf{H}(com_2) = h_2$	$\xleftarrow{com_2}$	
$c \leftarrow \mathsf{H}(com_1 + com_2, m, \mathbf{t})$		
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : (\mathbf{z}_1, r_1) := (\perp, \perp)$	$\xrightarrow{\mathbf{z}_1, r_1}$	
If $\mathbf{z}_i = \perp :$ restart	$\xleftarrow{\mathbf{z}_2, r_2}$	
Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$		

First step: Three-round protocol from "double" commitments

$\boxed{P_1(\mathbf{s}_1, \mathbf{t} = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2), ck)}$		$\boxed{P_2(\mathbf{s}_2, \mathbf{t}, ck)}$
$\mathbf{y}_1 \leftarrow\$\, D^{\ell+k}; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$	$\xrightarrow{h_1 = \mathsf{H}(com_1)}$	
$com_1 \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_1; r_1)$	$\xleftarrow{h_2 = \mathsf{H}(com_2)}$	
	$\xrightarrow{com_1}$	
Check $\mathsf{H}(com_2) = h_2$	$\xleftarrow{com_2}$	
$c \leftarrow \mathsf{H}(com_1 + com_2, m, \mathbf{t})$		
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : (\mathbf{z}_1, r_1) := (\perp, \perp)$	$\xrightarrow{\mathbf{z}_1, r_1}$	
If $\mathbf{z}_i = \perp$: restart	$\xleftarrow{\mathbf{z}_2, r_2}$	
Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$		

Signature verification

- Vf$(com, \mathbf{z}, r, m, ck, (\mathbf{A}, \mathbf{t}))$:
  1. Get a challenge $c \leftarrow \mathsf{H}(com, m, \mathbf{t})$
  2. Reconstruct committed $\mathbf{w} = \mathbf{A}\mathbf{z} - c\mathbf{t}$
  3. Verify

$$\|\mathbf{z}\| \text{ is small} \wedge \mathsf{Open}_{ck}(com, r, \mathbf{w}) = 1$$

- Correctness holds since
  - Linearity of $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$:

$$\mathbf{A}\mathbf{z} - c\mathbf{t} = \mathbf{A}(\mathbf{z}_1 + \mathbf{z}_2) - c(\mathbf{A}\mathbf{s}_1 + \mathbf{A}\mathbf{s}_2) = \mathbf{w}_1 + \mathbf{w}_2$$

  - Homomorphism of the commitment: $\mathsf{Open}_{ck}(com, r, \mathbf{w}) = \mathsf{Open}_{ck}(com_1 + com_2, r_1 + r_2, \mathbf{w}_1 + \mathbf{w}_2) = 1$
  - If $\mathbf{z}_i$ follows Gaussian centered at $\mathbf{0}$ then $\|\mathbf{z}\| \approx \sqrt{2}\,\|\mathbf{z}_i\|$

Signature verification

- Vf$(com, \mathbf{z}, r, m, ck, (\mathbf{A}, \mathbf{t}))$:
  1. Get a challenge $c \leftarrow \mathsf{H}(com, m, \mathbf{t})$
  2. Reconstruct committed $\mathbf{w} = \mathbf{A}\mathbf{z} - c\mathbf{t}$
  3. Verify

$$\|\mathbf{z}\| \text{ is small} \wedge \mathsf{Open}_{ck}(com, r, \mathbf{w}) = 1$$

- Correctness holds since
  - Linearity of $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$:

$$\mathbf{A}\mathbf{z} - c\mathbf{t} = \mathbf{A}(\mathbf{z}_1 + \mathbf{z}_2) - c(\mathbf{A}\mathbf{s}_1 + \mathbf{A}\mathbf{s}_2) = \mathbf{w}_1 + \mathbf{w}_2$$

  - Homomorphism of the commitment:
    $\mathsf{Open}_{ck}(com, r, \mathbf{w}) = \mathsf{Open}_{ck}(com_1 + com_2, r_1 + r_2, \mathbf{w}_1 + \mathbf{w}_2) = 1$
  - If $\mathbf{z}_i$ follows Gaussian centered at $\mathbf{0}$ then $\|\mathbf{z}\| \approx \sqrt{2}\,\|\mathbf{z}_i\|$

Security

☺ Provably Secure!

