Lattice-based distributed signing from the Fiat–Shamir with aborts paradigm

NIST MPTS Workshop

Based on “Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices”

(eprint 2020/1110)

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

[1]Aarhus University, Denmark

[2]NTT Corporation, Japan

AARHUS UNIVERSITET

NTT

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

- FSwA-style signature has a structure similar to the DL-based counterparts.
  - Many existing works on round-efficient n-party Schnorr-style signatures.
  - Drijvers et al. [DEF$^+$19] recently attacked & proposed **2-round** protocols.

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
- FSwA-style signature has a structure similar to the DL-based counterparts.
  - Many existing works on round-efficient n-party Schnorr-style signatures.
  - Drijvers et al. [DEF+19] recently attacked & proposed **2-round** protocols.

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

- FSwA-style signature has a structure similar to the DL-based counterparts.
  - Many existing works on round-efficient n-party Schnorr-style signatures.
  - Drijvers et al. [DEF$^+$19] recently attacked & proposed **2-round** protocols.

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

Bare-bone 2-party signing: Schnorr vs Dilithium

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

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

$P_1(\mathbf{s}_1, pk = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2))$		$P_2(\mathbf{s}_2, pk)$
$\mathbf{y}_1 \leftarrow_\$ D; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$	$\xrightarrow{\mathbf{w}_1}$	
$c \leftarrow \mathsf{H}(\mathbf{w}_1 + \mathbf{w}_2, m, pk)$	$\xleftarrow{\mathbf{w}_2}$	
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
If 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

$P_1(\mathbf{s}_1, pk = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2))$		$P_2(\mathbf{s}_2, pk)$
$\mathbf{y}_1 \leftarrow\$ D; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$	$\xrightarrow{\mathbf{w}_1}$	
$c \leftarrow \mathsf{H}(\mathbf{w}_1 + \mathbf{w}_2, m, pk)$	$\xleftarrow{\mathbf{w}_2}$	
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
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**

Recent observations in the DL-setting apply!

1. Variant of the **concurrent attack** against bare-bone 2-round protocols in DL
   - Idea: corrupt $\widetilde{P}_2$ adaptively chooses $\mathbf{w}_2$ after seeing honest P_1's $\mathbf{w}_1$
   - Vectorial variant of Wagner's k-list sum algorithm to find a valid forgery
2. **Homomorphic commitment** to the first message $\mathbf{w}_i$ saves!
   - Per-message commitment key $ck = \mathsf{H}(m, pk)$ is crucial to achieve secure 2-round protocol!

Recent observations in the DL-setting apply!

1. Variant of the **concurrent attack** against bare-bone 2-round protocols in DL
   - Idea: corrupt $\widetilde{P}_2$ adaptively chooses $\mathbf{w}_2$ after seeing honest P_1's $\mathbf{w}_1$
   - Vectorial variant of Wagner's k-list sum algorithm to find a valid forgery
2. **Homomorphic commitment** to the first message $\mathbf{w}_i$ saves!
   - Per-message commitment key $ck = \mathsf{H}(m, pk)$ is crucial to achieve secure 2-round protocol!

Provably secure 2-round protocol: the final form

$P_1(\mathbf{s}_1, pk = \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2))$		$P_2(\mathbf{s}_2, pk)$
$ck \leftarrow \mathsf{H}(m, pk)$		$ck \leftarrow \mathsf{H}(m, pk)$
$\mathbf{y}_1 \leftarrow\$ D; \mathbf{w}_1 = \mathbf{A}\mathbf{y}_1$	$com_1 = \mathsf{Commit}_{ck}(\mathbf{w}_1; r_1)$ $\longrightarrow$	
$c \leftarrow \mathsf{H}(com_1 + com_2, m, pk)$	$\longleftarrow$ $com_2 = \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)$	
$\mathbf{z}_1 = c\mathbf{s}_1 + \mathbf{y}_1$		
If $\mathsf{RejSamp}(c\mathbf{s}_1, \mathbf{z}_1) = 0 : (\mathbf{z}_1, \mathbf{w}_1, r_1) := (\bot, \bot)$	$\mathbf{z}_1, r_1$ $\longrightarrow$	
If $\mathbf{z}_i = \bot :$ restart	$\longleftarrow$ $\mathbf{z}_2, r_2$	
Output $((com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2, r_1 + r_2), m)$		

Provably secure 2-round protocol: the final form

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

Provably secure 2-round protocol: the final form

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

Takeaways

- Progress in multi-party DL signing highly affects lattice-based counterparts!
- Several subtle differences:
  - Issue with "aborts"
    - Security proof is more involved
    - Need for many parallel repetitions in the n-party setting for large n
  - Poor quality of SIS solution in the security reduction for large n
  - Unclear if the same approach generalizes to t-out-of-n signing

Thank you!

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

Takeaways

- Progress in multi-party DL signing highly affects lattice-based counterparts!
- Several subtle differences:
  - Issue with "aborts"
    - Security proof is more involved
    - Need for many parallel repetitions in the n-party setting for large n
  - Poor quality of SIS solution in the security reduction for large n
  - Unclear if the same approach generalizes to t-out-of-n signing

Thank you!

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

Takeaways

- Progress in multi-party DL signing highly affects lattice-based counterparts!
- Several subtle differences:
  - Issue with "aborts"
    - Security proof is more involved
    - Need for many parallel repetitions in the n-party setting for large n
  - Poor quality of SIS solution in the security reduction for large n
  - Unclear if the same approach generalizes to t-out-of-n signing

Thank you!

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

References i

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.

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.

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

$\mathcal{A}$ (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 $(C = \left\{ c \in \mathbb{Z}^N : \|c\|_1 = \kappa \wedge \|c\|_\infty = 1 \right\}, +)$

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

$\mathcal{A}$ (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

$\mathcal{A}$ (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

$\mathcal{A}$ (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

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

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 $\leadsto$ k should be sufficiently small.
  - Attack becomes easier for a general n-party setting

Concurrent attack against bare-bone protocol

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

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