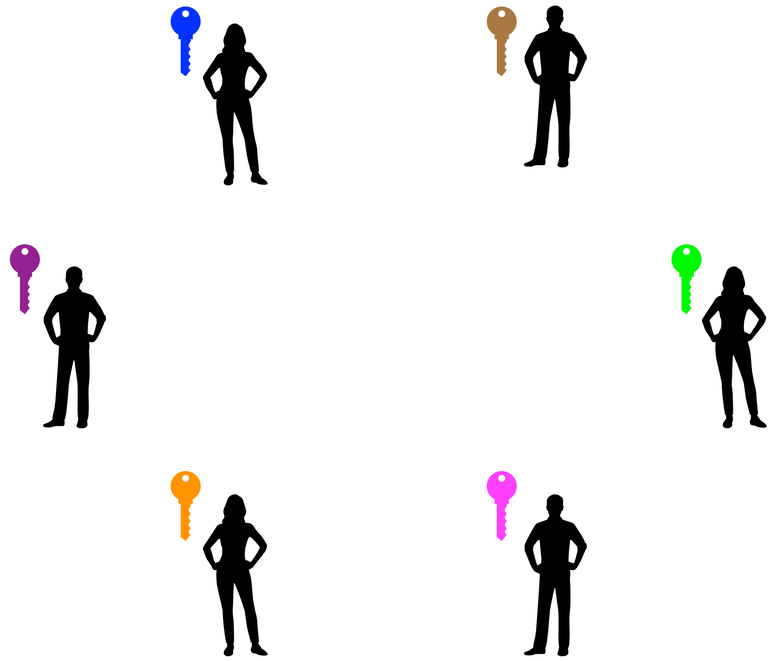
Amber: Lattice-Based Threshold KEM with Active Security