- If protocol doesn't abort: Honest party oracle can be simulated with the NA-HVZK simulator
- If protocol aborts: Hiding commitment reveals nothing about $\mathbf{w}_i$
- Security reduction to (Module) LWE without the forking lemma, thanks to the **lossy ID technique** (Abdalla et al. [AFLT16])

Security

☺ Provably Secure!

- If protocol doesn't abort: Honest party oracle can be simulated with the NA-HVZK simulator
- If protocol aborts: Hiding commitment reveals nothing about $\mathbf{w}_i$
- Security reduction to (Module) LWE without the forking lemma, thanks to the **lossy ID technique** (Abdalla et al. [AFLT16])

☺ Provably Secure!

- If protocol doesn't abort: Honest party oracle can be simulated with the NA-HVZK simulator
- If protocol aborts: Hiding commitment reveals nothing about $\mathbf{w}_i$
- Security reduction to (Module) LWE without the forking lemma, thanks to the **lossy ID technique** (Abdalla et al. [AFLT16])

Efficiency

☺ No expensive machinery like FHE, MPC, Gaussian sampling over lattices, etc.

- L^2-norm of $\mathbf{z}$ grows by a factor of $\sqrt{n}$: given n discrete Gaussian samples $\mathbf{z}_i \sim D_\sigma$, their sum $\mathbf{z} = \mathbf{z}_1 + \ldots + \mathbf{z}_n$ is statistically close to $D_{\sqrt{n}\sigma}$.
- Need to wait for all n parties to pass the rejection sampling: if each party succeeds with prob. $1/M$ then the entire protocol restarts M^n times
  - To keep M^n constant, σ grows by a factor of n.
  - Or parallel repetition is required.

Efficiency

☺ No expensive machinery like FHE, MPC, Gaussian sampling over lattices, etc.

- L^2-norm of $\mathbf{z}$ grows by a factor of $\sqrt{n}$: given n discrete Gaussian samples $\mathbf{z}_i \sim D_\sigma$, their sum $\mathbf{z} = \mathbf{z}_1 + \ldots + \mathbf{z}_n$ is statistically close to $D_{\sqrt{n}\sigma}$.
- Need to wait for all n parties to pass the rejection sampling: if each party succeeds with prob. $1/M$ then the entire protocol restarts M^n times
  - To keep M^n constant, σ grows by a factor of n.
  - Or parallel repetition is required.

Efficiency

☺ No expensive machinery like FHE, MPC, Gaussian sampling over lattices, etc.

- L^2-norm of $\mathbf{z}$ grows by a factor of $\sqrt{n}$: given n discrete Gaussian samples $\mathbf{z}_i \sim D_\sigma$, their sum $\mathbf{z} = \mathbf{z}_1 + \ldots + \mathbf{z}_n$ is statistically close to $D_{\sqrt{n}\sigma}$.
- Need to wait for all n parties to pass the rejection sampling: if each party succeeds with prob. $1/M$ then the entire protocol restarts M^n times
    - To keep M^n constant, σ grows by a factor of n.
    - Or parallel repetition is required.

Two-round protocol

How to drop the extra round?

$P_1(\mathbf{s}_1, \mathbf{t} = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2), ck)$		$P_2(\mathbf{s}_2, \mathbf{t}, ck)$
$\mathbf{y}_1 \leftarrow_\$ D^{\ell+k}; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$	$\xleftarrow{h_2 = \mathsf{H}(com_2)}$	$com_2 \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)$
	$\xrightarrow{com_1 = \mathsf{Commit}_{ck}(\mathbf{w}_1; r_1)}$	
Check $\mathsf{H}(com_2) = h_2$	$\xleftarrow{com_2}$	
$c \leftarrow \mathsf{H}(com_1 + com_2, m, \mathbf{t})$		
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : (\mathbf{z}_1, r_1) := (\perp, \perp)$	$\xrightarrow{\mathbf{z}_1, r_1}$	
If $\mathbf{z}_i = \perp$: restart	$\xleftarrow{\mathbf{z}_2, r_2}$	
Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$		

How to drop the extra round?

