

On the Adaptive Security of Key-Unique Threshold Signatures

Elizabeth Crites

Web3 Foundation

Chelsea Komlo

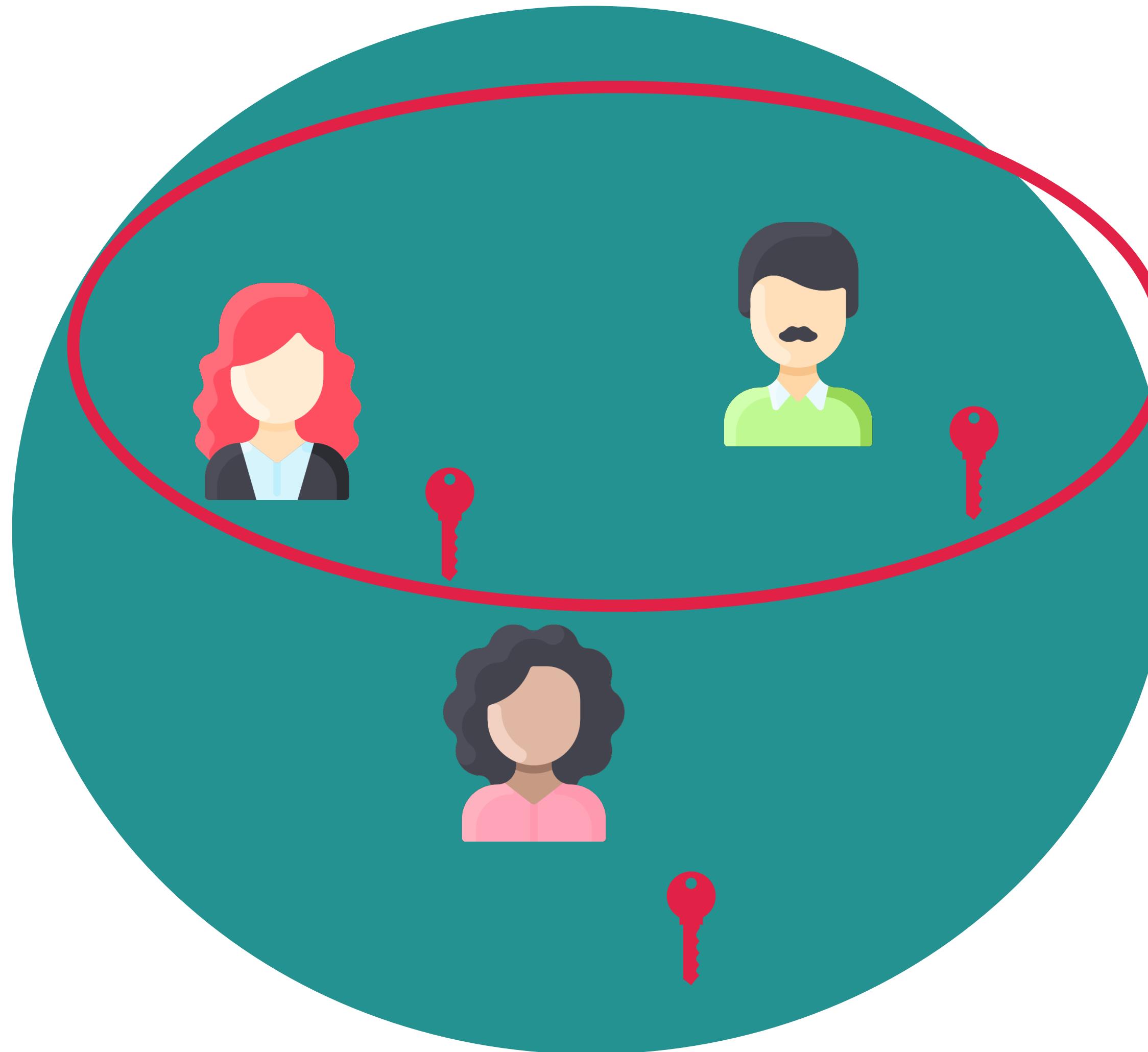
University of Waterloo
Near One

Mary Maller

Ethereal Foundation
PQShield

August 7, 2025

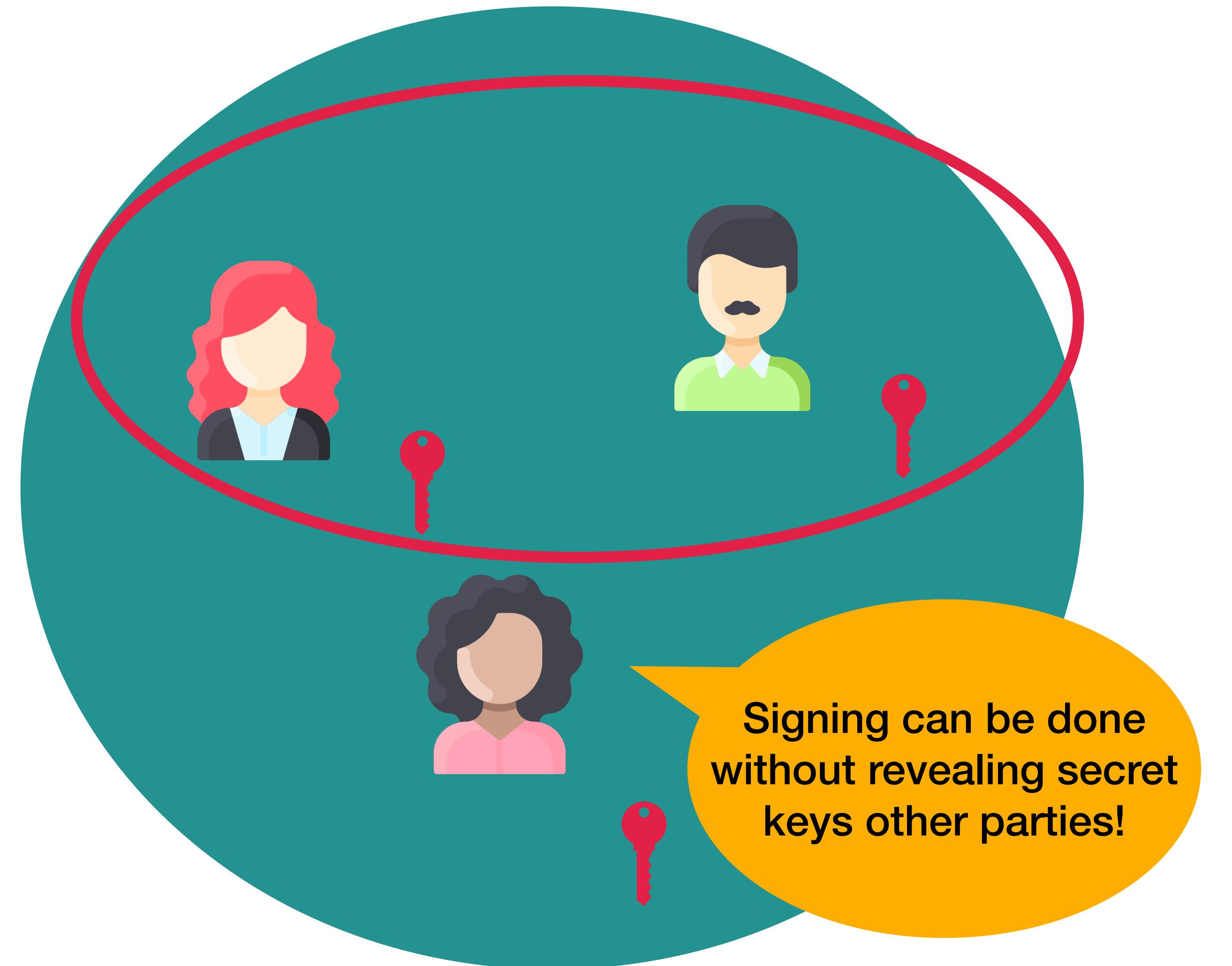
Threshold Signatures: Special Case of MPC



- Ideally t -out-of- n
- Key generation via trusted dealer or DKG
- Secure up to $(t-1)$ corruptions

(2,3) Example

Threshold Signatures: Special Case of MPC



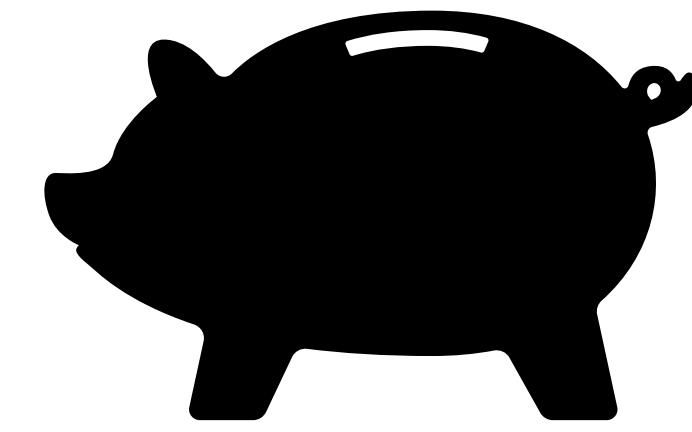
- Ideally t -out-of- n
- Key generation via trusted dealer or DKG
- Secure up to $(t-1)$ corruptions

(2,3) Example

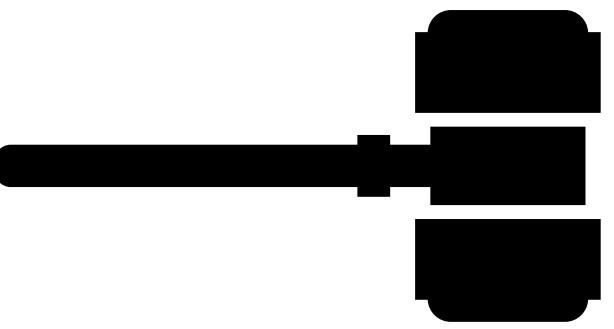
Uses for Threshold Signatures



Banks and Exchanges



Cryptocurrency Wallets



Trust Authorities (CAs)

Practical Interest in Threshold Signatures



Standards



NISTIR 8214C (Draft)

NIST First Call for Multi-Party Threshold Schemes

Date Published: January 25, 2023

Comments Due: April 10, 2023

Email Comments to: nistir-8214C-comments@nist.gov

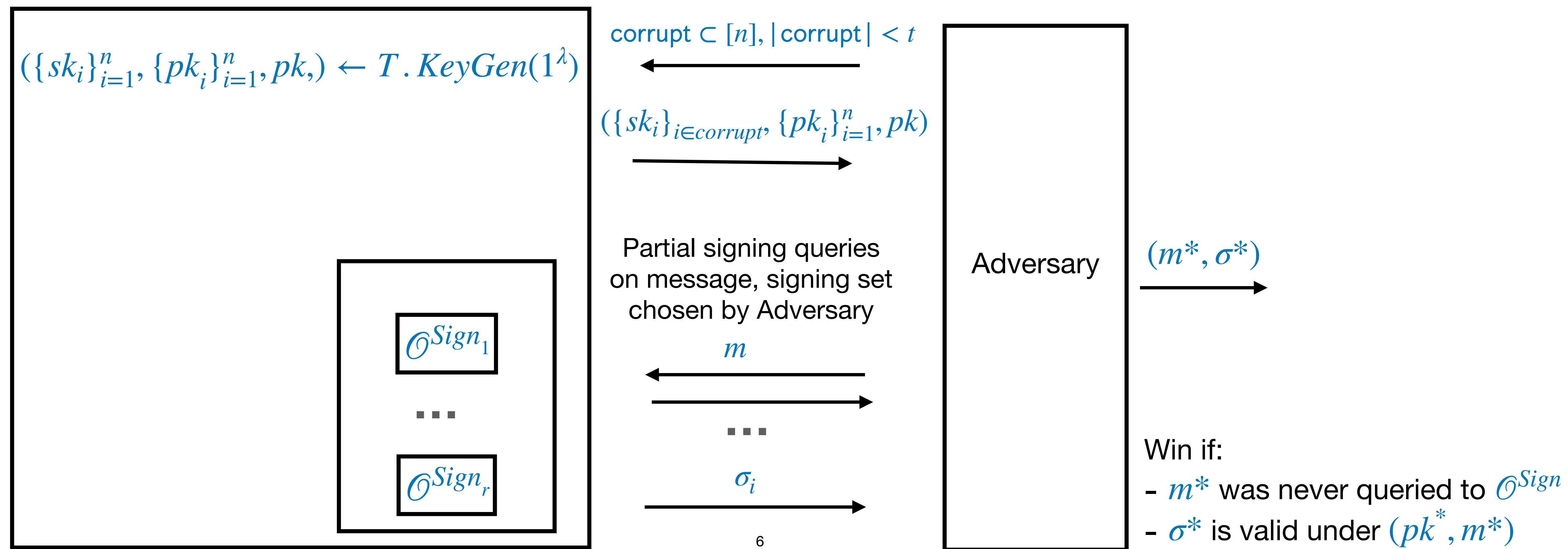
Author(s)

Luís T. A. N. Brandão (Strativia), Rene Peralta (NIST)

<https://csrc.nist.gov/publications/detail/nistir/8214c/draft>

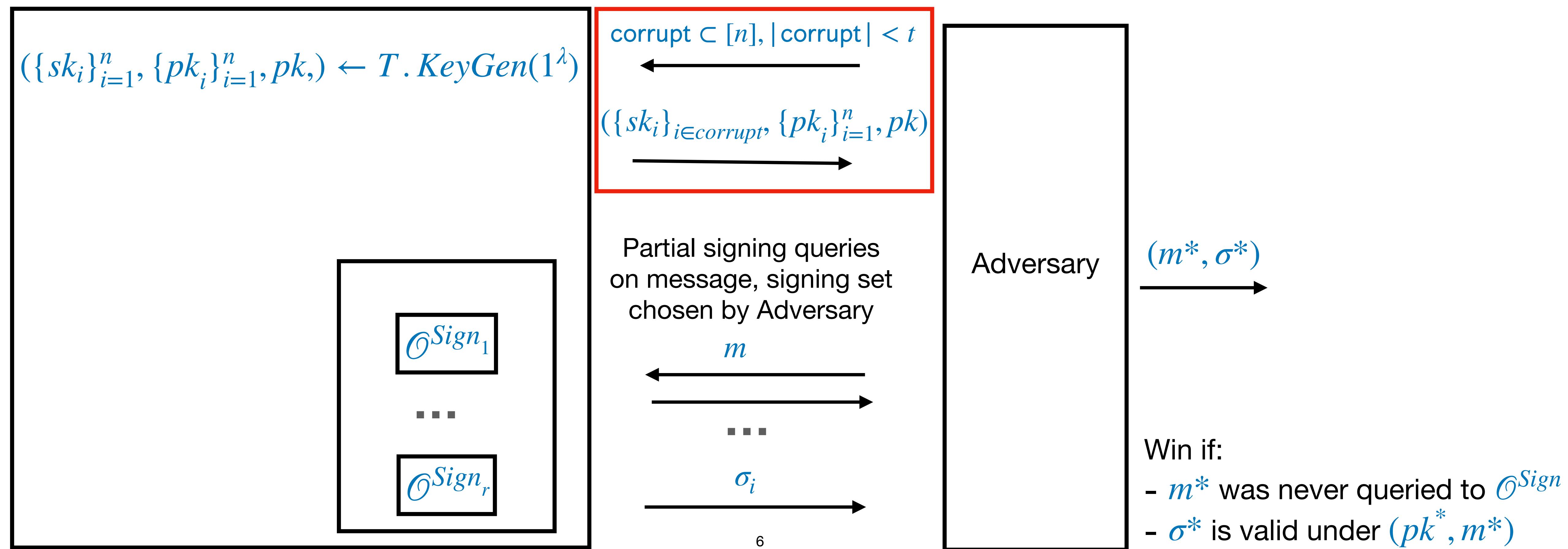
Static Unforgeability

A threshold signature scheme T is secure if no PPT adversary can win the following game with non-negligible advantage:



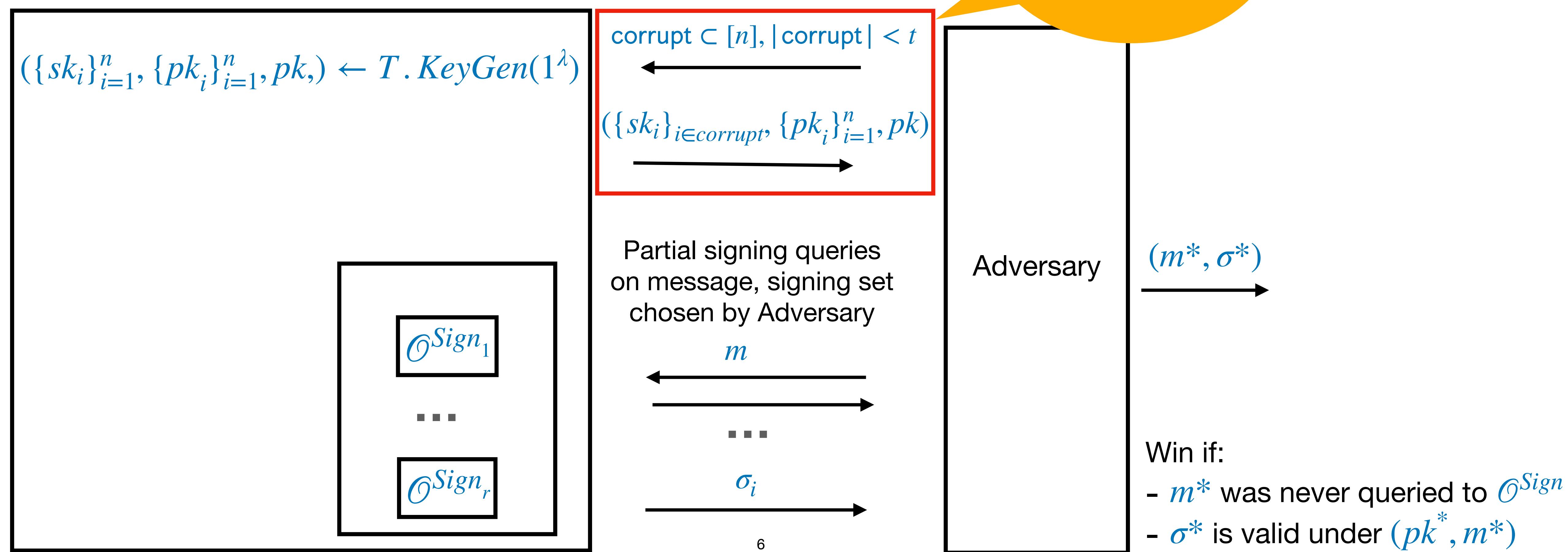
Static Unforgeability

A threshold signature scheme T is secure if no PPT adversary can win the following game with non-negligible advantage:



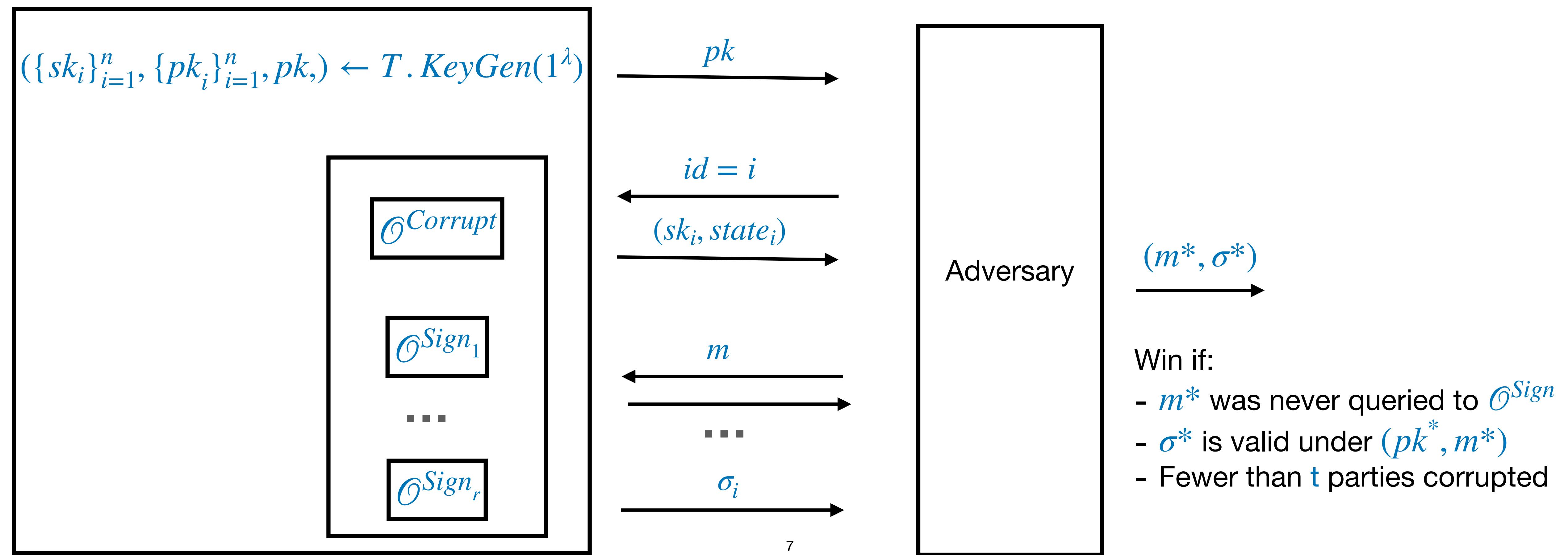
Static Unforgeability

A threshold signature scheme T is secure if no PPT adversary with non-negligible advantage:



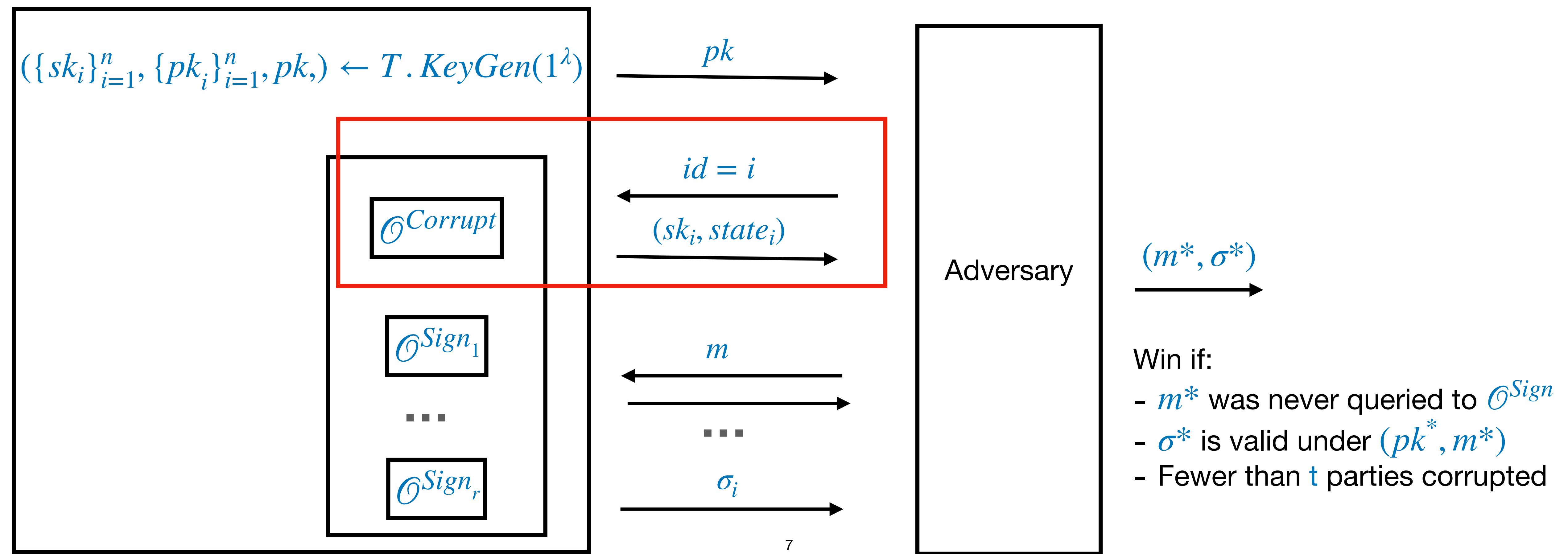
Adaptive Unforgeability

An adaptive adversary can choose on the fly which parties to corrupt. Receives not only the secret key but also the private state of the corrupted party.



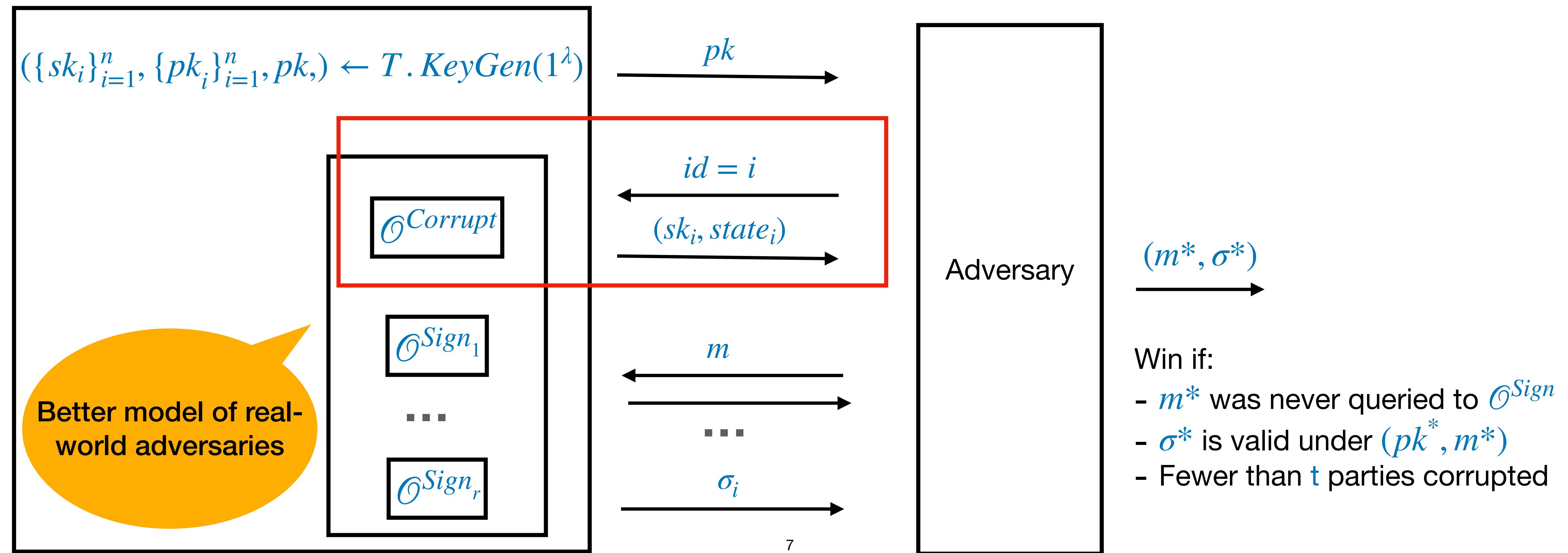
Adaptive Unforgeability

An adaptive adversary can choose on the fly which parties to corrupt. Receives not only the secret key but also the private state of the corrupted party.



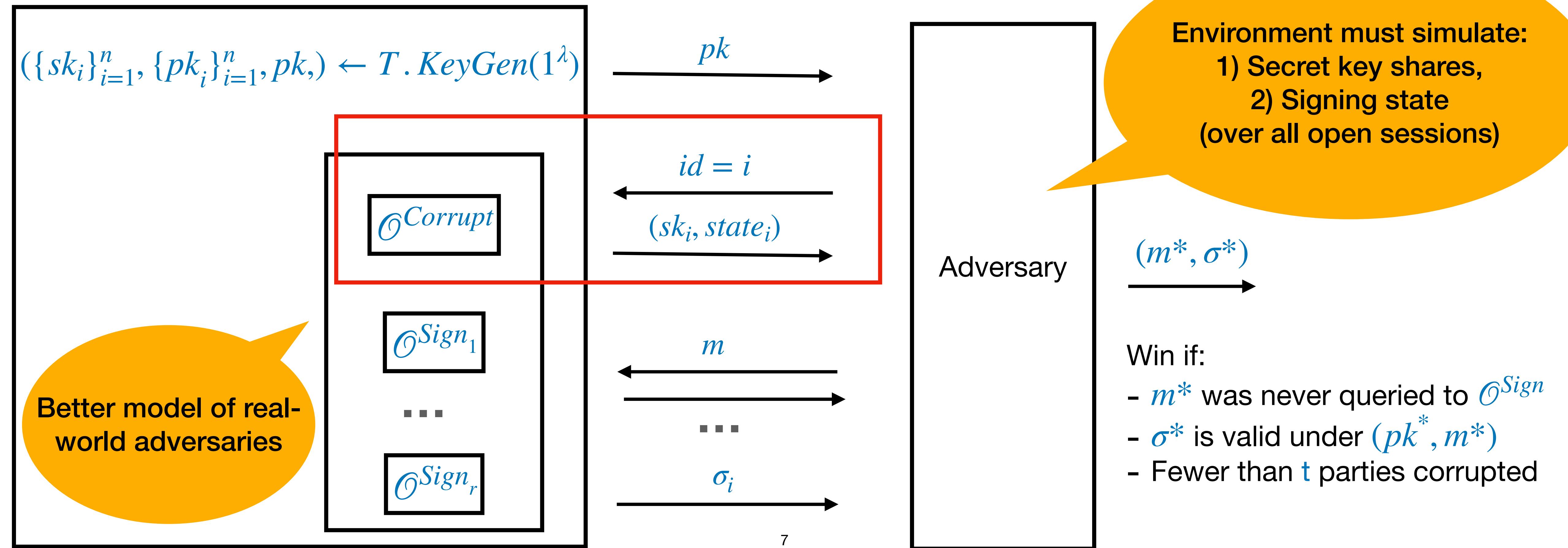
Adaptive Unforgeability

An adaptive adversary can choose on the fly which parties to corrupt. Receives not only the secret key but also the private state of the corrupted party.

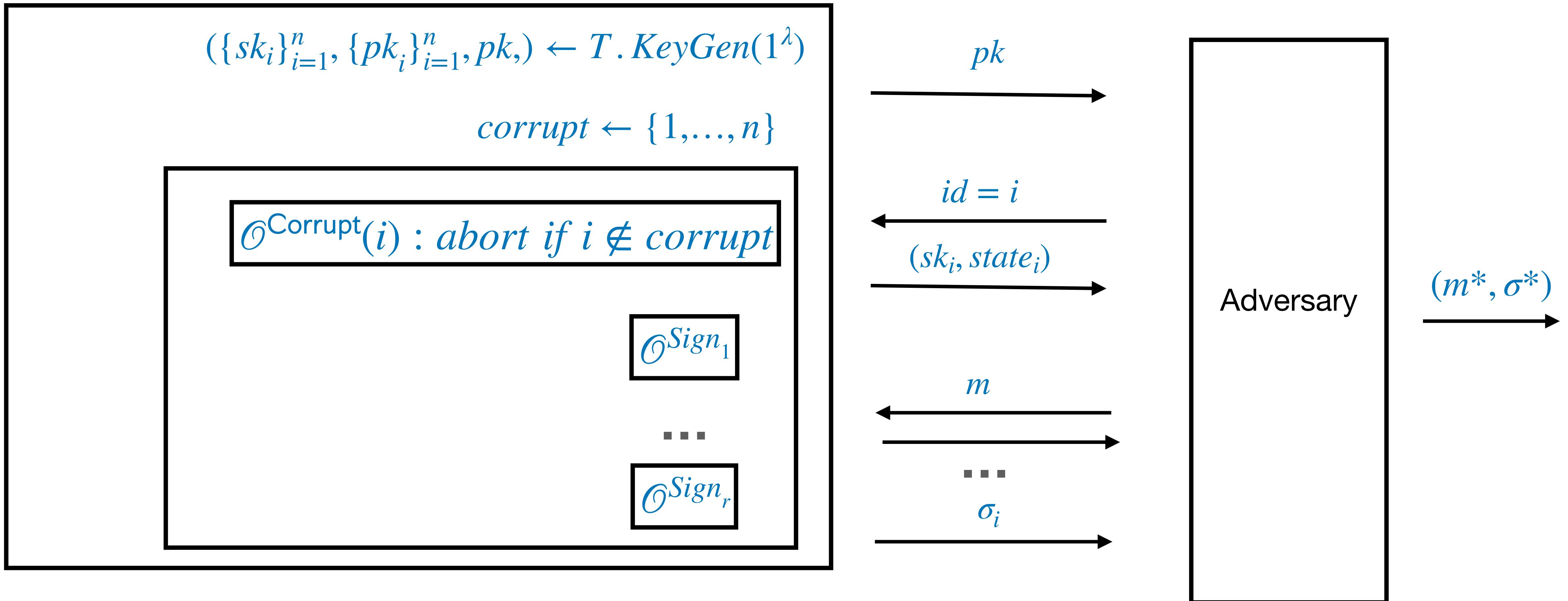


Adaptive Unforgeability

An adaptive adversary can choose on the fly which parties to corrupt. Receives not only the secret key but also the private state of the corrupted party.

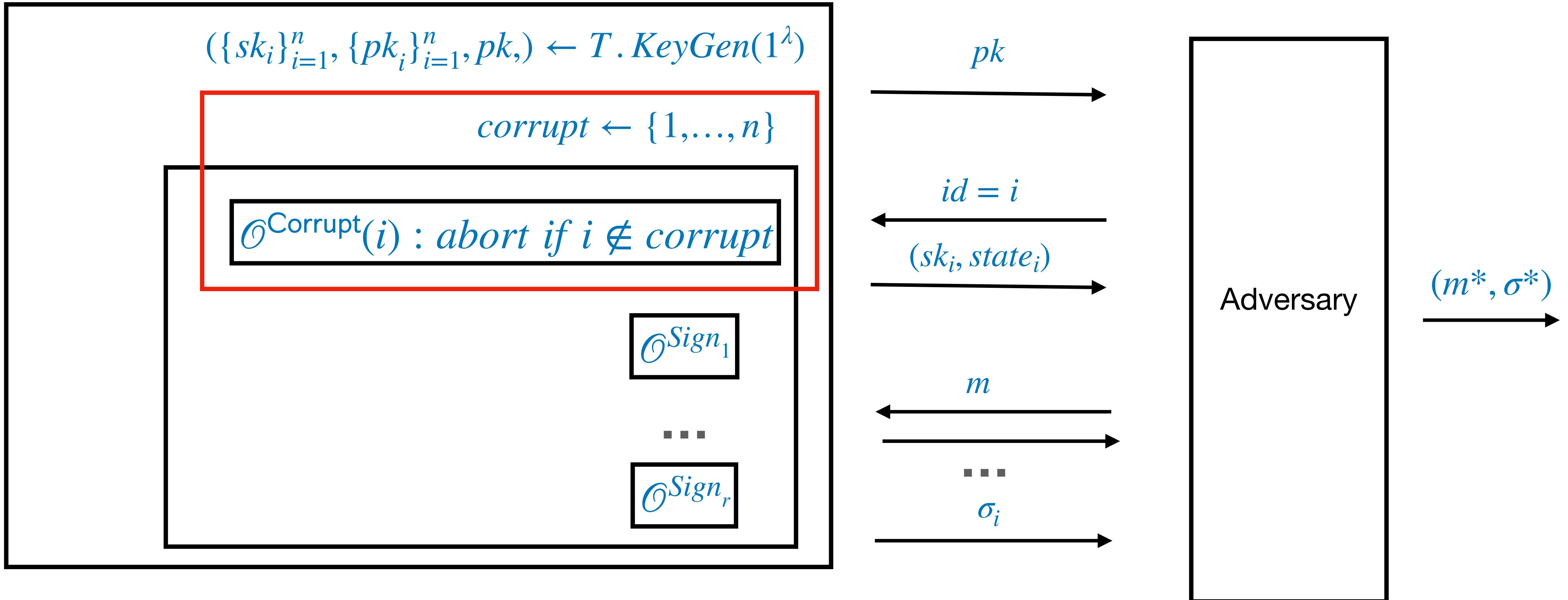


Trivial Adaptive Unforgeability from Guessing



- Tightness loss of $\binom{n}{t}$ in number of parties and corruption threshold.

Trivial Adaptive Unforgeability from Guessing



- Tightness loss of $\binom{n}{t}$ in number of parties and corruption threshold.

Is (Non-Trivial) Adaptive Security Meaningful in Practice?



Is (Non-Trivial) Adaptive Security Meaningful in Practice?

- (Opinion) - Maybe- we need more theoretical understanding.



Is (Non-Trivial) Adaptive Security Meaningful in Practice?

- (Opinion) - Maybe- we need more theoretical understanding.
- Not yet at the point to choose adaptive security over implementation considerations (performance, simplicity).



Is (Non-Trivial) Adaptive Security Meaningful in Practice?

- (Opinion) - Maybe- we need more theoretical understanding.
- Not yet at the point to choose adaptive security over implementation considerations (performance, simplicity).
- More cryptanalysis is needed [CS25, CKKTZ25].



Research Question: what security assumptions are required to prove (non-trivial) adaptive security for threshold signatures?

Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



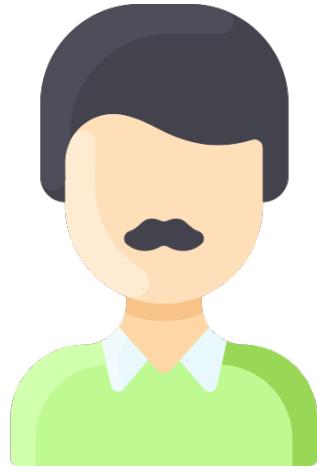
Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1

$$r_1, s_1 \leftarrow \mathbb{Z}_q$$



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

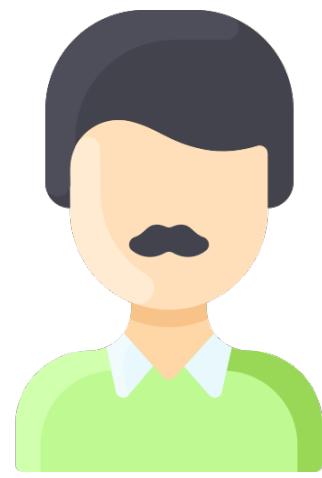


Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$(R_1, S_1)$$



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

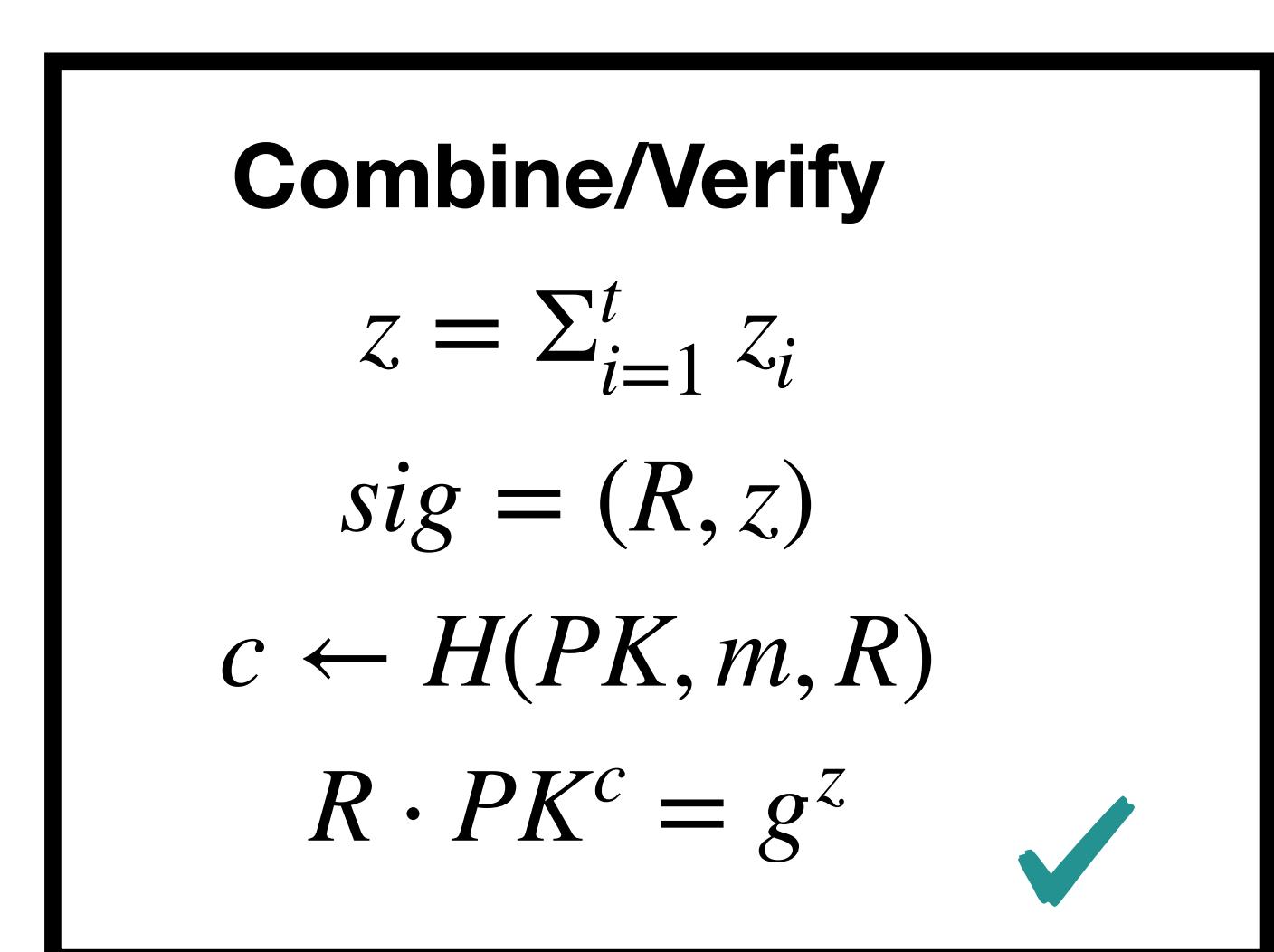
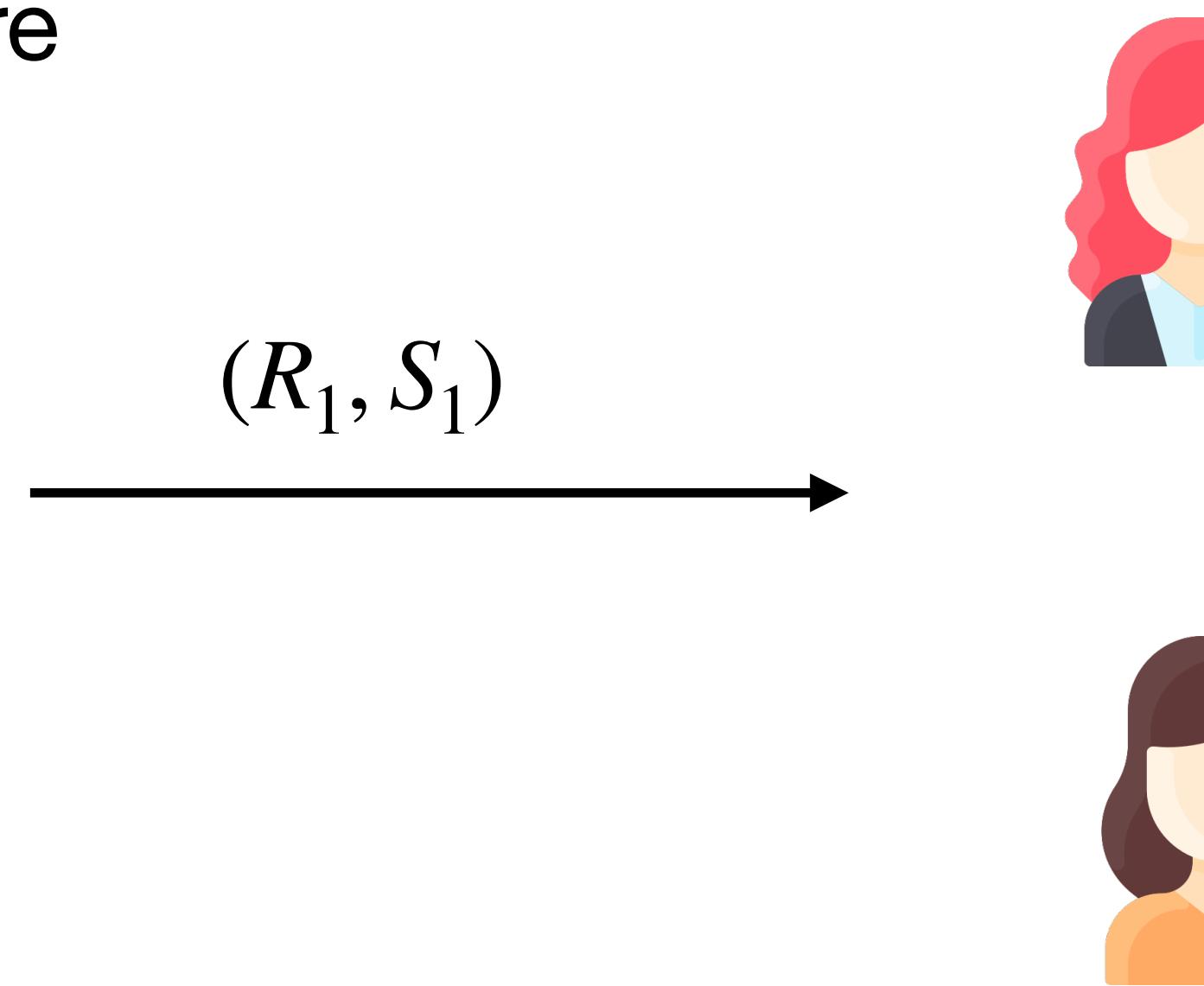
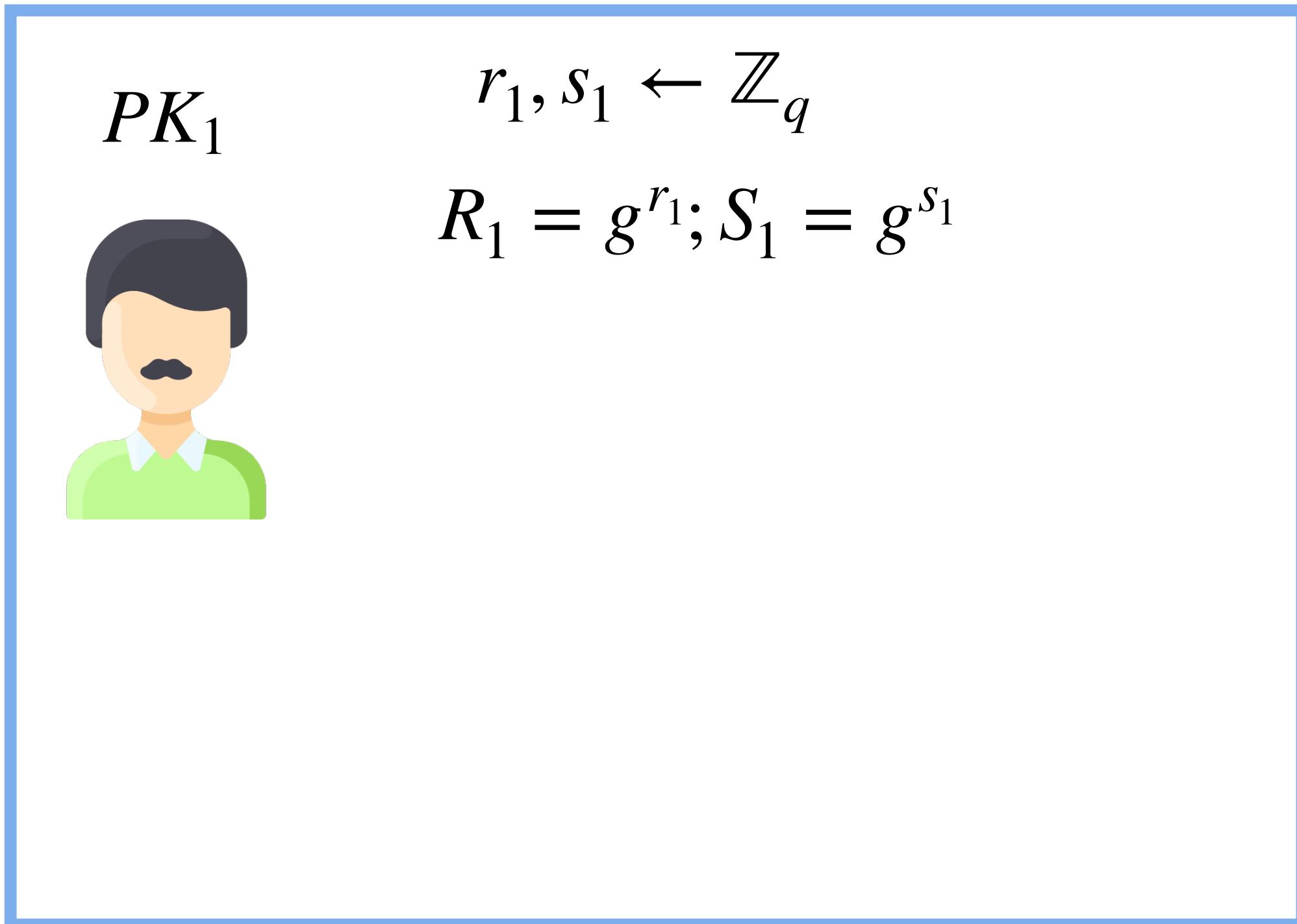
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature



Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$(R_1, S_1)$$

$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

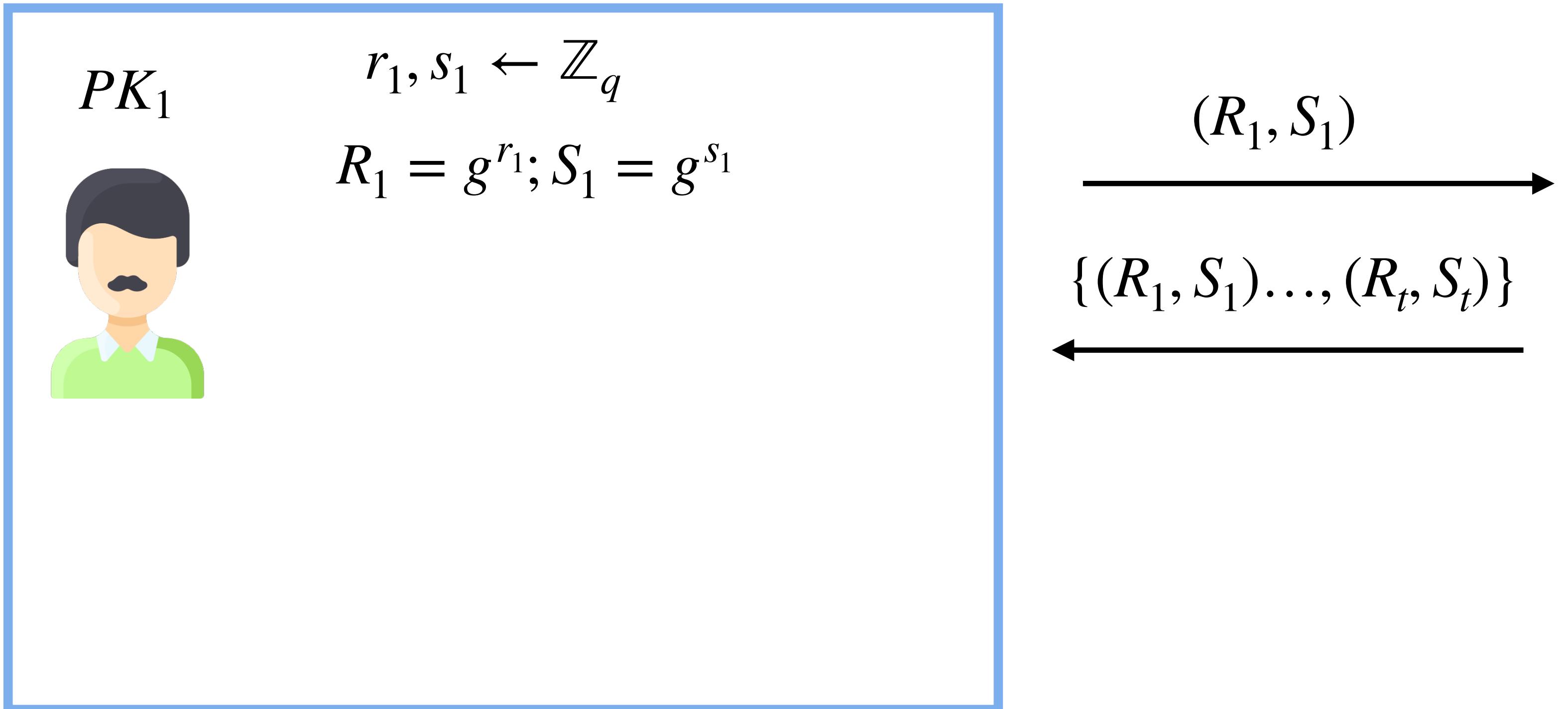
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature



Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^t)$$

$$(R_1, S_1)$$

$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$

PK_2



⋮

PK_t



Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

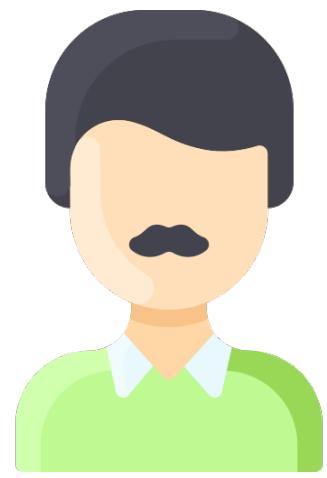


Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^t)$$

$$R = \prod_{i=1}^t R_i S_i^a$$

$$(R_1, S_1)$$

$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

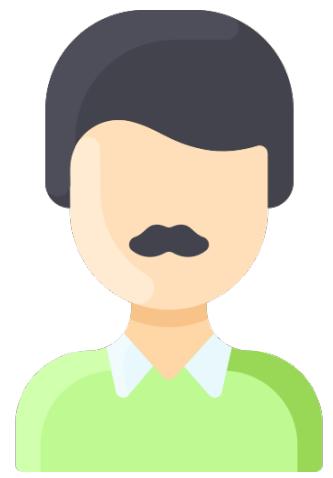


Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^t)$$

$$R = \prod_{i=1}^t R_i S_i^a$$

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

$$(R_1, S_1)$$

$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$

PK_2



⋮



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

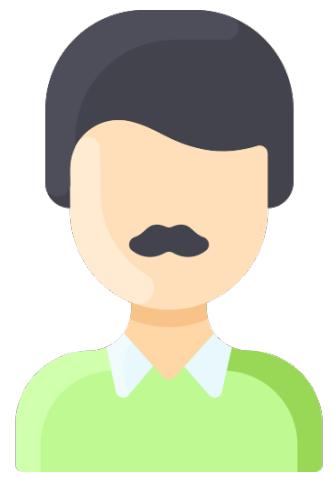


Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^t)$$

$$R = \prod_{i=1}^t R_i S_i^a$$

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

$$z_1 = r_1 + s_1 a + c s k_1 \lambda_1$$

$$(R_1, S_1)$$

$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

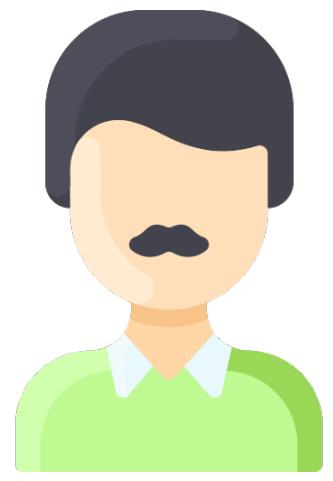


Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^t)$$

$$R = \prod_{i=1}^t R_i S_i^a$$

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

$$z_1 = r_1 + s_1 a + c s k_1 \lambda_1$$

$$(R_1, S_1)$$

$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$

PK_2



⋮



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

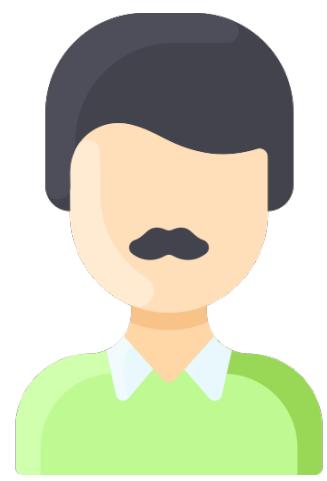


Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^t)$$

$$R = \prod_{i=1}^t R_i S_i^a$$

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

$$z_1 = r_1 + s_1 a + c s k_1 \lambda_1$$

$$(R_1, S_1) \rightarrow$$

$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$

$$\downarrow$$

$$(z_1, \dots, z_t)$$



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

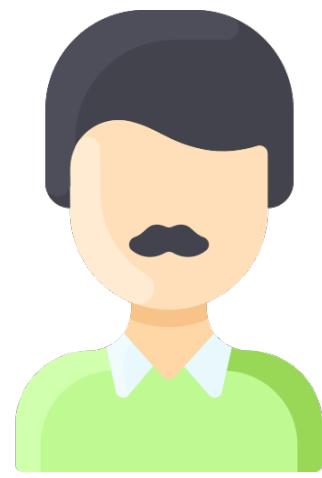


Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^t)$$

$$R = \prod_{i=1}^t R_i S_i^a$$

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

$$z_1 = r_1 + s_1 a + c s k_1 \lambda_1$$

$$(R_1, S_1)$$

$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$



PK_2



PK_t



$$(z_1, \dots, z_t)$$

Reminder: $sk = \sum_{i=1}^t \lambda_i sk_i$

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

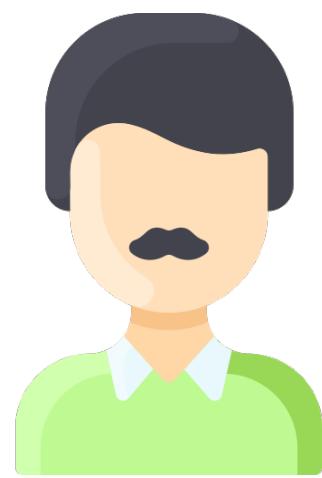


Case Study: FROST

[KG20, BCKM TZ22]

Two-Round threshold Schnorr signature

PK_1



$$r_1, s_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}; S_1 = g^{s_1}$$

$$a = H(PK, m, \{(i, R_i, S_i)\}_{i=1}^t)$$

$$R = \prod_{i=1}^t R_i S_i^a$$

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

$$z_1 = r_1 + s_1 a + c s k_1 \lambda_1$$

Desirable: Message-independent first round, simplicity, backwards-compatibility

$$(R_1, S_1) \rightarrow$$

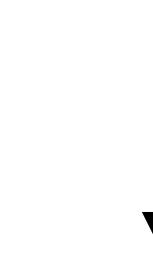
$$\{(R_1, S_1), \dots, (R_t, S_t)\}$$



PK_2



PK_t



$$(z_1, \dots, z_t)$$

Reminder: $sk = \sum_{i=1}^t \lambda_i sk_i$

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



FROST in Practice



CFRG
Internet-Draft
Intended status: Informational
Expires: 28 July 2023

D. Connolly
Zcash Foundation
C. Komlo
University of Waterloo, Zcash Foundation
I. Goldberg
University of Waterloo
C. A. Wood
Cloudflare
24 January 2023

Two-Round Threshold Schnorr Signatures with FROST
[draft-irtf-cfrg-frost-12](#)

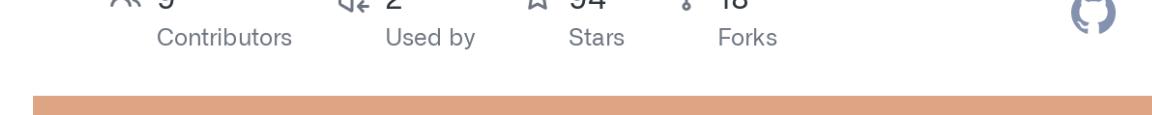


N near



serai-dex/serai

9 Contributors 2 Issues 94 Stars 18 Forks



BFRS



jesseposner/
FROST-BIP340

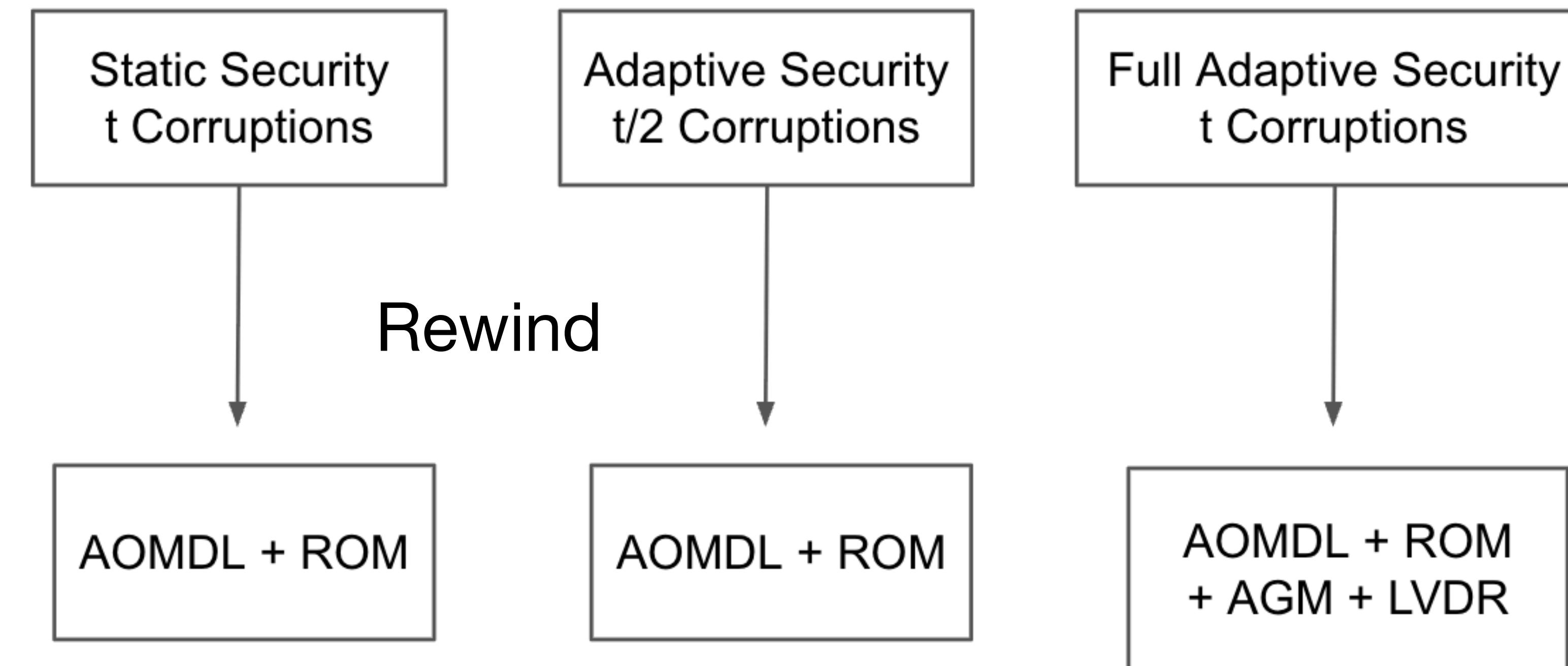


BIP340 compatible implementation of
Flexible Round-Optimized Schnorr
Threshold Signatures (FROST). This work
is made possible with the support of
Brink.
2 Contributors 2 Issues 20 Stars 3 Forks



Adaptive Security of FROST

[CKKTZ25]

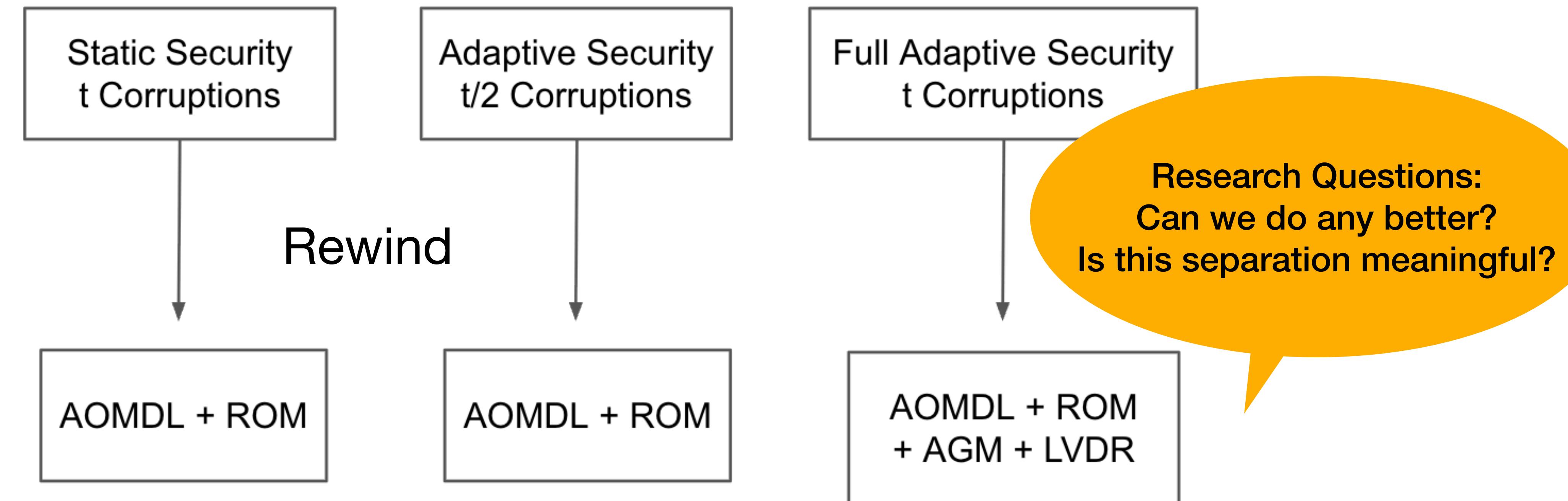


(Threshold = $t + 1$)

AOMDL = Algebraic One-More Discrete Logarithm
AGM = Algebraic Group Model
LDVR: Low-Dimensional Vector Representation Assumption

Adaptive Security of FROST

[CKKTZ25]



(Threshold = $t + 1$)

AOMDL = Algebraic One-More Discrete Logarithm
AGM = Algebraic Group Model
LDVR: Low-Dimensional Vector Representation Assumption

Adaptively Secure Schemes (Past ~Year Only)

Fully Adaptive Schnorr Threshold Signatures

Elizabeth Crites¹, Chelsea Komlo², and Mary Maller³

¹ Web3 Foundation

² University of Waterloo & Zcash Foundation

³ Ethereum Foundation & PQShield, UK

elizabeth@web3.foundation, ckomlo@uwaterloo.ca, mary.maller@ethereum.org

Adaptively Secure 5 Round Threshold Signatures from MLWE/MSIS and DL with Rewinding

Shuichi Katsumata^{1,2}, Michael Reichle³, Kaoru Takemure^{*1,2}

¹PQShield

{shuichi.katsumata, kaoru.takemure}@pqshield.com

²AIST

³ETH Zürich

michael.reichle@inf.ethz.ch

Twinkle: Threshold Signatures from DDH with Full Adaptive Security

Rena Bacho^{1,3}, Julian Loss¹, Stefano Tessaro²

Benedikt Wagner^{1,3}, Chenzhi Zhu²

February 26, 2024

Adaptively Secure Three-Round Threshold Schnorr Signatures from DDH

Rena Bacho^{1,2}, Sourav Das³, Julian Loss^{1,2}, and Ling Ren³

¹ CISPA Helmholtz Center for Information Security

² Saarland University, Saarbrücken, Germany

³ University of Illinois at Urbana-Champaign

{renas.bacho,loss}@cispa.de, {souravd2,renling}@illinois.edu

HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures

Rena Bacho^{1,3}, Julian Loss¹, Gilad Stern²

Benedikt Wagner^{* 4}

October 5, 2024

Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security

Rena Bacho^{1,2}, Sourav Das³, Julian Loss^{1,2}, and Ling Ren³

¹ CISPA Helmholtz Center for Information Security

² Saarland University, Saarbrücken, Germany

³ University of Illinois at Urbana-Champaign

{renas.bacho,loss}@cispa.de, {souravd2,renling}@illinois.edu

Adaptively Secure Schemes (Past ~Year Only)

Fully Adaptive Schnorr Threshold Signatures

Elizabeth Crites¹, Chelsea Komlo², and Mary Maller³

¹ Web3 Foundation

² University of Waterloo & Zcash Foundation

³ Ethereum Foundation & PQShield, UK

elizabeth@web3.foundation, ckomlo@uwaterloo.ca, mary.maller@ethereum.org

Twinkle: Threshold Signatures from DDH with Full Adaptive Security

Rena Bacho^{1,3}, Julian Loss¹, Stefano Tessaro²

Benedikt Wagner^{1,3}, Chenzhi Zhu²

February 26, 2024

Adaptively Secure Three-Round Threshold Schnorr Signatures from DDH

Rena Bacho^{1,2}, Sourav Das³, Julian Loss^{1,2}, and Ling Ren³

¹ CISPA Helmholtz Center for Information Security

² Saarland University, Saarbrücken, Germany

³ University of Illinois at Urbana-Champaign

{renas.bacho,loss}@cispa.de, {souravd2,renling}@illinois.edu

HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures

Rena Bacho^{1,3}, Julian Loss¹, Gilad Stern²

Benedikt Wagner^{* 4}

October 5, 2024

Glacius: Threshold Schnorr Signatures from DDH with Full Adaptive Security

Rena Bacho^{1,2}, Sourav Das³, Julian Loss^{1,2}, and Ling Ren³

¹ CISPA Helmholtz Center for Information Security

² Saarland University, Saarbrücken, Germany

³ University of Illinois at Urbana-Champaign

{renas.bacho,loss}@cispa.de, {souravd2,renling}@illinois.edu

Research Question:
How can we classify adaptively secure schemes?

Adaptive Schemes (Recent, Non-Exhaustive)

	Scheme	Static Assumptions	Adaptive Assumptions (Full)
Schnorr	Sparkle [CKM23]	DL+ROM	AOMDL+ROM, $t/2$
FROST/2/3 [KG20, CGRS23, BCMTZ22]		(A)OMDL+ROM	AOMDL + ROM + AGM + LVDR
	Twinkle [BLTWZ23]		OMCDH/DDH+ROM
	Threshold BLS (1) [B03, BL22]	GDH+ROM	OMDL+AGM+ROM
Non-Schnorr	Threshold BLS (3) [DR24]	co-CDH	Co-CDH+DDH
	Threshold ECDSA [GG18, CGGMP20]	Various	✗

Adaptive Schemes (Recent, Non-Exhaustive)

Scheme	Static Assumptions	Adaptive Assumptions (Full)
Schnorr	Sparkle [CKM23]	DL+ROM
FROST/2/3 [KG20, CGRS23, BCMTZ22]	(A)OMDL+ROM	AOMDL+ROM, $t/2$
	Twinkle [BLTWZ23]	AOMDL + ROM + AGM + LVDR
	Threshold BLS (1) [B03, BL22]	OMCDH/DDH+ROM
Non-Schnorr	Threshold BLS (3) [DR24]	OMDL+AGM+ROM
	Threshold ECDSA [GG18, CGGMP20]	Co-CDH+DDH



Research Question:
What assumptions are inherent,
and why?

Our Results

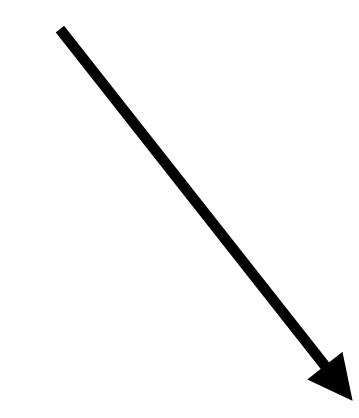
Definition

Key-Unique Threshold Signatures

Impossibility Result #1



**Impossible under Non-Interactive,
Computational Assumptions**



**Key-Unique
Schnorr Threshold Signatures**

Impossibility Result #2



**Impossible with rewinding
under Interactive (One-More) DL**

Reviewer #2 (Curse of Expertise)

“I do not think the results are surprising”



Reviewer #2 (Curse of Expertise)



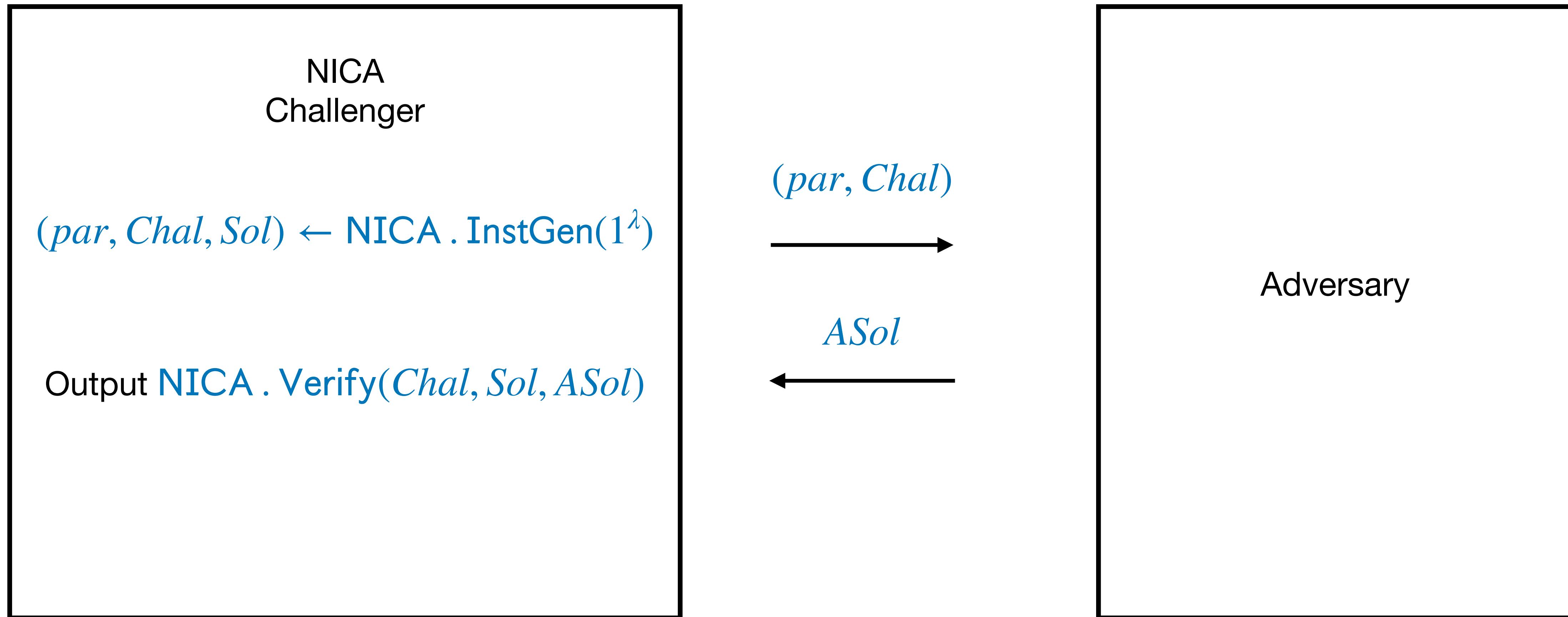
“I do not think the results are surprising”



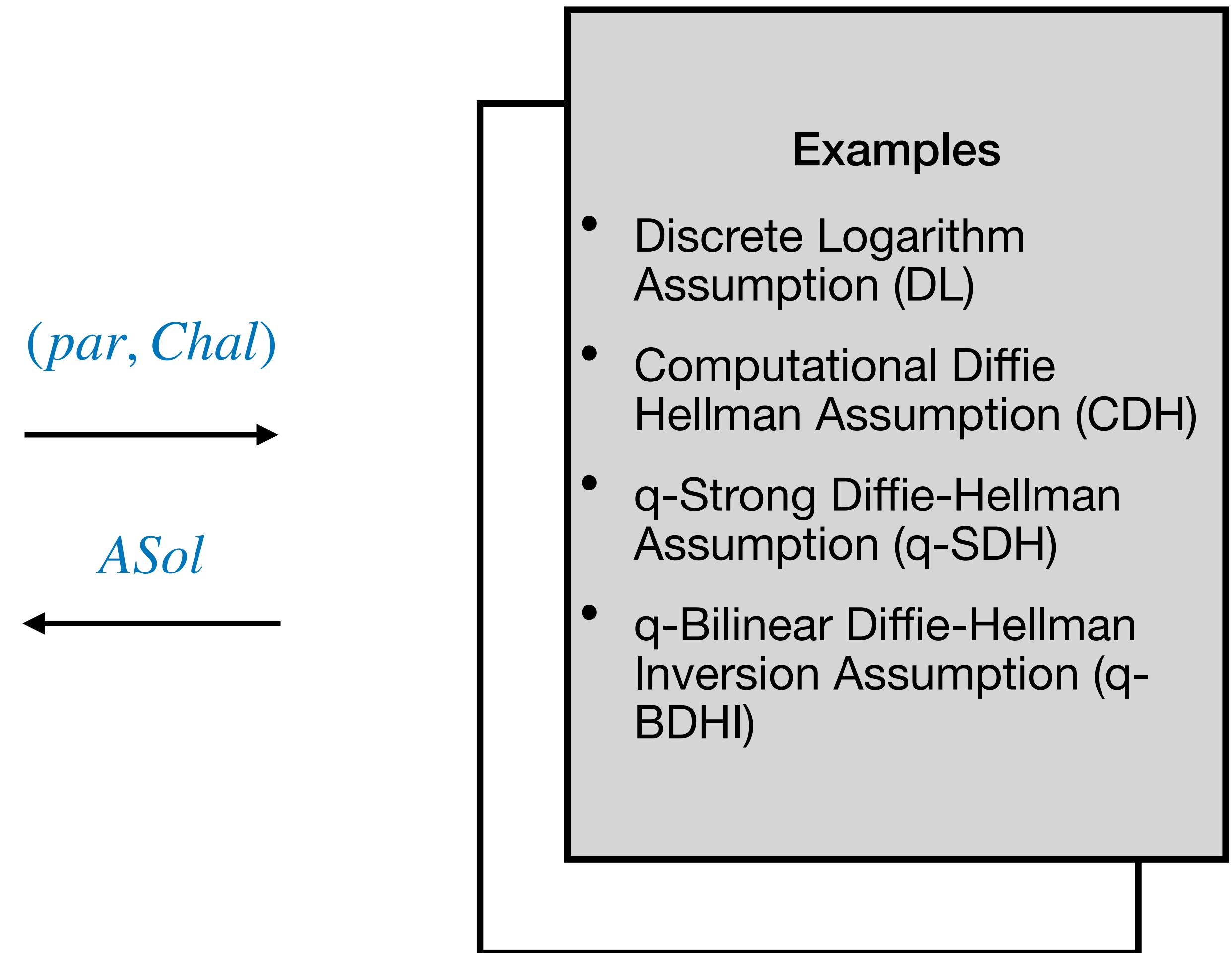
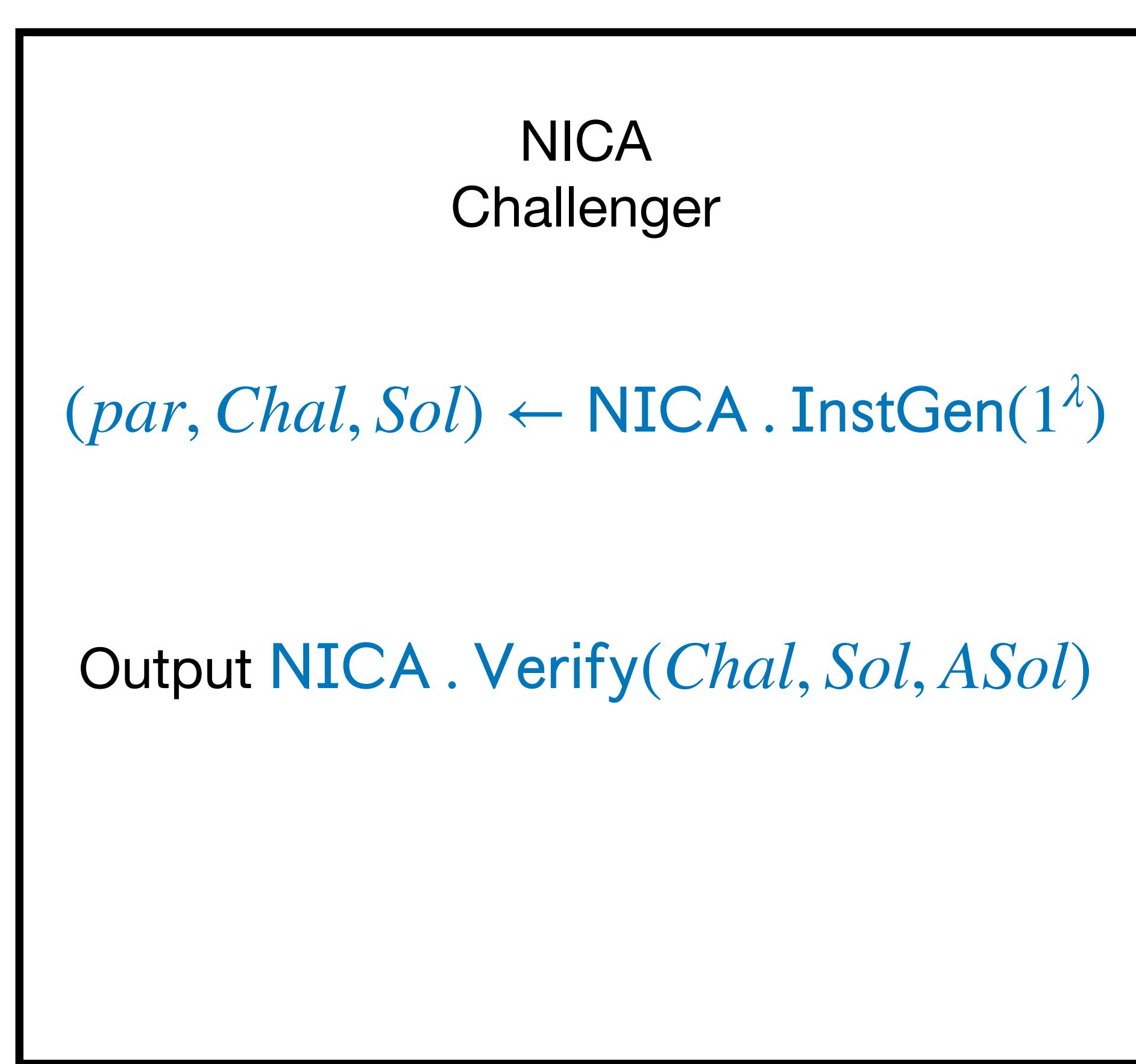
We should have better tools to understand the research landscape, and implications in practice.