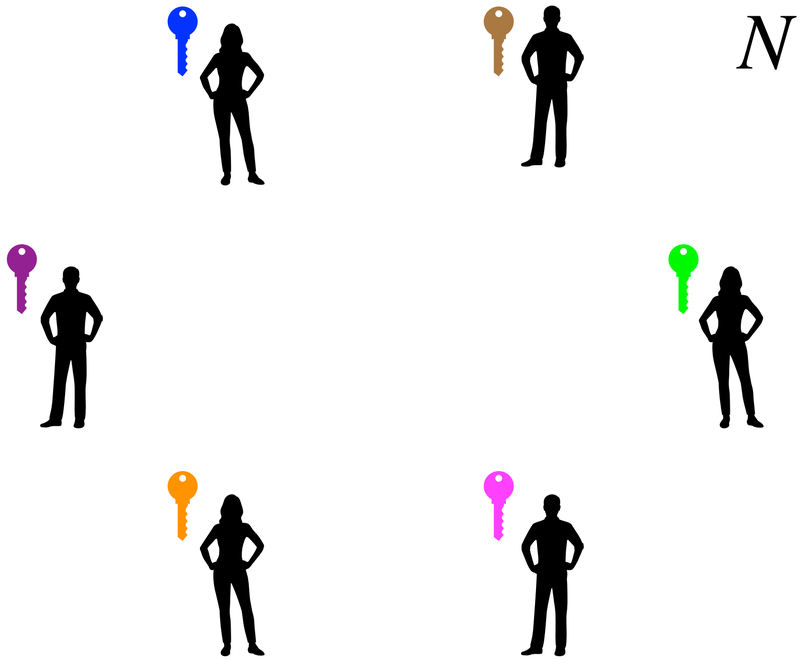
Sasha Lapiha @ Silence Labs 19 Jan 2026

*joint work with Katharina Boudgoust, Rafael del Pino and Thomas Prest

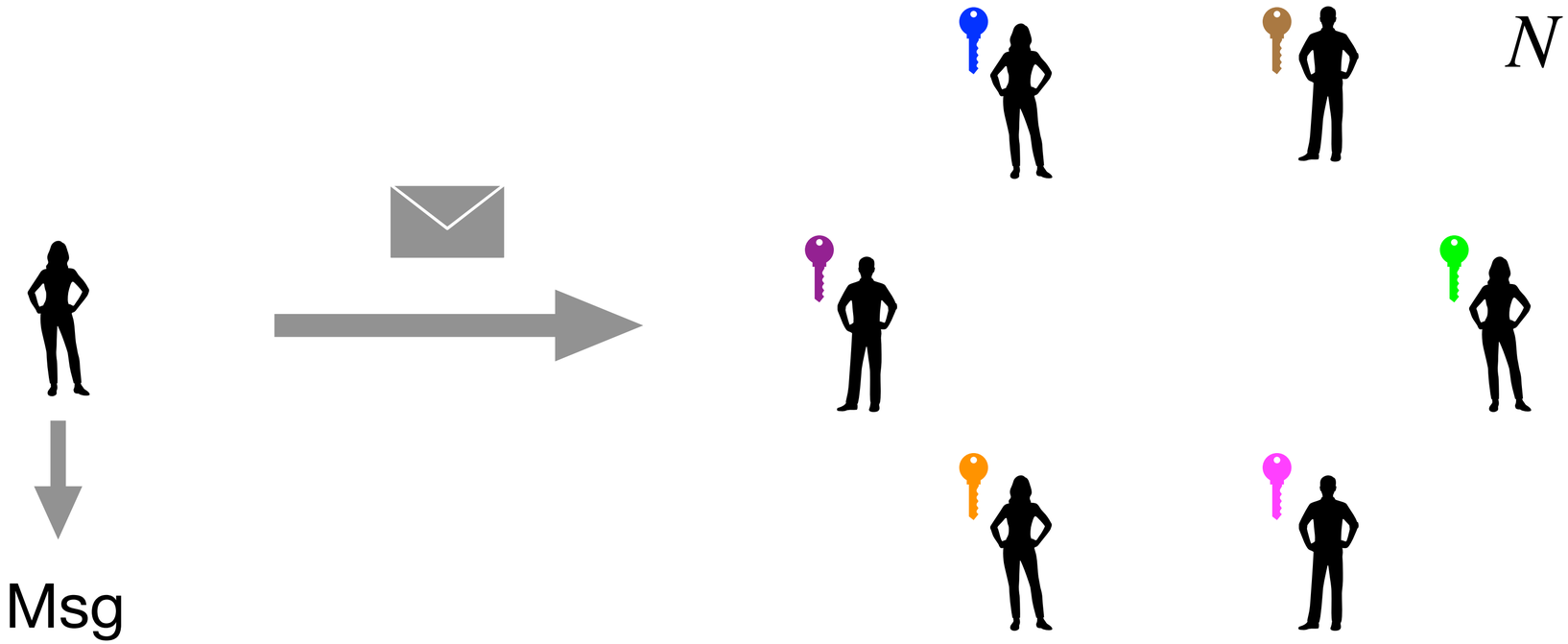
Threshold Cryptography



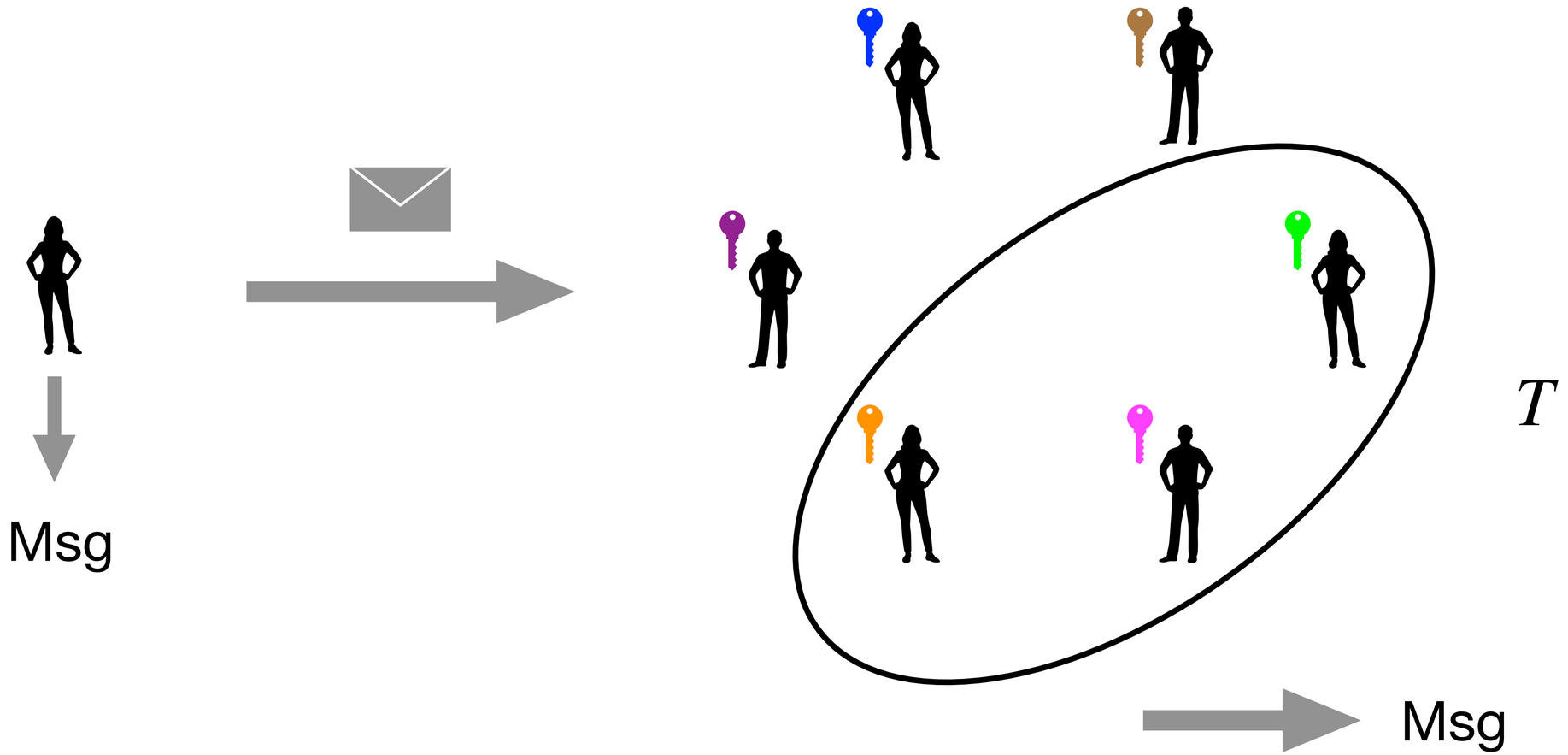