$P_1(\mathbf{s}_1, \mathbf{t} = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2), ck)$ $\qquad$ $P_2(\mathbf{s}_2, \mathbf{t}, ck)$

$\mathbf{y}_1 \leftarrow_\$ D^{\ell+k}; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$ $\quad \xrightarrow{com_1 = \mathsf{Commit}_{ck}(\mathbf{w}_1; r_1)}$

$c \leftarrow \mathsf{H}(com_1 + com_2, m, \mathbf{t})$ $\quad \xleftarrow{com_2 = \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)}$

$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$

If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : (\mathbf{z}_1, r_1) := (\perp, \perp)$ $\quad \xrightarrow{\mathbf{z}_1, r_1}$

If $\mathbf{z}_i = \perp$: restart $\quad \xleftarrow{\mathbf{z}_2, r_2}$

Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$

Simulation fails!

$\boxed{\mathsf{Sim}(\mathbf{t}_1, \mathbf{t} = \mathbf{t}_1 + \mathbf{A}\mathbf{s}_2, ck)}$ $\boxed{\mathcal{A}(\mathbf{s}_2, \mathbf{t}, ck)}$

$\mathbf{z}_1 \leftarrow_\$ D^{\ell+k}; c \leftarrow_\$ C; \mathbf{w}_1 = \mathbf{A}\mathbf{z}_1 - c\mathbf{t}_1$ $\xrightarrow{com_1 = \mathsf{Commit}_{ck}(\mathbf{w}_1; r_1)}$

com_2 is not known! $\rightsquigarrow$ can't program RO such that

$\mathsf{H}(com_1 + com_2, m, \mathbf{t}) := c$ $\xleftarrow{com_2 = \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)}$

With prob. $1 - 1/M : (\mathbf{z}_1, r_1) := (\perp, \perp)$ $\xrightarrow{\mathbf{z}_1, r_1}$

If $\mathbf{z}_i = \perp :$ restart $\xleftarrow{\mathbf{z}_2, r_2}$

Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$

Simulation fails!

$\mathsf{Sim}(\mathbf{t}_1, \mathbf{t} = \mathbf{t}_1 + \mathbf{A}\mathbf{s}_2, ck)$		$\mathcal{A}(\mathbf{s}_2, \mathbf{t}, ck)$
$\mathbf{z}_1 \leftarrow_\$ D^{\ell+k}; c \leftarrow_\$ C; \mathbf{w}_1 = \mathbf{A}\mathbf{z}_1 - c\mathbf{t}_1$	$\xrightarrow{com_1 = \mathsf{Commit}_{ck}(\mathbf{w}_1; r_1)}$	
com_2 is not known! $\rightsquigarrow$ can't program RO such that		
$\mathsf{H}(com_1 + com_2, m, \mathbf{t}) := c$	$\xleftarrow{com_2 = \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)}$	
With prob. $1 - 1/M : (\mathbf{z}_1, r_1) := (\perp, \perp)$	$\xrightarrow{\mathbf{z}_1, r_1}$	
If $\mathbf{z}_i = \perp$: restart	$\xleftarrow{\mathbf{z}_2, r_2}$	
Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$		

Also: If ck is fixed then the same concurrent attack applies!

$\rightsquigarrow$ Need per-message keys $ck = \mathsf{H}(m, \mathbf{t})$

Solution: Straight-line simulation with trapdoor commitment (Damgård ’00)

- Commitment key generation outputs an extra trapdoor td
- Given td a commitment can be opened to any message!
- **Simulation sketch**
  1. Honest party simulator sends out a “fake” commitment $com_1 = \mathsf{TCommit}_{ck}(td)$ in the first round
  2. com_1 can be later equivocated to anything depending on the derived joint challenge c.

Solution: Straight-line simulation with trapdoor commitment (Damgård ’00)

- Commitment key generation outputs an extra trapdoor td
- Given td a commitment can be opened to any message!
- Simulation sketch
  1. Honest party simulator sends out a “fake” commitment $com_1 = \mathsf{TCommit}_{ck}(td)$ in the first round
  2. com_1 can be later equivocated to anything depending on the derived joint challenge c.

Solution: Straight-line simulation with trapdoor commitment (Damgård '00)

- Commitment key generation outputs an extra trapdoor td
- Given td a commitment can be opened to any message!
- **Simulation sketch**
    1. Honest party simulator sends out a "fake" commitment $com_1 = \mathsf{TCommit}_{ck}(td)$ in the first round
    2. com_1 can be later equivocated to anything depending on the derived joint challenge c.