Background

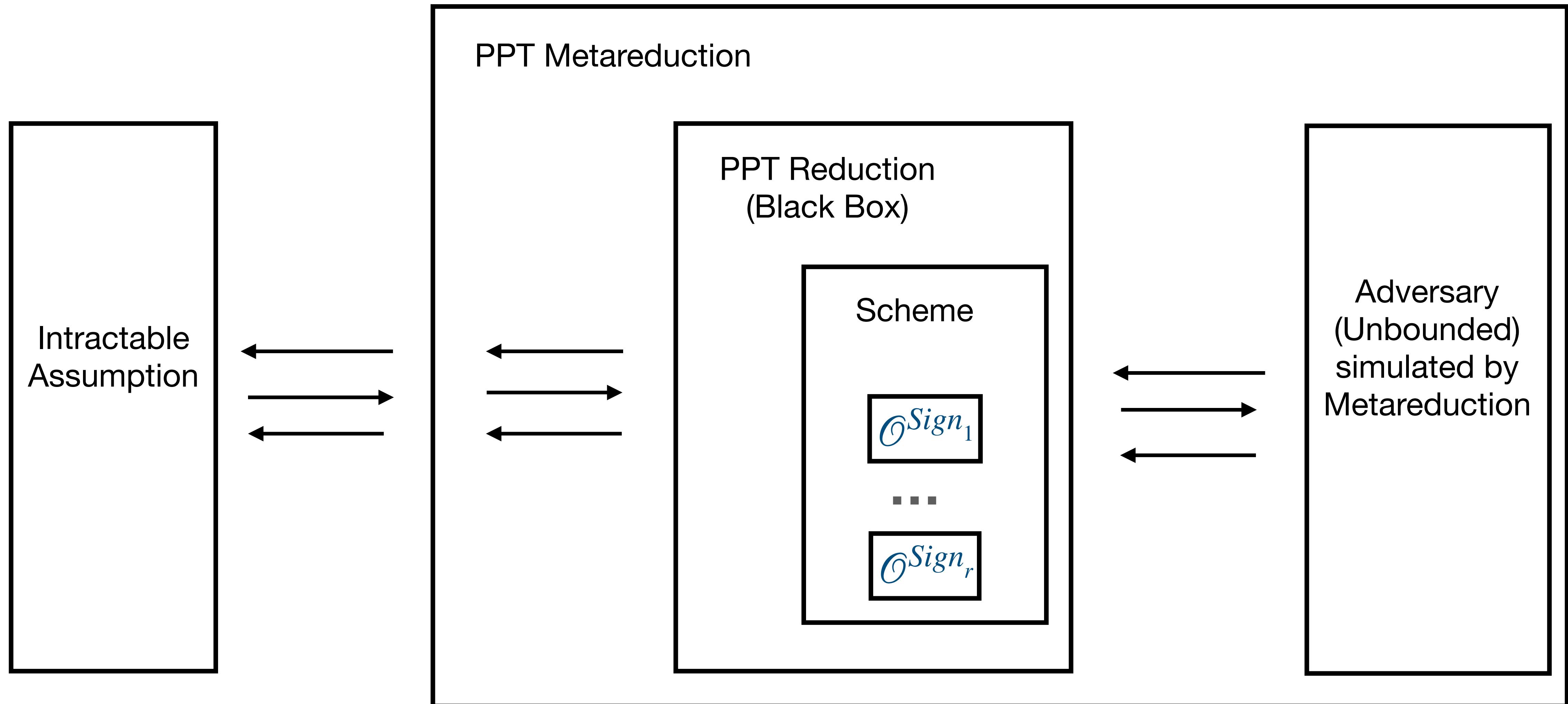
Non-Interactive Computational Assumption (NICA)



Non-Interactive Computational Assumption (NICA)

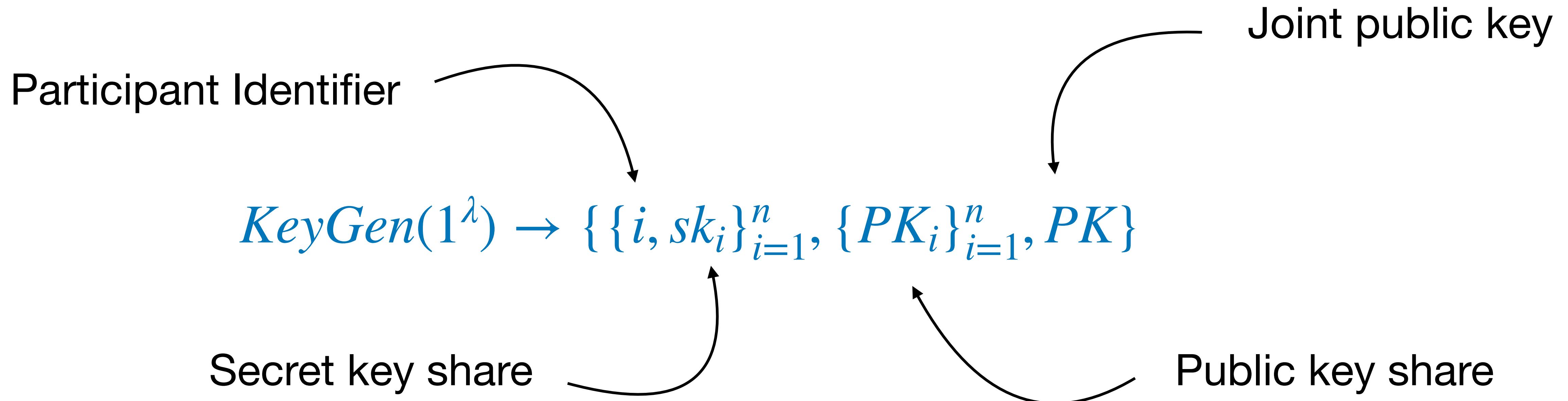


Metareduction: Prove impossibility of a reduction



Key-Uniqueness

Key Generation for Threshold Signatures



Key Uniqueness

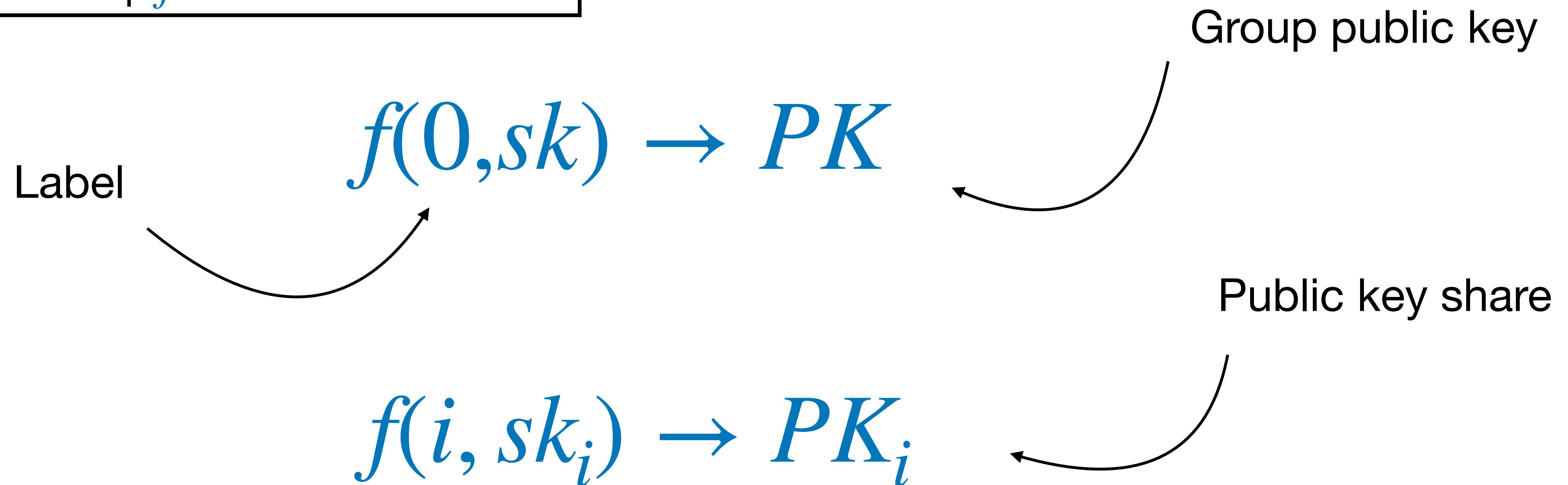
Property 1: Unique correspondence

Property 2: Efficient verification

Key Uniqueness

Property 1: Unique correspondence

Public key and shares have a unique correspondence via an (efficient) injective map f .



Key Uniqueness

If

VerifyPKs($PK, \{PK_i\}_{i=1}^n\}$) $\rightarrow 1$

Then

Recover($\{sk_j\}_{j \in C}\}) \rightarrow sk$

Property 2: Efficient verification

An efficient Verify algorithm exists to check that public keys are well-formed.

Key Uniqueness

If

$\text{VerifyPKs}(\text{PK}, \{\text{PK}_i\}_{i=1}^n) \rightarrow 1$

Then

Property 2: Efficient verification

An efficient Verify algorithm exists to check that public keys are well-formed.

Our simulation fails against a reduction that outputs invalid keys

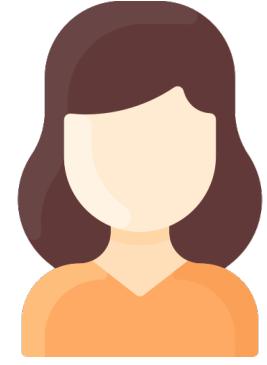
$\text{Recover}(\{sk_j\}_{j \in C}) \rightarrow sk$

Example #1: Perfectly Binding Commitment



Key unique. Employed by: FROST/2/3 [KG20, BCKM TZ22CKM21, CGRS23], Sparkle [CKM23], Lindell22, ZeroS [M23]

Example #1: Perfectly Binding Commitment



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Key unique. Employed by: FROST/2/3 [KG20, BCKMTZ22CKM21, CGRS23], Sparkle [CKM23], Lindell22, ZeroS [M23]

Example #1: Perfectly Binding Commitment



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Key unique. Employed by: FROST/2/3 [KG20, BCKMTZ22CKM21, CGRS23], Sparkle [CKM23], Lindell22, ZeroS [M23]

Example #1: Perfectly Binding Commitment



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Key unique. Employed by: FROST/2/3 [KG20, BCKMTZ22CKM21, CGRS23], Sparkle [CKM23], Lindell22, ZeroS [M23]

Example #1: Perfectly Binding Commitment



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Step 4: Define $PK_i = g^{E(i)} = g^{sk_i}$ and $PK = g^{E(0)} = g^{sk}$

Key unique. Employed by: FROST/2/3 [KG20, BCKMTZ22CKM21, CGRS23], Sparkle [CKM23], Lindell22, ZeroS [M23]

Example #1: Perfectly Binding Commitment



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Shamir secret sharing

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Step 4: Define $PK_i = g^{E(i)} = g^{sk_i}$ and $PK = g^{E(0)} = g^{sk}$

Key unique. Employed by: FROST/2/3 [KG20, BCKMTZ22CKM21, CGRS23], Sparkle [CKM23], Lindell22, ZeroS [M23]

Example #1: Perfectly Binding Commitment



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Shamir secret sharing

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Step 4: Define $PK_i = g^{E(i)} = g^{sk_i}$ and $PK = g^{E(0)} = g^{sk}$

Key unique. Employed by: FROST/2/3 [KG20, BCKMTZ22CKM21, CGRS23], Sparkle [CKM23], Lindell22, ZeroS [M23]

$((i, sk_i), \{PK_i\}_{i=1}^n, PK)$

Example #1: Perfectly Binding Commitment



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Shamir secret sharing

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Step 4: Define $PK_i = g^{E(i)} = g^{sk_i}$ and $PK = g^{E(0)} = g^{sk}$

Key-uniqueness allows for
identifiable abort “for free.”

‘2/3 [KG20, BCKMTZ22CKM21,
22, ZeroS [M23]

$((i, sk_i), \{PK_i\}_{i=1}^n, PK)$

Example #2: No Public Key Shares



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Example #2: No Public Key Shares



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Step 4: Define $PK = g^{E(0)} = g^{sk}$

Example #2: No Public Key Shares



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Step 4: Define $PK = g^{E(0)} = g^{sk}$

$((i, sk_i), PK)$

Example #2: No Public Key Shares



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Step 4: Define $PK = g^{E(0)} = g^{sk}$

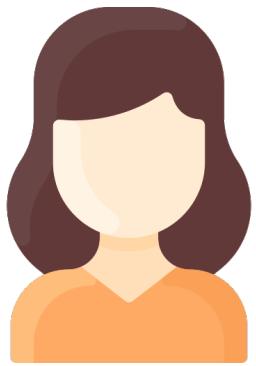
Not key unique. Employed by: KRT24

$$f(0, sk) \rightarrow g^{sk} = PK$$

$$f(i, sk_i) \rightarrow \perp$$

$$\xrightarrow{\hspace{1cm}} ((i, sk_i), PK)$$

Example #2: No Public Key Shares



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$

Step 4: Define $PK = g^{E(0)} = g^{sk}$

Tradeoff: No identifiable abort

Not key unique. Employed by: KRT24

$$f(0, sk) \rightarrow g^{sk} = PK$$

$$f(i, sk_i) \rightarrow \perp$$

$$\xrightarrow{\hspace{1cm}} ((i, sk_i), PK)$$

Example #3: Pedersen Commitments



Example #3: Pedersen Commitments



Step 1: Sample $sk \leftarrow \mathbb{Z}_q^n$, $\{a_{i1}, \dots, a_{i,t-1}\}_{i=1}^2 \leftarrow \mathbb{Z}_q^t$

Example #3: Pedersen Commitments



Step 1: Sample $sk \leftarrow \mathbb{Z}_q^n$, $\{a_{i1}, \dots, a_{i,t-1}\}_{i=1}^2 \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E_i(x) = sk + \sum_{j=1}^{t-1} a_{1j}x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_{2j}x^j$

Example #3: Pedersen Commitments

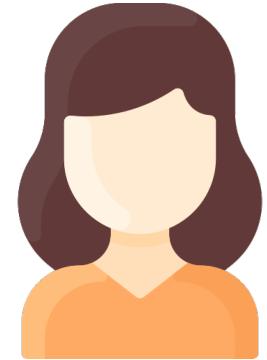


Step 1: Sample $sk \leftarrow \mathbb{Z}_q^n$, $\{a_{i1}, \dots, a_{i,t-1}\}_{i=1}^2 \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E_i(x) = sk + \sum_{j=1}^{t-1} a_{1j}x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_{2j}x^j$

Step 3: Define $sk'_i = (E(i), F(i))$

Example #3: Pedersen Commitments



Step 1: Sample $sk \leftarrow \mathbb{Z}_q^n$, $\{a_{i1}, \dots, a_{i,t-1}\}_{i=1}^2 \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E_i(x) = sk + \sum_{j=1}^{t-1} a_{1j}x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_{2j}x^j$

Step 3: Define $sk'_i = (E(i), F(i))$

Step 4: Define $PK_i = g^{E(i)}h^{F(i)}$ and $PK = g^{E(0)} = g^{sk}$

Example #3: Pedersen Commitments



Step 1: Sample $sk \leftarrow \mathbb{Z}_q^n$, $\{a_{i1}, \dots, a_{i,t-1}\}_{i=1}^2 \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E_i(x) = sk + \sum_{j=1}^{t-1} a_{1j}x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_{2j}x^j$

Step 3: Define $sk'_i = (E(i), F(i))$

Step 4: Define $PK_i = g^{E(i)}h^{F(i)}$ and $PK = g^{E(0)} = g^{sk}$

$((i, sk'_i), \{PK_i\}_{i=1}^n, PK)$

Example #3: Pedersen Commitments



Step 1: Sample $sk \leftarrow \mathbb{Z}_q^n$, $\{a_{i1}, \dots, a_{i,t-1}\}_{i=1}^2 \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E_i(x) = sk + \sum_{j=1}^{t-1} a_{1j}x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_{2j}x^j$

Step 3: Define $sk'_i = (E(i), F(i))$

Step 4: Define $PK_i = g^{E(i)}h^{F(i)}$ and $PK = g^{E(0)} = g^{sk}$

Not key unique. Employed by: DR24

$((i, sk'_i)', \{PK_i\}_{i=1}^n, PK)$

Example #3: Pedersen Commitments



Step 1: Sample $sk \leftarrow \mathbb{Z}_q^n$, $\{a_{i,1}, \dots, a_{i,t-1}\}_{i=1}^2 \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E_i(x) = sk + \sum_{j=1}^{t-1} a_{1j}x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_{2j}x^j$

Step 3: Define $sk'_i = (E(i), F(i))$

Step 4: Define $PK_i = g^{E(i)}h^{F(i)}$ and $PK = g^{E(0)} = g^{sk}$

Tradeoff:
Identifiable abort via
additional NIZK.

Not key unique. Employed by: DR24

$((i, sk'_i)', \{\underline{PK_i}\}_{i=1}^n, PK)$

Impossibility Result #1

TLDR;

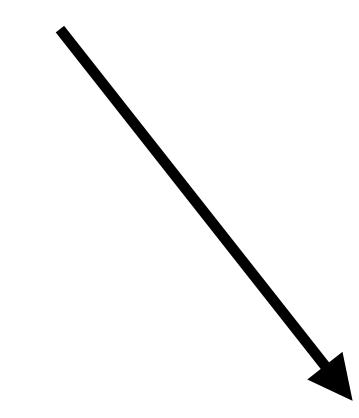
Definition

Key-Unique Threshold Signatures

Impossibility Result #1



**Impossible under Non-Interactive,
Computational Assumptions**



**Key-Unique
Schnorr Threshold Signatures**

Impossibility Result #2



**Impossible with rewinding
under Interactive (One-More) DL**

TLDR;

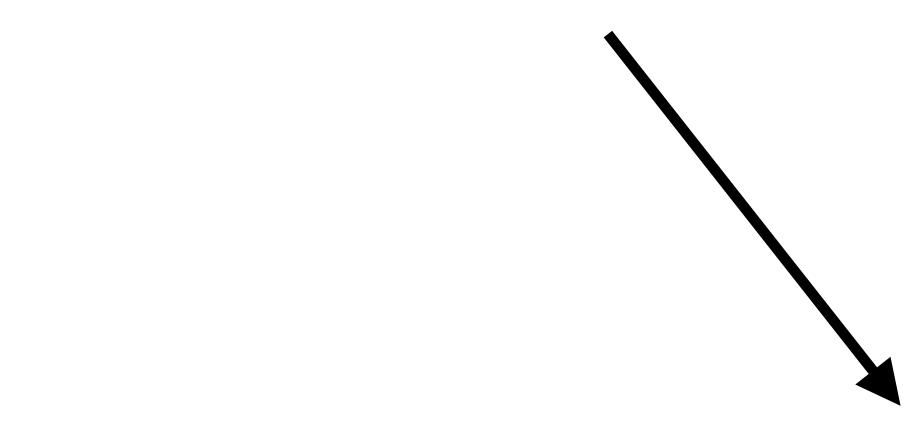
Definition

Key-Unique Threshold Signatures

Impossibility Result #1

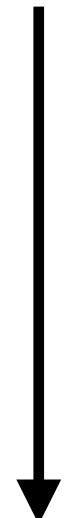


**Impossible under Non-Interactive,
Computational Assumptions**



**Key-Unique
Schnorr Threshold Signatures**

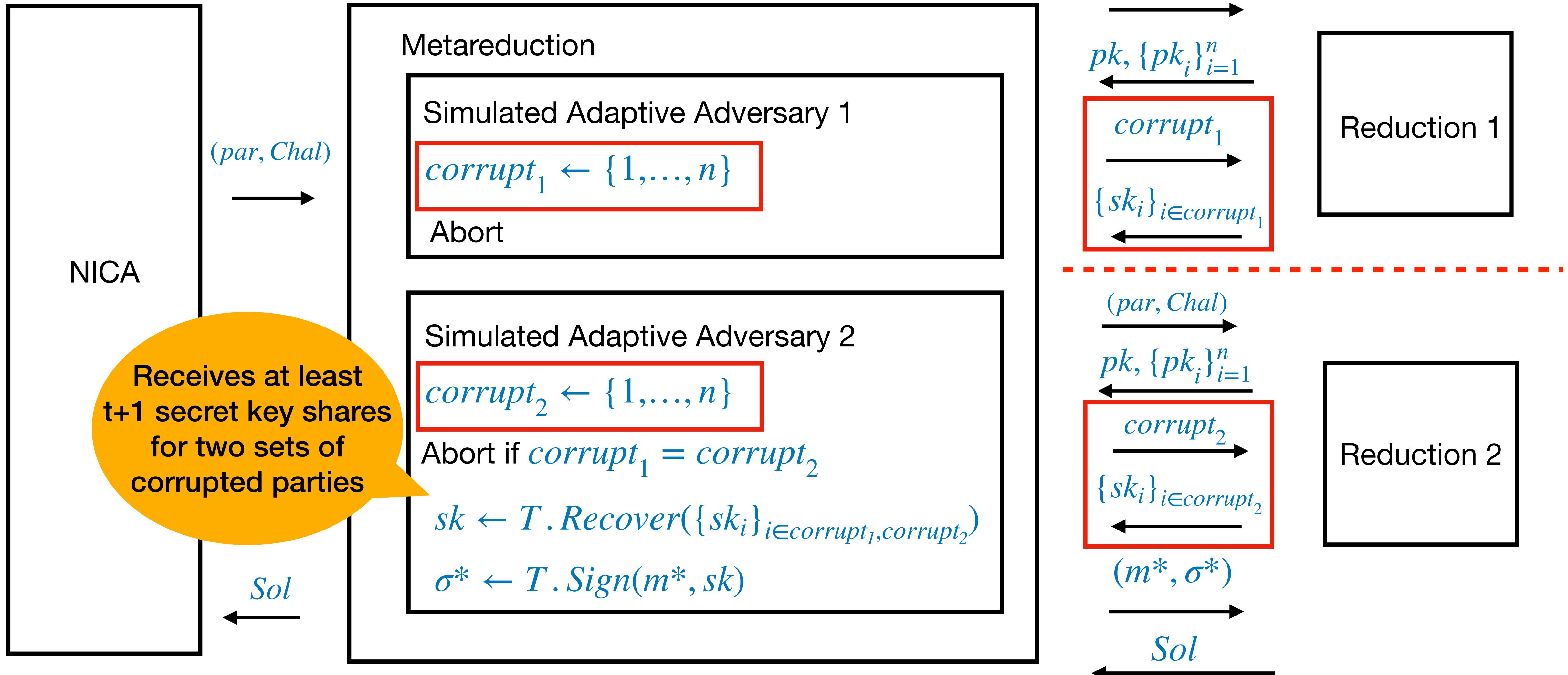
Impossibility Result #2



**Impossible with rewinding
under Interactive (One-More) DL**

- * **Applicable in:** Standard model, ROM
- * **Extensible to:** Algebraic adversaries (AGM)
- * **Assumes:** Reduction does not fork at key generation

Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



Generalization of result

(Threshold = $t + 1$)

Generalization of result

- Metareduction assumes full adaptive security, where $|corrupt| = t$

(Threshold = $t + 1$)

Generalization of result

- Metareduction assumes full adaptive security, where $|\text{corrupt}| = t$
 - . We can generalize impossibility result to any $\frac{t}{2} < |\text{corrupt}| \leq t$, with a combinatorial argument

(Threshold = $t + 1$)

Generalization of result

- Metareduction assumes full adaptive security, where $|\text{corrupt}| = t$
- We can generalize impossibility result to any $\frac{t}{2} < |\text{corrupt}| \leq t$, with a combinatorial argument
- Restricting $|\text{corrupt}| = \frac{t}{2}$ circumvents this impossibility result (FROST, [CKM23])

(Threshold = $t + 1$)

Adaptive Schemes (Recent, Non-Exhaustive)

Scheme	Static Assumptions	Adaptive Assumptions (Full)	Our Results Apply
Schnorr	Sparkle [CKM23]	DL+ROM	AOMDL+ROM, $t/2$
FROST/2/3 [KG20, CGRS23, BCMTZ22]		(A)OMDL+ROM	AOMDL+ROM +AGM+LVDR
	Twinkle [BLTWZ23]		OMCDH/DDH+ROM
	Threshold BLS (1) [B03, BL22]	GDH+ROM	OMDL+AGM+ROM
Non-Schnorr	Threshold ECDSA [GG18, CGGMP20]	Various	
	Threshold BLS (3) [DR24]	co-CDH	Co-CDH+DDH

Tradeoff:
extra NIZK

Impossibility Result #2

TLDR;

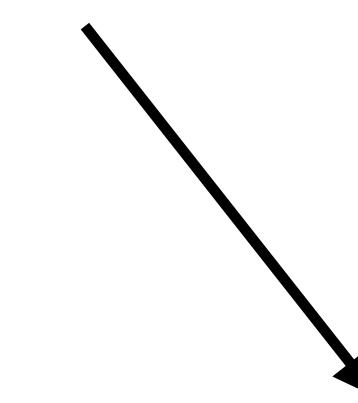
Definition

Key-Unique Threshold Signatures

Impossibility Result #1



**Impossible under Non-Interactive,
Computational Assumptions**



**Key-Unique
Schnorr Threshold Signatures**

Impossibility Result #2



**Impossible with rewinding
under Interactive (One-More DL)**

TLDR;

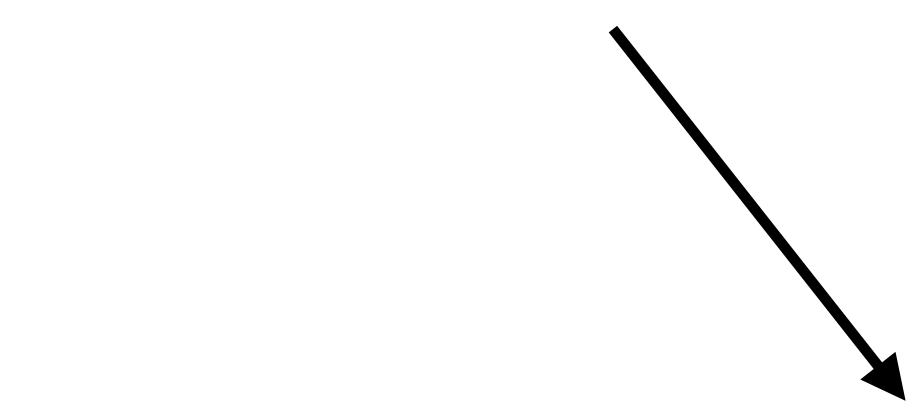
Definition

Key-Unique Threshold Signatures

Impossibility Result #1

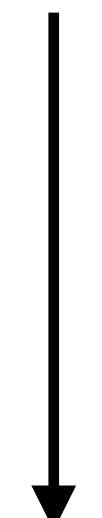


**Impossible under Non-Interactive,
Computational Assumptions**



**Key-Unique
Schnorr Threshold Signatures**

Impossibility Result #2

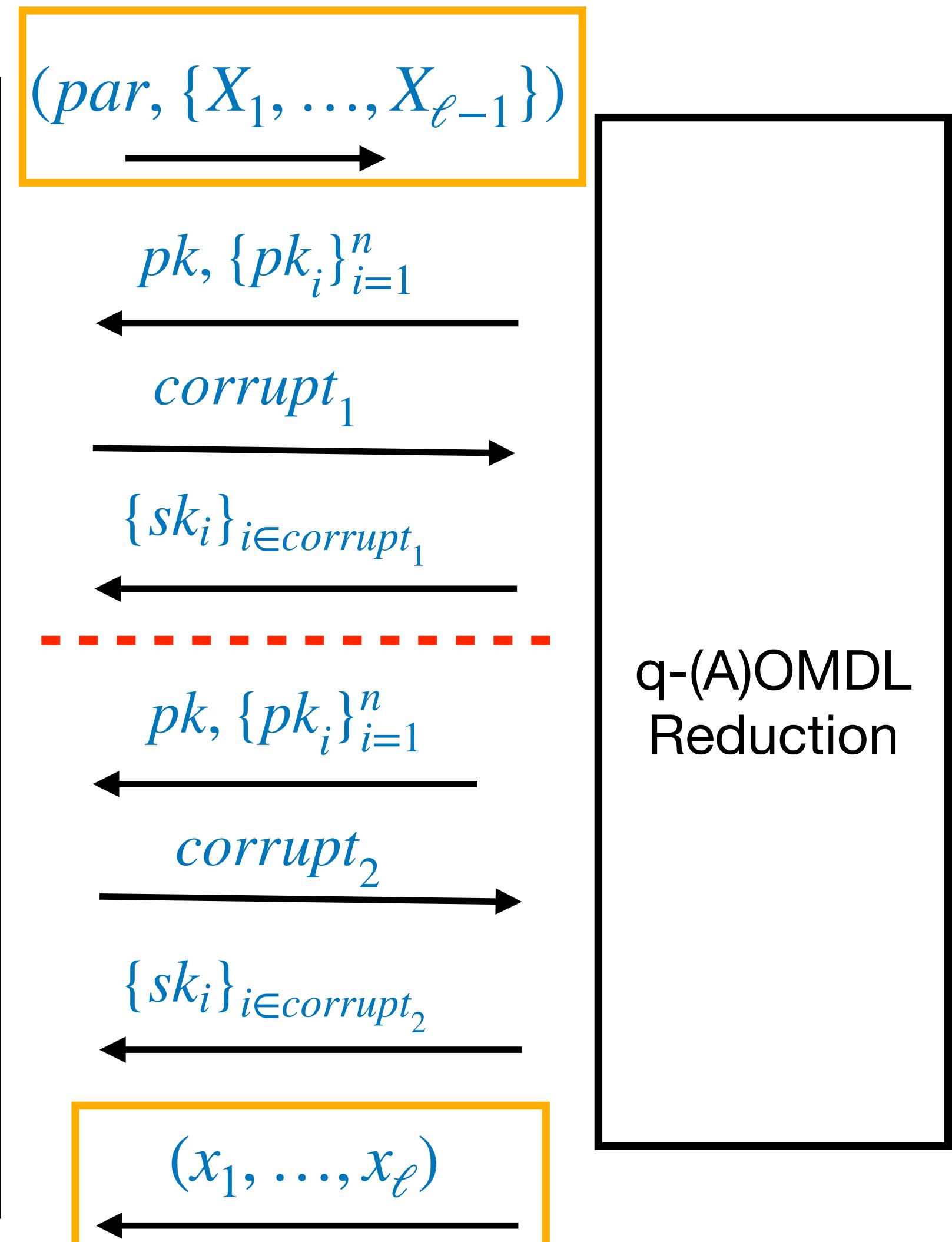
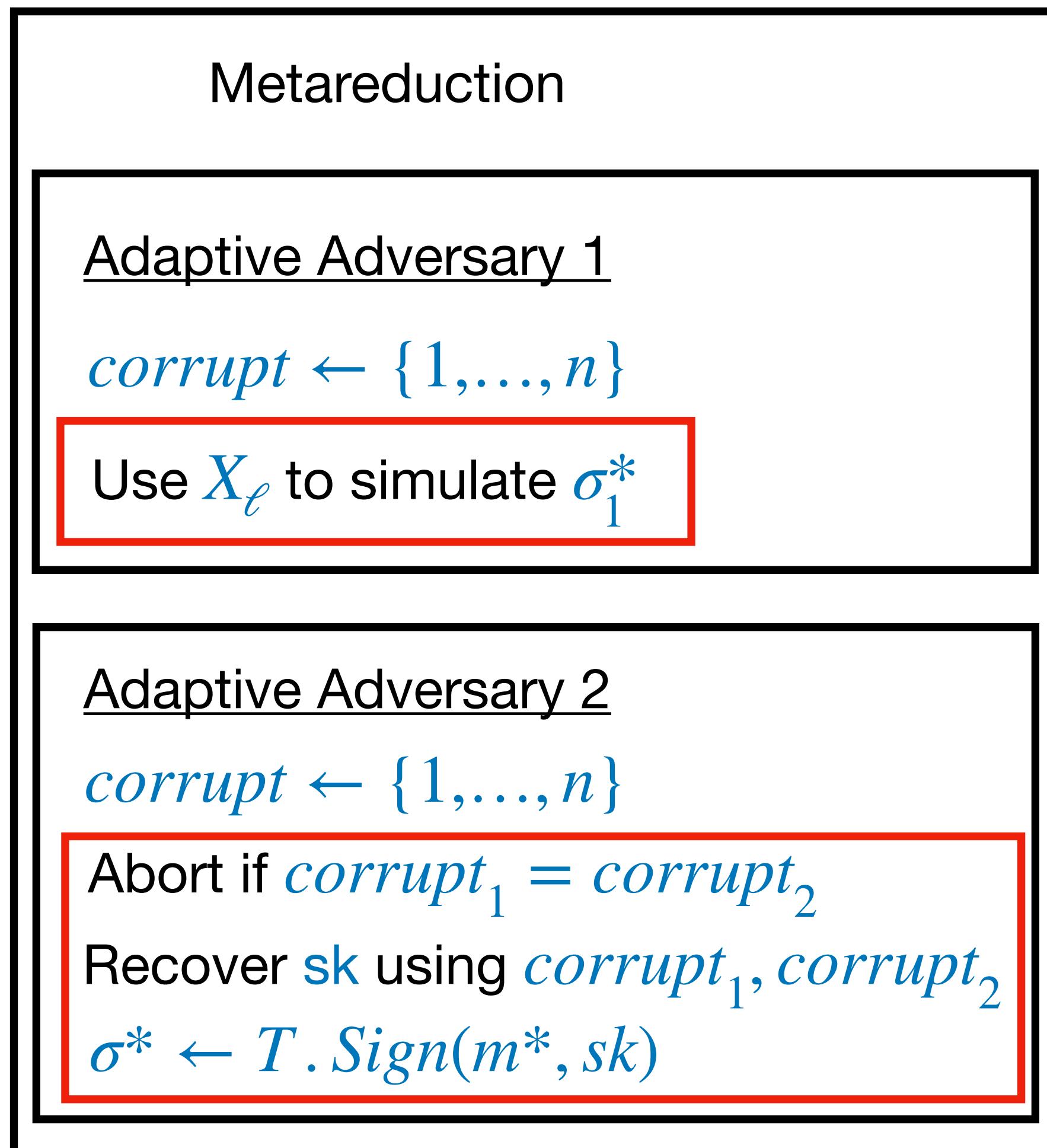
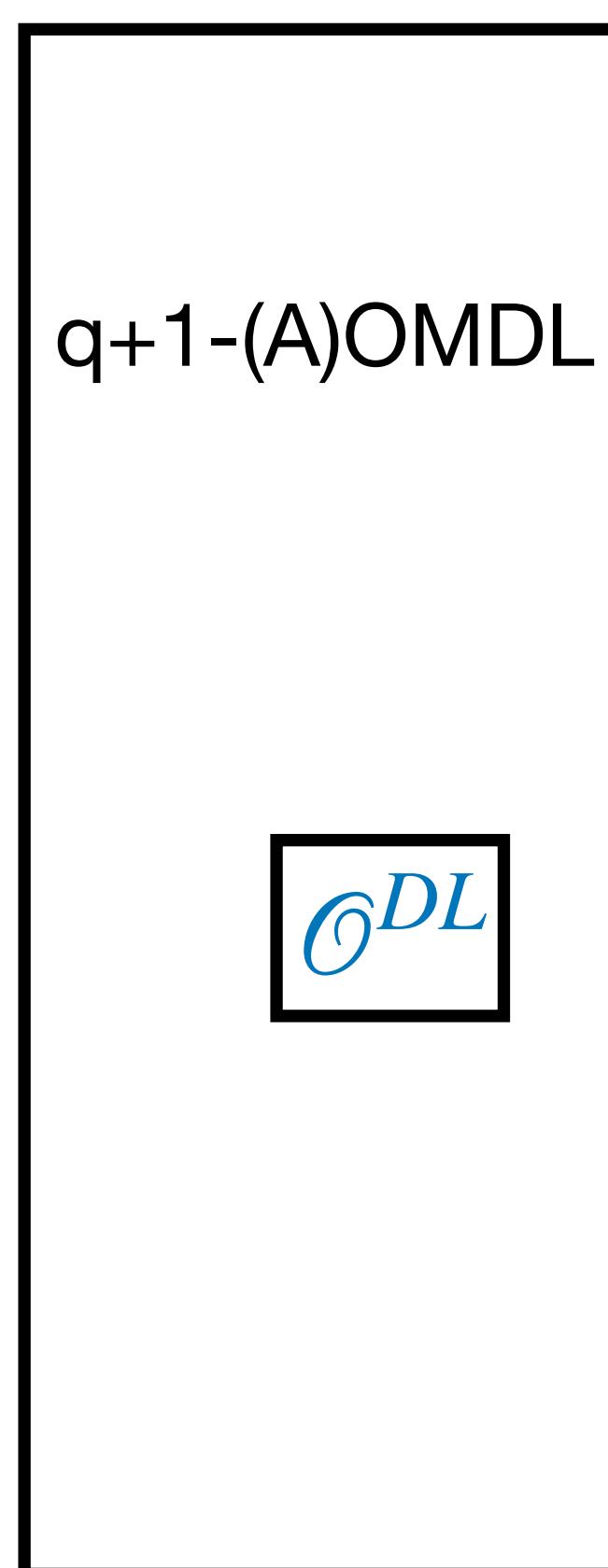


* **Applicable in:** standard model, ROM

* **Assumes:** Reduction that forks at a critical point (i.e., hash query)

**Impossible with rewinding
under Interactive (One-More DL)**

Impossibility of Key-Unique Adaptive Schnorr Threshold Signatures Under Rewinding+(A)OMDL



* Solves q+1-(A)OMDL given a q-(A)OMDL reduction

Adaptive Schemes (Recent, Non-Exhaustive)

Scheme	Static Assumptions	Adaptive Assumptions (Full)	Our Results Apply
Schnorr			
Lindell22	Schnorr	X	
Classic S [M23]			✓
Sparkle [CKM23]	DL+ROM	AOMDL+ROM, $t/2$	
FROST/2/3 [\underline{K} G20, CGRS23, BC <u>K</u> M TZ22]	(A)OMDL+ROM	AOMDL + ROM + AGM + LVDR	
Gargos [BDLR25]		DDH+ROM	X



Takeaways

Takeaways

- Adaptive security is a stronger notion of unforgeability for threshold signatures; closer to a real-world adversary.

Takeaways

- Adaptive security is a stronger notion of unforgeability for threshold signatures; closer to a real-world adversary.
- However, proving adaptive security under standard(-ish) assumptions is challenging.

Takeaways

- Adaptive security is a stronger notion of unforgeability for threshold signatures; closer to a real-world adversary.
- However, proving adaptive security under standard(-ish) assumptions is challenging.
- Our results frame how adaptively secure schemes can be constructed, and classify the tradeoffs incurred (key-uniqueness).

Takeaways

- Adaptive security is a stronger notion of unforgeability for threshold signatures; closer to a real-world adversary.
- However, proving adaptive security under standard(-ish) assumptions is challenging.
- Our results frame how adaptively secure schemes can be constructed, and classify the tradeoffs incurred (key-uniqueness).
- More cryptanalysis is needed to determine if adaptive security is a meaningful property in practice.

Takeaways

- Adaptive security is a stronger notion of unforgeability for threshold signatures; closer to a real-world adversary.
- However, proving adaptive security under standard(-ish) assumptions is challenging.
- Our results frame how adaptively secure schemes can be constructed, and classify the tradeoffs incurred (key-uniqueness).
- More cryptanalysis is needed to determine if adaptive security is a meaningful property in practice.

Thank you!

Assumptions

Assumptions

- q-strong Diffie-Hellman Problem Given $(g_1, g_2, g_2^x, g_2^{x^2}, g_2^{x^3}, \dots)$, compute $g_1^{\frac{1}{x+c}}$, for an adaptively chosen c.

Assumptions

- q-strong Diffie-Hellman Problem Given $(g_1, g_2, g_2^x, g_2^{x^2}, g_2^{x^3}, \dots)$, compute $g_1^{\frac{1}{x+c}}$, for an adaptively chosen c .
- q-Bilinear Diffie-Hellman Inversion Assumption: Given $(g_1, g_2, g_2^x, g_2^{x^2}, g_2^{x^3}, \dots)$, compute $e(g_1, g_2)^{1/x}$

Definition

$KeyGen(1^\lambda) \rightarrow \{\{i, sk_i\}_{i=1}^n, \{PK_i\}_{i=1}^n, PK\}$

$Sign(m, sk_i) \rightarrow z_i$

$Combine(\{z_i\}_{i=1}^t) \rightarrow \sigma$

$Verify(PK, m, \sigma) \rightarrow 0/1$

$Recover(\{i, sk_i\}_{i=1}^t) \rightarrow sk$

Impossibility of Key-Unique Adaptive Schnorr Threshold Signatures Under Rewinding+(A)OMDL

Impossibility of Key-Unique Adaptive Schnorr Threshold Signatures Under Rewinding+(A)OMDL

- Given a reduction to q -(A)OMDL in the ROM adaptive security, we define a metareduction to $q+1$ -(A)OMDL.

Impossibility of Key-Unique Adaptive Schnorr Threshold Signatures Under Rewinding+(A)OMDL

- Given a reduction to q -(A)OMDL in the ROM adaptive security, we define a metareduction to $q+1$ -(A)OMDL.
- Applies to full adaptive security (t corruptions) or greater than $t/2$.