Threshold Decryption



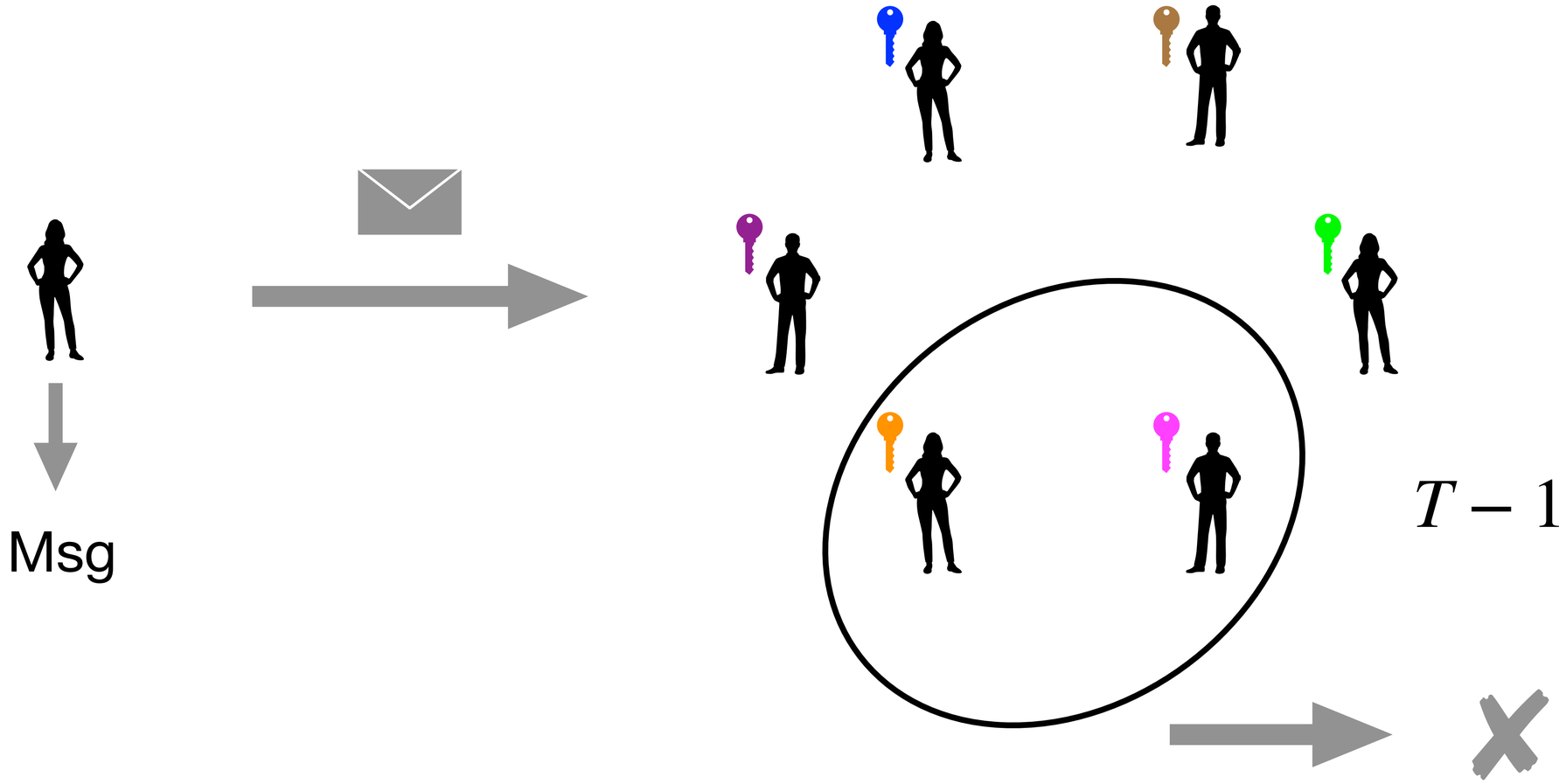
Threshold Decryption



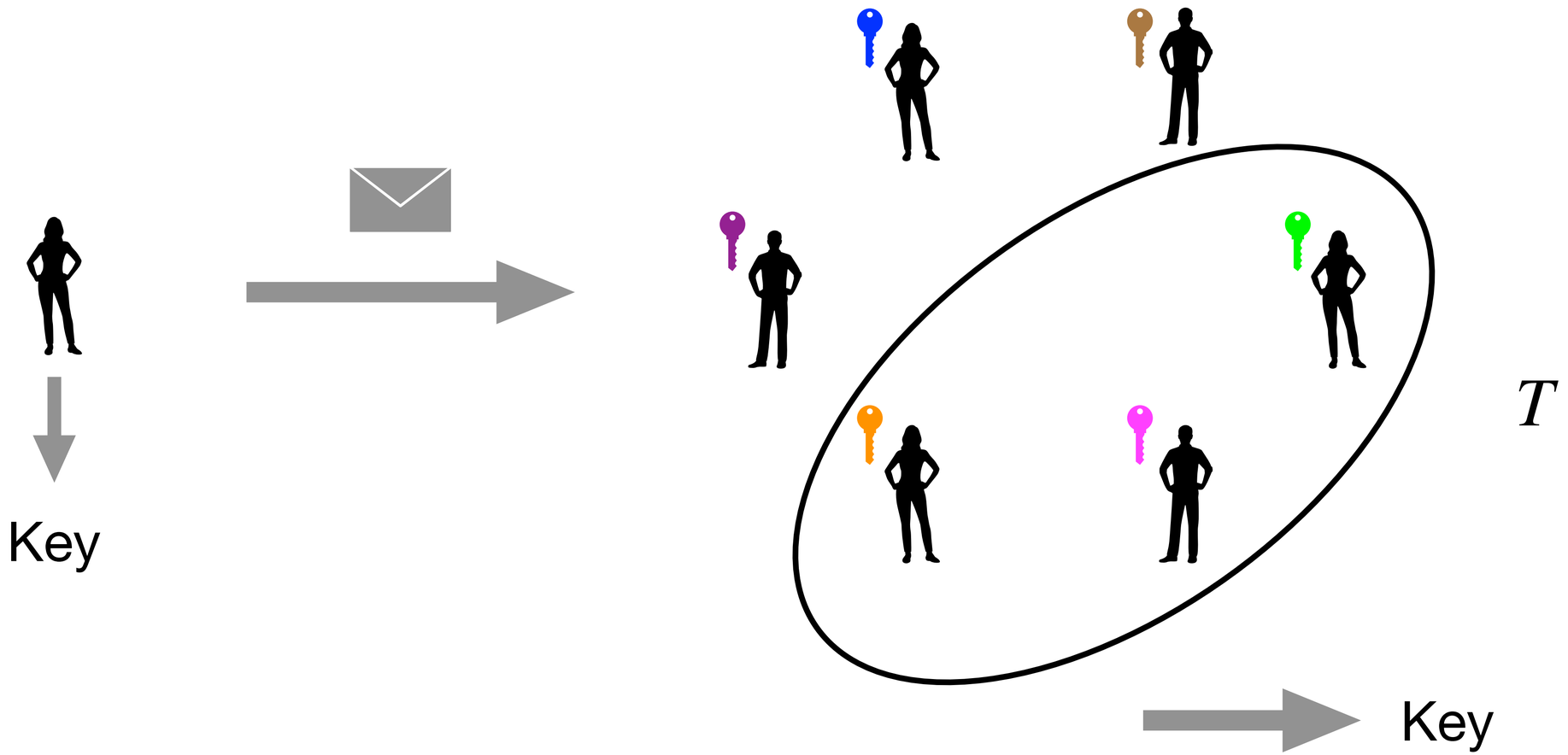
Threshold Decryption



Threshold Decryption

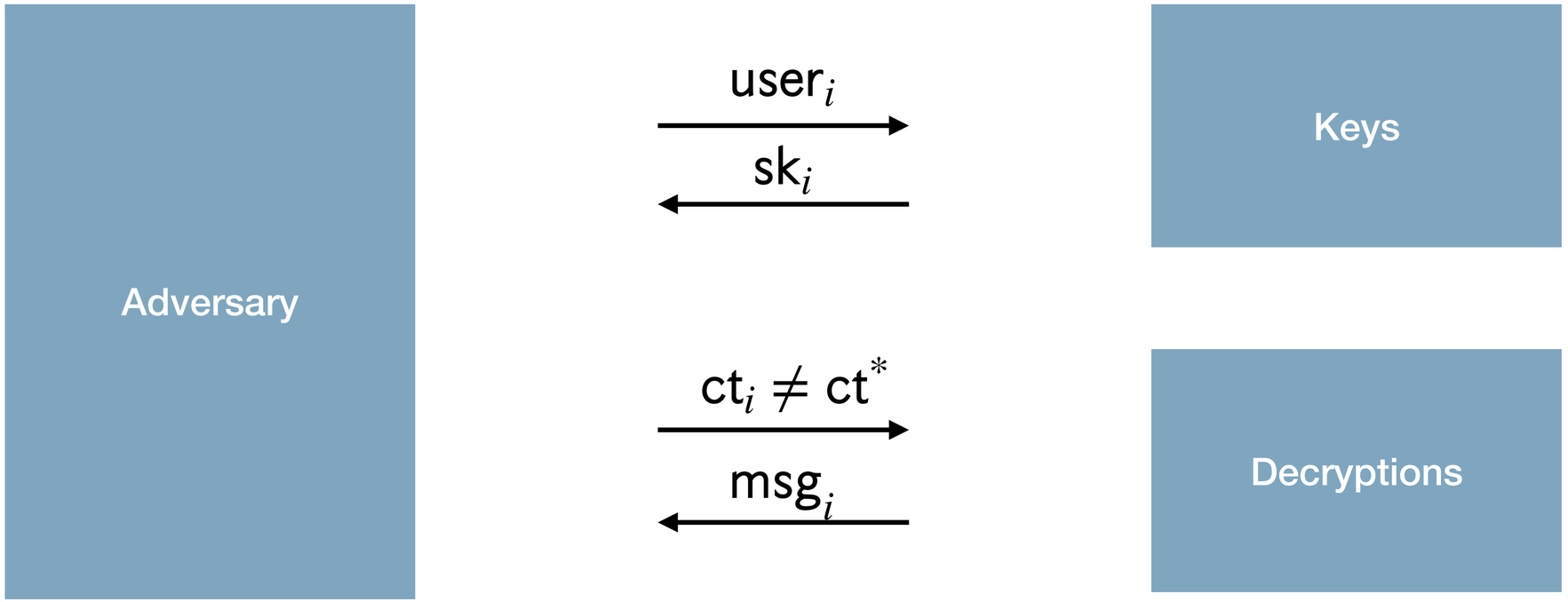


Threshold Decapsulation (TKEM)