Simulation with TDCOM

$\boxed{\mathsf{Sim}(\mathbf{t}_1, \mathbf{t} = \mathbf{t}_1 + \mathbf{A}\mathbf{s}_2, ck, td)}$ $\boxed{\mathcal{A}(\mathbf{s}_2, \mathbf{t}, ck)}$

$\xrightarrow{com_1 \ = \ \mathsf{TCommit}_{ck}(td)}$

$c \leftarrow \mathsf{H}(com_1 + com_2, m, \mathbf{t})$ $\xleftarrow{com_2 \ = \ \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)}$

$\mathbf{z}_1 \leftarrow_\$ D^{\ell+k}; \mathbf{w}_1 = \mathbf{A}\mathbf{z}_1 - c\mathbf{t}_1$

$r_1 \leftarrow \mathsf{Eqv}_{ck}(td, com_1, \mathbf{w}_1)$

With prob. $1 - 1/M : (\mathbf{z}_1, r_1) := (\bot, \bot)$ $\xrightarrow{\mathbf{z}_1, r_1}$

If $\mathbf{z}_i = \bot :$ restart $\xleftarrow{\mathbf{z}_2, r_2}$

Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$

Simulation with TDCOM

$\mathsf{Sim}(\mathbf{t}_1, \mathbf{t} = \mathbf{t}_1 + \mathbf{A}\mathbf{s}_2)$		$\mathcal{A}(\mathbf{s}_2, \mathbf{t})$
$ck \leftarrow \mathsf{H}(m, \mathbf{t})$		
//Invoke $(ck, td) \leftarrow \mathsf{TCGen}$ and program $\mathsf{H}(m, \mathbf{t}) := ck$	$\xrightarrow{com_1 = \mathsf{TCommit}_{ck}(td)}$	
$c \leftarrow \mathsf{H}(com_1 + com_2, m, \mathbf{t})$	$\xleftarrow{com_2 = \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)}$	
$\mathbf{z}_1 \leftarrow_\$ D^{\ell+k}; \mathbf{w}_1 = \mathbf{A}\mathbf{z}_1 - c\mathbf{t}_1$		
$r_1 \leftarrow \mathsf{Eqv}_{ck}(td, com_1, \mathbf{w}_1)$		
With prob. $1 - 1/M : (\mathbf{z}_1, r_1) := (\perp, \perp)$	$\xrightarrow{\mathbf{z}_1, r_1}$	
If $\mathbf{z}_i = \perp$: restart	$\xleftarrow{\mathbf{z}_2, r_2}$	
Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$		

Our two-round protocol: the final form

$P_1(\mathbf{s}_1, \mathbf{t} = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2))$		$P_2(\mathbf{s}_2, \mathbf{t})$
$ck \leftarrow \mathsf{H}(m, \mathbf{t})$		$ck \leftarrow \mathsf{H}(m, \mathbf{t})$
$\mathbf{y}_1 \leftarrow_\$ D^{\ell+k}; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$	$\xrightarrow{com_1 = \mathsf{Commit}_{ck}(\mathbf{w}_1; r_1)}$	
$c \leftarrow \mathsf{H}(com_1 + com_2, m, \mathbf{t})$	$\xleftarrow{com_2 = \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)}$	
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : (\mathbf{z}_1, r_1) := (\perp, \perp)$	$\xrightarrow{\mathbf{z}_1, r_1}$	
If $\mathbf{z}_i = \perp :$ restart	$\xleftarrow{\mathbf{z}_2, r_2}$	
Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$		

Summary of the two-round protocol

- Per-message ck prevents the concurrent k-list sum attack.
- TDCOM requires **computationally binding** $\rightsquigarrow$ security proof relies on the forking lemma (leading to a larger security loss)
- Paper describes how to instantiate a lattice-based TDCOM from Baum et al's commitment [BDL+18] + Micciancio–Peikert lattice trapdoor [MP12].

Takeaways

- Multi-party FSwA signing with low round complexity & without FHE/MPC
- By deriving **per-user challenges** $c_i = \mathsf{H}(com, \mu, \mathbf{t}_i, L)$ our construction can be turned into a **two-round multi-signature** secure in the plain public-key model (= no dedicated key generation protocol is needed)
- Open questions:
  - Make the signature size less dependent on the number of parties n
  - Tighter security reduction & proof in QROM

Thank you! & Questions?