Example #3: Non-Key Unique Key Generation

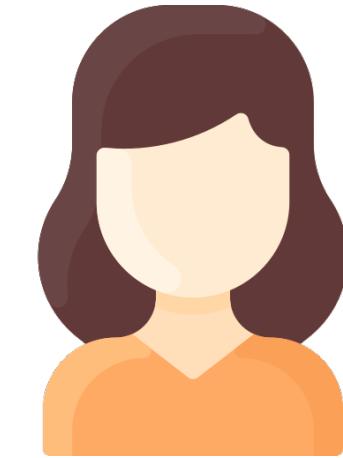


Example #3: Non-Key Unique Key Generation



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}, \{a'_1, \dots, a'_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Example #3: Non-Key Unique Key Generation



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}, \{a'_1, \dots, a'_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$ and $E'(x) = 0 + \sum_{j=1}^{t-1} a'_j x^j$

Example #3: Non-Key Unique Key Generation



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}, \{a'_1, \dots, a'_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$ and $E'(x) = 0 + \sum_{j=1}^{t-1} a'_j x^j$

Step 3: Define $sk_i = (E(i), E'(i))$

Example #3: Non-Key Unique Key Generation



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}, \{a'_1, \dots, a'_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$ and $E'(x) = 0 + \sum_{j=1}^{t-1} a'_j x^j$

Step 3: Define $sk_i = (E(i), E'(i))$

Step 4: Define $PK_i = g^{E(i)} h^{E'(i)}$ and $PK = g^{E(0)} = g^{sk}$

Example #3: Non-Key Unique Key Generation



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}, \{a'_1, \dots, a'_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$ and $E'(x) = 0 + \sum_{j=1}^{t-1} a'_j x^j$

Step 3: Define $sk_i = (E(i), E'(i))$

Step 4: Define $PK_i = g^{E(i)} h^{E'(i)}$ and $PK = g^{E(0)} = g^{sk}$
 $((i, sk_i), PK_i, PK)$

Non-Key Uniqueness for Example #3

Public key shares are not generated via an *injective* map f .

$$f(0, sk) \rightarrow g^{E(0)} = PK$$

$$f(i, sk_i) \rightarrow g^{E(i)} h^{E'(i)} = PK_i$$

(Single-Party) Schnorr Signatures



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

(Single-Party) Schnorr Signatures



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

To sign a message m :

$$r \xleftarrow{\$} \mathbb{Z}_q; \quad R \leftarrow g^r$$

$$c \leftarrow H(PK, m, R)$$

$$z \leftarrow r + csk$$

(Single-Party) Schnorr Signatures



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



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

To sign a message m :

$$r \xleftarrow{\$} \mathbb{Z}_q; \quad R \leftarrow g^r$$

$$c \leftarrow H(PK, m, R)$$

$$z \leftarrow r + csk$$

(Single-Party) Schnorr Signatures



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



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

To sign a message m :

$$r \xleftarrow{\$} \mathbb{Z}_q; \quad R \leftarrow g^r$$

$$c \leftarrow H(PK, m, R)$$

$$z \leftarrow r + csk$$

To verify (PK, σ, m) :

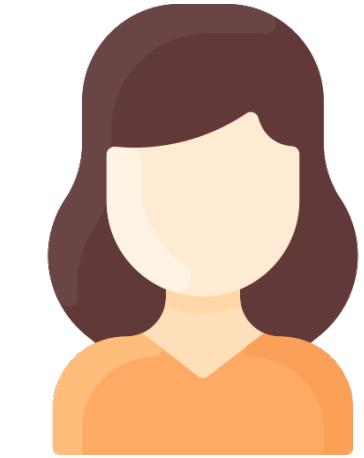
$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c \stackrel{?}{=} g^z$$

output accept/reject

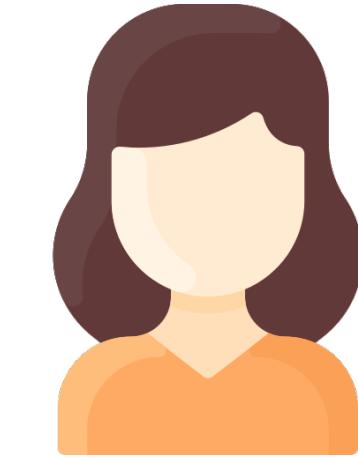
Case Study: Single Inconsistent Player

Key Generation:



Case Study: Single Inconsistent Player

Key Generation:



Step 1: Sample $(sk_i, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$ for each $i \in \{1, \dots, n\}$

Case Study: Single Inconsistent Player

Key Generation:



Step 1: Sample $(sk_1, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$ for each $i \in \{1, \dots, n\}$

Step 2: Define $f_1(x) = sk_1 + \sum_{j=1}^{t-1} a_j x^j$

Case Study: Single Inconsistent Player

Key Generation:



Step 1: Sample $(sk_1, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$ for each $i \in \{1, \dots, n\}$

Step 2: Define $f_1(x) = sk_1 + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = \sum_{j=1}^n f_j(i)$

Case Study: Single Inconsistent Player

Key Generation:



Step 1: Sample $(sk_1, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$ for each $i \in \{1, \dots, n\}$

Step 2: Define $f_1(x) = sk_1 + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = \sum_{j=1}^n f_j(i)$

Step 4: Set $sk = \sum_{i=1}^n sk_i$ and $PK = g^{f(0)} = g^{sk}$

Multi-Party Schnorr Signatures

How to share r ?

How to share sk ?

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

$sig = (R, z)$

What do we want?

- output signature that verifies like standard Schnorr signature
 - public key looks like standard Schnorr signature public key
- few (2-3) rounds
 - Stinson & Strobl 2001 uses DKG for signing
- reasonable security assumptions
- concurrent security

	Scheme	Assumptions	Signing Rounds
Multi-sigs	MuSig [MPSW18, BDN18]	DL+ROM	3
	SimpleMuSig [BDN18, CKM21]		
Threshold	MuSig2 [NRS21]	OMDL+ROM	2
	DWMS [AB21]		
Threshold	SpeedyMuSig [CKM21]		
	Lindell22	Schnorr	3
	Sparkle [CKM23]	DL+ROM	
	FROST [KG20, BCKMTZ22]	OMDL+ROM	2
	FROST2 [CKM21]		

One-More Discrete Log (OMDL):
 - stronger assumption
 + partially non-interactive schemes

	Scheme	Assumptions	Signing Rounds
Multi-sigs	MuSig [MPSW18, BDN18]	DL+ROM	3
	SimpleMuSig [BDN18, CKM21]		
Threshold	MuSig2 [NRS21]	OMDL+ROM	2
	DWMS [AB21]		
Threshold	SpeedyMuSig [CKM21]		
	Lindell22	Schnorr	3
	Sparkle [CKM23]	DL+ROM	
	FROST [KG20, BCKMTZ22]	OMDL+ROM	2
	FROST2 [CKM21]		

All are concurrently secure ✓

One-More Discrete Log (OMDL):

- stronger assumption
- + partially non-interactive schemes

Concurrent Security: ROS Attacks

[NKDM03, DEFKLNS19,
BLLOR21]

Session 1

sk_1



...

Session k



sk_2



Concurrent Security: ROS Attacks

[NKDM03, DEFKLNS19,
BLLOR21]

Session 1

sk_1



$R_1^{(1)}$

...

Session k

$R_1^{(k)}$



sk_2



Concurrent Security: ROS Attacks

[NKDM03, DEFKLNS19,
BLLOR21]

Session 1

sk_1



$R_1^{(1)}$

$R_2^{(1)}$

...

Session k



$R_1^{(k)}$

$R_2^{(k)}$

sk_2



Concurrent Security: ROS Attacks

[NKDM03, DEFKLNS19,
BLLOR21]

Session 1

sk_1



$R_1^{(1)}$

$R_2^{(1)}$

...

Session k



$R_1^{(k)}$

$R_2^{(k)}$

Can forge!



Concurrent Security: ROS Attacks

[NKDM03, DEFKLNS19,
BLLOR21]

Session 1

sk_1



...

$R_1^{(1)}$

$R_2^{(1)}$

sk_2



Can forge!

Session k



$R_1^{(k)}$

$R_2^{(k)}$

Affected:

- multi-signatures
- threshold signatures
- blind signatures

Solution: Force adversary to commit to its nonces...

MuSig2 / SpeedyMuSig / FROST/2



Key Generation:

$$(sk_i, PK_i), PK$$



Combine / Verify:

MuSig2 / SpeedyMuSig / FROST/2

 R_i, S_i 

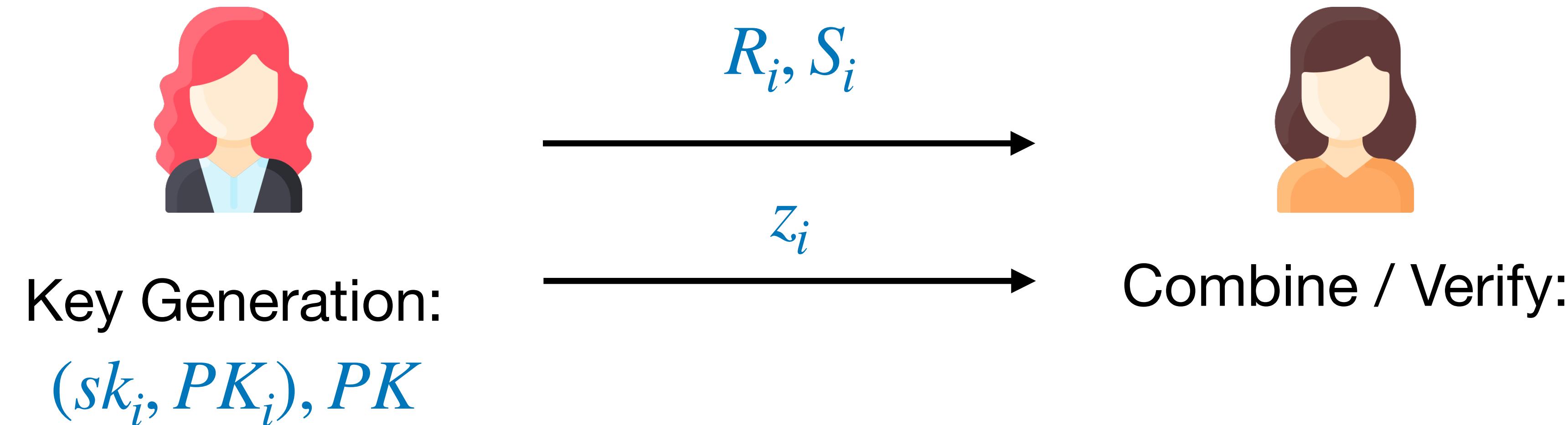
Key Generation:

 $(sk_i, PK_i), PK$

Round 1: Output $R_i \leftarrow g^{r_i}, S_i \leftarrow g^{s_i}$

Combine / Verify:

MuSig2 / SpeedyMuSig / FROST/2



Round 1: Output $R_i \leftarrow g^{r_i}, S_i \leftarrow g^{s_i}$

Round 2: $a \leftarrow H'(PK, m, \{R_i, S_i\}_{i=1}^n)$

$$R = \prod_{i=1}^n R_i S_i^a$$

$$c \leftarrow H(PK, m, R)$$

$$\text{Output } z_i \leftarrow r_i + a s_i + c sk_i$$

MuSig2 / SpeedyMuSig / FROST/2

 R_i, S_i  z_i

Key Generation:

$$(sk_i, PK_i), PK$$

Round 1: Output $R_i \leftarrow g^{r_i}, S_i \leftarrow g^{s_i}$

Round 2: $a \leftarrow H'(PK, m, \{R_i, S_i\}_{i=1}^n)$

$$R = \prod_{i=1}^n R_i S_i^a$$

$$c \leftarrow H(PK, m, R)$$

$$\text{Output } z_i \leftarrow r_i + a s_i + c sk_i$$

Combine / Verify:

$$z = \sum_{i=1}^n z_i$$

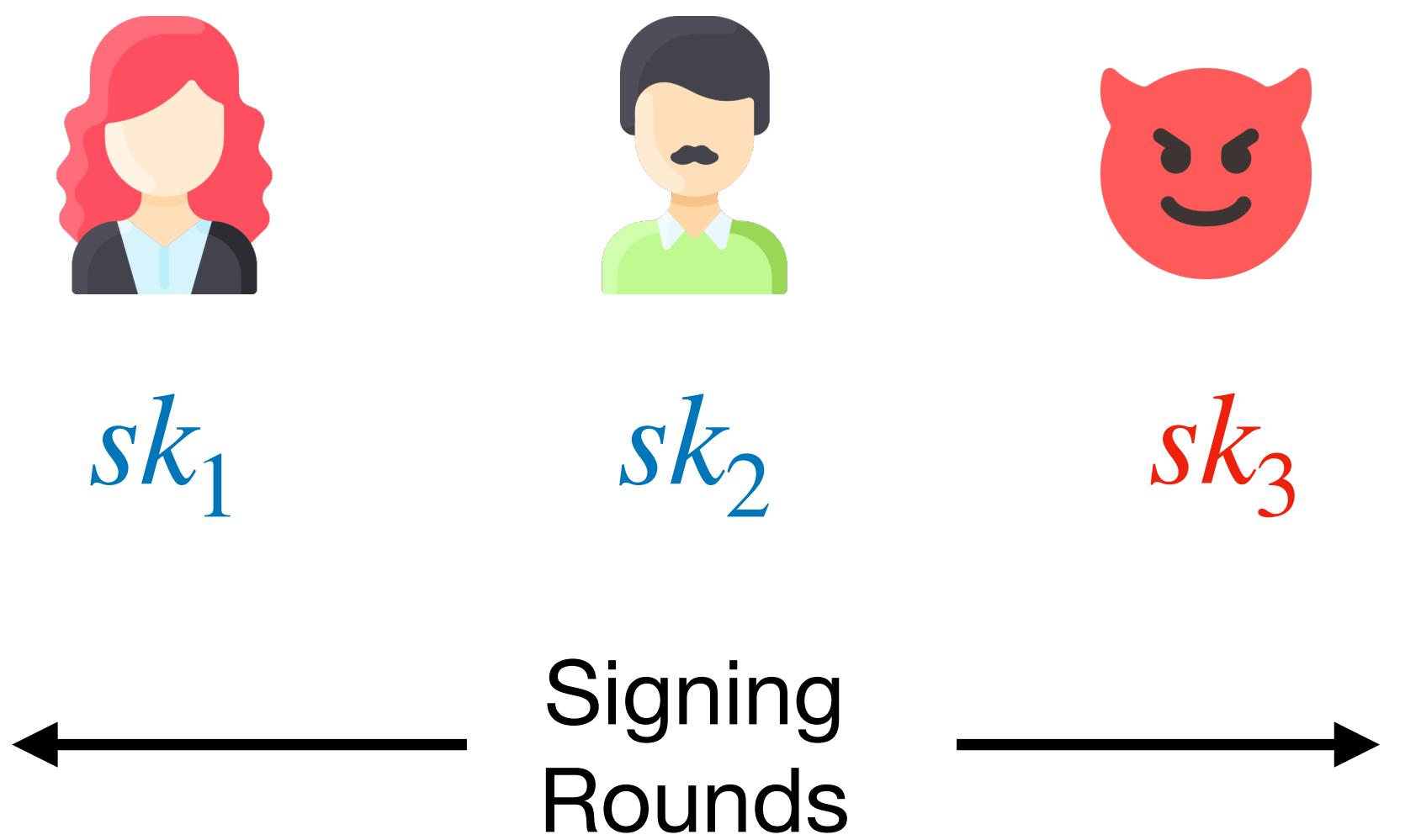
$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z \quad \checkmark$$

Adaptive Security

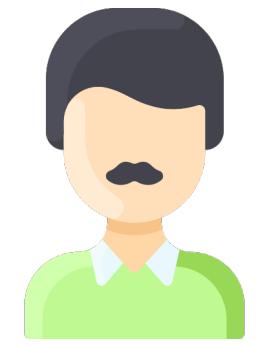
Static Corruption



Adaptive Security

Adaptive Corruption

Static Corruption

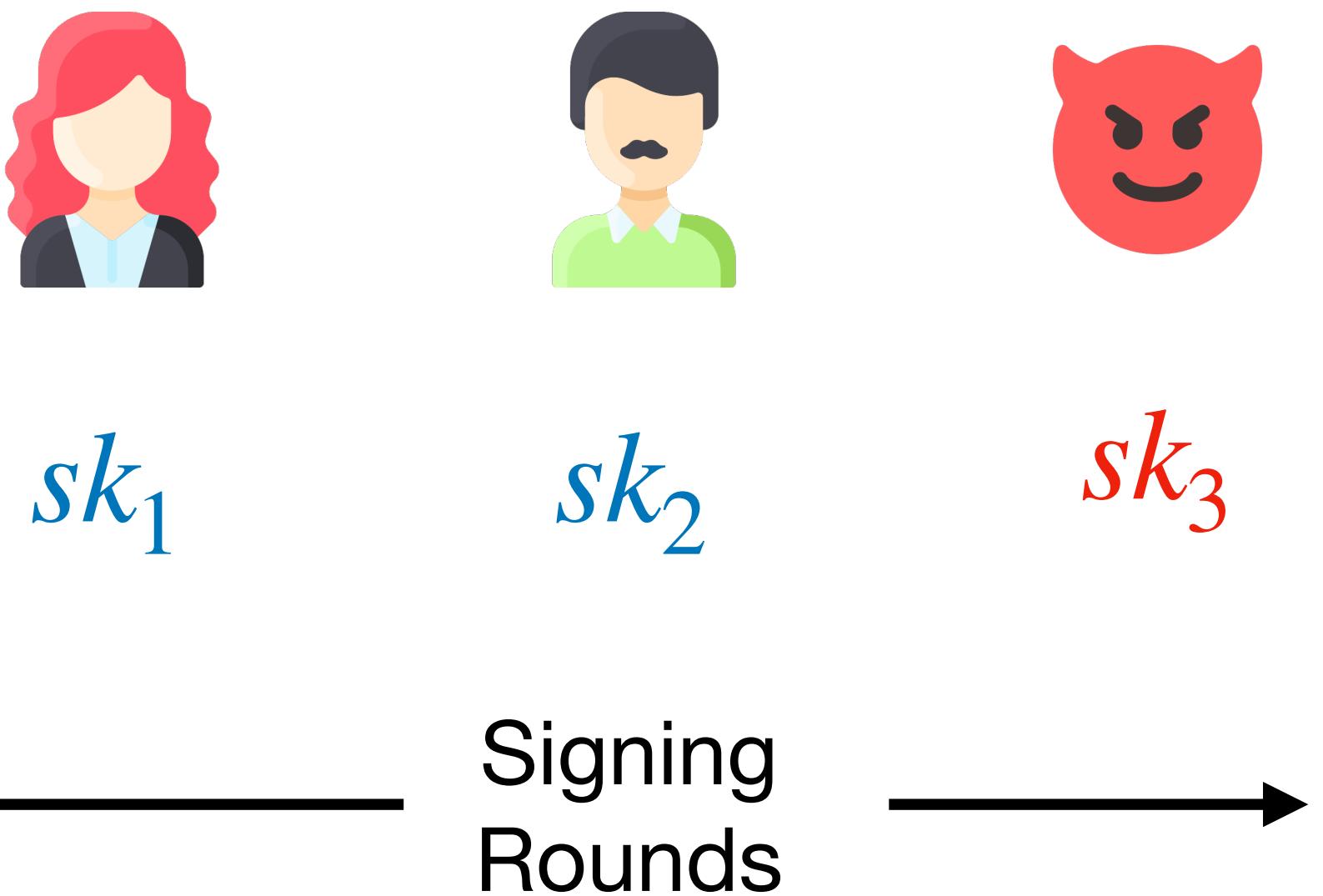


sk_1

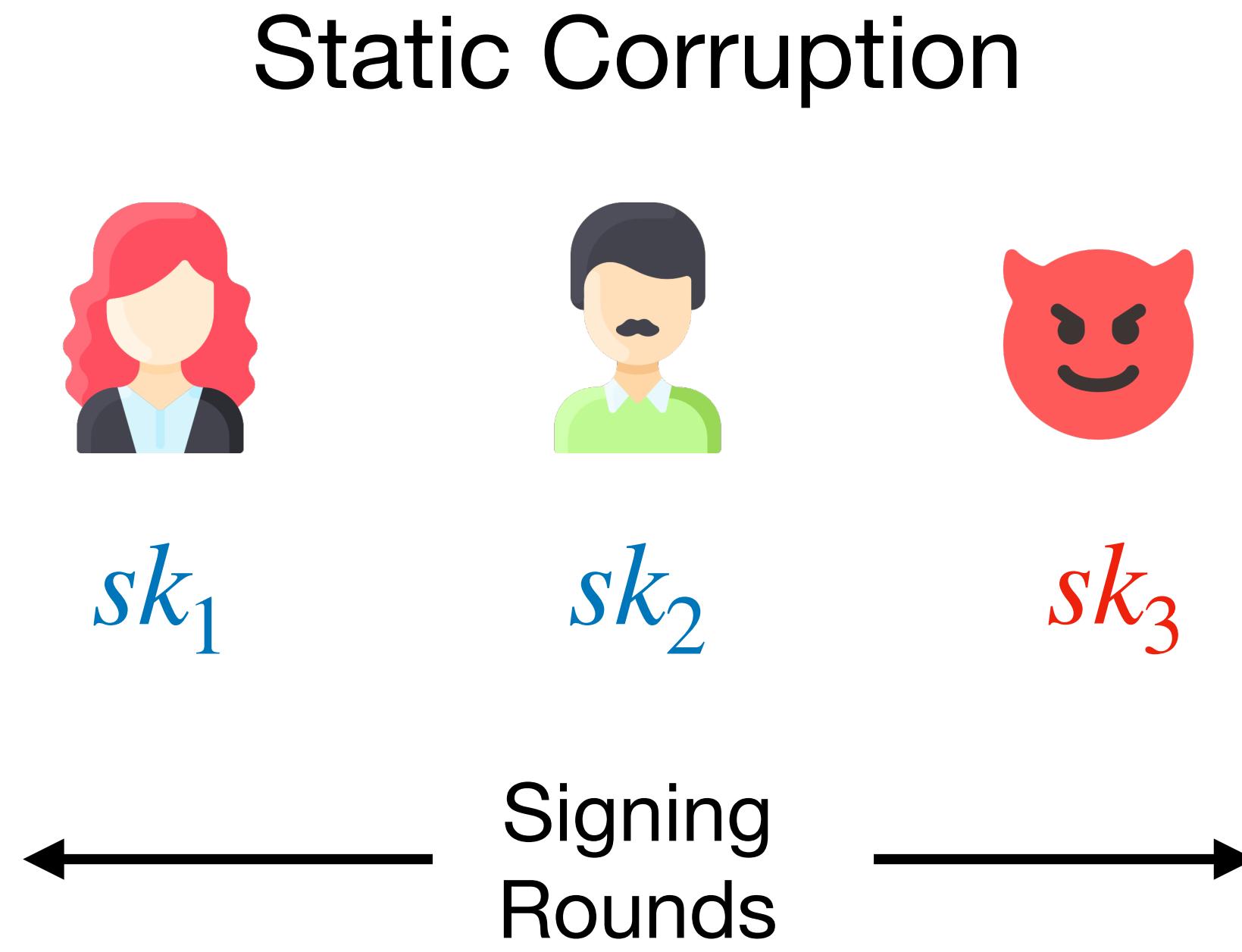
sk_2

sk_3

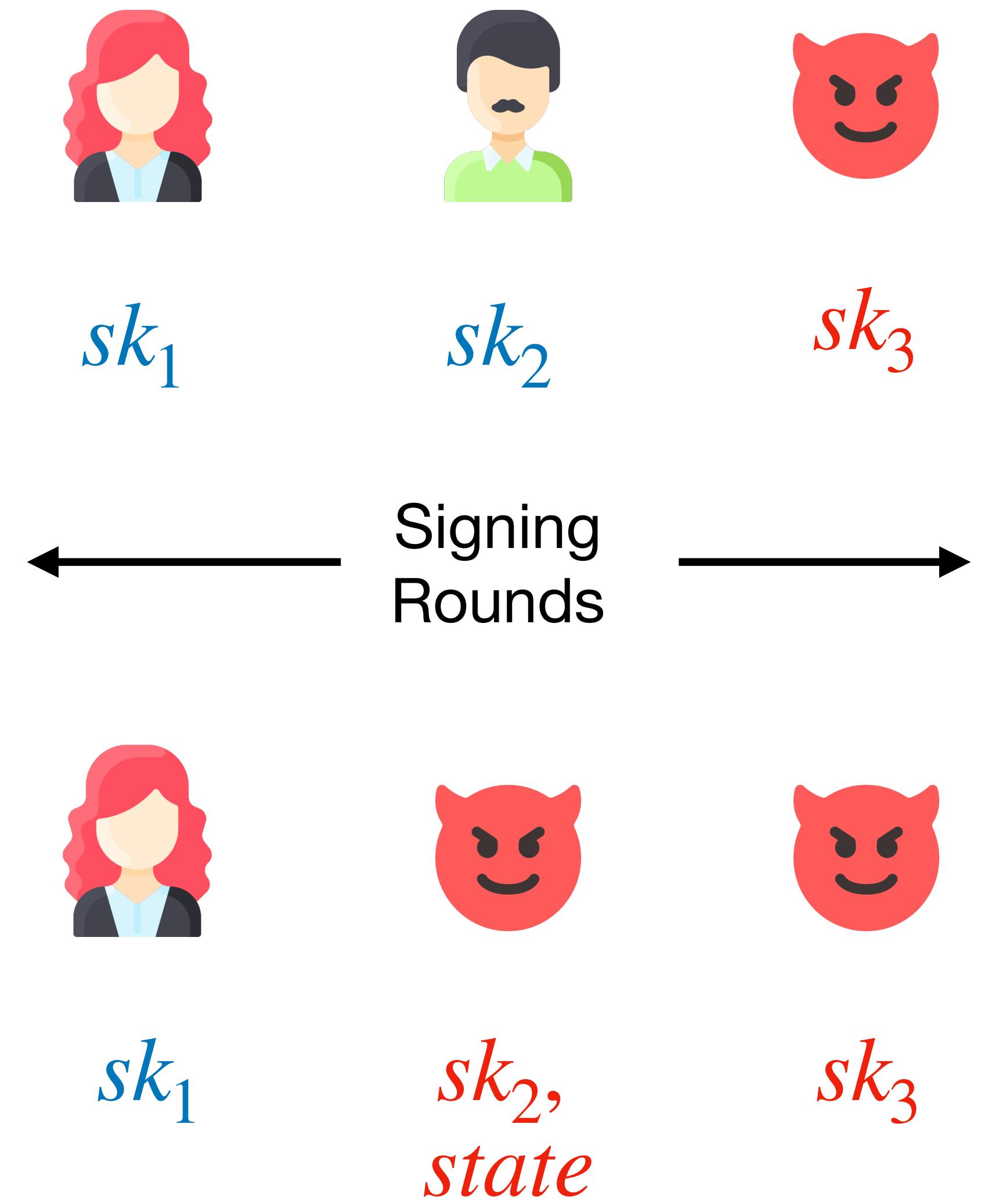
←
Signing
Rounds
→



Adaptive Security



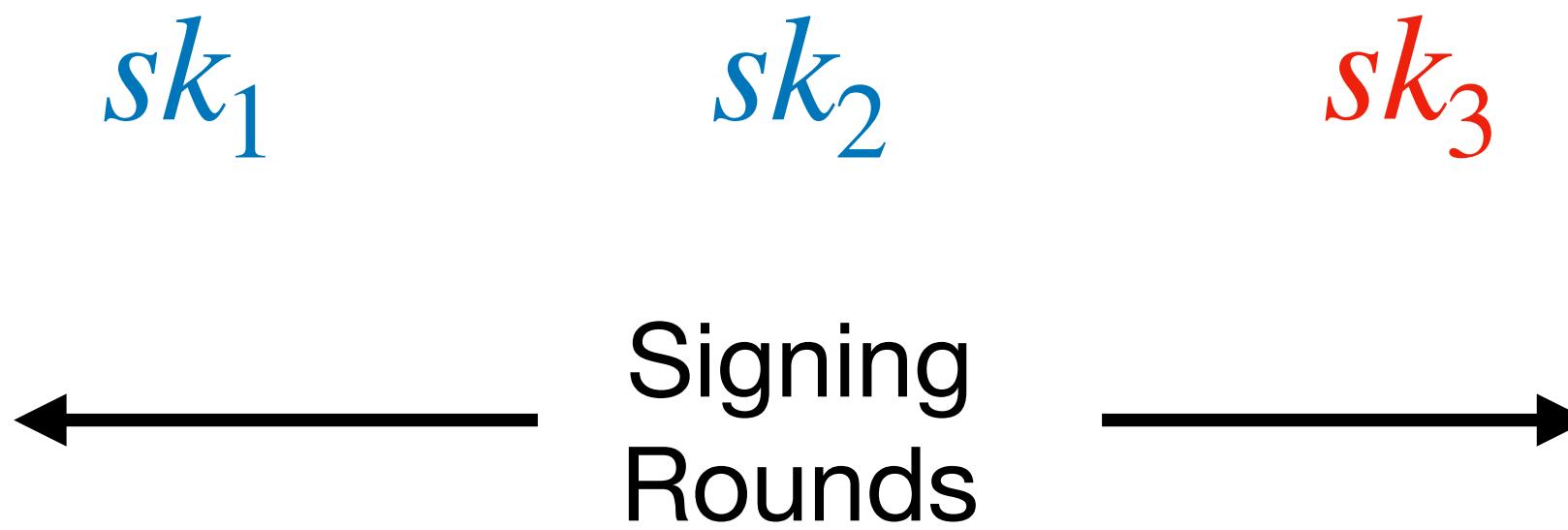
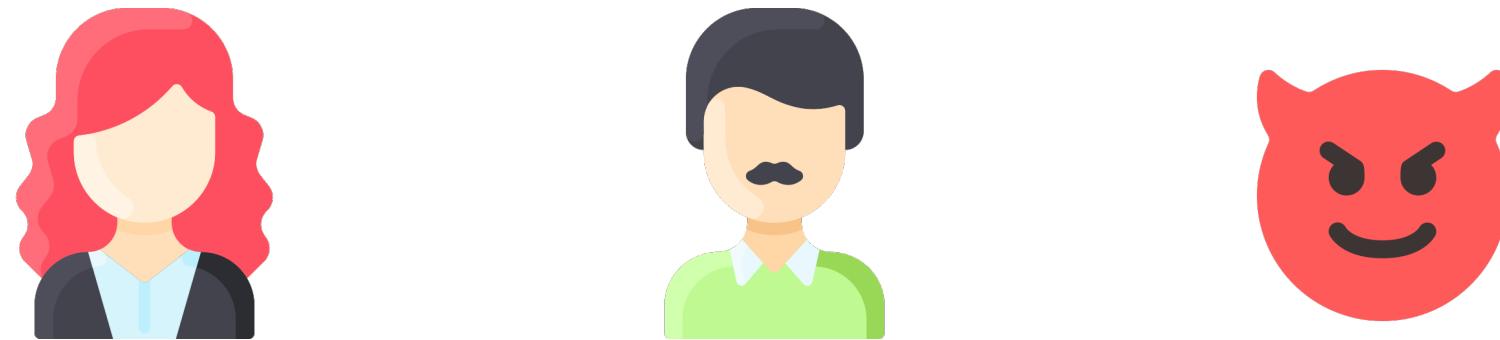
Adaptive Corruption



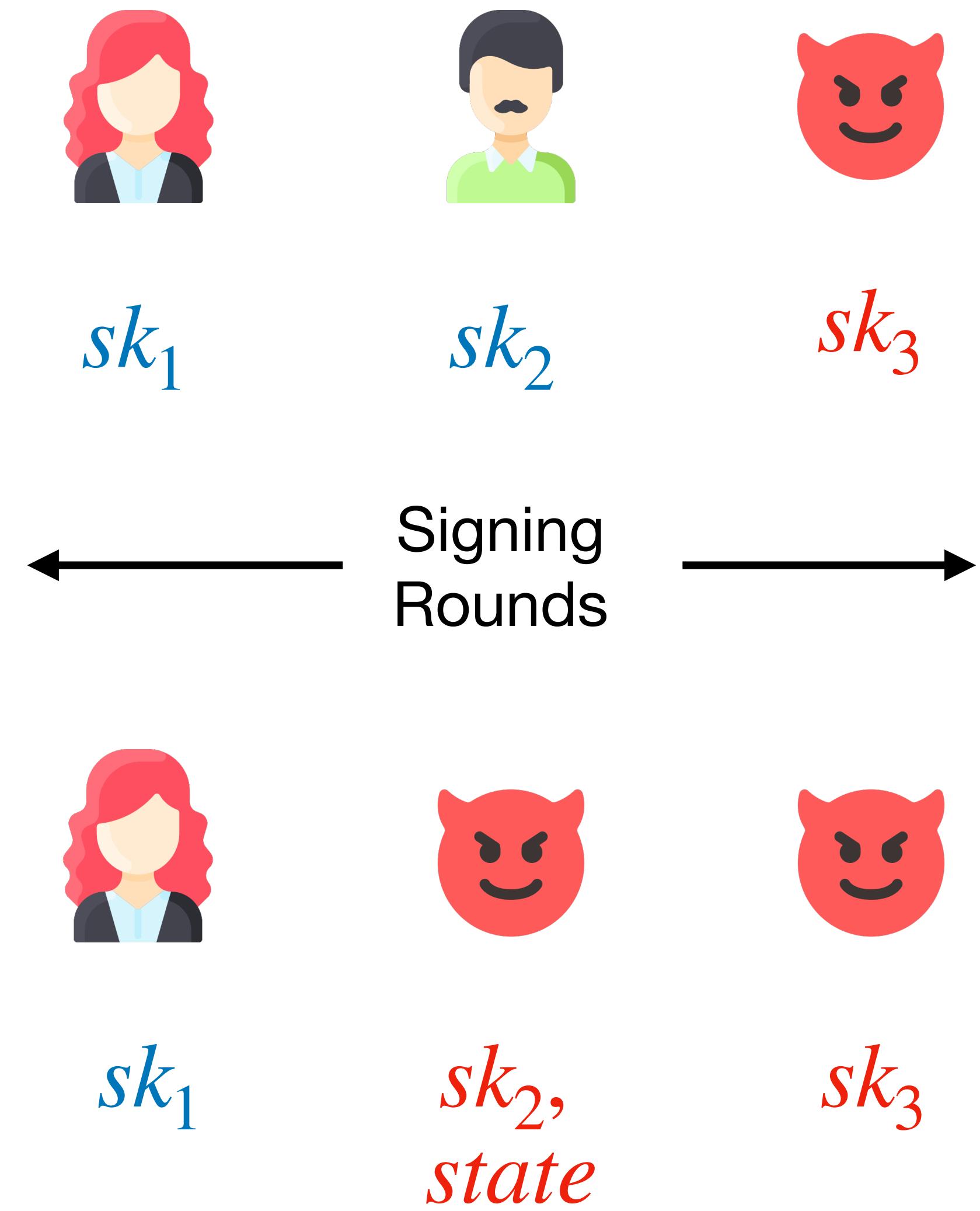
Adaptive Security

Adaptive Corruption

Static Corruption



- Adaptive security of Sparkle [CKM23], FROST forthcoming



MuSig2: Simple Two-Round Schnorr Multi-Signatures

Jonas Nick¹, Tim Ruffing¹, and Yannick Seurin²

How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures

Elizabeth Crites¹, Chelsea Komlo², and Mary Maller³

Better than Advertised Security for Non-interactive Threshold Signatures

Mihir Bellare¹ , Elizabeth Crites², Chelsea Komlo³, Mary Maller⁴,
Stefano Tessaro⁵, and Chenzhi Zhu⁵ 

Fully Adaptive Schnorr Threshold Signatures

Elizabeth Crites¹, Chelsea Komlo², and Mary Maller³

FROST: Flexible Round-Optimized Schnorr Threshold Signatures

Chelsea Komlo
University of Waterloo, Zcash Foundation
ckomlo@uwaterloo.ca

Ian Goldberg
University of Waterloo
iang@uwaterloo.ca

ROAST: Robust Asynchronous Schnorr Threshold Signatures

Tim Ruffing
Blockstream
crypto@timruffing.de

Viktoria Ronge
Friedrich-Alexander-Universität
Erlangen-Nürnberg
ronge@cs.fau.de

Elliott Jin
Blockstream
eyj@blockstream.com

Jonas Schneider-Bensch
CISPA Helmholtz Center for
Information Security
jonas.schneider-bensch@cispa.de

Dominique Schröder
Friedrich-Alexander-Universität
Erlangen-Nürnberg
dominique.schroeder@fau.de

A Formal Treatment of Distributed Key Generation, and New Constructions

Chelsea Komlo, Ian Goldberg, Douglas Stebila

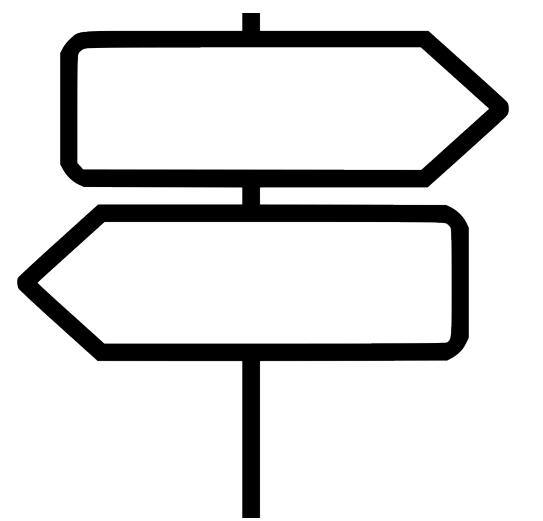
From Theory to Practice: A Hitchhiker's Guide



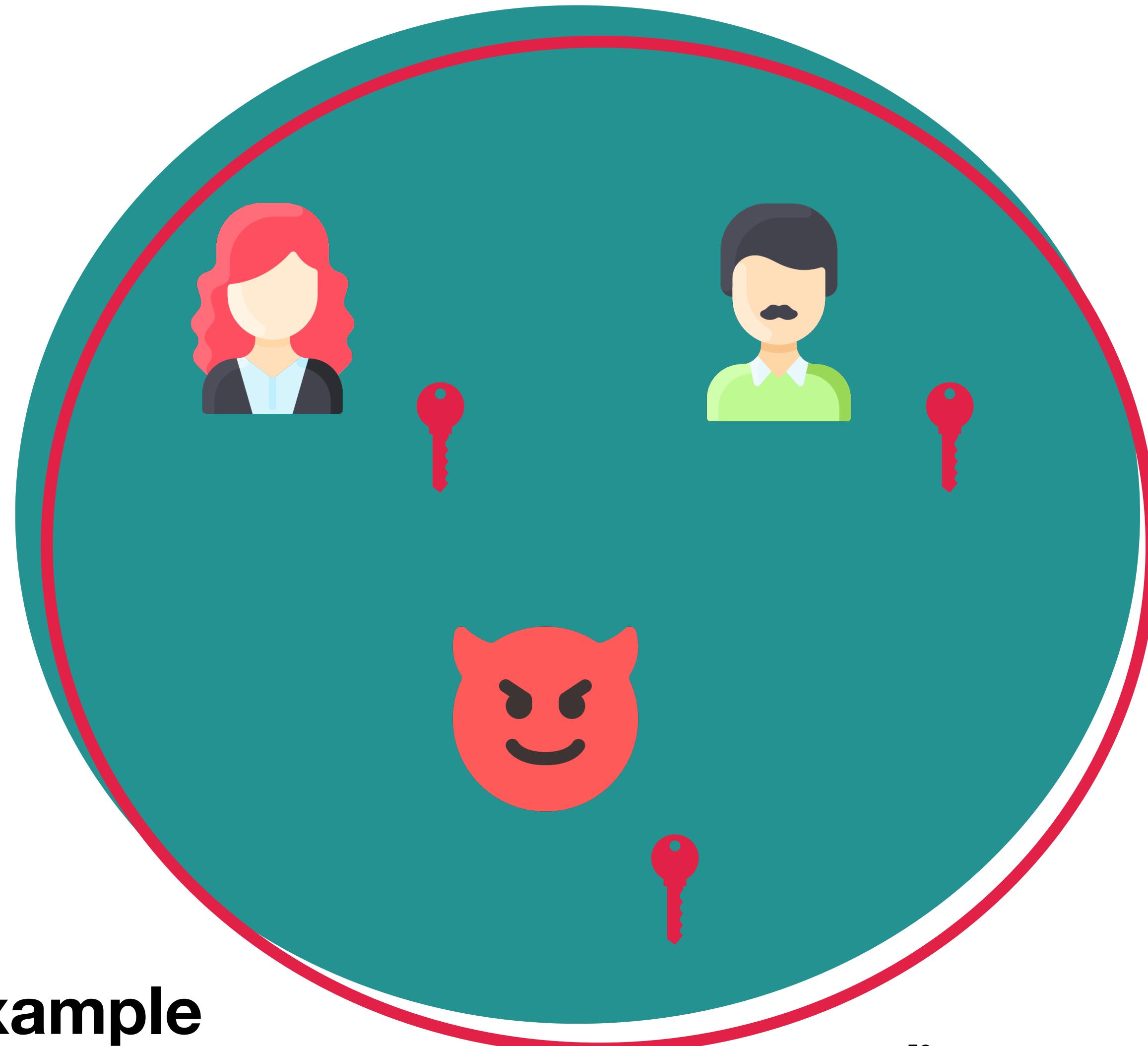
Unforgeability:
Attacker cannot forge signatures.

Liveness:
System can always create signatures.