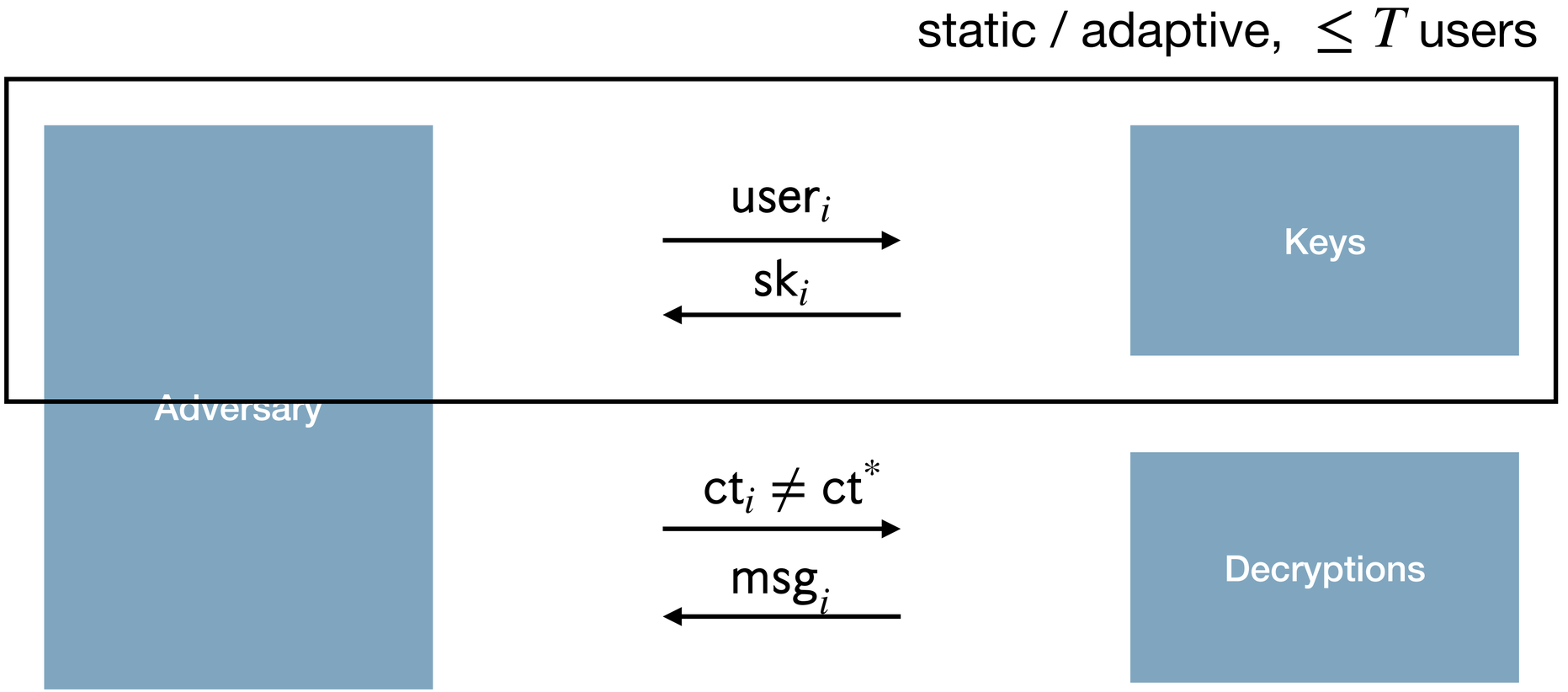


Threshold Active Security

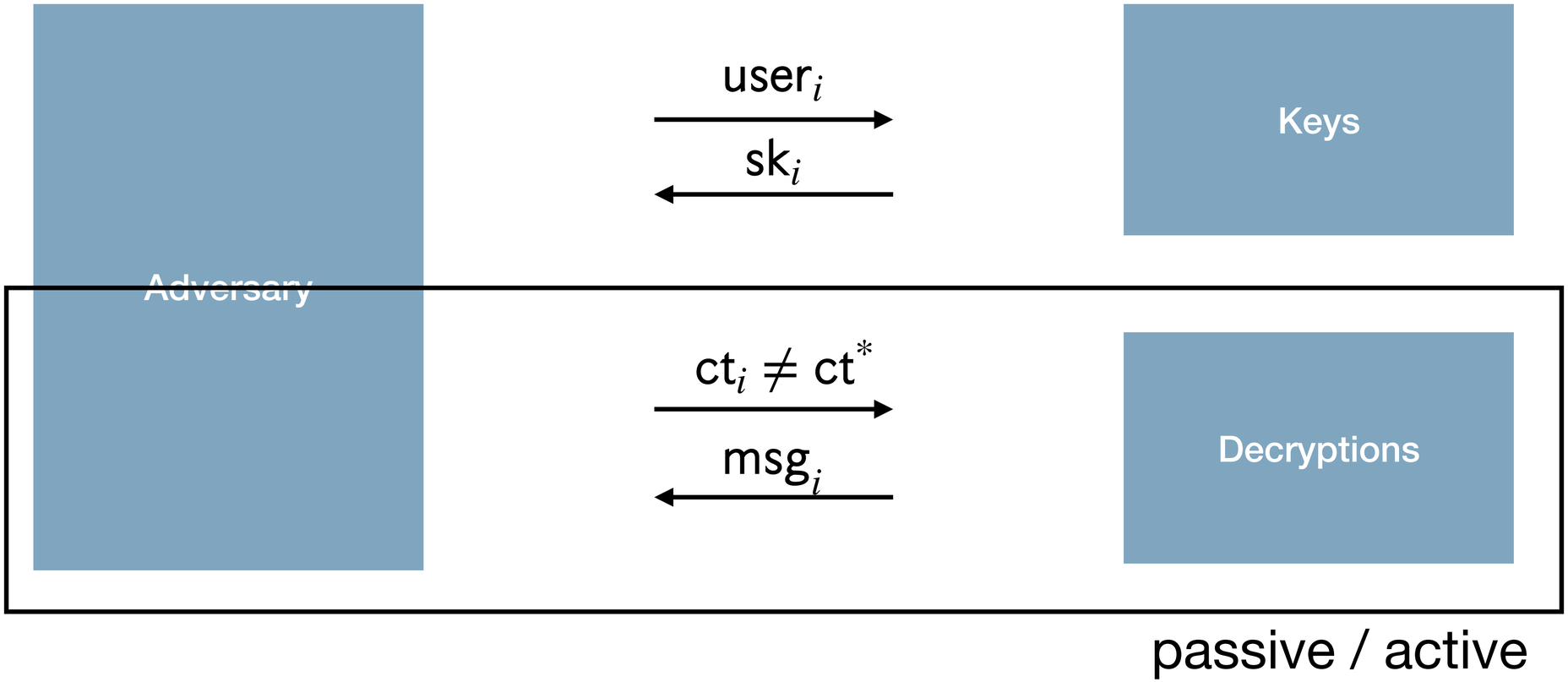
Security Model



Security Model



Security Model



FO transform (non-threshold case)

$$\text{msg} \xleftarrow{\$} \{0,1\}^{128}$$
$$\text{ct} = \text{Enc}(\text{msg}, \text{pk})$$

and

$$\text{msg} = \text{Dec}(\text{ct}, \text{sk})$$

FO transform (non-threshold case)

$$\text{msg} \xleftarrow{\$} \{0,1\}^{128}$$
$$\text{ct} = \text{Enc}(\text{msg}, \text{pk})$$

and

$$\text{msg} = \text{Dec}(\text{ct}, \text{sk})$$

Enforce honest encryption

- Generate encryption randomness via PRG
- Encrypt the PRG seed

FO transform (non-threshold case)

$$\text{msg} \xleftarrow{\$} \{0,1\}^{128}$$
$$\text{ct} = \text{Enc}(\text{msg}, \text{pk})$$

and

$$\text{msg} = \text{Dec}(\text{ct}, \text{sk})$$

Enforce honest encryption

- Generate encryption randomness via PRG
- Encrypt the PRG seed
- Recover the seed from Dec()
- Re-encrypt with the same seed and compare ciphertexts

FO transform (non-threshold case)

$\text{msg} \xleftarrow{\$} \{0,1\}^{128}$
$\text{ct} = \text{Enc}(\text{msg}, \text{pk})$

and

$\text{msg} = \text{Dec}(\text{ct}, \text{sk})$

$\text{msg} \xleftarrow{\$} \{0,1\}^{128}$
$\text{rand} = \text{PRG}(\text{msg})$
$\text{ct} = \text{Enc}(\text{msg}, \text{pk}; \text{rand})$

and

$\text{msg} = \text{Dec}(\text{ct}, \text{sk})$
$\text{rand} = \text{PRG}(\text{msg})$
$\text{Enc}(\text{msg}, \text{pk}; \text{rand}) == \text{ct}?$

Obtaining active security

MPC

FHE

ZK Proofs

Generic ?