More details at `https://ia.cr/2020/1110`

Takeaways

- Multi-party FSwA signing with low round complexity & without FHE/MPC
- By deriving **per-user challenges** $c_i = \mathsf{H}(com, \mu, \mathbf{t}_i, L)$ our construction can be turned into a **two-round multi-signature** secure in the plain public-key model (= no dedicated key generation protocol is needed)
- Open questions:
  - Make the signature size less dependent on the number of parties n
  - Tighter security reduction & proof in QROM

Thank you! & Questions?

More details at `https://ia.cr/2020/1110`

Takeaways

- Multi-party FSwA signing with low round complexity & without FHE/MPC
- By deriving **per-user challenges** $c_i = \mathsf{H}(com, \mu, \mathbf{t}_i, L)$ our construction can be turned into a **two-round multi-signature** secure in the plain public-key model (= no dedicated key generation protocol is needed)
- Open questions:
  - Make the signature size less dependent on the number of parties n
  - Tighter security reduction & proof in QROM

Thank you! & Questions?

More details at `https://ia.cr/2020/1110`

Takeaways

- Multi-party FSwA signing with low round complexity & without FHE/MPC
- By deriving **per-user challenges** $c_i = \mathsf{H}(com, \mu, \mathbf{t}_i, L)$ our construction can be turned into a **two-round multi-signature** secure in the plain public-key model (= no dedicated key generation protocol is needed)
- Open questions:
  - Make the signature size less dependent on the number of parties n
  - Tighter security reduction & proof in QROM

Thank you! & Questions?

More details at `https://ia.cr/2020/1110`

Michel Abdalla, Pierre-Alain Fouque, Vadim Lyubashevsky, and Mehdi Tibouchi.
Tightly secure signatures from lossy identification schemes.
Journal of Cryptology, 29(3):597–631, July 2016.

Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert.
More efficient commitments from structured lattice assumptions.
In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 368–385. Springer, Heidelberg, September 2018.

- Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai.
  Threshold cryptosystems from threshold fully homomorphic encryption.
  In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.
- Rikke Bendlin, Sara Krehbiel, and Chris Peikert.
  How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE.
  In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 218–236. Springer, Heidelberg, June 2013.

Katharina Boudgoust and Adeline Roux-Langlois.
Compressed linear aggregate signatures based on module lattices.
Cryptology ePrint Archive, Report 2021/263, 2021.
https://eprint.iacr.org/2021/263.

Rachid El Bansarkhani and Jan Sturm.
An efficient lattice-based multisignature scheme with applications to bitcoins.
In Sara Foresti and Giuseppe Persiano, editors, *CANS 16*, volume 10052 of *LNCS*, pages 140–155. Springer, Heidelberg, November 2016.

Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs.
On the security of two-round multi-signatures.
In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE Computer Society Press, May 2019.

Yarkın Doröz, Jeffrey Hoffstein, Joseph H. Silverman, and Berk Sunar.
Mmsat: A scheme for multimessage multiuser signature aggregation.
Cryptology ePrint Archive, Report 2020/520, 2020.
`https://eprint.iacr.org/2020/520`.

Masayuki Fukumitsu and Shingo Hasegawa.
A lattice-based provably secure multisignature scheme in quantum random oracle model.
In *ProvSec 2020*, LNCS, pages xxx–xxx. Springer, 2020.

Freepik.
Icons made by Freepik from Flaticon.com.
http://www.flaticon.com.

Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan.
Trapdoors for hard lattices and new cryptographic constructions.
In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

Vadim Lyubashevsky.
Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures.
In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009.

Daniele Micciancio and Chris Peikert.
Trapdoors for lattices: Simpler, tighter, faster, smaller.
In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

David Wagner.
A generalized birthday problem.
In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002.

Concurrent attack against bare-bone protocol I

$\mathcal{A}$ (malicious) has $\mathbf{s}'$; P (honest) has $\mathbf{s}$; joint public key is $\mathbf{t} = \mathbf{A}(\mathbf{s}' + \mathbf{s})$