Multi-Signatures vs. Threshold Signatures

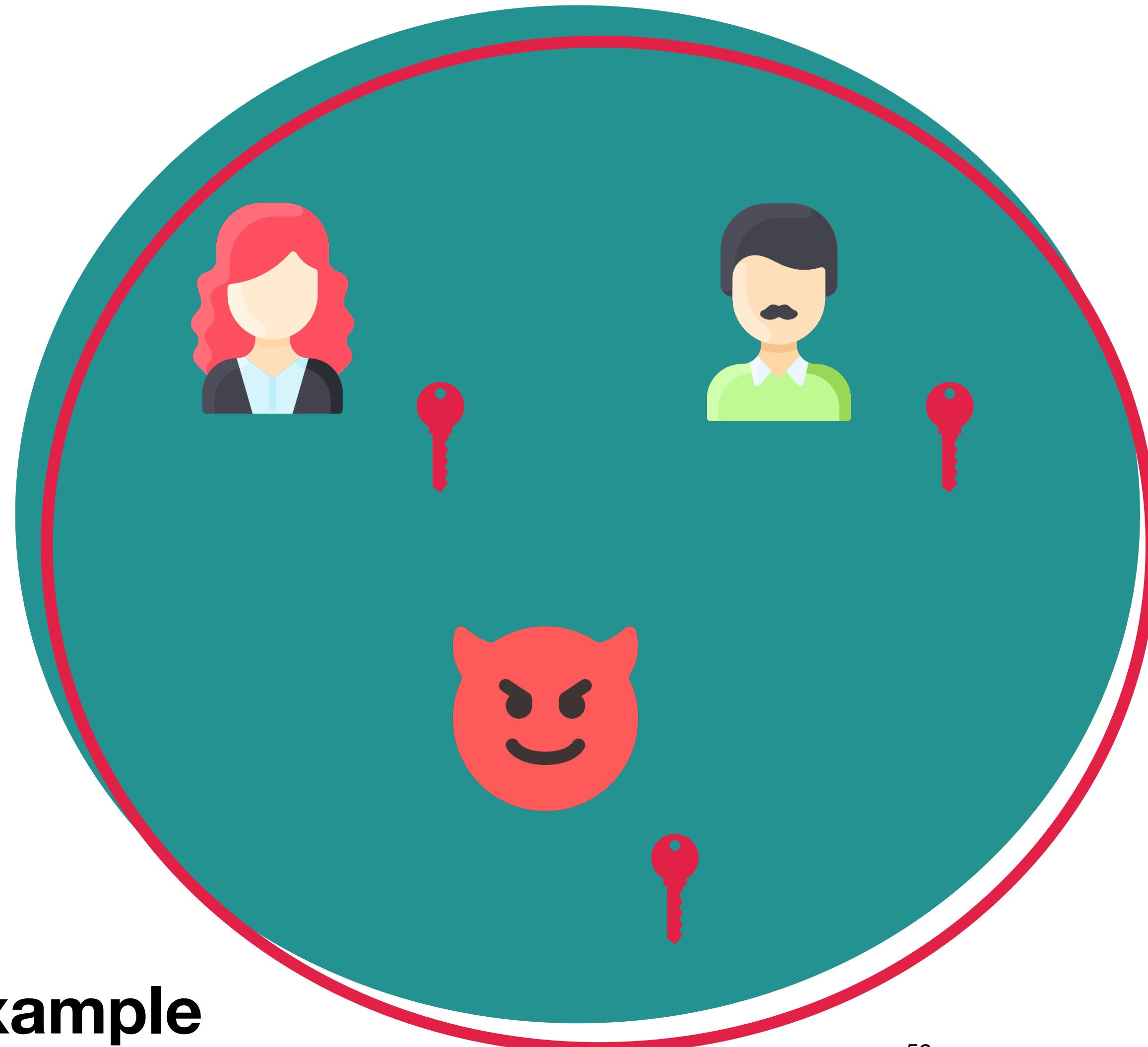


Multi-Signatures do not Guarantee Liveness



(3,3) Example

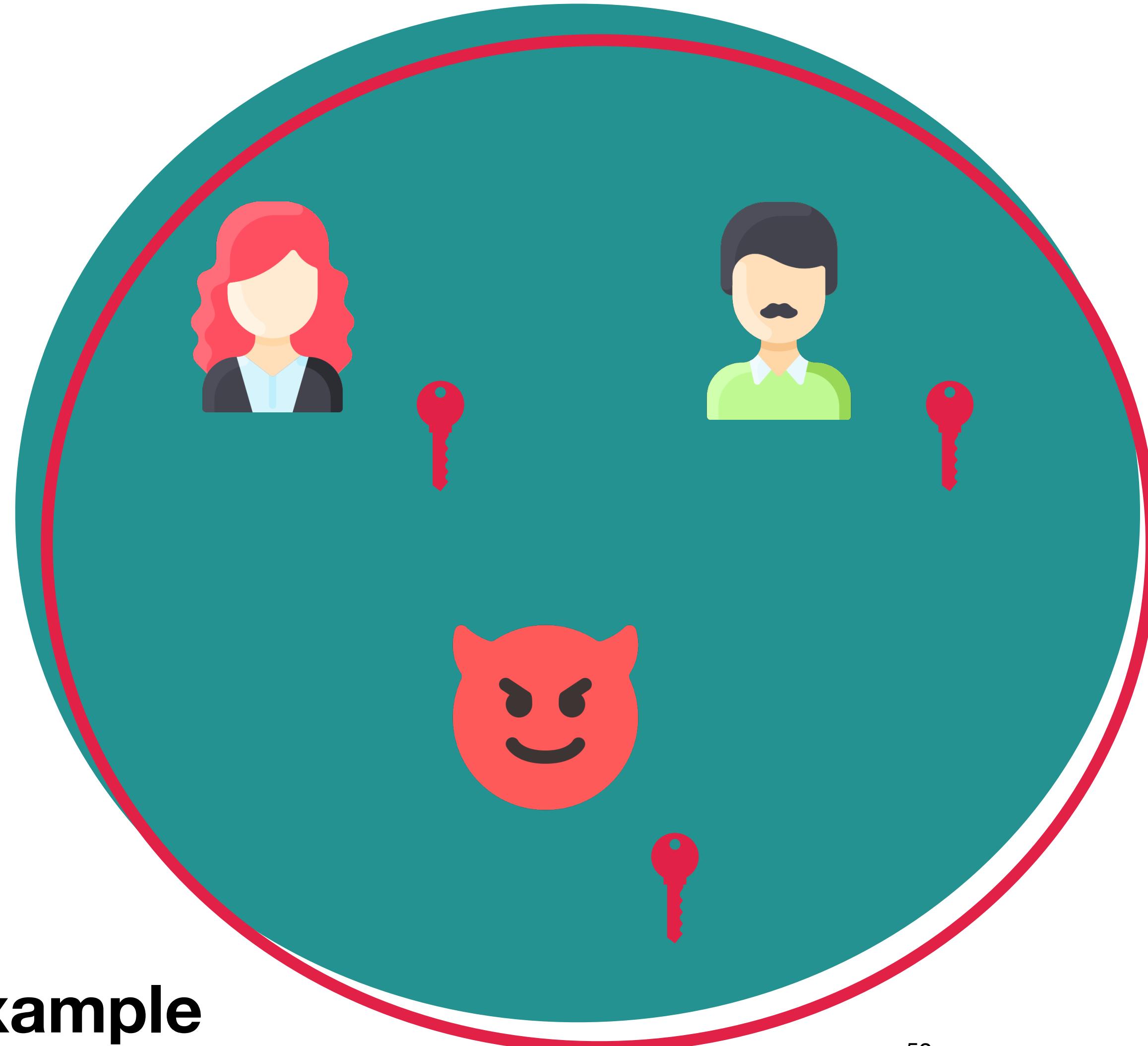
Multi-Signatures do not Guarantee Liveness



(3,3) Example

- If only one signer is unavailable, signing is not possible.

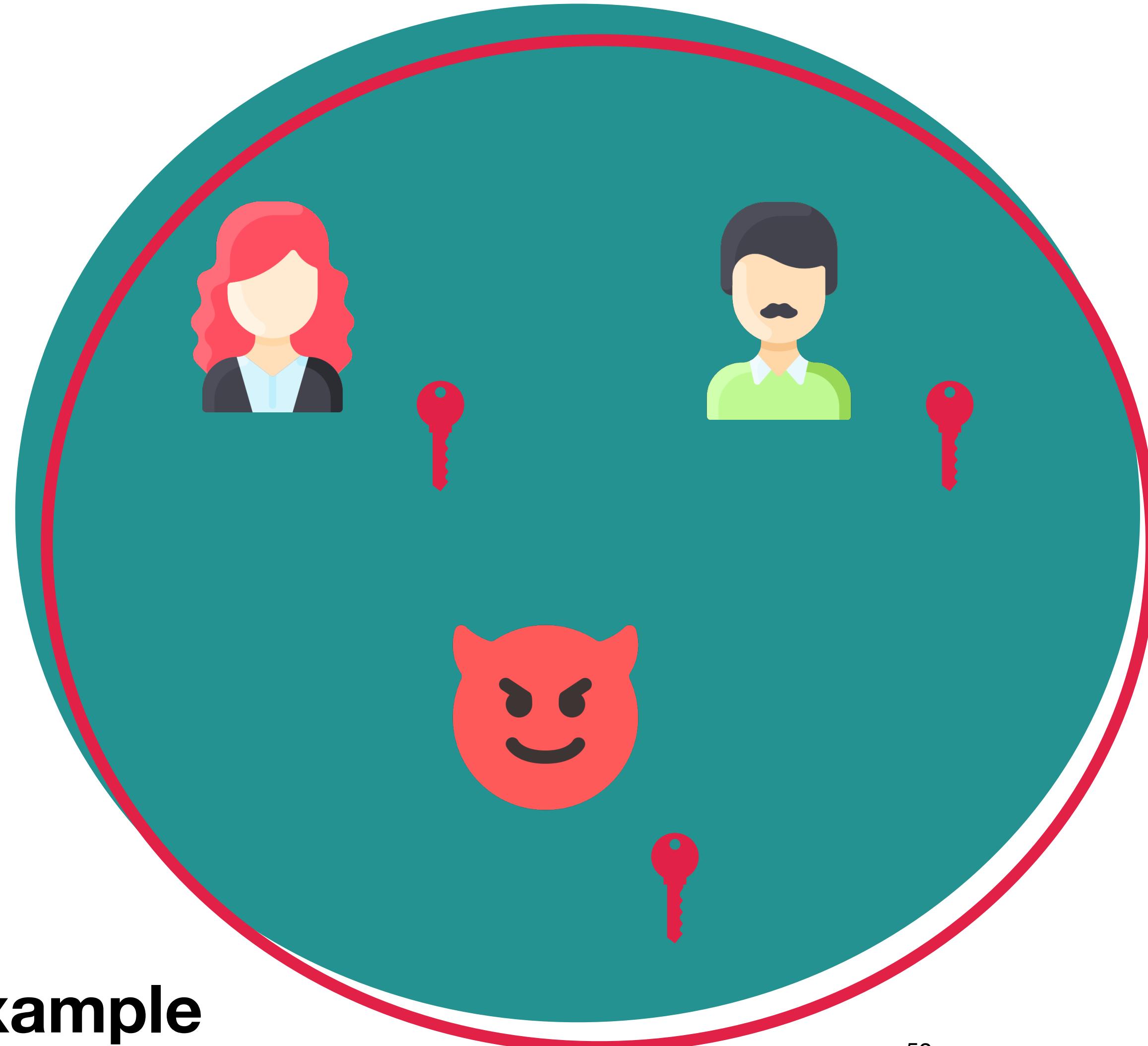
Multi-Signatures do not Guarantee Liveness



(3,3) Example

- If only one signer is unavailable, signing is not possible.
- Needs to be handled on a different layer of the system.

Multi-Signatures do not Guarantee Liveness



(3,3) Example

- If only one signer is unavailable, signing is not possible.
- Needs to be handled on a different layer of the system.
- Possible to use non-interactive key aggregation instead of DKG.

DKGs can be Cumbersome

DKGs can be Cumbersome

- Distributed Key Generation (DKG)

DKGs can be Cumbersome

- Distributed Key Generation (DKG)
- Major pain point: DKGs require some kind of broadcast channel

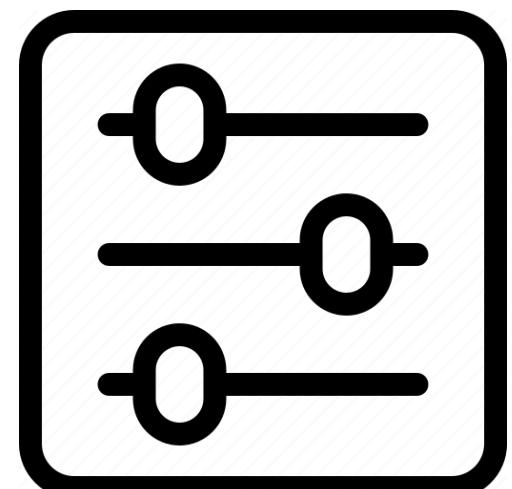
DKGs can be Cumbersome

- Distributed Key Generation (DKG)
- Major pain point: DKGs require some kind of broadcast channel
 - Protocol descriptions often just assume that all communication takes place over reliable broadcast (= consensus/BFT)

DKGs can be Cumbersome

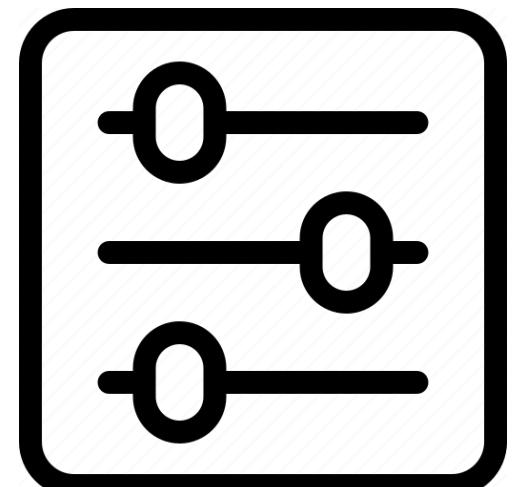
- Distributed Key Generation (DKG)
- Major pain point: DKGs require some kind of broadcast channel
 - Protocol descriptions often just assume that all communication takes place over reliable broadcast (= consensus/BFT)
 - Implementers often fail to understand this, or simply ignore it

How to Choose (n, t) for Threshold Signatures ?



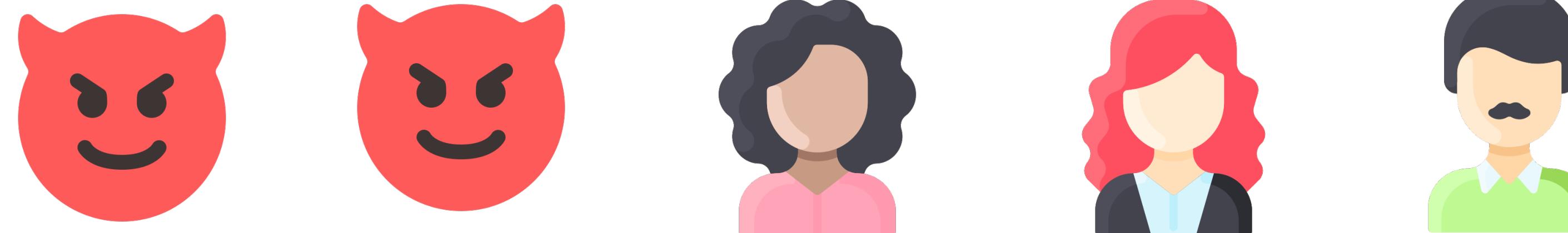
How to Choose (n, t) for Threshold Signatures ?

**FROST supports any choice,
but that just makes the problem harder!**



Honest Majority (Classic)

$n = 5$



$t = 3$

**Maximum number of tolerable bad signers for
unforgeability:** $t - 1 = 2$ **liveness:** $n - t = 2$

Honest Majority (Classic)

$n = 5$



$t = 3$

**Maximum number of tolerable bad signers for
unforgeability:**

$$t - 1 = 2$$

liveness:
 $n - t = 2$

may be required in consensus
systems anyway

Honest Minority

$n = 5$



$t = 4$

**Maximum number of tolerable bad signers for
unforgeability:** $t - 1 = 3$

liveness: $n - t = 1$

Honest Minority

$n = 5$



$t = 4$

**Maximum number of tolerable bad signers for
unforgeability:**

$$t - 1 = 3$$

liveness:
 $n - t = 1$

**no progress with 2 bad signers
but also no forgery**

Full Threshold

$n = 5$



$t = 5$

**Maximum number of tolerable bad signers for
unforgeability:** $t - 1 = 4$

liveness: $n - t = 0$

Full Threshold

$n = 5$



$t = 5$

**Maximum number of tolerable bad signers for
unforgeability:**

$$t - 1 = 4$$

**multi-signatures possible:
(non-interactive
key aggregation, no DKG)**

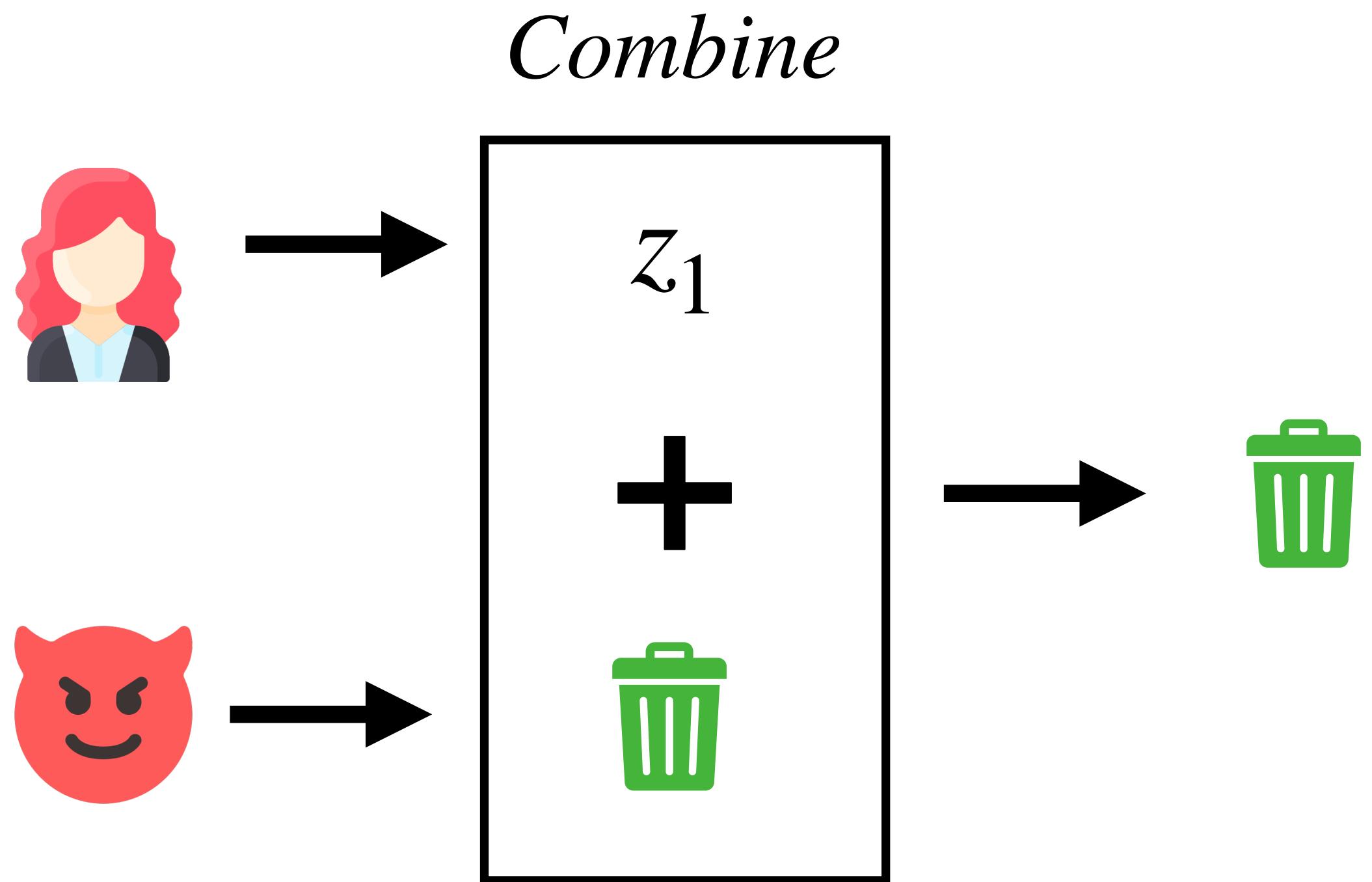
liveness:
 $n - t = 0$

Robustness: the protocol succeeds so long as at least t players participate honestly.

(required for liveness!)

FROST and Robustness

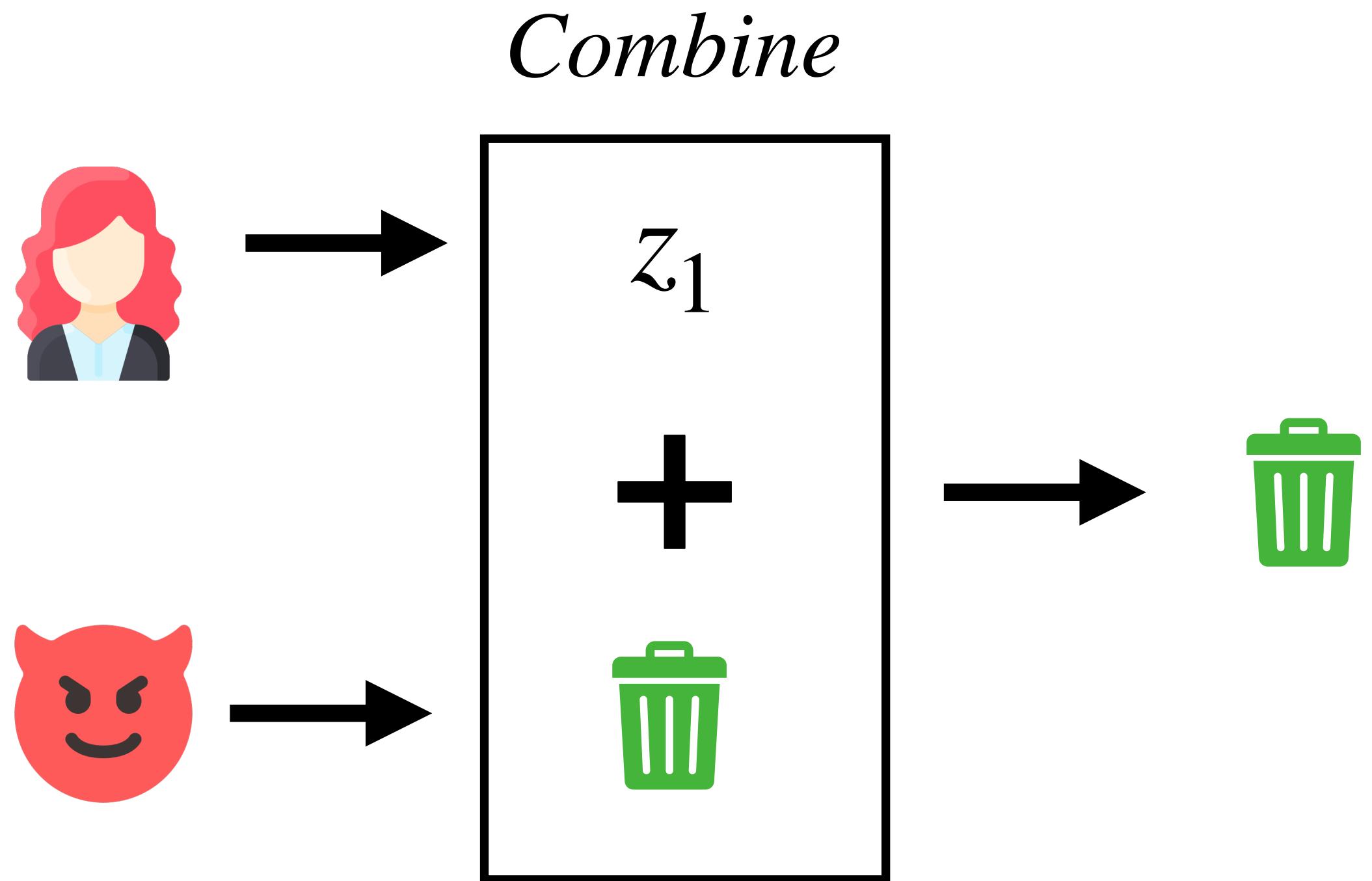
- FROST is **not** robust.



(2,3) Example

FROST and Robustness

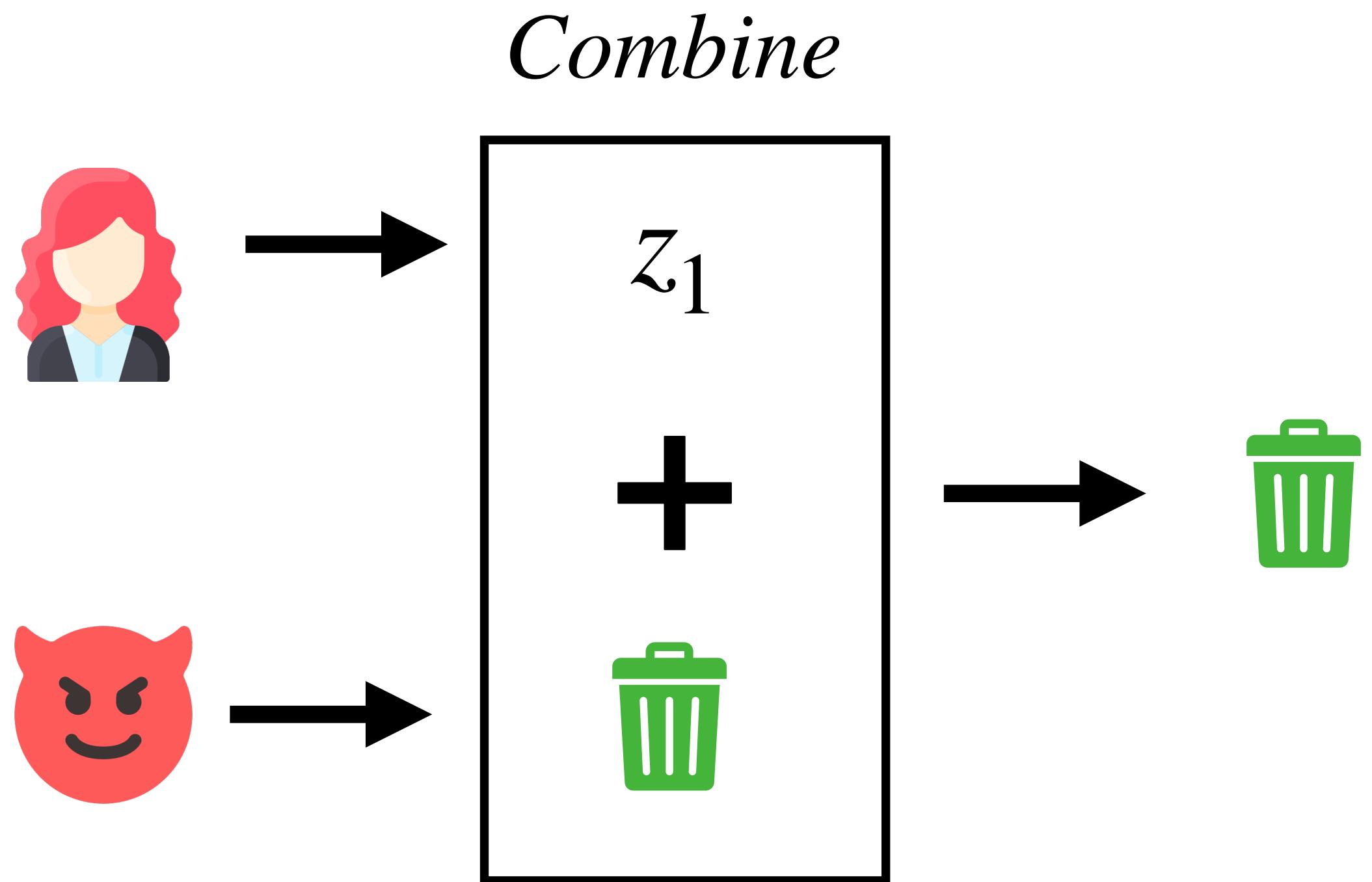
- FROST is **not** robust.
- If even one FROST signer issues garbage, the resulting signature is garbage



(2,3) Example

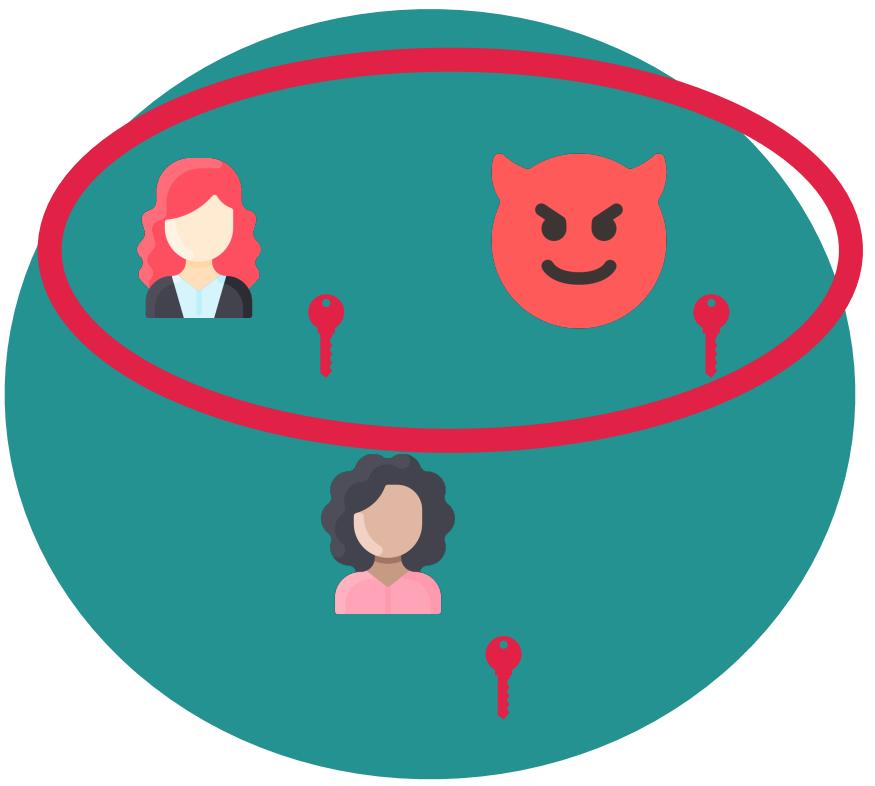
FROST and Robustness

- FROST is **not** robust.
- If even one FROST signer issues garbage, the resulting signature is garbage
- Then the protocol must be re-run with a different subset of signers.

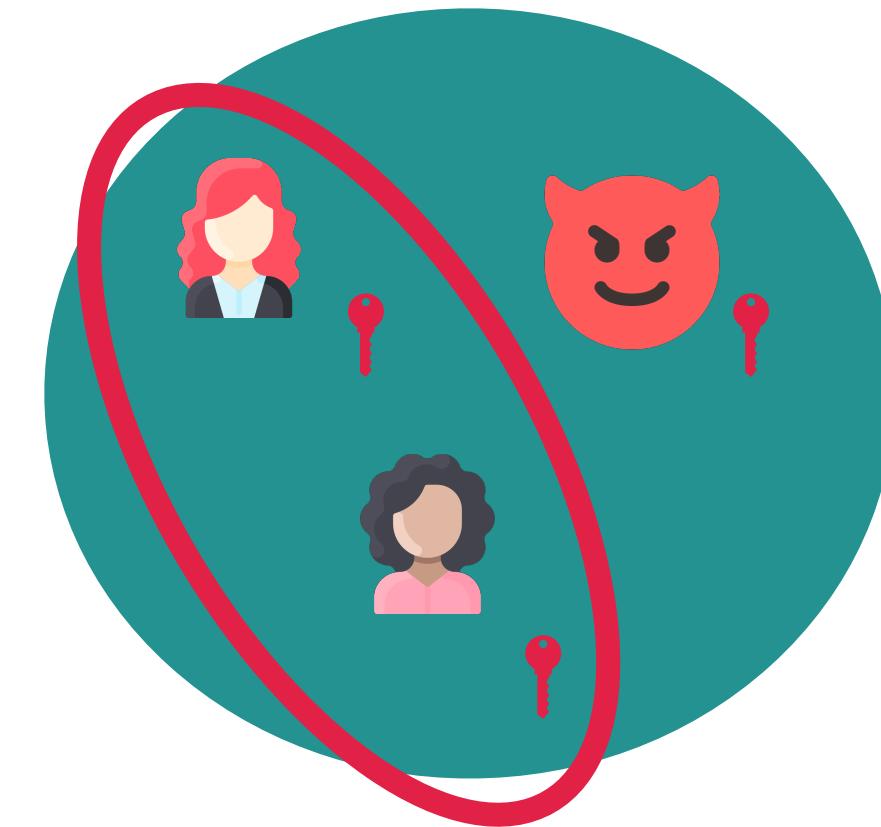
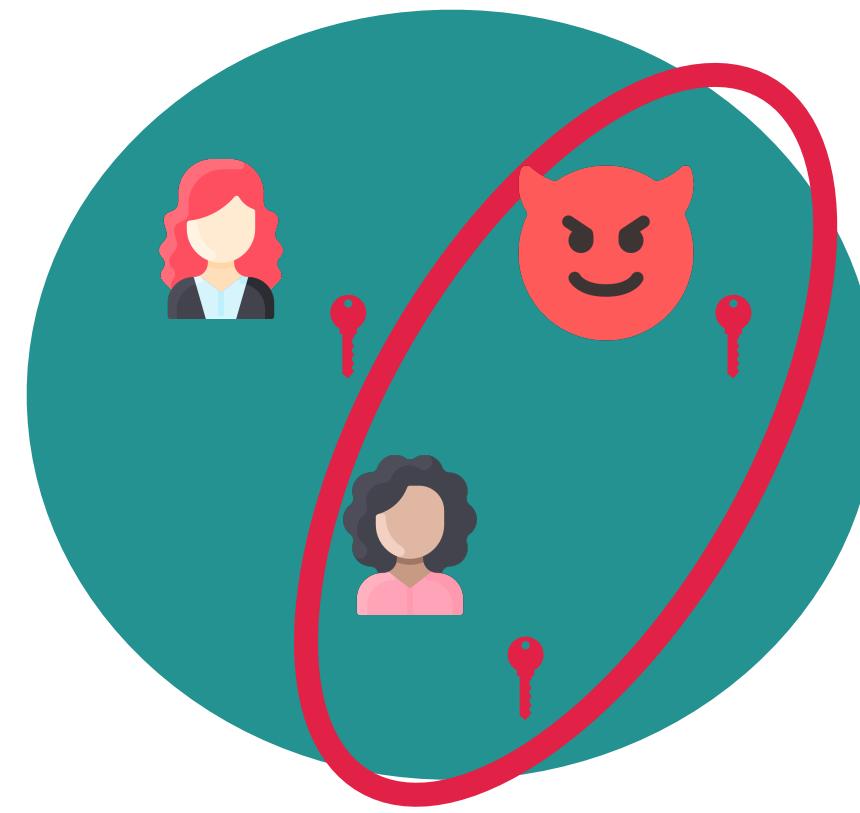
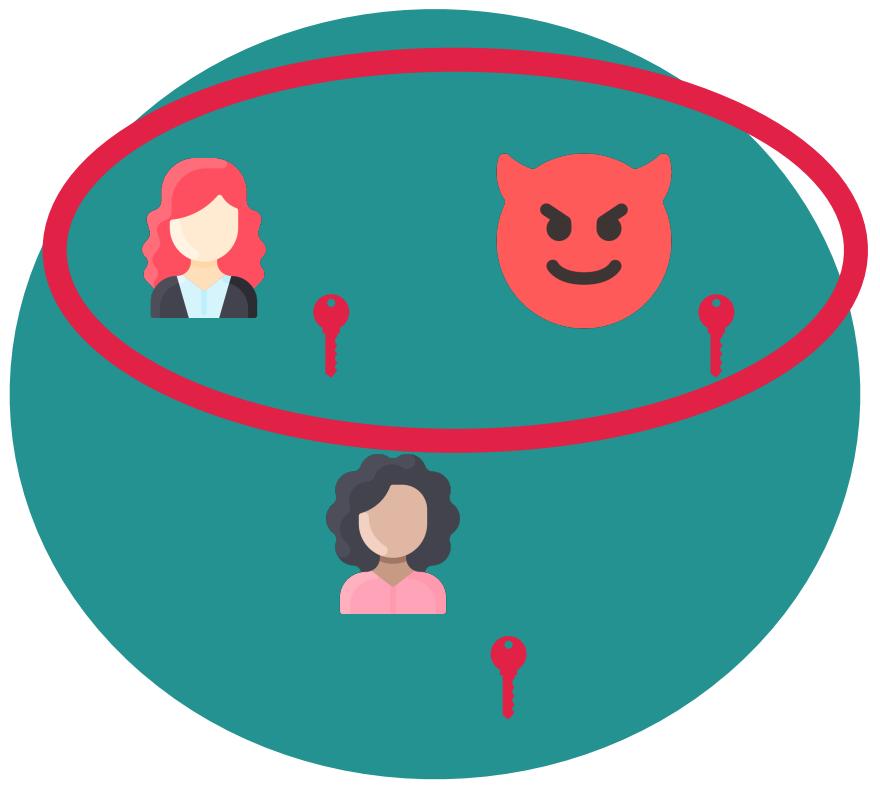


(2,3) Example

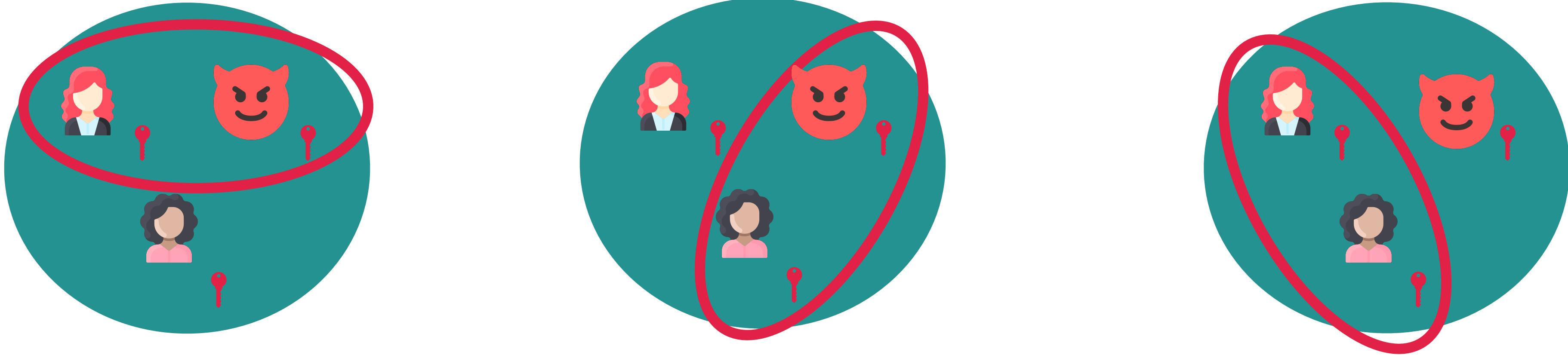
ROAST: Making FROST Robust



ROAST: Making FROST Robust

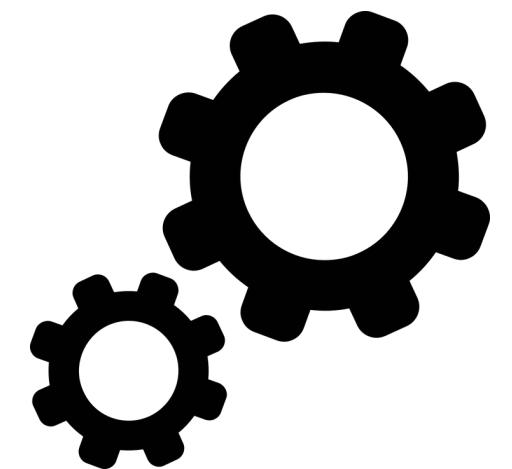


ROAST: Making FROST Robust



- ROAST is a wrapper picks subsets in a clever way
- At most $n - t + 1$ FROST runs necessary
- Resulting protocol is robust and asynchronous (no timeouts)

Standardization and Deployment

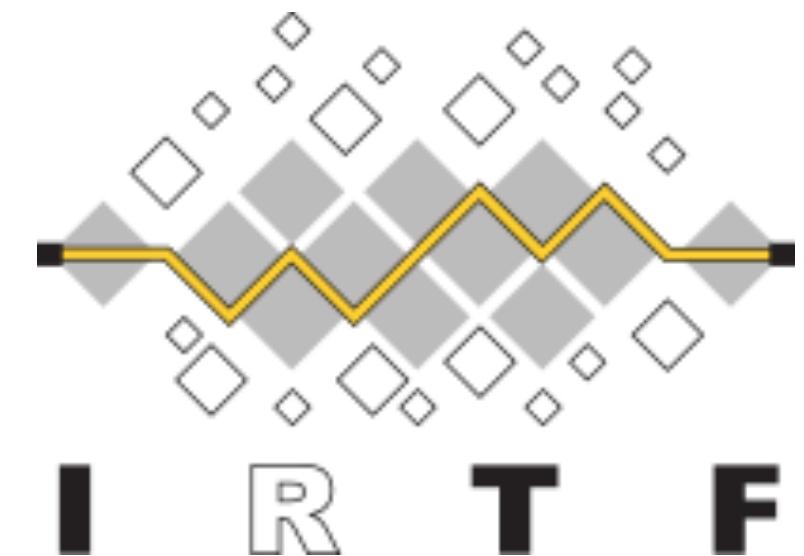


Standardization

CFRG
Internet-Draft
Intended status: Informational
Expires: 28 July 2023

D. Connolly
Zcash Foundation
C. Komlo
University of Waterloo, Zcash Foundation
I. Goldberg
University of Waterloo
C. A. Wood
Cloudflare
24 January 2023

Two-Round Threshold Schnorr Signatures with FROST
[draft-irtf-cfrg-frost-12](https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost-12)



<https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost/>

Standardization



ZIP: 312

Title: FROST for Spend Authorization Signatures

Owners: Conrado Gouvea <conrado@zfnd.org>

Chelsea Komlo <ckomlo@uwaterloo.ca>

Deirdre Connolly <deirdre@zfnd.org>

Status: Draft

Category: Wallet

Created: 2022-08-dd

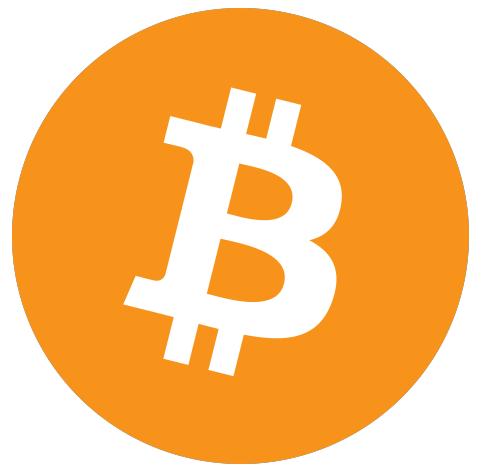
License: MIT

Discussions-To: <<https://github.com/zcash/zips/issues/382>>

Pull-Request: <<https://github.com/zcash/zips/pull/662>>

<https://github.com/ZcashFoundation/zips/blob/zip-frost/zip-0312.rst>

Standardization



BIP: 327

Title: MuSig2 for BIP340-compatible Multi-Signatures

Author: Jonas Nick <jonasd.nick@gmail.com>

Tim Ruffing <crypto@timruffing.de>

Elliott Jin <elliott.jin@gmail.com>

Status: Draft

License: BSD-3-Clause

Type: Informational

Created: 2022-03-22

<https://github.com/bitcoin/bips/blob/master/bip-0327.mediawiki>

Standardization



NISTIR 8214C (Draft)

NIST First Call for Multi-Party Threshold Schemes

Date Published: January 25, 2023

Comments Due: April 10, 2023

Email Comments to: nistir-8214C-comments@nist.gov

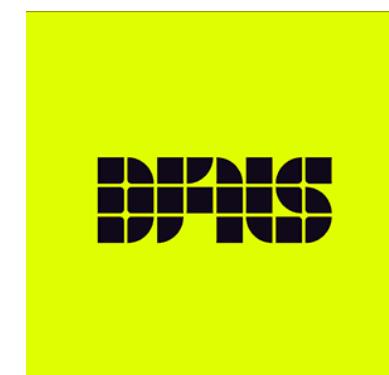
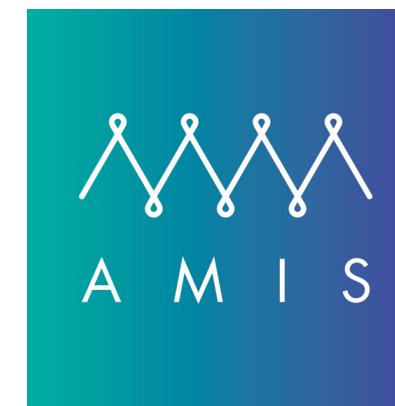
Author(s)

Luís T. A. N. Brandão (Strativia), Rene Peralta (NIST)

<https://csrc.nist.gov/publications/detail/nistir/8214c/draft>

FROST and MuSig2 in Practice, Today

FROST



serai-dex/**serai**



Contributors 9 Used by 2 Stars 94 Forks 18

[jesseposner/FROST-BIP340](#)



BIP340 compatible implementation of Flexible Round-Optimized Schnorr Threshold Signatures (FROST). This work is made possible with the support of Brink.