BCHK Transform

BCHK Transform

Identity-based Encryption

$(sk, pp) \leftarrow$ ibe-KeyGen()
$ct =$ ibe-Enc(msg, pp, id)

and

$sk_{id} =$ ibe-Extract(sk, id)
$msg =$ ibe-Dec(ct, sk_{id})

BCHK Transform

$ct = Enc(msg, pk)$

and

$msg = Dec(ct, sk)$

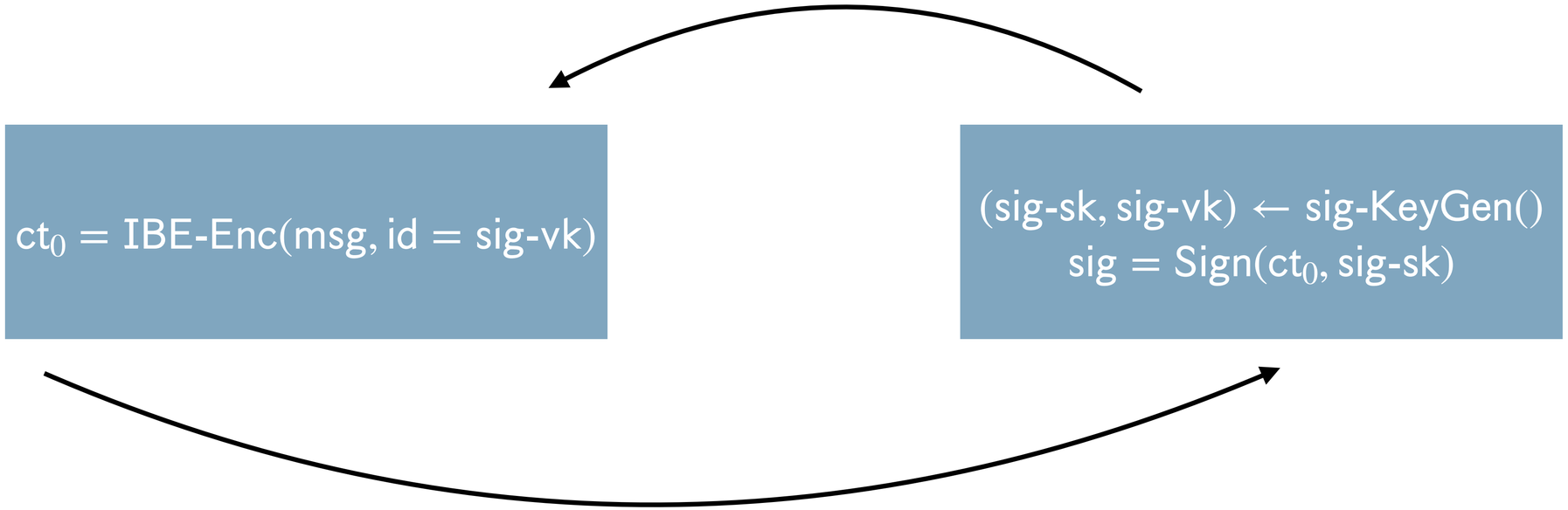
$(sig\text{-}sk, sig\text{-}vk) \leftarrow sig\text{-}KeyGen()$

$ct_0 = ibe\text{-}Enc(msg, id = sig\text{-}vk)$

$sig = Sign(ct_0, sig\text{-}sk)$

$ct = (ct_0, sig, sig\text{-}vk)$

BCHK Transform



BCHK Transform

$ct = Enc(msg, pk)$

and

$msg = Dec(ct, sk)$

$(sig\text{-}sk, sig\text{-}vk) \leftarrow sig\text{-}KeyGen()$
$ct_0 = ibe\text{-}Enc(msg, id = sig\text{-}vk)$
$sig = Sign(ct_0, sig\text{-}sk)$
$ct = (ct_0, sig, sig\text{-}vk)$

and

$Verify(ct_0, sig, sig\text{-}vk) == 1?$
$sk_{id} = Extract(sk, id = sig\text{-}vk)$
$msg = Dec(ct_0, sk_{id})$

Threshold BCHK

$ct = Enc(msg, pk)$

and

$msg = Dec(ct, sk)$

$(sig\text{-}sk, sig\text{-}vk) \leftarrow sig\text{-}KeyGen()$
$ct_0 = ibe\text{-}Enc(msg, id = sig\text{-}vk)$
$sig = Sign(ct_0, sig\text{-}sk)$
$ct = (ct_0, sig, sig\text{-}vk)$

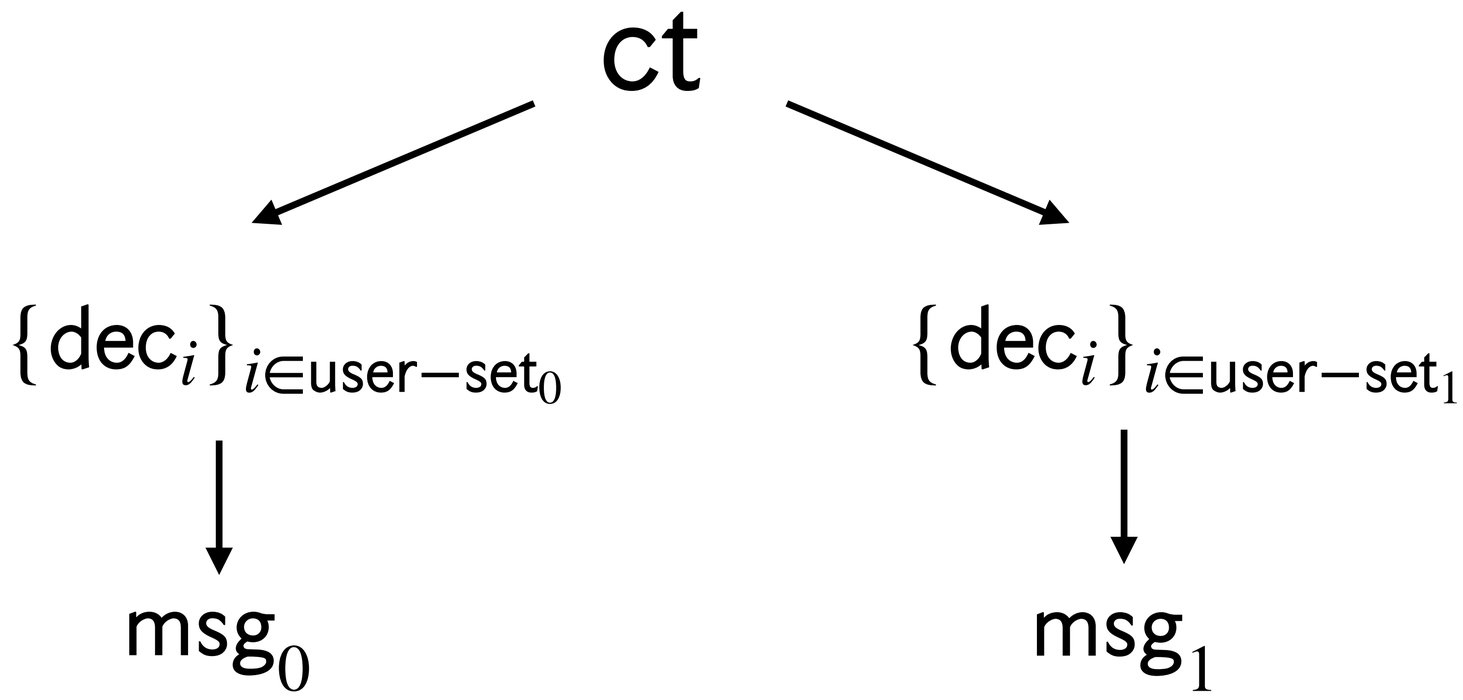
and

$Verify(ct_0, sig, sig\text{-}vk) == 1?$
$sk_{id} = Extract(sk, id = sig\text{-}vk)$
$msg = Dec(ct_0, sk_{id})$