1. $\mathcal{A}$ starts k concurrent sessions on the same m; receive $\mathbf{w}_1, \ldots, \mathbf{w}_k$ from P
2. Let $\mathbf{w}^* = \mathbf{w}_1 + \ldots + \mathbf{w}_k$; Find m^*, $\mathbf{w}'_1, \ldots, \mathbf{w}'_k$ such that
$$\begin{aligned} c^* = \mathsf{H}(\mathbf{w}^*, m^*, \mathbf{t}) &= \mathsf{H}(\mathbf{w}_1 + \mathbf{w}'_1, m, \mathbf{t}) + \ldots + \mathsf{H}(\mathbf{w}_k + \mathbf{w}'_k, m, \mathbf{t}) \\ &= c_1 + \ldots + c_k \end{aligned}$$
by solving a **sparse, ternary variant of the generalized birthday problem for** $(k+1)$ **trees** [Wag02]: GBP over $\left(C = \left\{ c \in \mathbb{Z}^N : \|c\|_1 = \kappa \wedge \|c\|_\infty = 1 \right\}, +\right)$
3. $\mathcal{A}$ resumes the sessions by sending $\mathbf{w}'_1, \ldots, \mathbf{w}'_k$; P returns $\mathbf{z}_1 = c_1\mathbf{s} + \mathbf{y}_1, \ldots, \mathbf{z}_k = c_k\mathbf{s} + \mathbf{y}_k$.
4. Output a forgery $(\mathbf{w}^*, \mathbf{z}^*, m^*)$ where
$$\mathbf{z}^* = c^*\mathbf{s}' + \mathbf{z}_1 + \ldots + \mathbf{z}_k$$

Concurrent attack against bare-bone protocol I

$\mathcal{A}$ (malicious) has $\mathbf{s}'$; P (honest) has $\mathbf{s}$; joint public key is $\mathbf{t} = \mathbf{A}(\mathbf{s}' + \mathbf{s})$

1. $\mathcal{A}$ starts k concurrent sessions on the same m; receive $\mathbf{w}_1, \ldots, \mathbf{w}_k$ from P
2. Let $\mathbf{w}^* = \mathbf{w}_1 + \ldots + \mathbf{w}_k$; Find $m^*, \mathbf{w}'_1, \ldots, \mathbf{w}'_k$ such that

$$\begin{aligned} c^* = \mathsf{H}(\mathbf{w}^*, m^*, \mathbf{t}) &= \mathsf{H}(\mathbf{w}_1 + \mathbf{w}'_1, m, \mathbf{t}) + \ldots + \mathsf{H}(\mathbf{w}_k + \mathbf{w}'_k, m, \mathbf{t}) \\ &= c_1 + \ldots + c_k \end{aligned}$$

   by solving a **sparse, ternary variant of the generalized birthday problem for $(k+1)$ trees** [Wag02]: GBP over $\left(C = \left\{c \in \mathbb{Z}^N : \|c\|_1 = \kappa \wedge \|c\|_\infty = 1\right\}, +\right)$

3. $\mathcal{A}$ resumes the sessions by sending $\mathbf{w}'_1, \ldots, \mathbf{w}'_k$; P returns $\mathbf{z}_1 = c_1\mathbf{s} + \mathbf{y}_1, \ldots, \mathbf{z}_k = c_k\mathbf{s} + \mathbf{y}_k$.
4. Output a forgery $(\mathbf{w}^*, \mathbf{z}^*, m^*)$ where

$$\mathbf{z}^* = c^*\mathbf{s}' + \mathbf{z}_1 + \ldots + \mathbf{z}_k$$

Concurrent attack against bare-bone protocol I

$\mathcal{A}$ (malicious) has $\mathbf{s}'$; P (honest) has $\mathbf{s}$; joint public key is $\mathbf{t} = \mathbf{A}(\mathbf{s}' + \mathbf{s})$

1. $\mathcal{A}$ starts k concurrent sessions on the same m; receive $\mathbf{w}_1, \ldots, \mathbf{w}_k$ from P
2. Let $\mathbf{w}^* = \mathbf{w}_1 + \ldots + \mathbf{w}_k$; Find $m^*, \mathbf{w}'_1, \ldots, \mathbf{w}'_k$ such that

$$c^* = \mathsf{H}(\mathbf{w}^*, m^*, \mathbf{t}) = \mathsf{H}(\mathbf{w}_1 + \mathbf{w}'_1, m, \mathbf{t}) + \ldots + \mathsf{H}(\mathbf{w}_k + \mathbf{w}'_k, m, \mathbf{t})$$
$$= c_1 + \ldots + c_k$$