Contributors 2 Issues 2 Stars 20 Forks 3

MuSig2



[BlockstreamResearch/secp256k1-zkp](#)

[LLFourn/secp256kfun](#)



bitcoin/bips

#1372 Add BIP MuSig2



[input-output-hk/musig2](#)

BlockstreamResearch/secp256k1-zkp

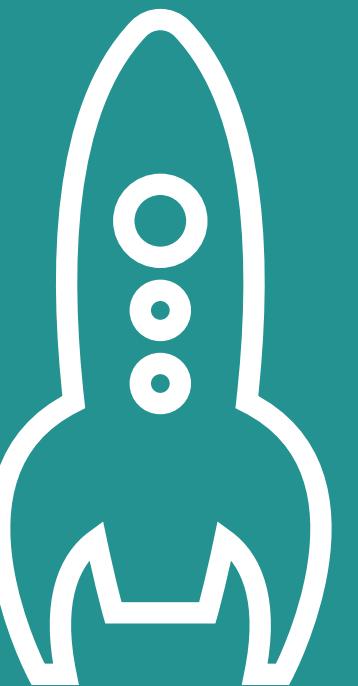
#223 musig: Update to BIP v1.0.0-rc.4 (Check pubnonce in NonceGe...)



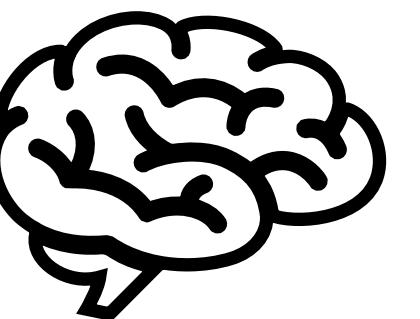
muun

Self-custodial wallet for bitcoin and lightning.

From Practice to Theory: What open problems exist?



Efficient Deterministic Signatures



(Single-Party) EdDSA Signature Scheme



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

(Single-Party) EdDSA Signature Scheme



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

To sign a message m :

$$r \leftarrow H(m, sk); \quad R \leftarrow g^r$$

$$c \leftarrow H(PK, m, R)$$

$$z \leftarrow r + csk$$

(Single-Party) EdDSA Signature Scheme



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

To sign a message m :

$$r \leftarrow H(m, sk); \quad R \leftarrow g^r$$

$$c \leftarrow H(PK, m, R)$$

$$z \leftarrow r + csk$$

(Single-Party) EdDSA Signature Scheme



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

To sign a message m :

$$r \leftarrow H(m, sk); \quad R \leftarrow g^r$$

$$c \leftarrow H(PK, m, R)$$

$$z \leftarrow r + csk$$

Helps prevent
issues arising from
bad randomness.

(Single-Party) EdDSA Signature Scheme



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



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

To sign a message m :

$$r \leftarrow H(m, sk); \quad R \leftarrow g^r$$

$$c \leftarrow H(PK, m, R)$$

$$z \leftarrow r + csk$$

Helps prevent
issues arising from
bad randomness.

(Single-Party) EdDSA Signature Scheme



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



To generate a key pair:

$$sk \xleftarrow{\$} \mathbb{F}; \quad PK \leftarrow g^{sk}$$

To sign a message m :

$$r \leftarrow H(m, sk); \quad R \leftarrow g^r$$

$$c \leftarrow H(PK, m, R)$$

$$z \leftarrow r + csk$$

To verify (PK, σ, m) :

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c \stackrel{?}{=} g^z$$

output accept/reject

Helps prevent
issues arising from
bad randomness.

Naively applying EdDSA-style
determinism to randomized
multi-party Schnorr schemes
is not secure!



Naively applying EdDSA-style
determinism to randomized
multi-party Schnorr schemes
is not secure!



Deterministic multi-party Schnorr schemes exist,
but are performance-intensive. [NRSW20, GKMN21]

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Corrupt
 PK_2

Honest

PK_1



m

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Corrupt
 PK_2

Honest

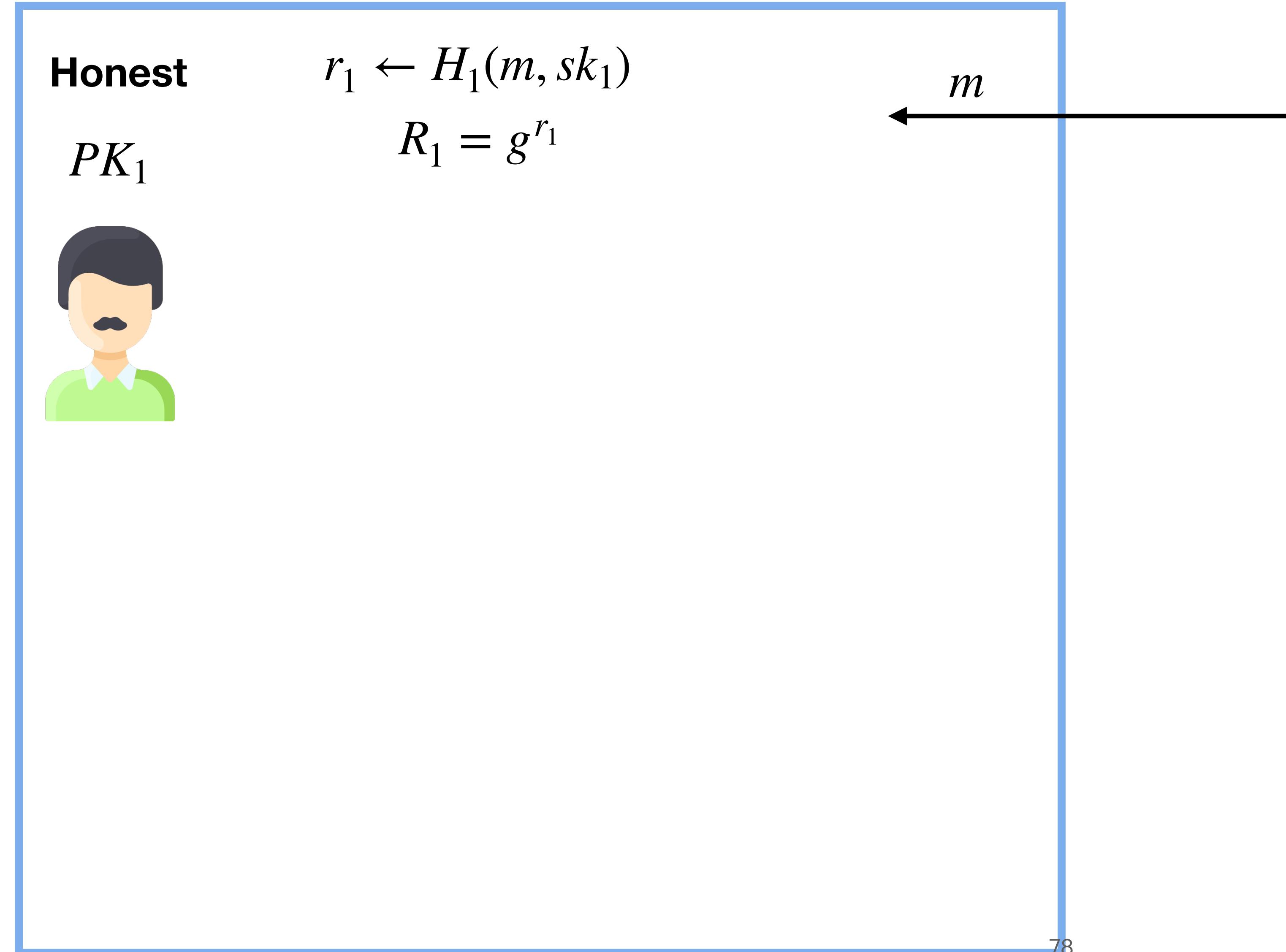
PK_1



$$r_1 \leftarrow H_1(m, sk_1)$$

m

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Corrupt
 PK_2

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Corrupt
 PK_2

Honest

PK_1



$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$\begin{array}{c} m \\ \xleftarrow{\hspace{1cm}} \\ R_1 \end{array}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Corrupt
 PK_2

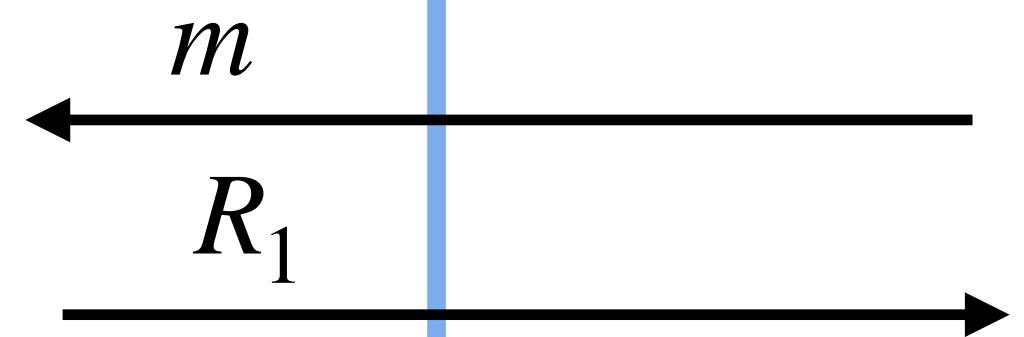
Honest

PK_1

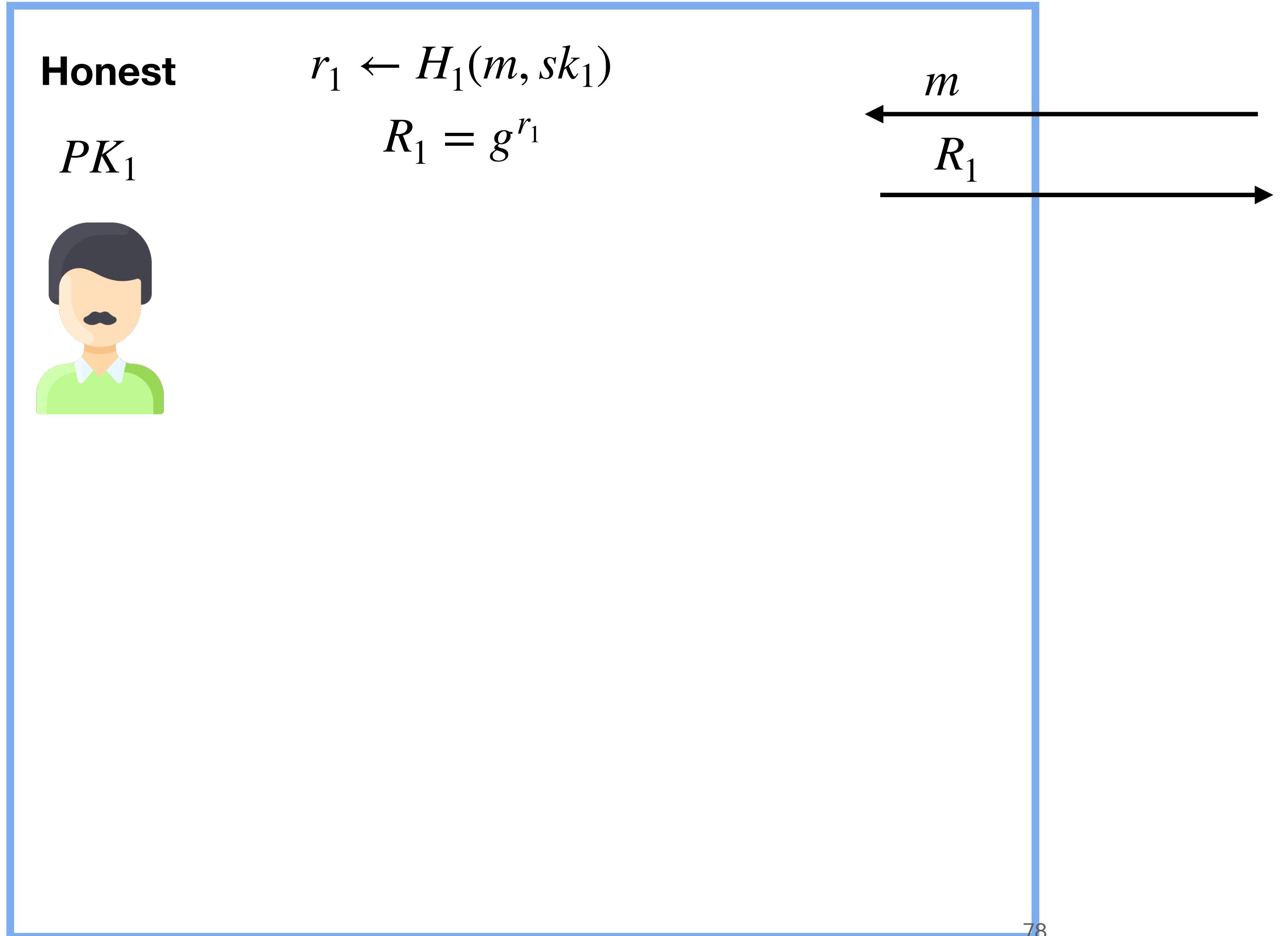


$$r_1 \leftarrow H_1(m, sk_1)$$

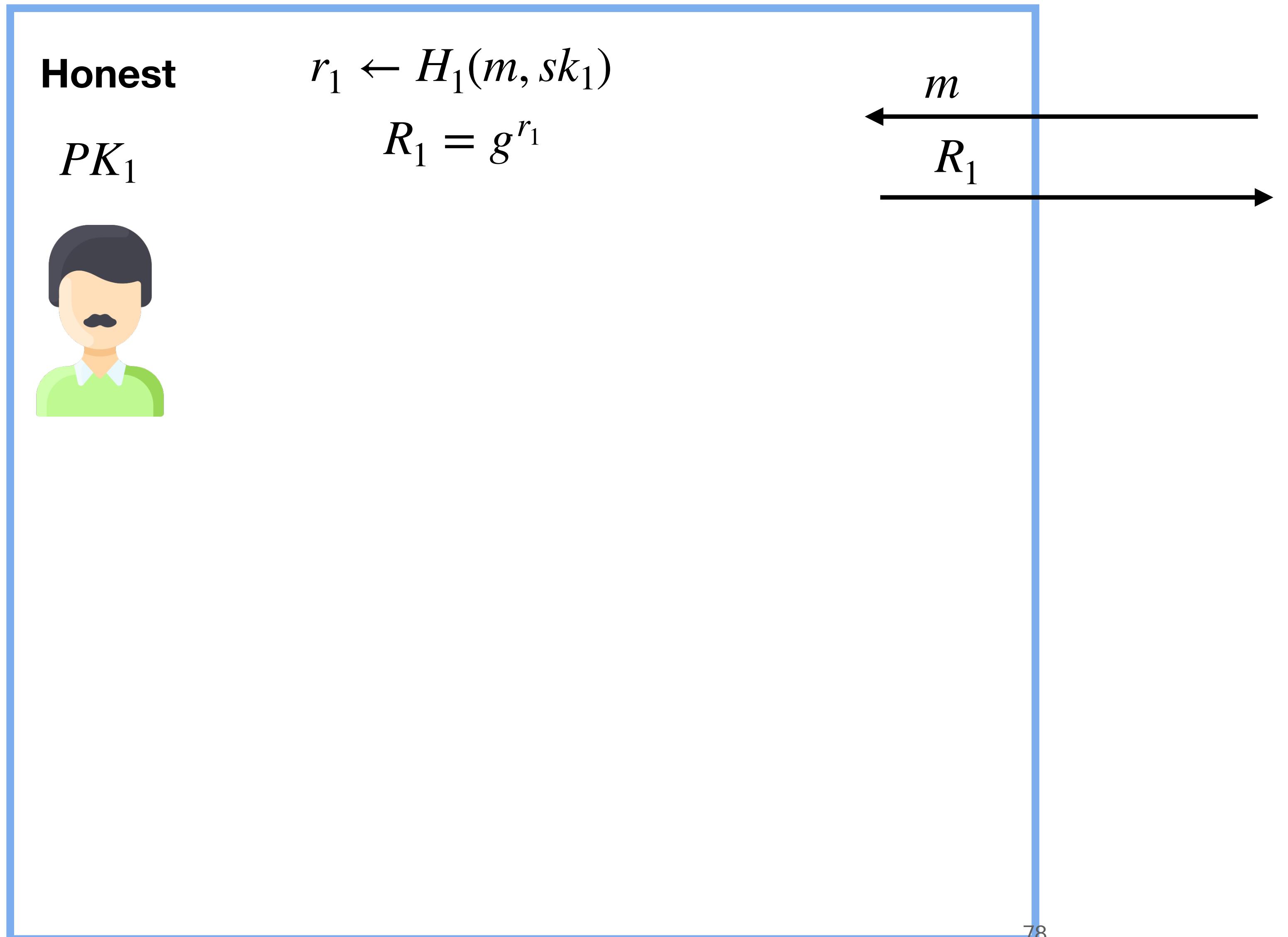
$$R_1 = g^{r_1}$$



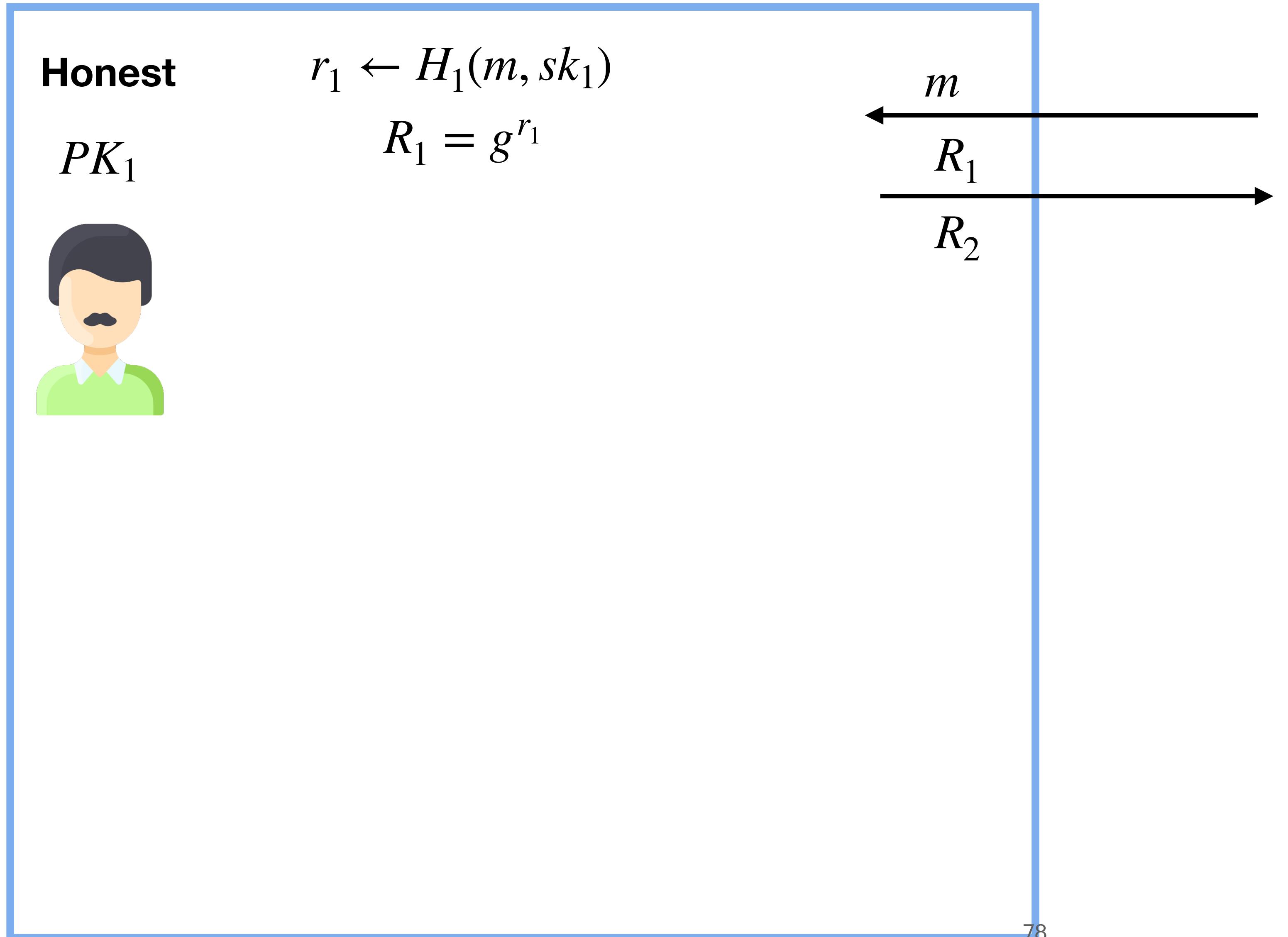
Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



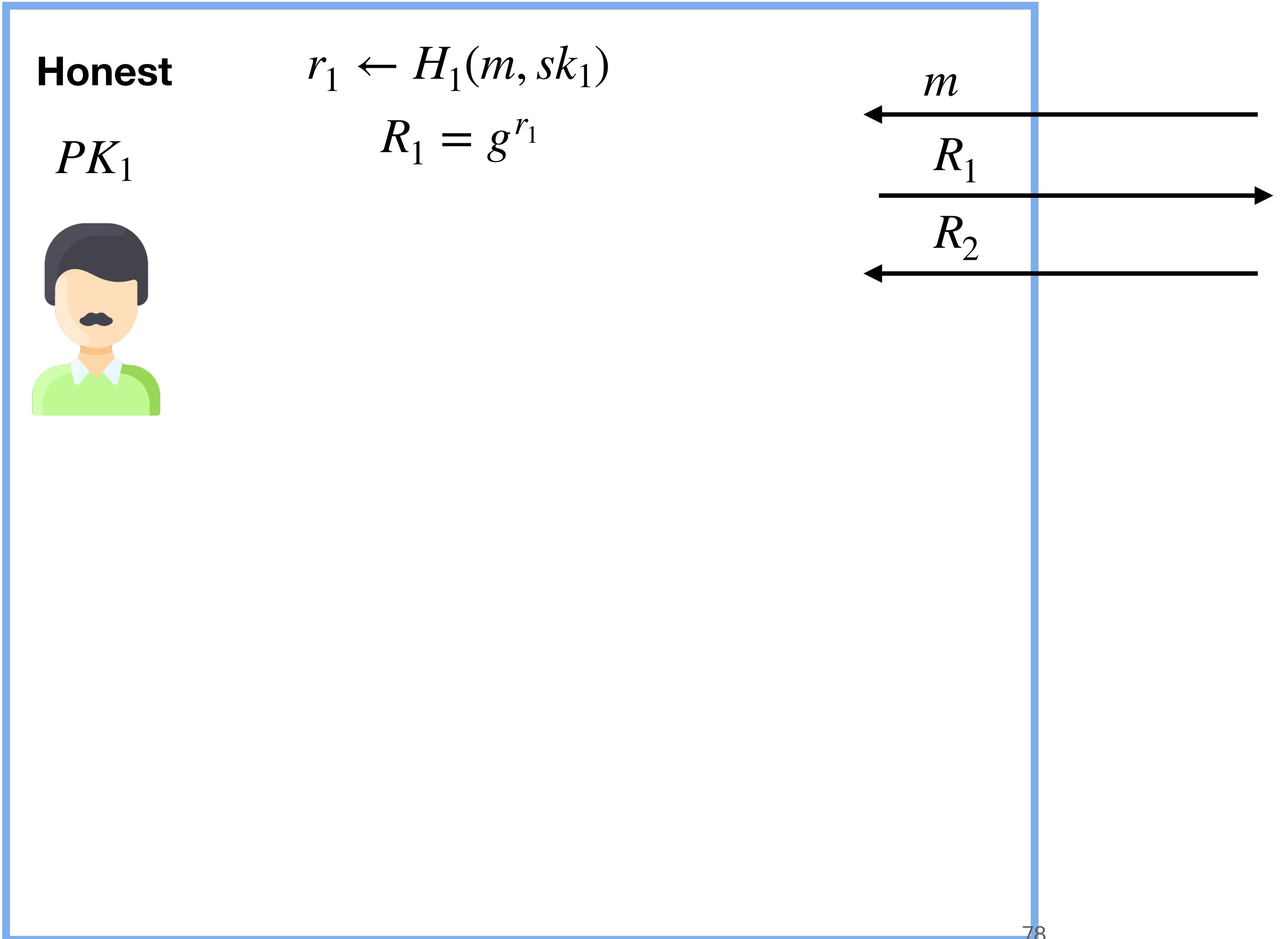
Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



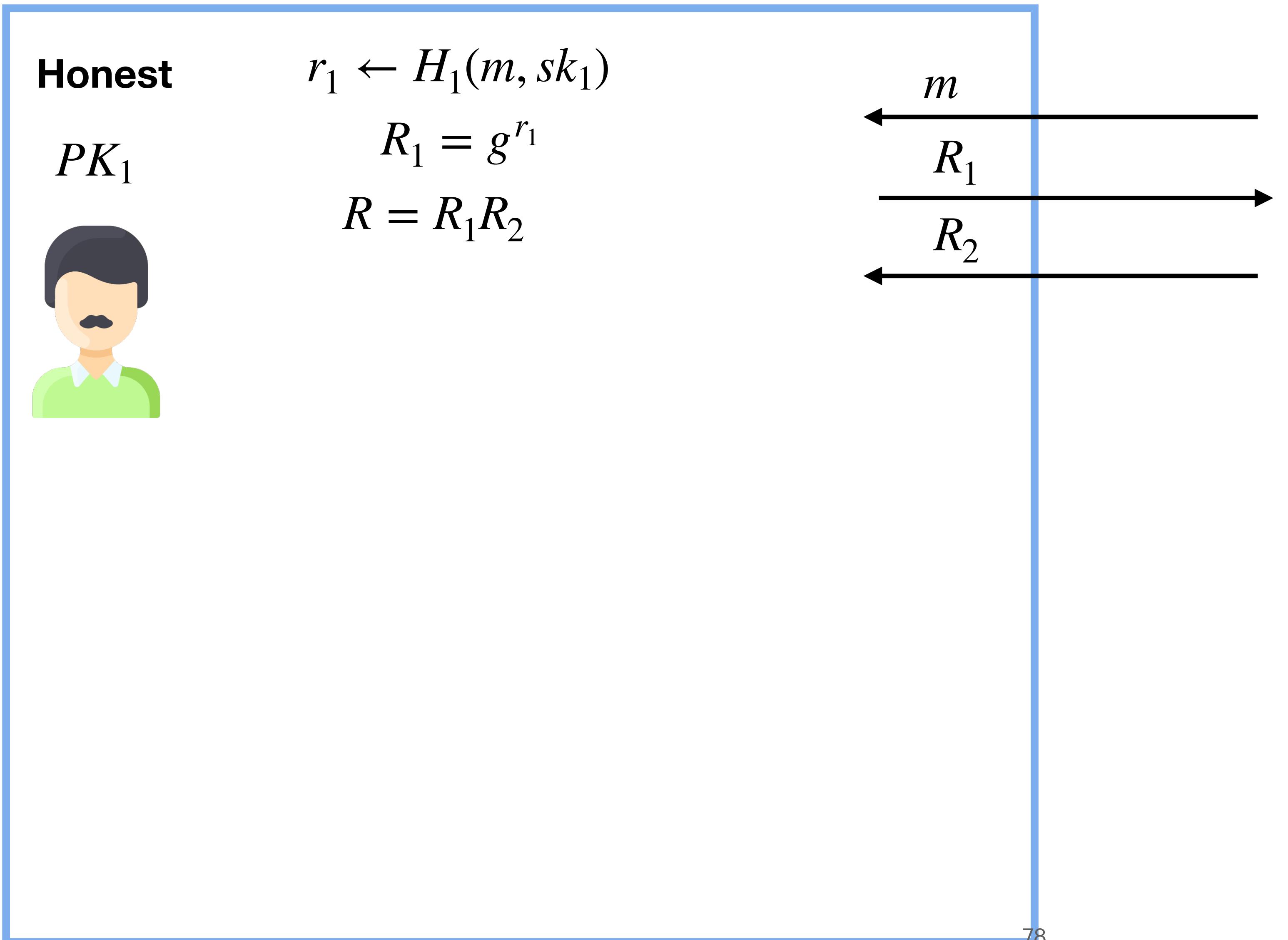
Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

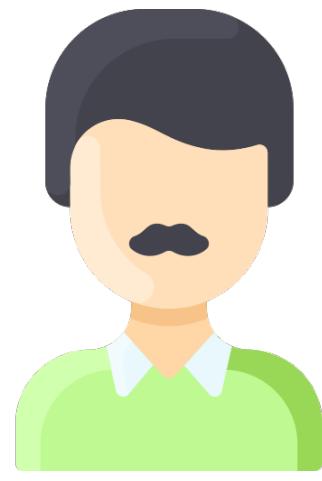


Corrupt
 PK_2

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1

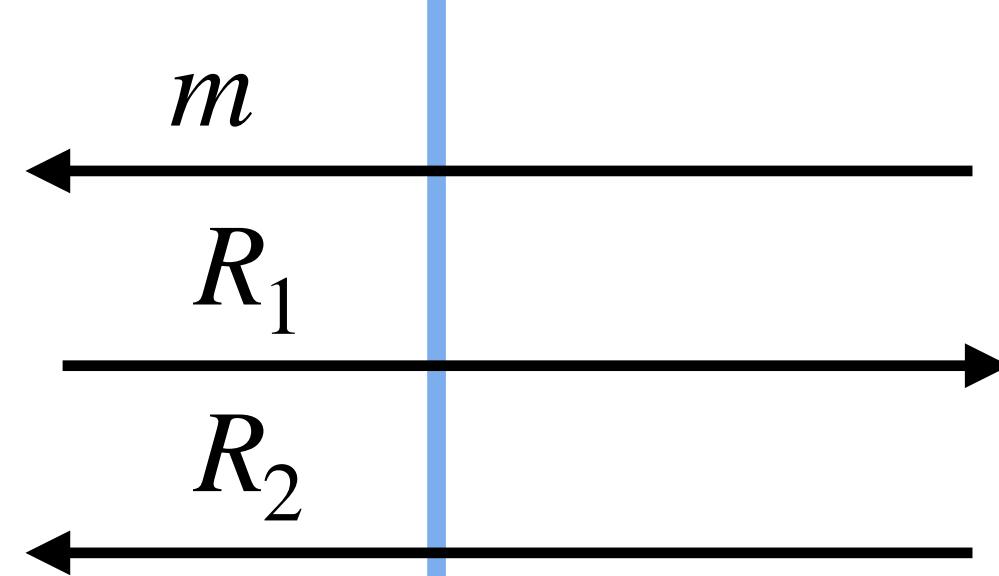


$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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



Corrupt

PK_2

$$\begin{aligned} r_2 &\leftarrow H_1(m, sk_2) \\ R_2 &= g^{r_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



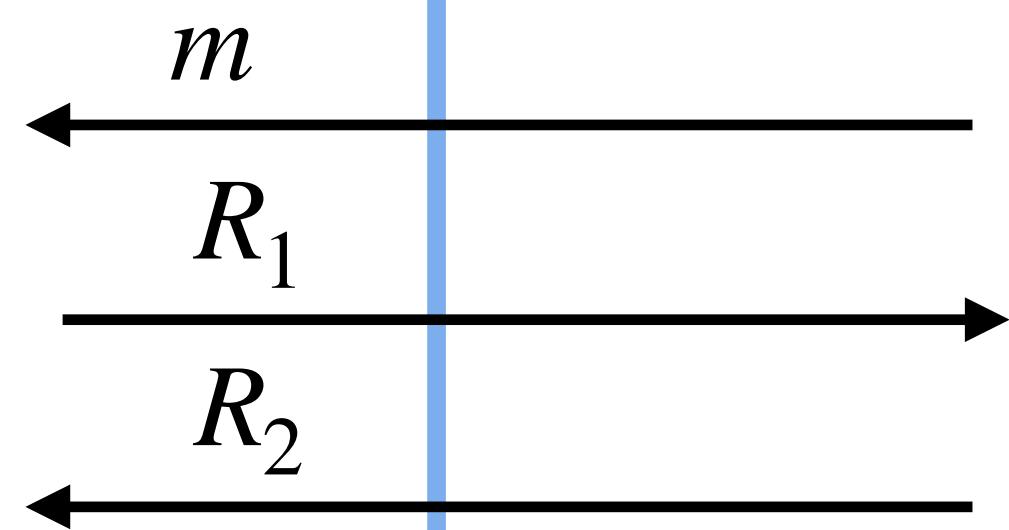
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

PK_2

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



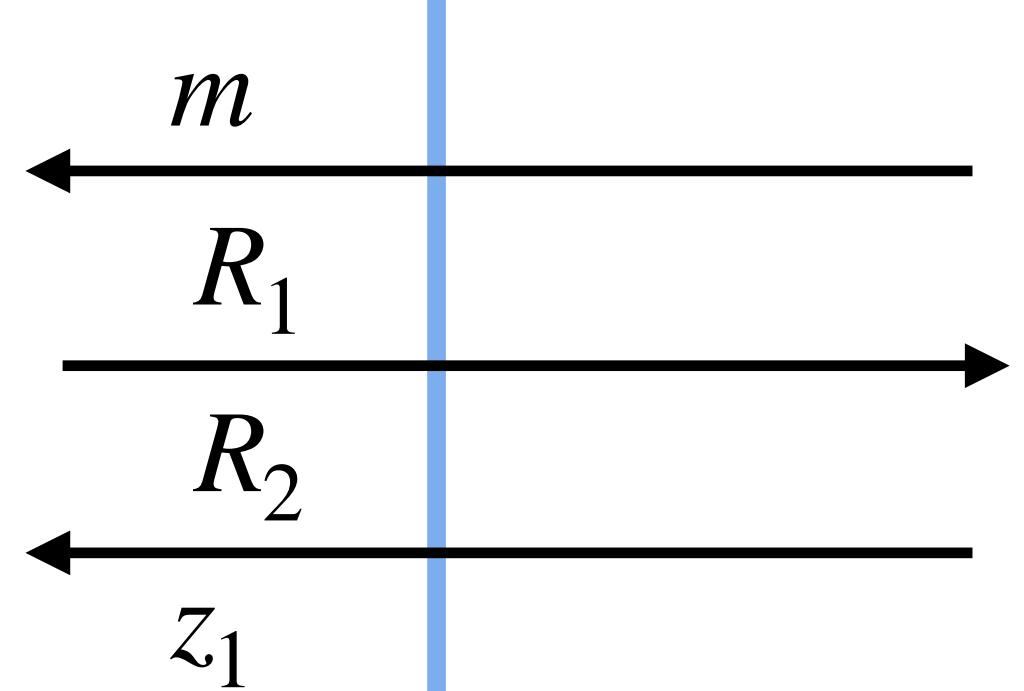
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

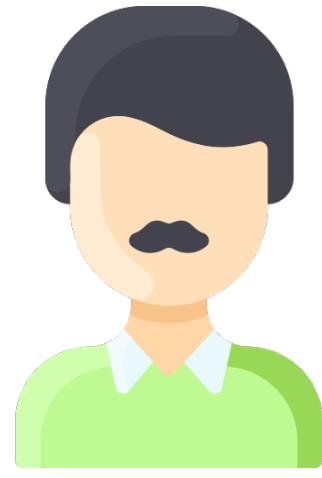
PK_2

$$\begin{aligned} r_2 &\leftarrow H_1(m, sk_2) \\ R_2 &= g^{r_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



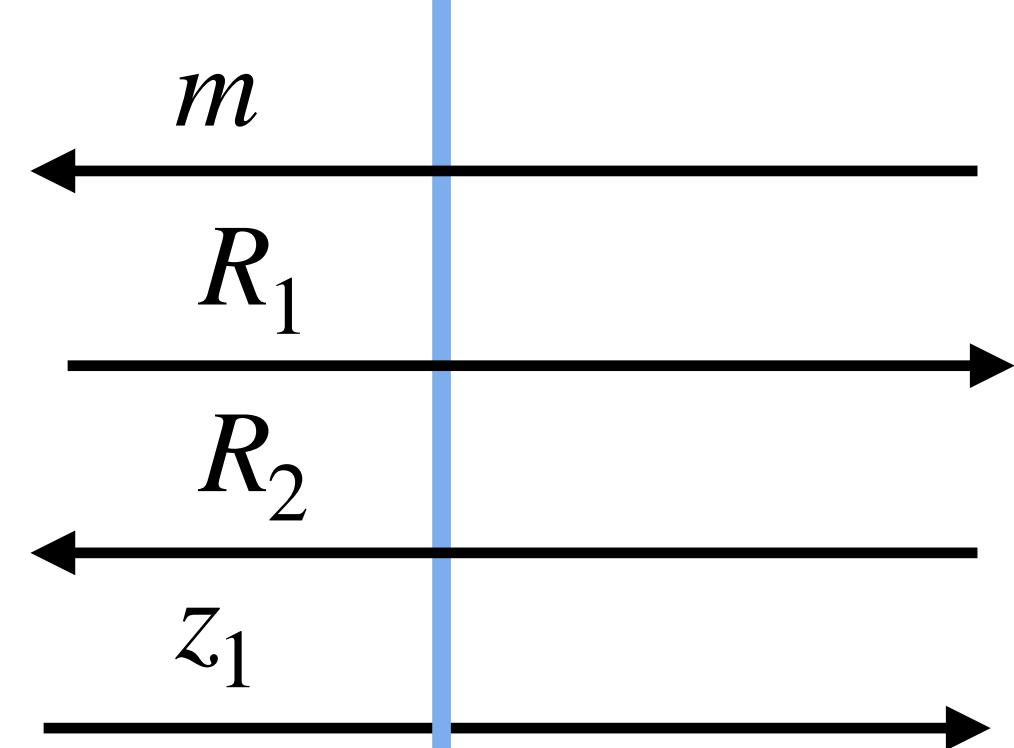
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