Threshold BCHK

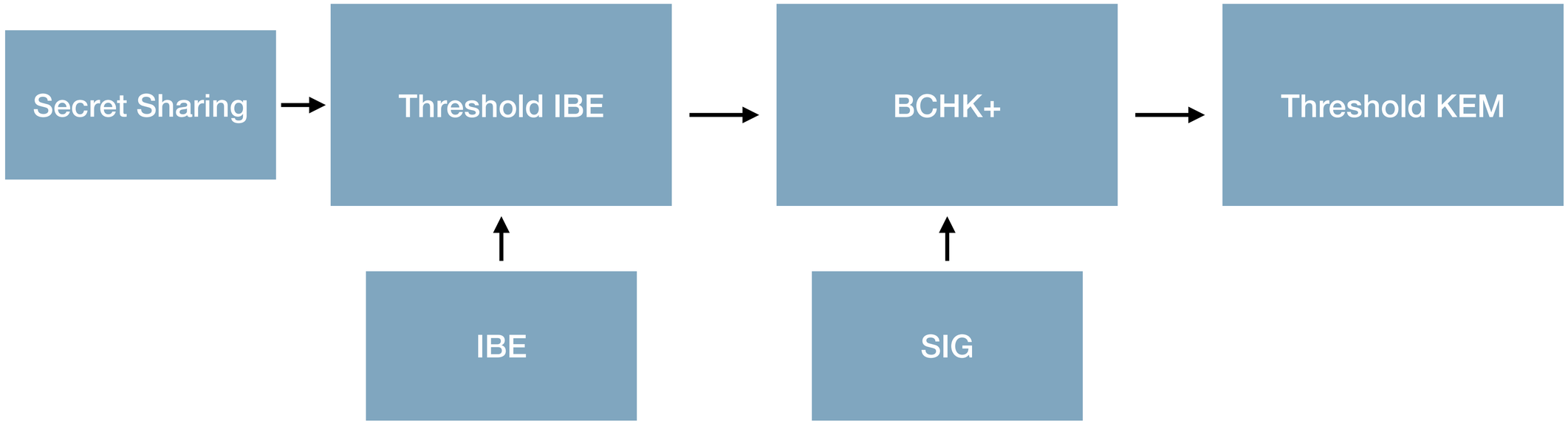
BCHK+ = BCHK + FO

Decryption Consistency



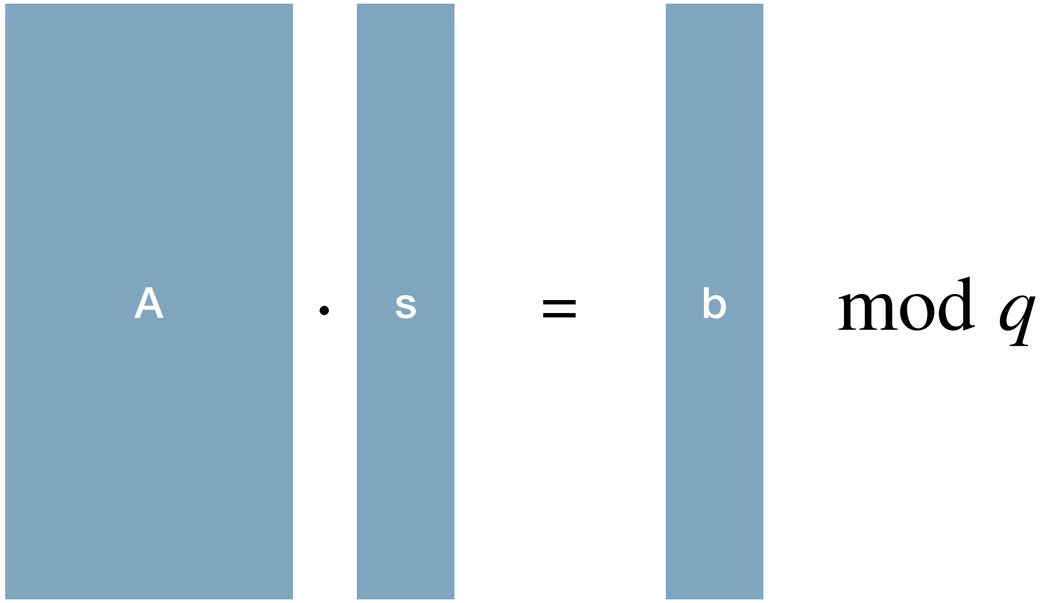
Amber: The Lattice Construction

The Blueprint



Learning with Errors Problem

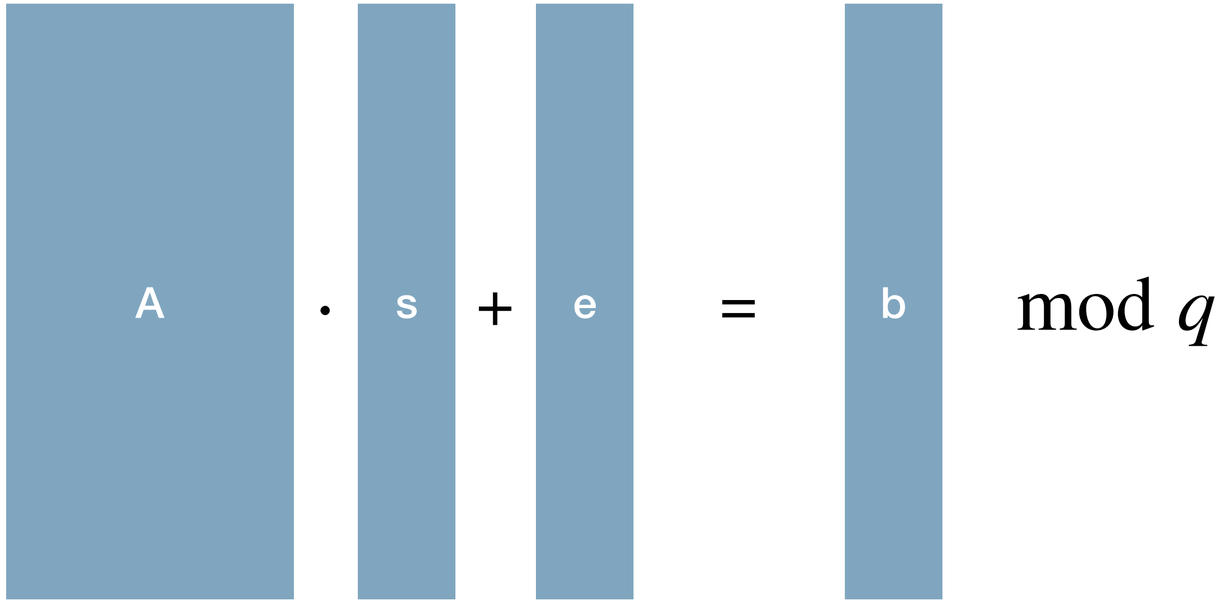
Easy problem: find unique **s** such that



$$\cdot \quad = \quad \bmod q$$

Learning with Errors Problem

Hard problem: find **s** and a *short noise* **e** such that



$$\bmod q$$

Learning with Errors Problem

The set $\{\mathbf{y} \mid \mathbf{y} = \mathbf{A} \cdot \mathbf{s} \bmod q\}$ is a lattice.