   by solving a **sparse, ternary variant of the generalized birthday problem for** $(k+1)$ **trees** [Wag02]: GBP over $\left(C = \left\{ c \in \mathbb{Z}^N : \|c\|_1 = \kappa \wedge \|c\|_\infty = 1 \right\}, +\right)$
3. $\mathcal{A}$ resumes the sessions by sending $\mathbf{w}'_1, \ldots, \mathbf{w}'_k$; P returns $\mathbf{z}_1 = c_1\mathbf{s} + \mathbf{y}_1, \ldots, \mathbf{z}_k = c_k\mathbf{s} + \mathbf{y}_k$.
4. Output a forgery $(\mathbf{w}^*, \mathbf{z}^*, m^*)$ where

$$\mathbf{z}^* = c^*\mathbf{s}' + \mathbf{z}_1 + \ldots + \mathbf{z}_k$$

Concurrent attack against bare-bone protocol I

$\mathcal{A}$ (malicious) has $\mathbf{s}'$; P (honest) has $\mathbf{s}$; joint public key is $\mathbf{t} = \mathbf{A}(\mathbf{s}' + \mathbf{s})$

1. $\mathcal{A}$ starts k concurrent sessions on the same m; receive $\mathbf{w}_1, \ldots, \mathbf{w}_k$ from P
2. Let $\mathbf{w}^* = \mathbf{w}_1 + \ldots + \mathbf{w}_k$; Find m^*, $\mathbf{w}'_1, \ldots, \mathbf{w}'_k$ such that

$$\begin{aligned} c^* &= \mathsf{H}(\mathbf{w}^*, m^*, \mathbf{t}) = \mathsf{H}(\mathbf{w}_1 + \mathbf{w}'_1, m, \mathbf{t}) + \ldots + \mathsf{H}(\mathbf{w}_k + \mathbf{w}'_k, m, \mathbf{t}) \\ &= c_1 + \ldots + c_k \end{aligned}$$

   by solving a **sparse, ternary variant of the generalized birthday problem for** $(k+1)$ **trees** [Wag02]: GBP over $(C = \{c \in \mathbb{Z}^N : \|c\|_1 = \kappa \wedge \|c\|_\infty = 1\}, +)$
3. $\mathcal{A}$ resumes the sessions by sending $\mathbf{w}'_1, \ldots, \mathbf{w}'_k$; P returns $\mathbf{z}_1 = c_1\mathbf{s} + \mathbf{y}_1, \ldots, \mathbf{z}_k = c_k\mathbf{s} + \mathbf{y}_k$.
4. Output a forgery $(\mathbf{w}^*, \mathbf{z}^*, m^*)$ where

$$\mathbf{z}^* = c^*\mathbf{s}' + \mathbf{z}_1 + \ldots + \mathbf{z}_k$$

Concurrent attack against bare-bone protocol II

Why $(\mathbf{w}^*, \mathbf{z}^*, m^*)$ passes the verification:

- Thanks to the $(k+1)$-list sum solver $c^* = \mathsf{H}(\mathbf{w}^*, m^*, \mathbf{t}) = c_1 + \ldots + c_k$
- The forgery $\mathbf{z}^*$satisfies

$$\begin{aligned}\mathbf{z}^* &= c^*\mathbf{s}' + \mathbf{z}_1 + \ldots + \mathbf{z}_k \\ &= c^*\mathbf{s}' + (c_1 + \ldots + c_k)\mathbf{s} + (\mathbf{y}_1 + \ldots + \mathbf{y}_k) \\ &= c^*(\mathbf{s}' + \mathbf{s}) + (\mathbf{y}_1 + \ldots + \mathbf{y}_k)\end{aligned}$$

- Hence we have

$$\begin{aligned}\mathbf{A}\mathbf{z}^* - c^*\mathbf{t} &= \mathbf{A}(\mathbf{y}_1 + \ldots + \mathbf{y}_k) \\ &= \mathbf{w}^*\end{aligned}$$

- Verifier also checks $\|\mathbf{z}^*\|$ is small $\rightsquigarrow$ k should be sufficiently small.
  - Attack becomes easier for a general n-party setting

Concurrent attack against bare-bone protocol II

Why $(\mathbf{w}^*, \mathbf{z}^*, m^*)$ passes the verification:

- Thanks to the $(k+1)$-list sum solver $c^* = \mathsf{H}(\mathbf{w}^*, m^*, \mathbf{t}) = c_1 + \ldots + c_k$
- The forgery $\mathbf{z}^*$satisfies

$$\begin{aligned} \mathbf{z}^* &= c^*\mathbf{s}' + \mathbf{z}_1 + \ldots + \mathbf{z}_k \\ &= c^*\mathbf{s}' + (c_1 + \ldots + c_k)\mathbf{s} + (\mathbf{y}_1 + \ldots + \mathbf{y}_k) \\ &= c^*(\mathbf{s}' + \mathbf{s}) + (\mathbf{y}_1 + \ldots + \mathbf{y}_k) \end{aligned}$$

- Hence we have

$$\begin{aligned} \mathbf{A}\mathbf{z}^* - c^*\mathbf{t} &= \mathbf{A}(\mathbf{y}_1 + \ldots + \mathbf{y}_k) \\ &= \mathbf{w}^* \end{aligned}$$

- Verifier also checks $\|\mathbf{z}^*\|$ is small $\rightsquigarrow$ k should be sufficiently small.
  - Attack becomes easier for a general n-party setting

Concurrent attack against bare-bone protocol II

Why $(\mathbf{w}^*, \mathbf{z}^*, m^*)$ passes the verification:

- Thanks to the $(k+1)$-list sum solver $c^* = \mathsf{H}(\mathbf{w}^*, m^*, \mathbf{t}) = c_1 + \ldots + c_k$
- The forgery $\mathbf{z}^*$satisfies

$$\begin{aligned} \mathbf{z}^* &= c^*\mathbf{s}' + \mathbf{z}_1 + \ldots + \mathbf{z}_k \\ &= c^*\mathbf{s}' + (c_1 + \ldots + c_k)\mathbf{s} + (\mathbf{y}_1 + \ldots + \mathbf{y}_k) \\ &= c^*(\mathbf{s}' + \mathbf{s}) + (\mathbf{y}_1 + \ldots + \mathbf{y}_k) \end{aligned}$$

- Hence we have

$$\begin{aligned} \mathbf{A}\mathbf{z}^* - c^*\mathbf{t} &= \mathbf{A}(\mathbf{y}_1 + \ldots + \mathbf{y}_k) \\ &= \mathbf{w}^* \end{aligned}$$

- Verifier also checks $\|\mathbf{z}^*\|$ is small $\rightsquigarrow$ k should be sufficiently small.
  - Attack becomes easier for a general n-party setting

Concurrent attack against bare-bone protocol II

Why $(\mathbf{w}^*, \mathbf{z}^*, m^*)$ passes the verification:

- Thanks to the $(k+1)$-list sum solver $c^* = \mathsf{H}(\mathbf{w}^*, m^*, \mathbf{t}) = c_1 + \ldots + c_k$
- The forgery $\mathbf{z}^*$satisfies

$$\begin{aligned}\mathbf{z}^* &= c^*\mathbf{s}' + \mathbf{z}_1 + \ldots + \mathbf{z}_k \\ &= c^*\mathbf{s}' + (c_1 + \ldots + c_k)\mathbf{s} + (\mathbf{y}_1 + \ldots + \mathbf{y}_k) \\ &= c^*(\mathbf{s}' + \mathbf{s}) + (\mathbf{y}_1 + \ldots + \mathbf{y}_k)\end{aligned}$$

- Hence we have

$$\begin{aligned}\mathbf{A}\mathbf{z}^* - c^*\mathbf{t} &= \mathbf{A}(\mathbf{y}_1 + \ldots + \mathbf{y}_k) \\ &= \mathbf{w}^*\end{aligned}$$

- Verifier also checks $\|\mathbf{z}^*\|$ is small $\rightsquigarrow$ k should be sufficiently small.
  - Attack becomes easier for a general n-party setting

TDCOM

A trapdoor commitment scheme TCOM consists of the following algorithms in addition to (CSetup, CGen, Commit, Open).

- TCGen$(cpp) \to (ck, td)$: The trapdoor key generation algorithm that outputs a key ck and the trapdoor td.
- TCommit$_{ck}(td) \to com$: The trapdoor committing algorithm that outputs a commitment com.
- Eqv$_{ck}(td, com, msg) \to r$: The equivocation algorithm that outputs randomness r.
- **Security**: for any $msg \in S_{msg}$, the distribution of (msg, ck, com, r) generated by the above algorithms is indistinguishable from the one honestly generated by CGen and Commit.