PK_2

$$\begin{aligned} r_2 &\leftarrow H_1(m, sk_2) \\ R_2 &= g^{r_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



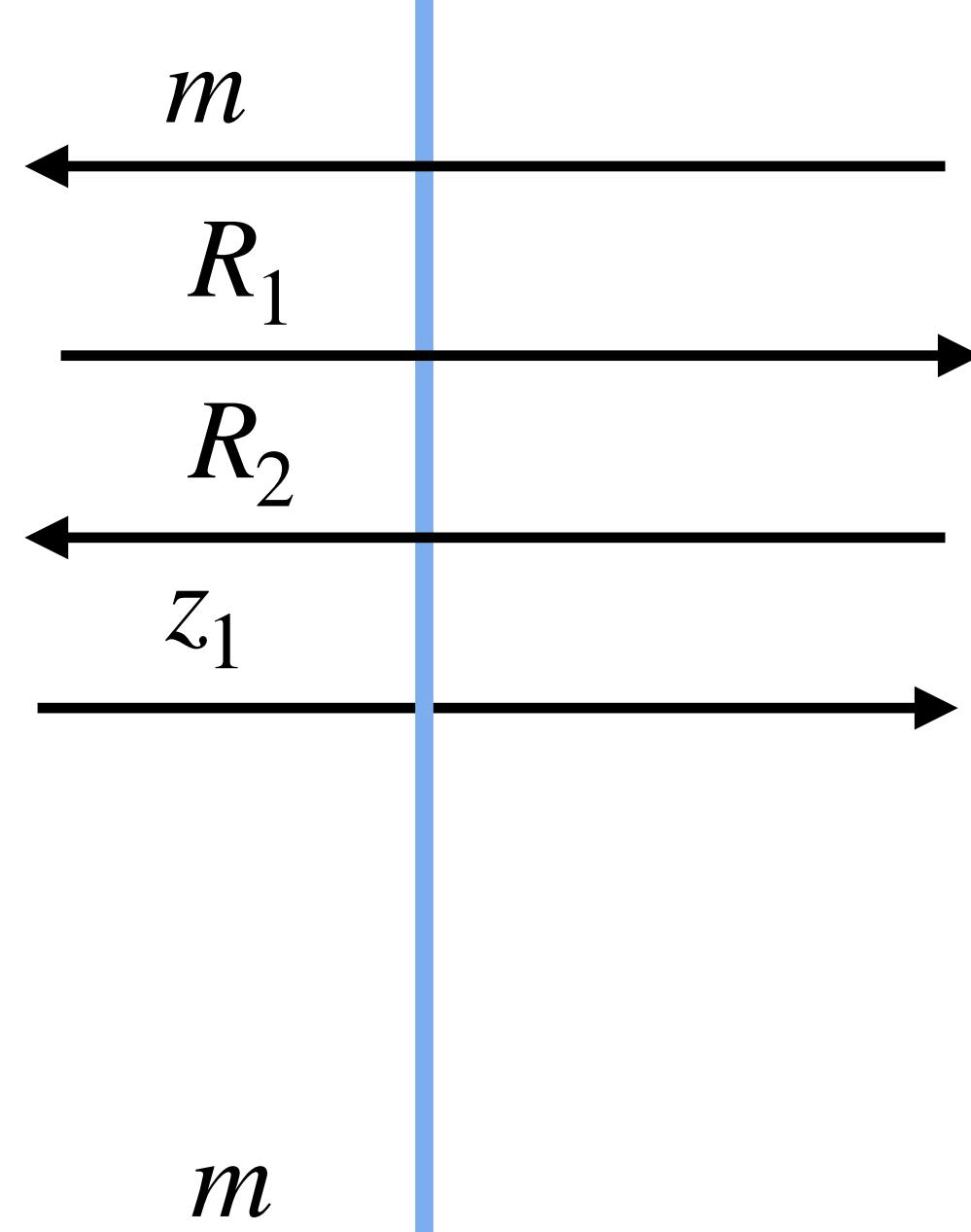
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



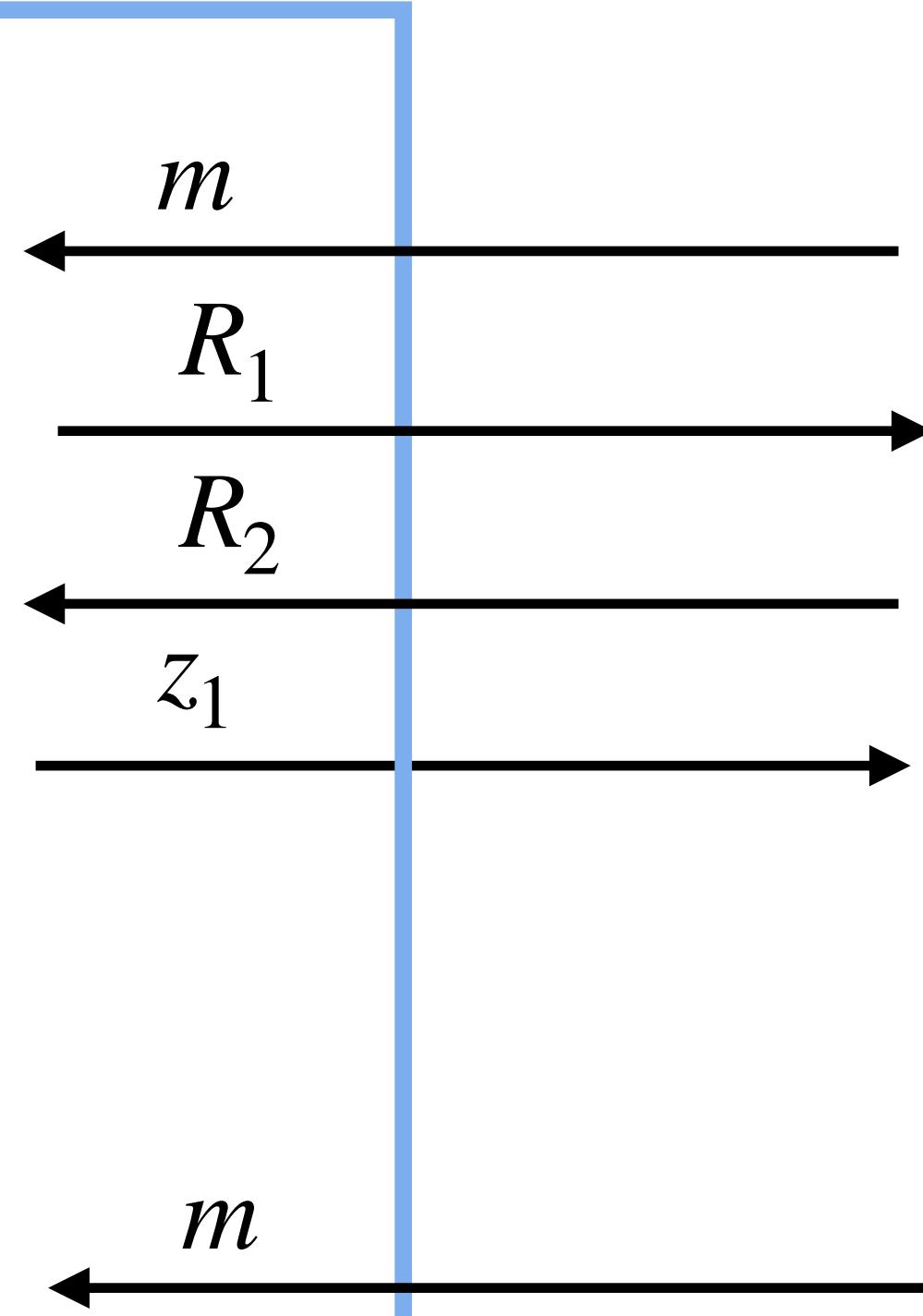
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

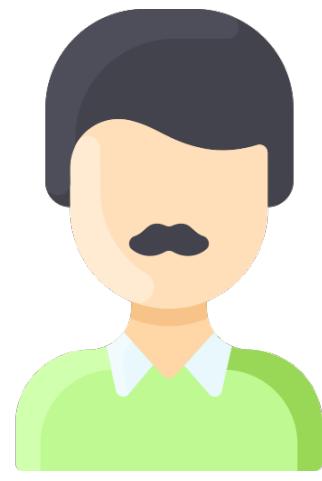
PK_2

$$\begin{aligned} r_2 &\leftarrow H_1(m, sk_2) \\ R_2 &= g^{r_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



$$r_1 \leftarrow H_1(m, sk_1)$$

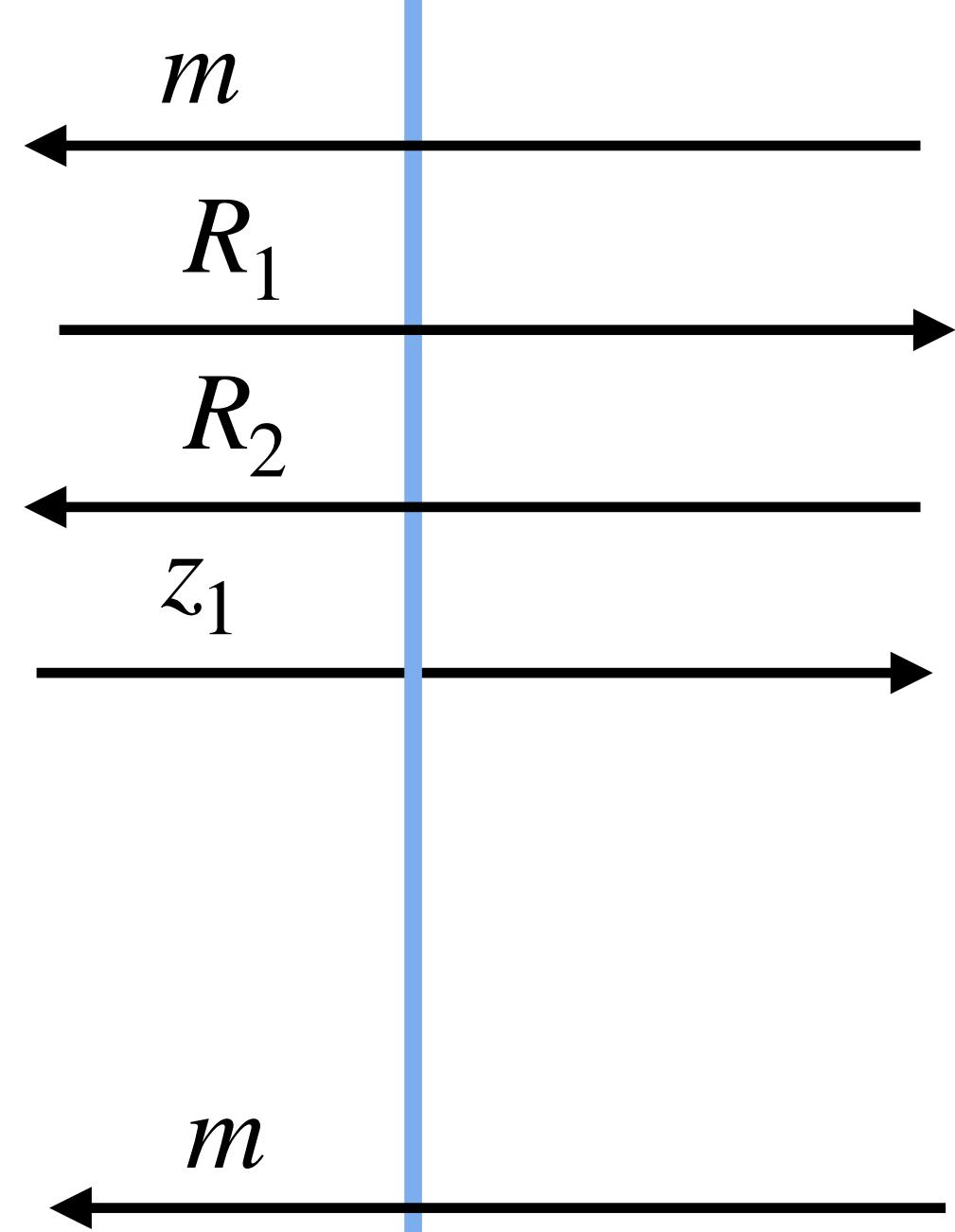
$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$

$$r_1 \leftarrow H_1(m, sk_1)$$



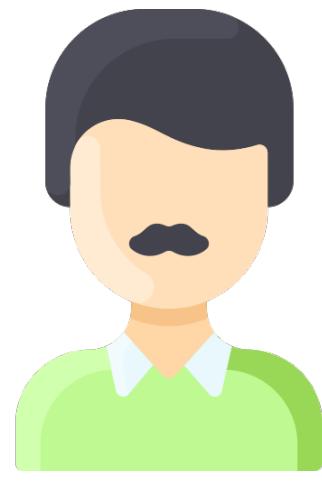
Corrupt

$$\begin{aligned} r_2 &\leftarrow H_1(m, sk_2) \\ R_2 &= g^{r_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



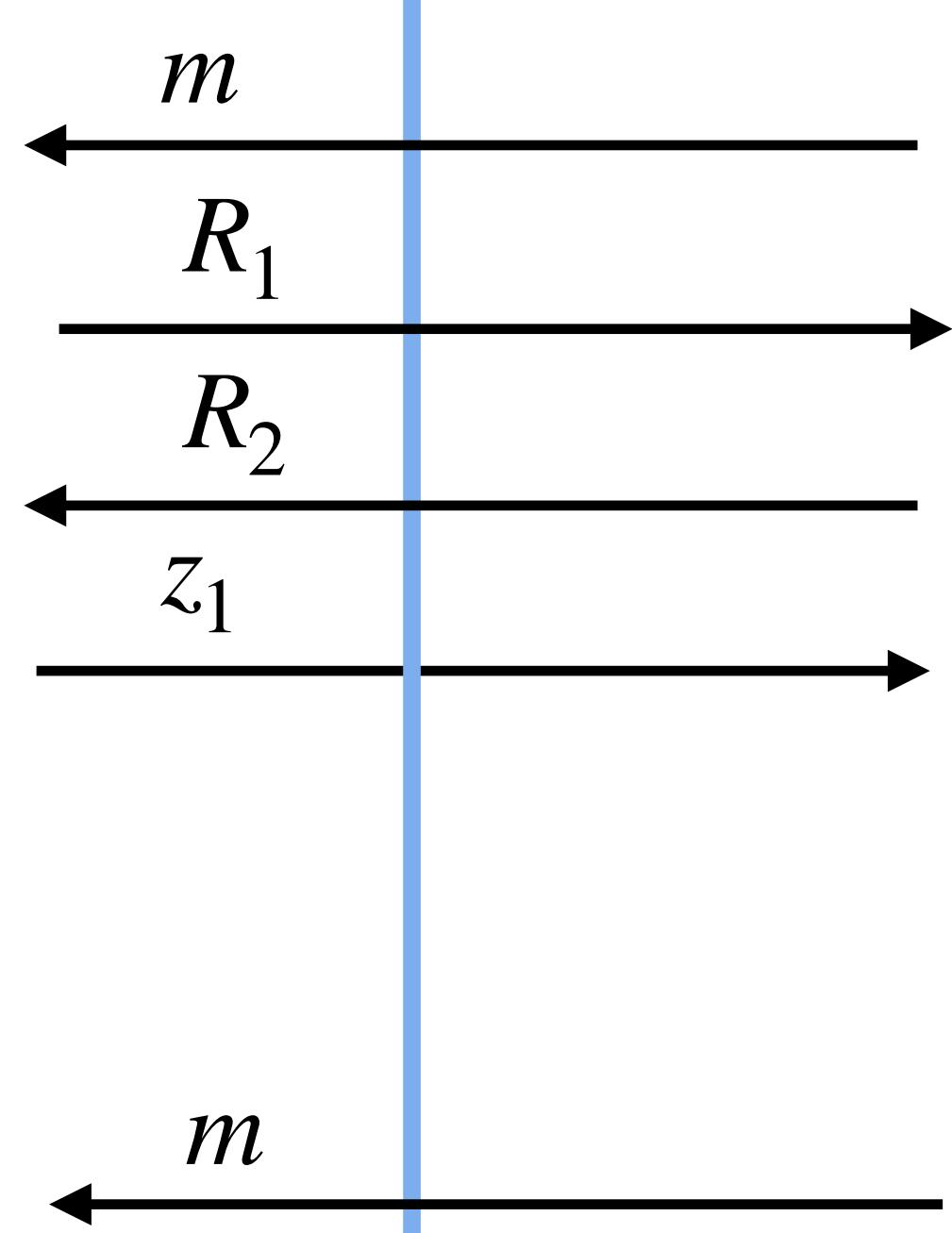
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



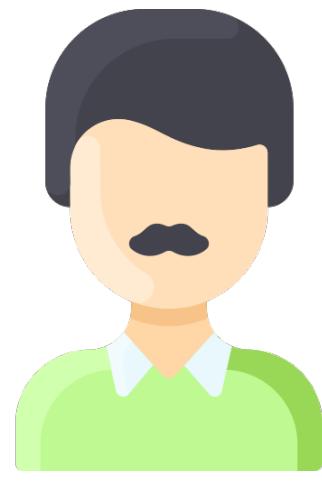
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



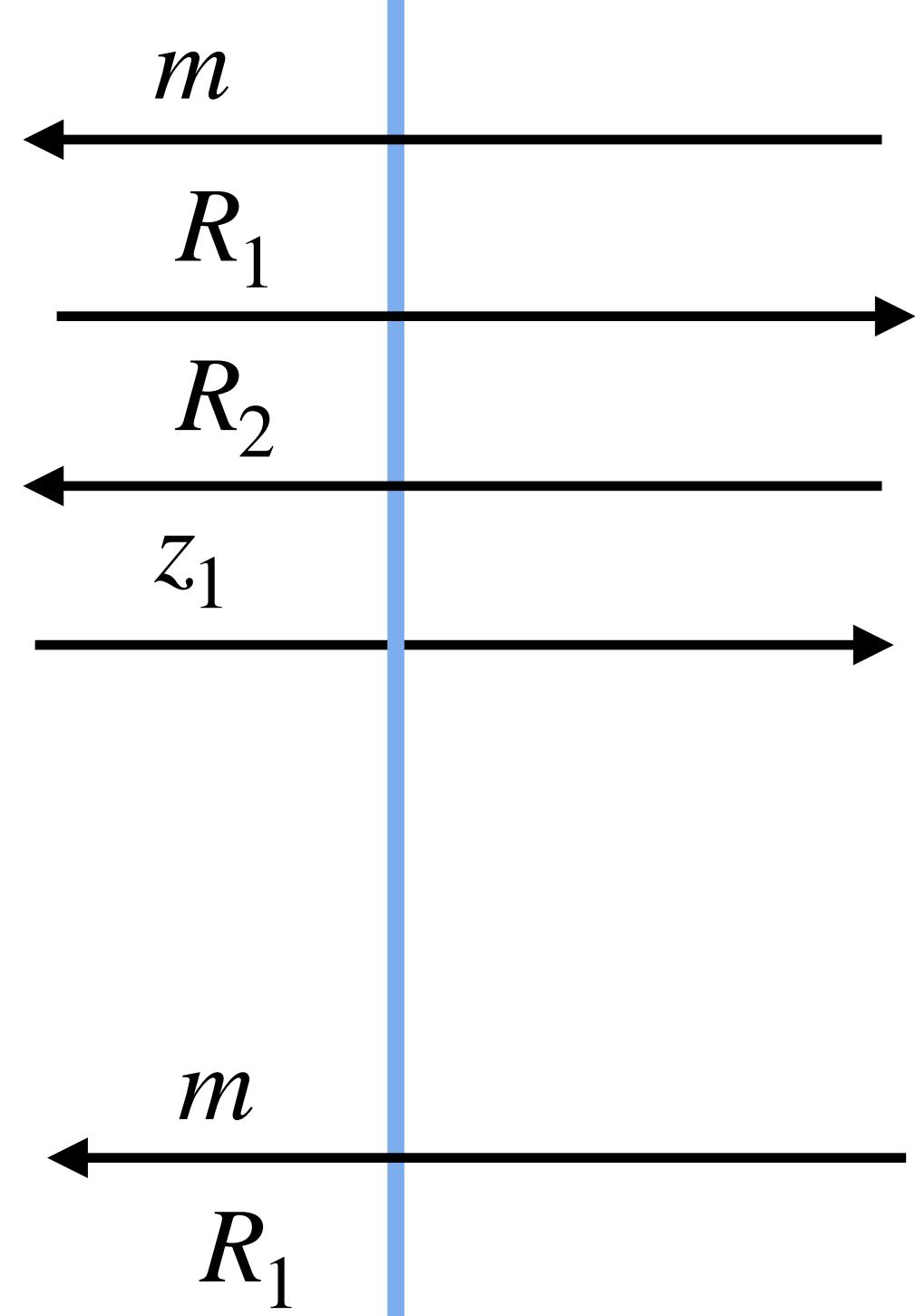
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

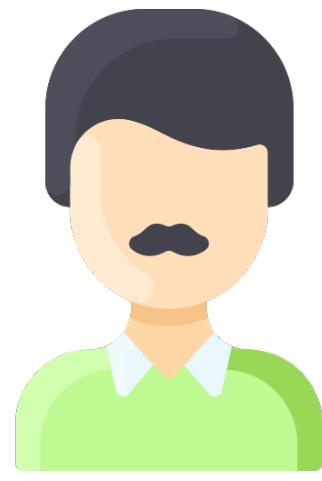
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



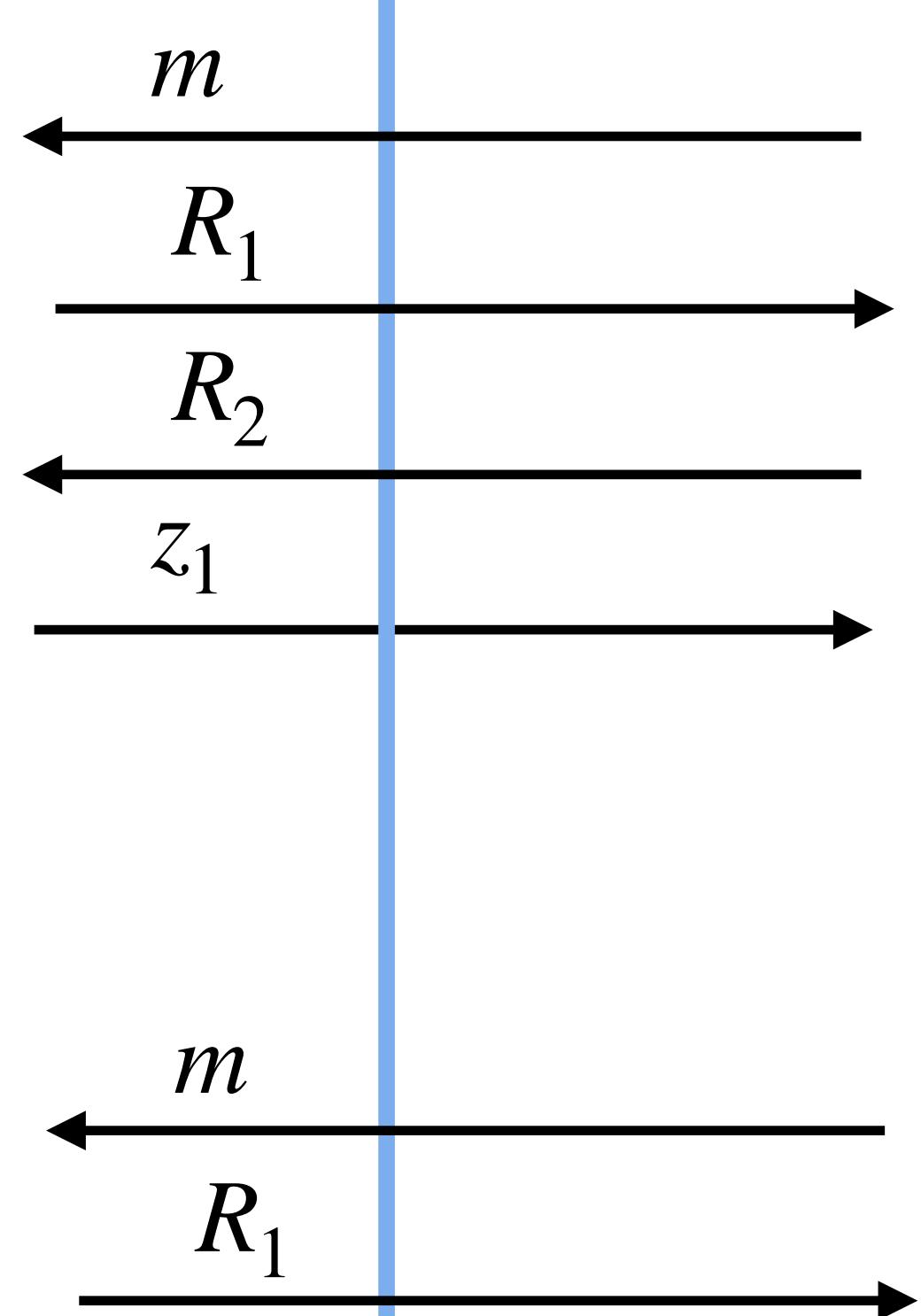
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

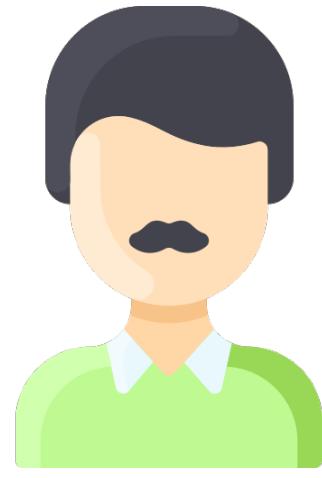
PK_2

$$\begin{aligned} r_2 &\leftarrow H_1(m, sk_2) \\ R_2 &= g^{r_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



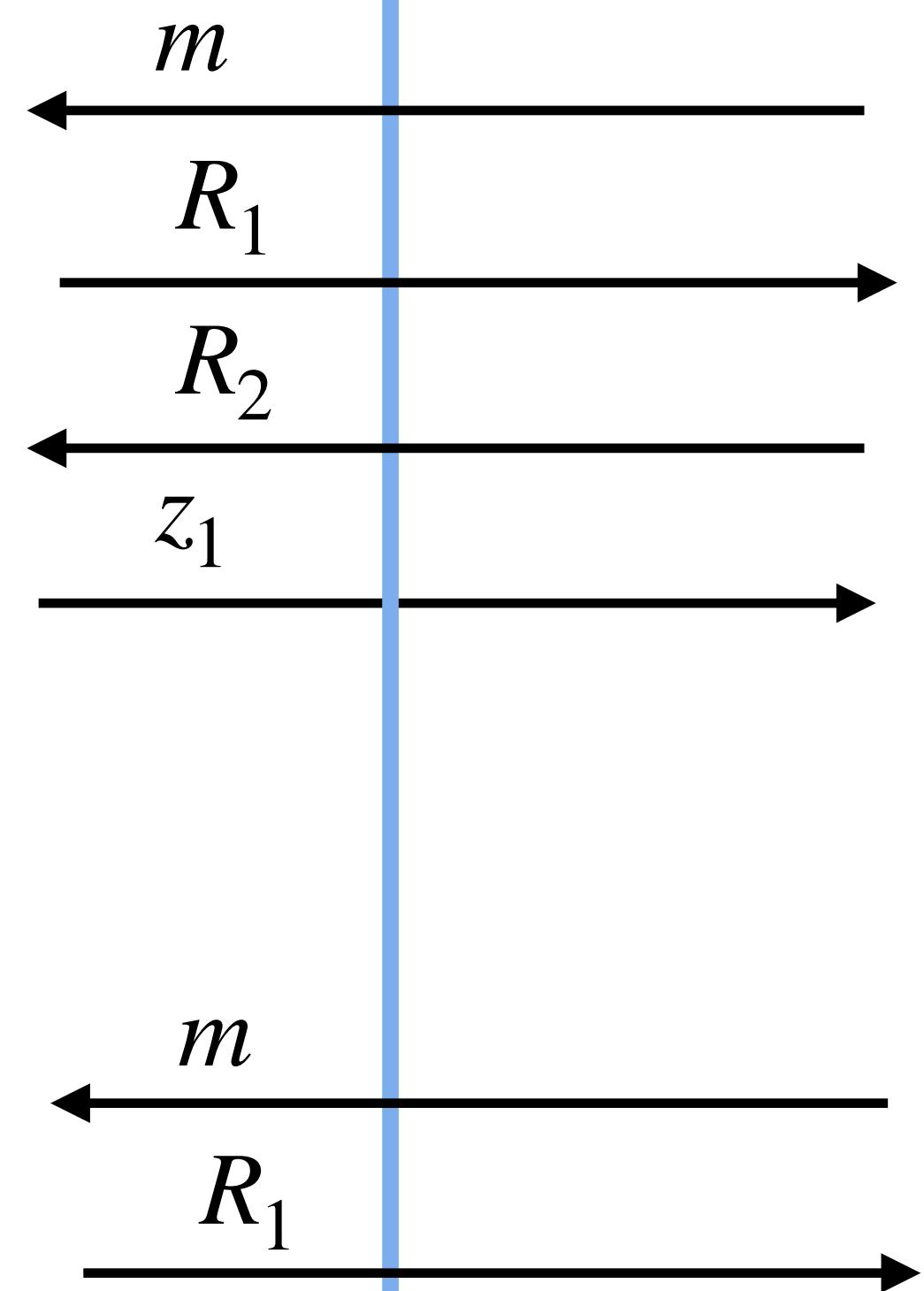
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$

$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



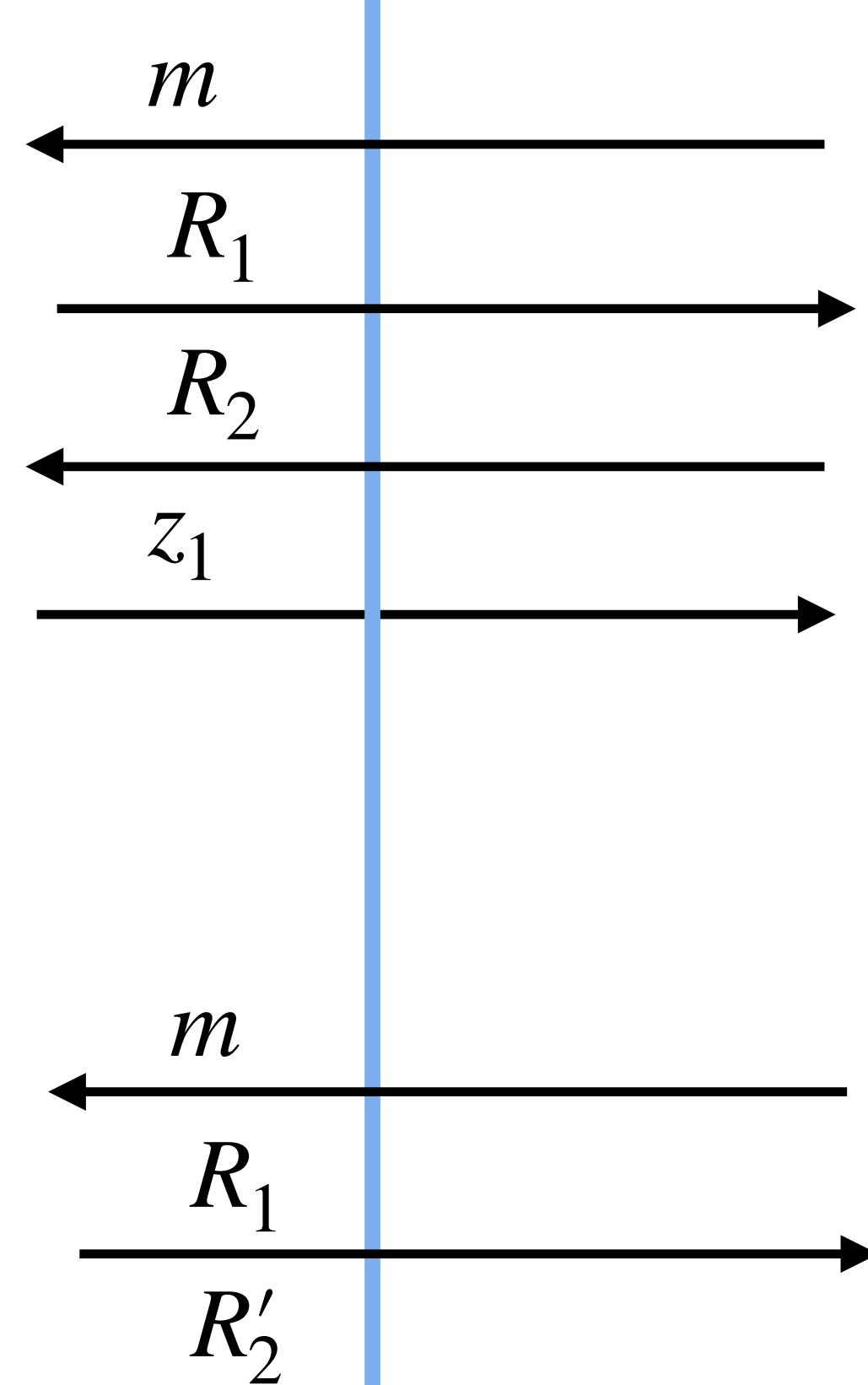
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

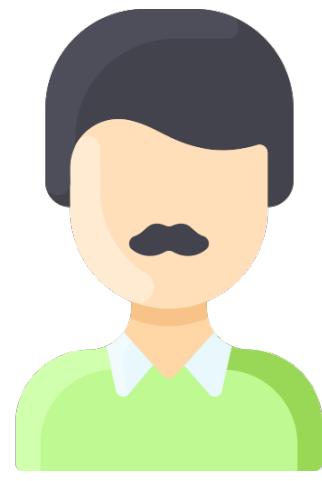
$$R_2 = g^{r_2}$$

$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



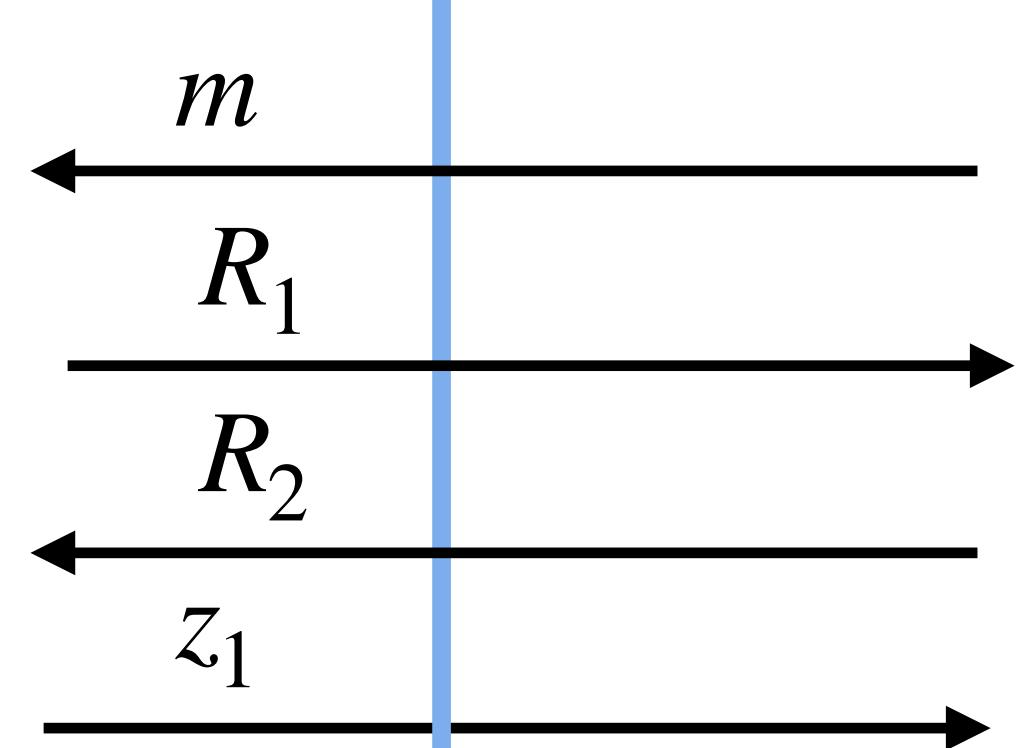
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

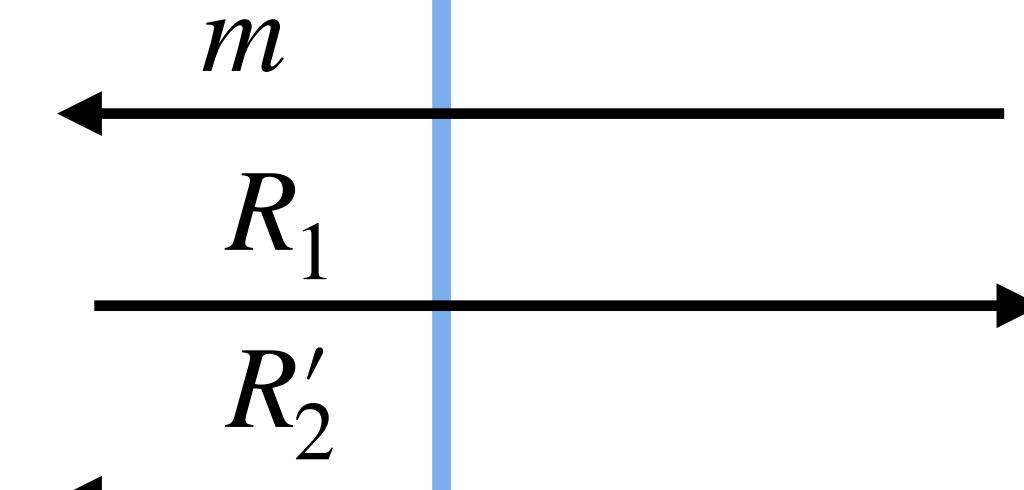
$$c = H(PK, R, m)$$

$$z_1 = r_1 + csk_1$$



$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

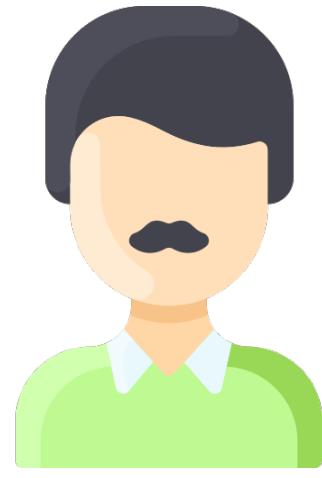
$$R_2 = g^{r_2}$$

$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



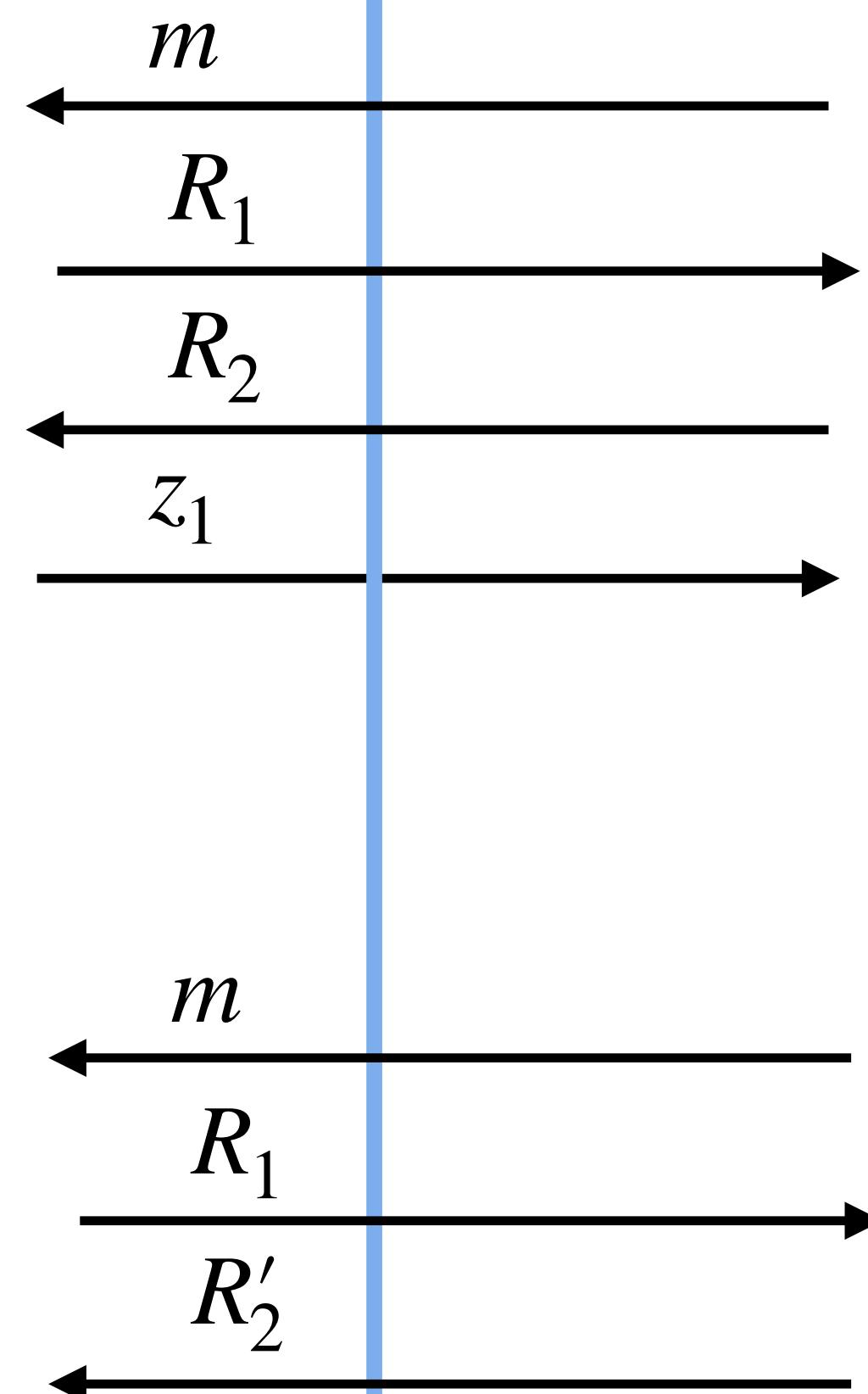
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$