Finding $\mathbf{e}$ from $\mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q$ is a *decoding problem.*

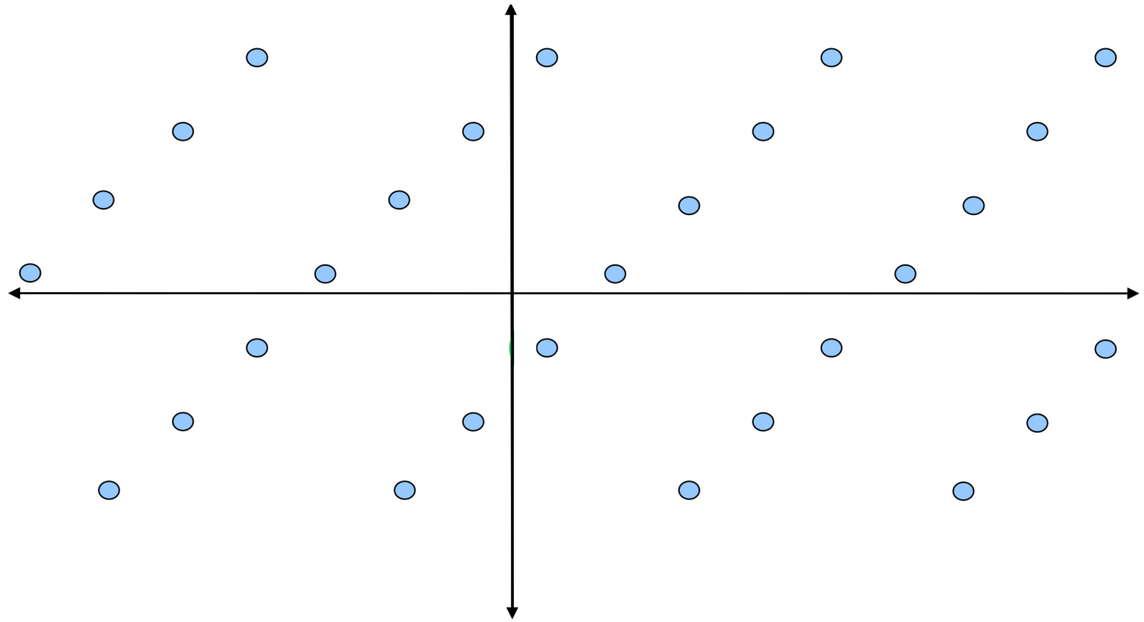


Image from V. Lyubashevsky, CRYSTALS-Dilithium Talk, 2018

Learning with Errors Problem

The set $\{\mathbf{y} \mid \mathbf{y} = \mathbf{A} \cdot \mathbf{s} \bmod q\}$ is a lattice.

Finding $\mathbf{e}$ from $\mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q$ is a *decoding problem.*

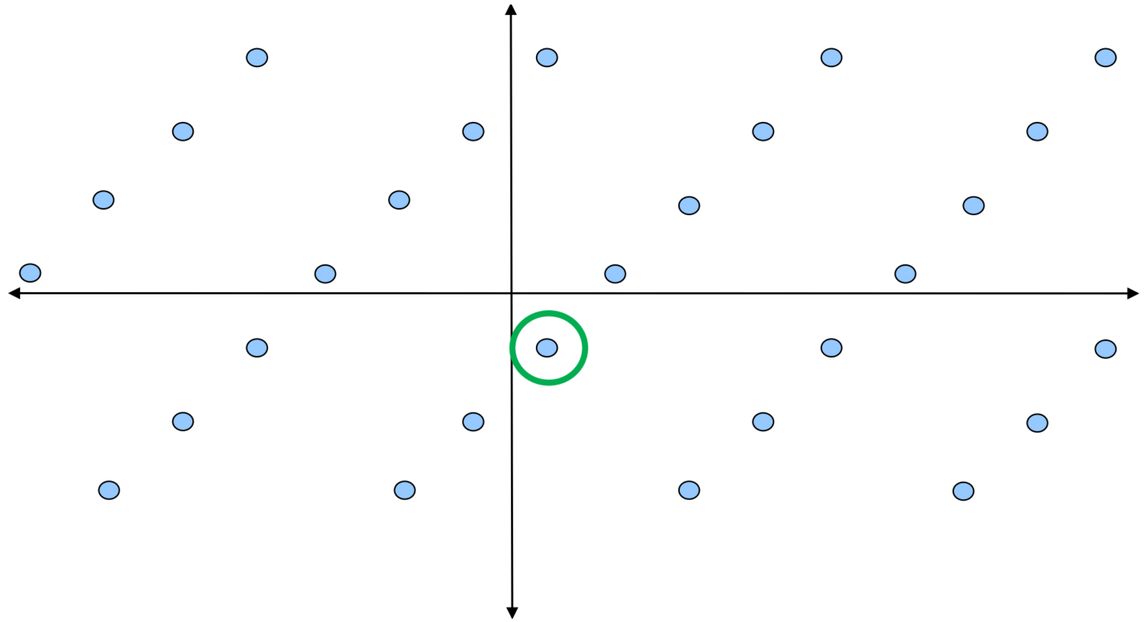


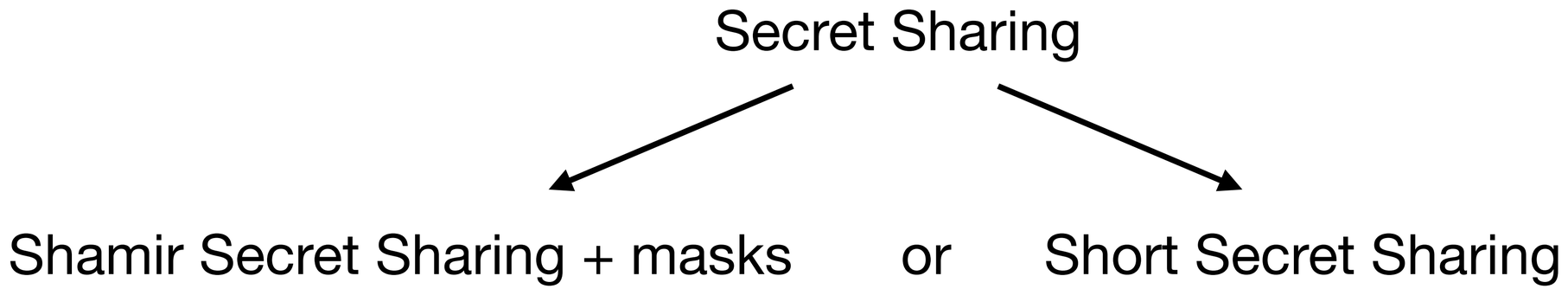
Image from V. Lyubashevsky, CRYSTALS-Dilithium Talk, 2018

Secret Sharing

Secret keys: short vectors **s** and **e**

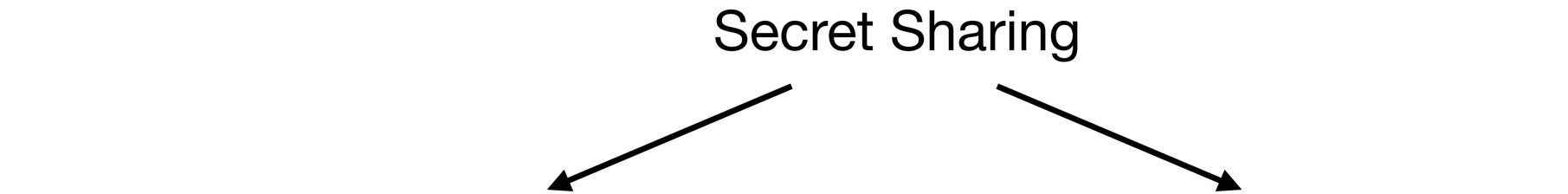
Secret Sharing

Secret keys: short vectors **s** and **e**



Secret Sharing

Secret keys: short vectors **s** and **e**

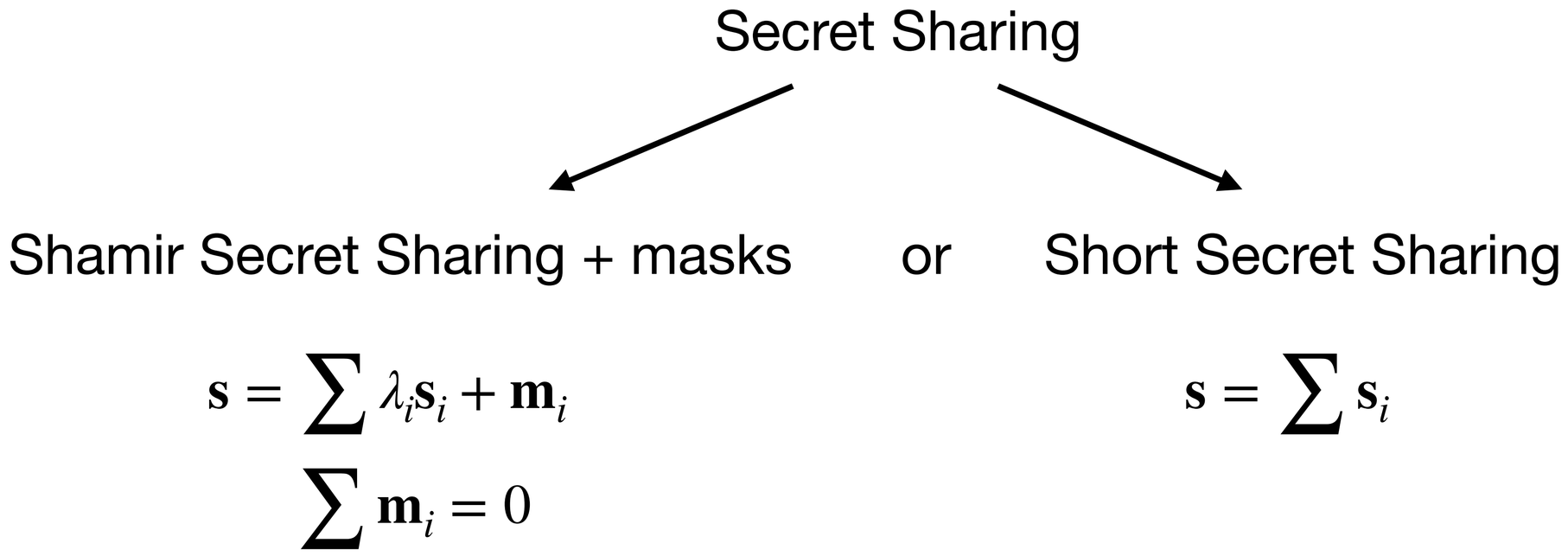


$$\mathbf{s} = \sum \lambda_i \mathbf{s}_i + \mathbf{m}_i$$

$$\sum \mathbf{m}_i = 0$$

Secret Sharing

Secret keys: short vectors **s** and **e**



Security Implications

Variant	Corruptions	Robustness	Group Size
Shamir SS	Adaptive	No	Large (< 1024)
Short SS	Static	Yes	Medium (< 64)

The parameters

Variant	Rounds	Decryption Queries	Pk size (bytes)	Ct size (bytes)
Shamir SS	3	2^46	6 688	28 544
Short SS	3	2^25	7 456	29 056

The number are stated for T = 32 and security level of 128 bits.

Future work

- NIST Threshold Call and Implementation
- Feedback on the security features
- Further reducing the sizes
- Kyber compatiblility

Summary

“Amber: Lattice-Based Threshold KEM with Active Security” | ia.cr/2025/1958, ia.cr/2026/021

- Active security definitions
- BCHK and FO transforms
- Amber blueprints
- Lattice-based secret sharing

Summary

“Amber: Lattice-Based Threshold KEM with Active Security” | ia.cr/2025/1958, ia.cr/2026/021

- Active security definitions
- BCHK and FO transforms
- Amber blueprints
- Lattice-based secret sharing

Thank you!

BCHK+ = BCHK + FO

TKEM.Keygen(1^κ)

$(\{\mathsf{dk}_i\}_{i\in[N]}, \mathsf{ek}) \leftarrow \mathsf{TIBE.Setup}(1^\kappa)$

$\mathbf{return}\ (\{\mathsf{dk}_i\}_{i\in[N]}, \mathsf{ek})$

TKEM.Encaps(ek)

$\mathsf{msg} \leftarrow \{0,1\}^d$

$\mathsf{rand} = G_{\mathsf{fo}}(\mathsf{msg})$

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SIG.Keygen}(1^\kappa)$

$\mathsf{ct} = \mathsf{TIBE.Encrypt}(\mathsf{ek}, \mathsf{id} = \mathsf{vk}, \mathsf{msg}; \mathsf{rand})$

$\mathsf{sig} \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}, \mathsf{ct})$

$K = H_{\mathsf{fo}}(\mathsf{msg}||\mathsf{ct})$

$\mathbf{return}\ (\mathsf{ct}, \mathsf{vk}, \mathsf{sig}), \mathrm{K}$

TKEM.Decaps$(\mathsf{dk}_i, (\mathsf{ct}, \mathsf{vk}, \mathsf{sig}))$

$\mathbf{assert}\ \mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{ct}, \mathsf{sig}) = 1$

$\mathsf{contrib}_i = \mathsf{TIBE.Extract}(\mathsf{dk}_i, \mathsf{id} = \mathsf{vk})$

$\mathbf{return}\ \mathsf{contrib}_i$

TKEM.Combine$(\{\mathsf{contrib}_i\}_{i\in\mathsf{act}}, (\mathsf{ct}, \mathsf{vk}, \mathsf{sig}))$

$\mathbf{assert}\ \mathsf{SIG.Verify}(\mathsf{vk}, \mathsf{ct}, \mathsf{sig}) = 1$

$\mathsf{msg} = \mathsf{TIBE.Combine}(\mathsf{ct}, \mathsf{vk}, \mathsf{act}, \{\mathsf{contrib}_i\}_i)$

$\mathsf{rand} = G_{\mathsf{fo}}(\mathsf{msg})$

$\mathbf{assert}\ \mathsf{ct} = \mathsf{TIBE.Encrypt}(\mathsf{ek}, \mathsf{vk}, \mathsf{msg}; \mathsf{rand})$

$K = H_{\mathsf{fo}}(\mathsf{msg}||\mathsf{ct})$

$\mathbf{return}\ K$

Dual-Regev Encryption Scheme

- Public key $(\mathbf{A} \in \mathbb{Z}_q^{m \times n}, \mathbf{r} \in \mathbb{Z}_q^n)$
- Secret key: short $\mathbf{T}_A \in \mathbb{Z}^{m \times m}$ s.t. $\mathbf{T}_A \cdot \mathbf{A} = \mathbf{0} \bmod q$
- Ciphertext: $\mathbf{u} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$, $v = \mathbf{r}^T\mathbf{s} + \mathbf{e}' + \text{msg}$
- Decryption key: short $\mathbf{x}$ s.t. $\mathbf{x}^T \cdot \mathbf{A} = \mathbf{r} \bmod q$

$$\mathbf{v} - \mathbf{x}^T \cdot \mathbf{u} = \mathbf{r}^T\mathbf{s} + \mathbf{e}' + \text{msg} - \mathbf{x}^T \cdot \mathbf{A} \cdot \mathbf{s} - \mathbf{x}^T \cdot \mathbf{e}$$

$$= \text{msg} + \mathbf{e}' - \mathbf{x}^T \cdot \mathbf{e} = \text{msg} + \text{noise}$$

Lattice-Based IBE

- Public key $(\mathbf{A} \in \mathbb{Z}_q^{m \times n}, \mathbf{r} \in \mathbb{Z}_q^n)$
- Master Secret key: short $\mathbf{T}_A \in \mathbb{Z}^{m \times m}$ s.t. $\mathbf{T}_A \cdot \mathbf{A} = \mathbf{0} \bmod q$
- Ciphertext: $\mathbf{u} = \begin{bmatrix} \mathbf{A} \\ \mathbf{H}(\mathrm{id}) \end{bmatrix} \cdot \mathbf{s} + \mathbf{e}$, $v = \mathbf{r}^T \mathbf{s} + \mathbf{e}' + \mathrm{msg}$
- Identity key: short $\mathbf{x}$ s.t. $\mathbf{x}^T \cdot \begin{bmatrix} \mathbf{A} \\ \mathbf{H}(\mathrm{id}) \end{bmatrix} = \mathbf{r}^T \bmod q$