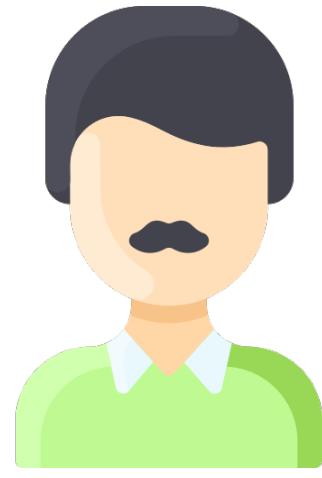
Corrupt

PK_2

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



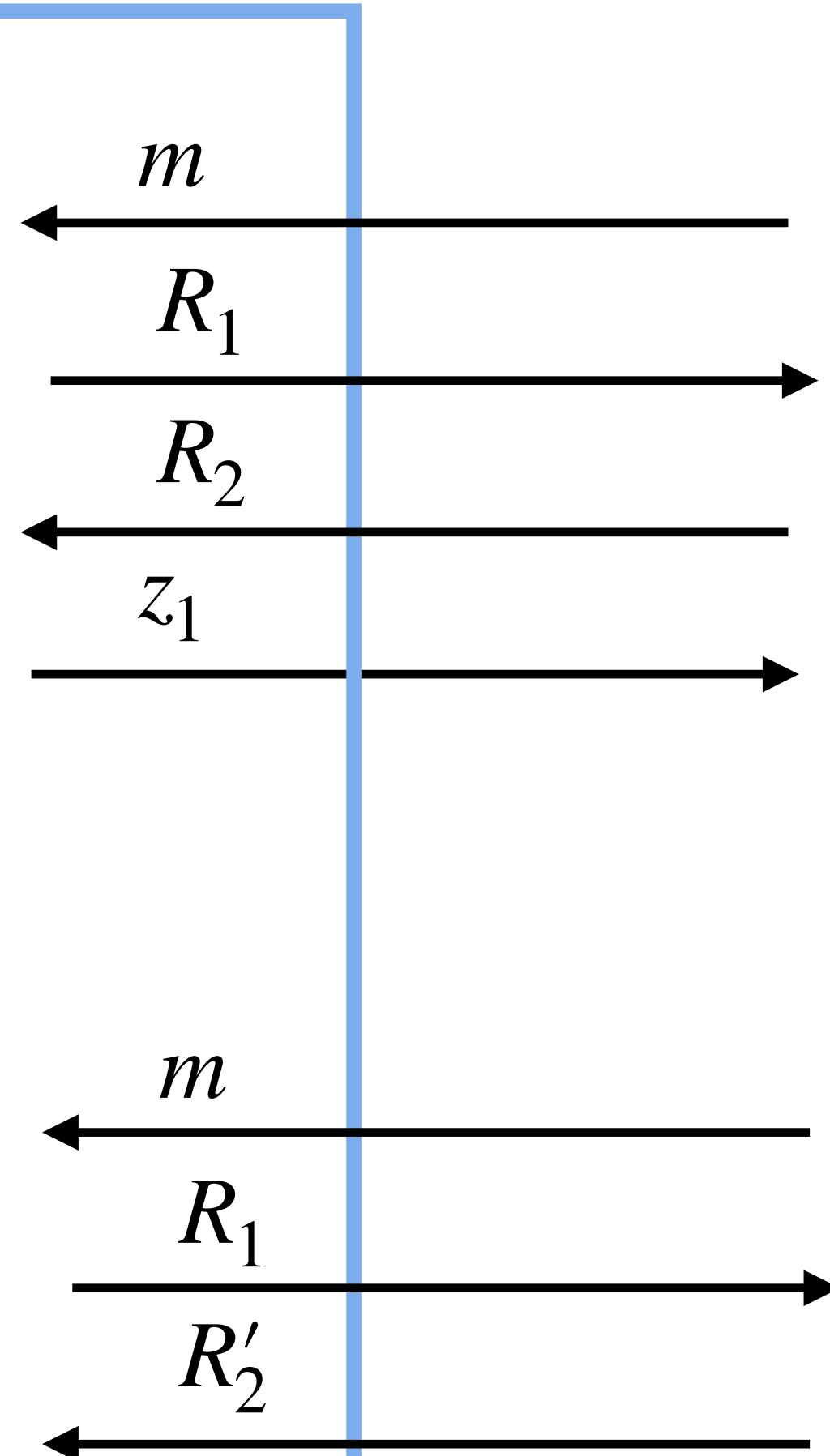
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

$$r'_2 \xleftarrow{\$} \mathbb{F}$$

$$R'_2 = g^{r'_2}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



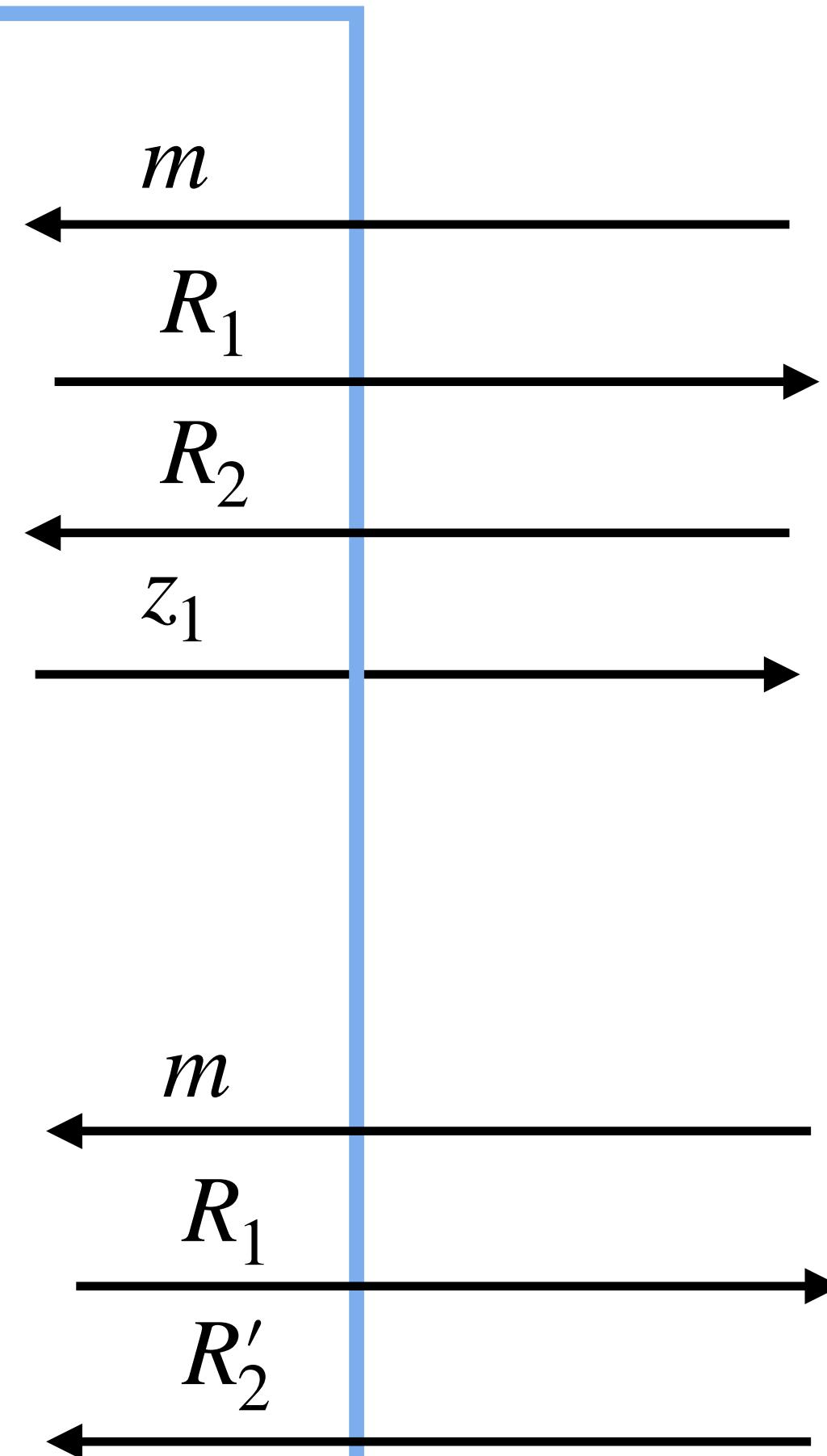
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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

$$z'_1 = r_1 + c'sk_1$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$

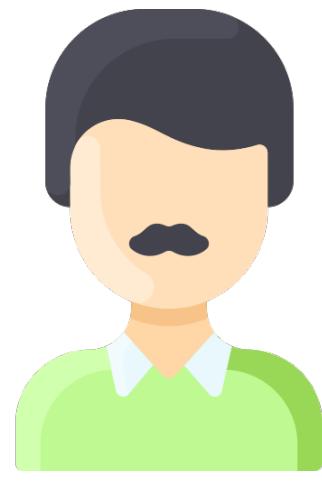
Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Corrupt
 PK_2

Honest

PK_1



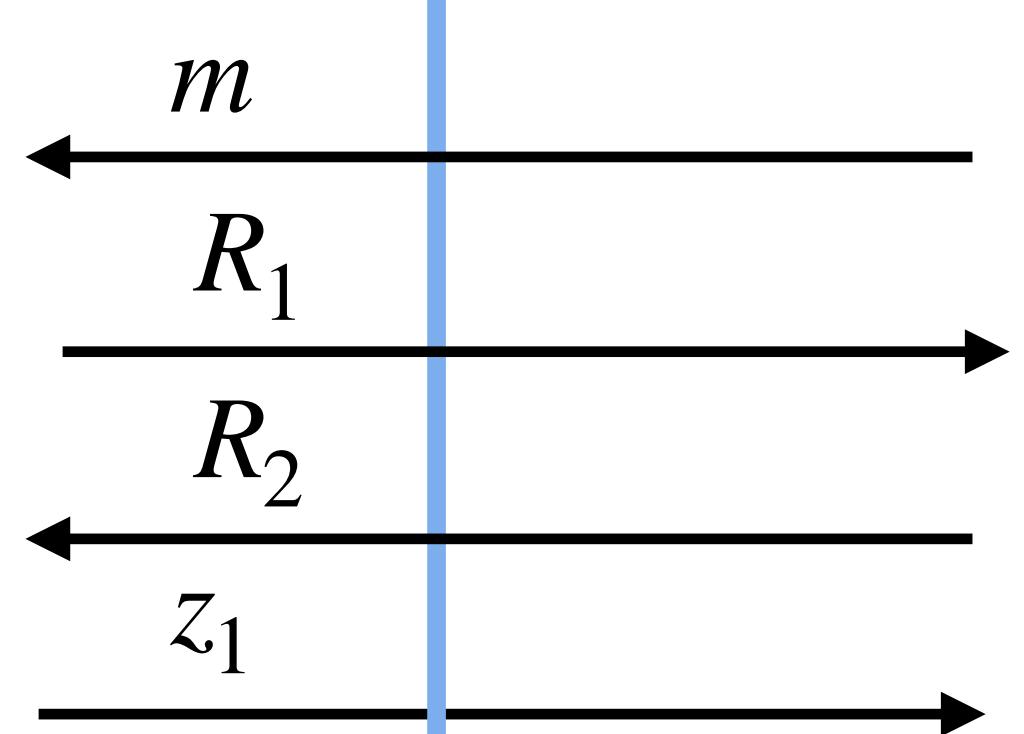
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



$$\begin{aligned} r_2 &\leftarrow H_1(m, sk_2) \\ R_2 &= g^{r_2} \end{aligned}$$

$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$

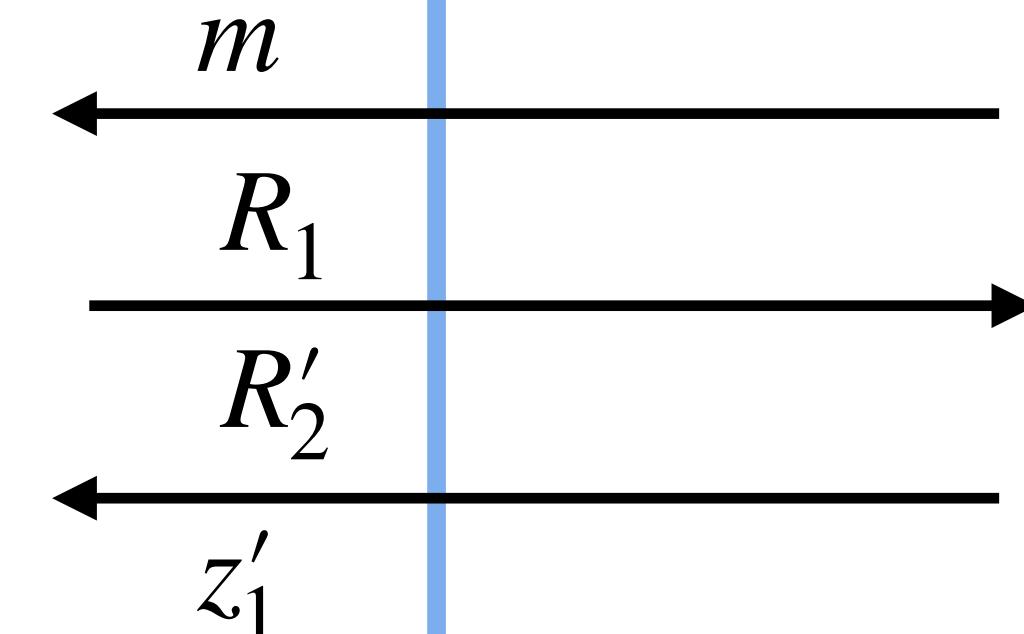
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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

$$z'_1 = r_1 + c'sk_1$$



Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



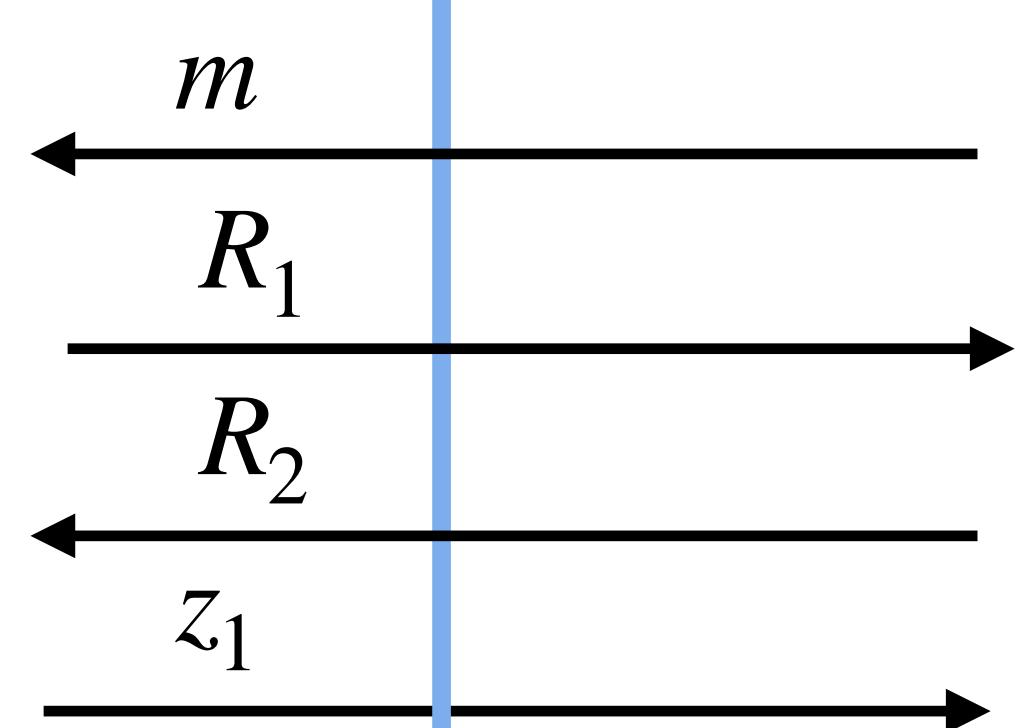
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



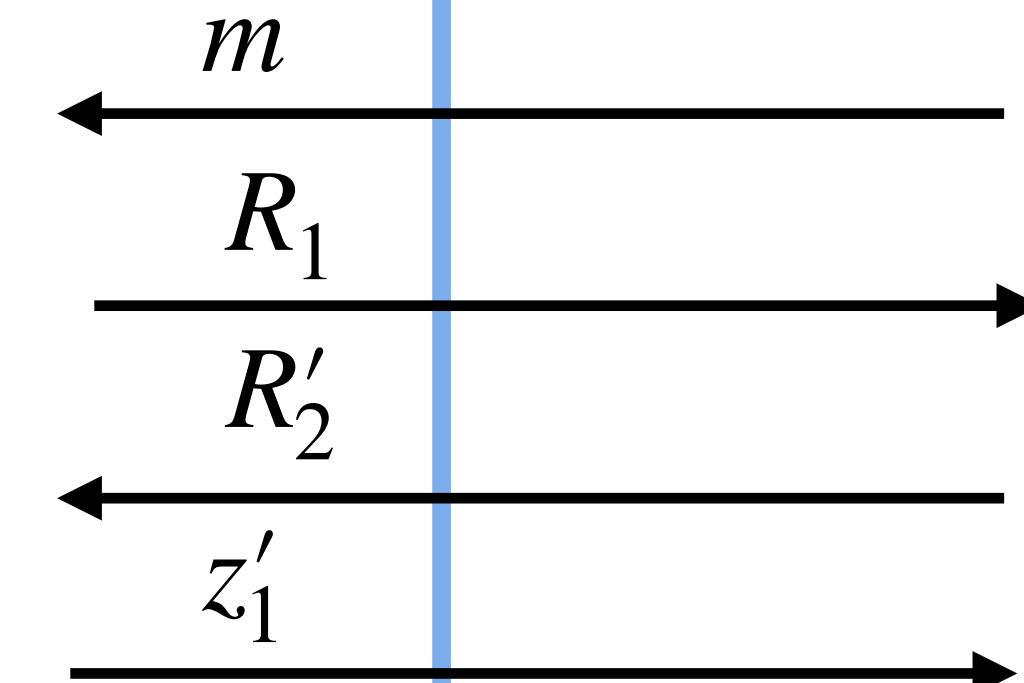
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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

$$z'_1 = r_1 + c'sk_1$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

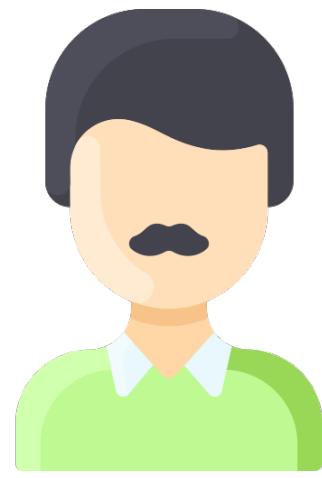
$$r'_2 \xleftarrow{\$} \mathbb{F}$$

$$R'_2 = g^{r'_2}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



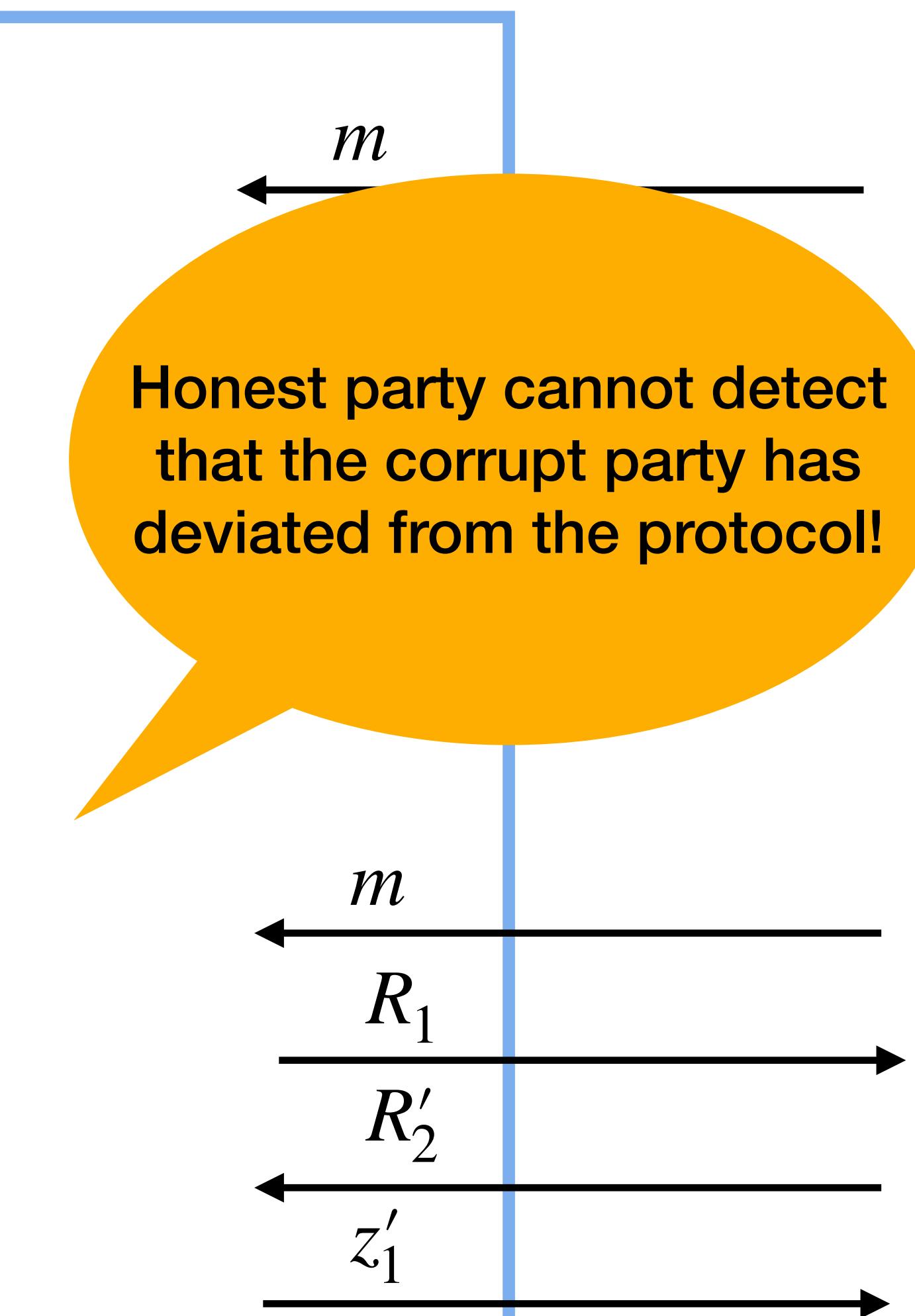
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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

$$z'_1 = r_1 + c'sk_1$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

$$r'_2 \xleftarrow{\$} \mathbb{F}$$

$$R'_2 = g^{r'_2}$$

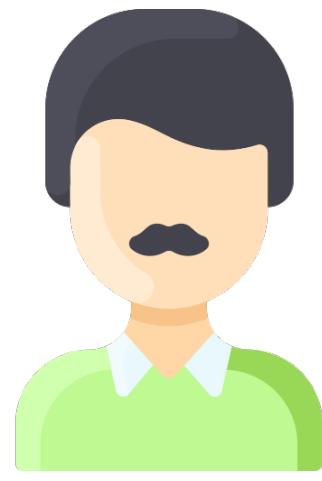
Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature



Corrupt
 PK_2

Honest

PK_1



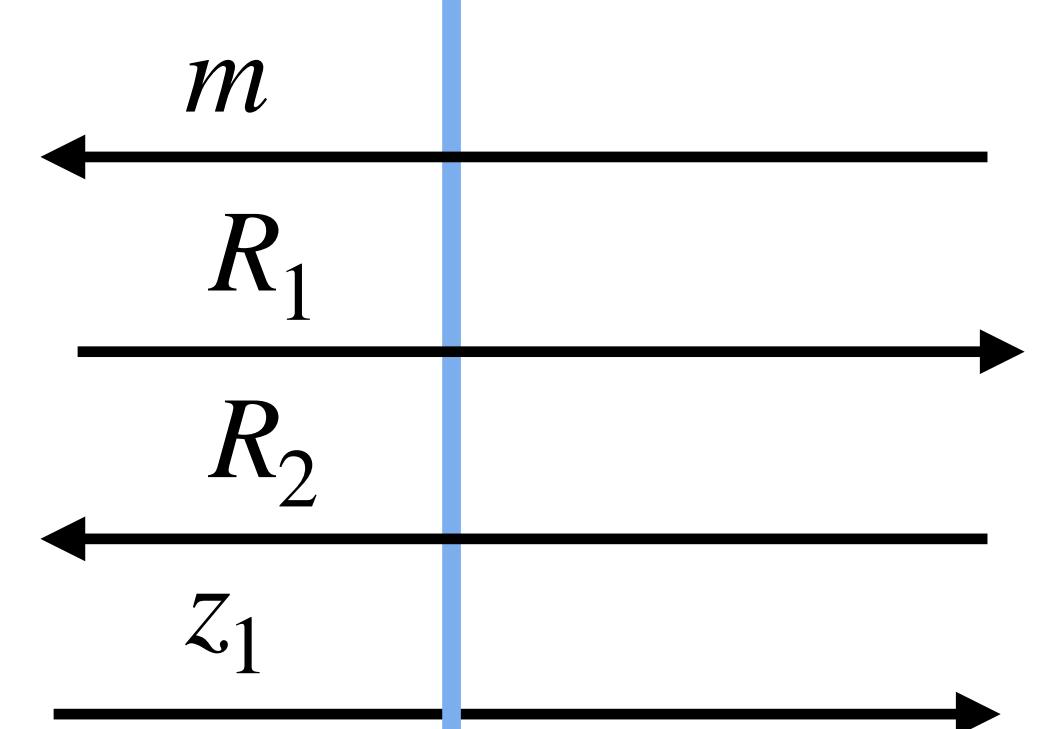
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



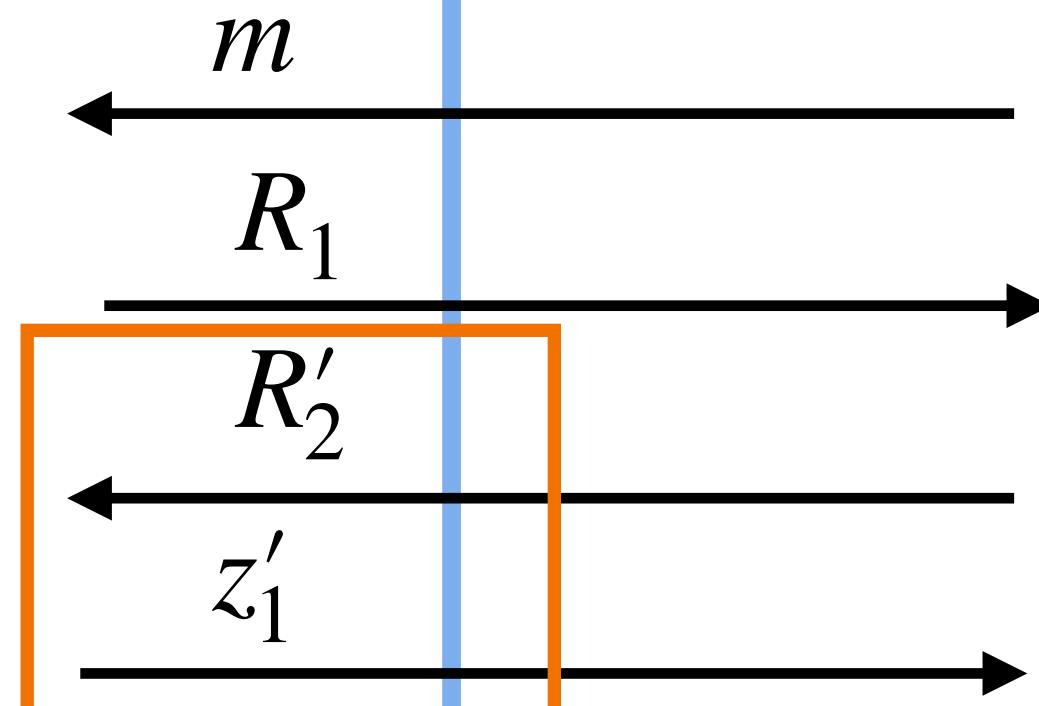
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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

$$z'_1 = r_1 + c'sk_1$$



$$r_2 \leftarrow H_1(m, sk_2)$$

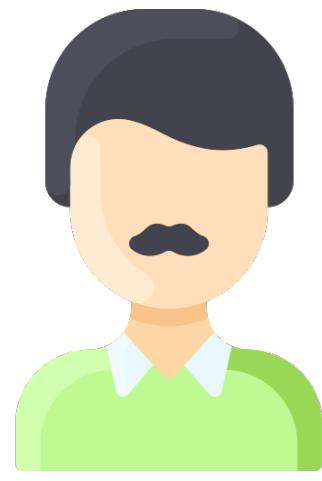
$$R_2 = g^{r_2}$$

$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



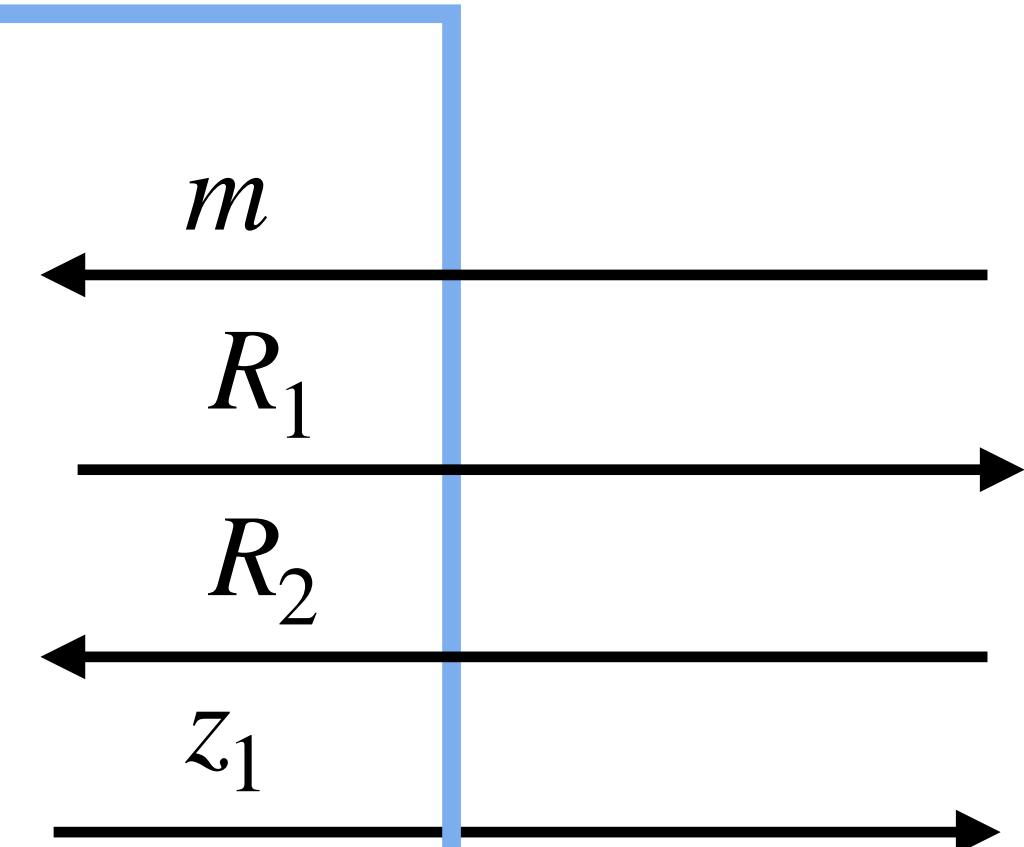
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



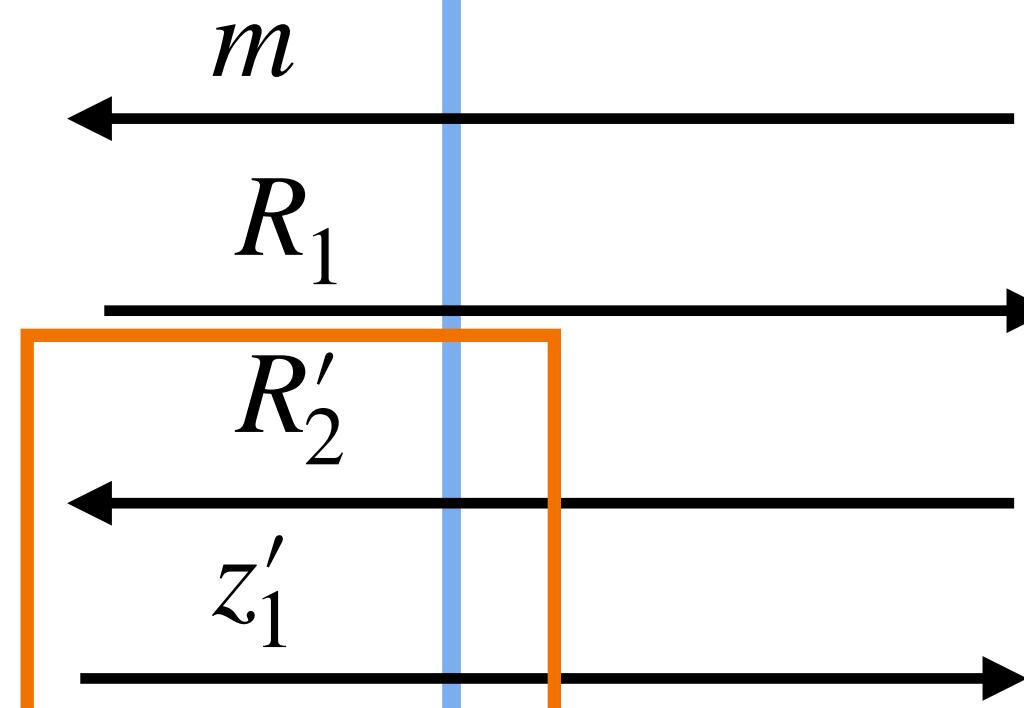
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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

$$z'_1 = r_1 + c'sk_1$$



Corrupt

PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

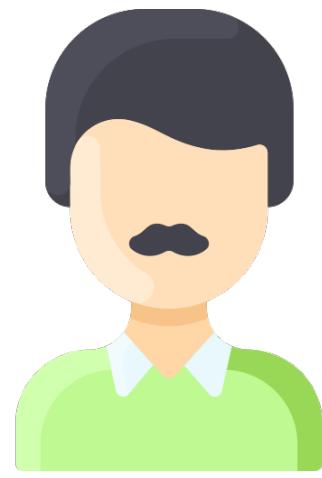
$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$

$$sk = \frac{z_v - z'_v}{c - c'}$$

Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



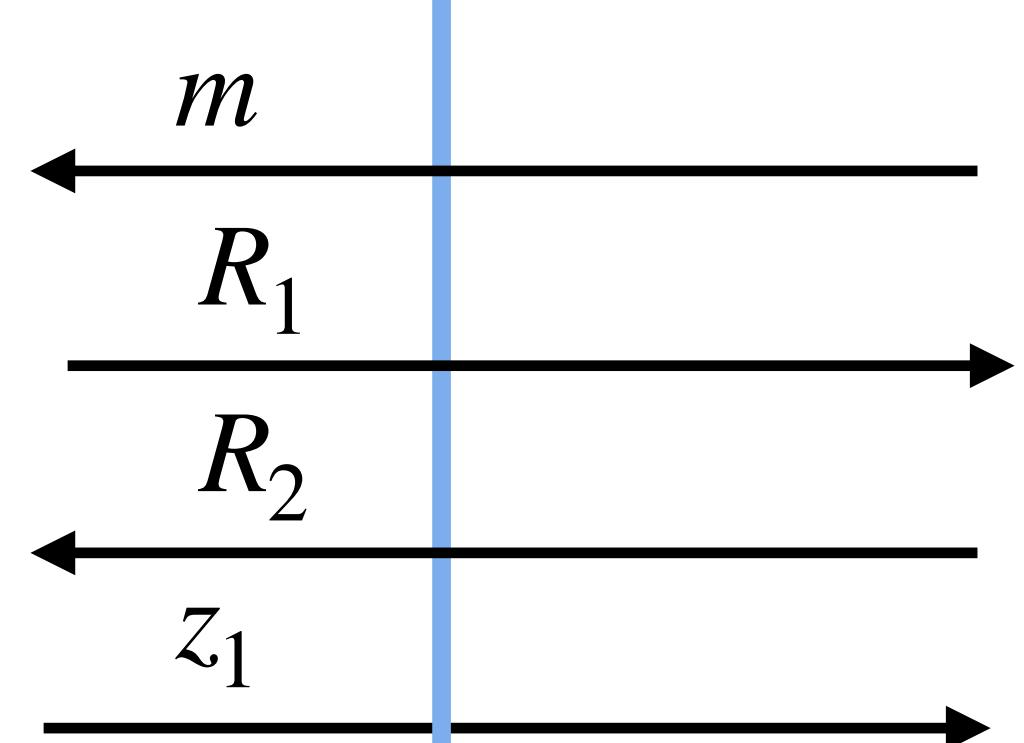
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + c sk_1$$



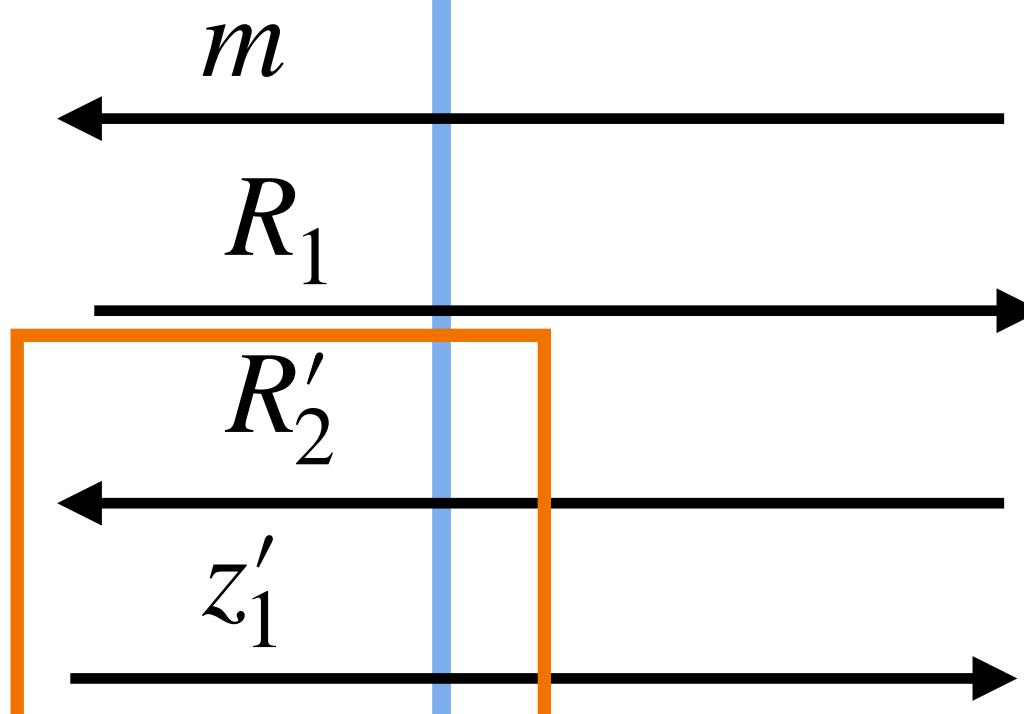
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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

$$z'_1 = r_1 + c' sk_1$$



$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$

$$sk = \frac{z_v - z'_v}{c - c'}$$



Corrupt

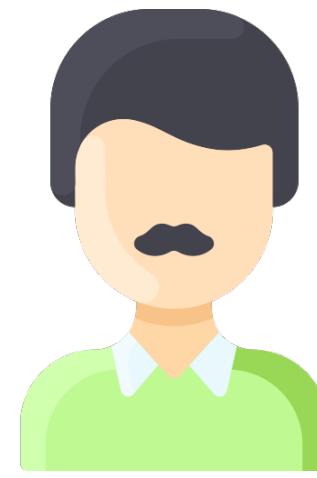
PK_2



Key Recovery Attack on Deterministic (Insecure) Multi-Party Signature

Honest

PK_1



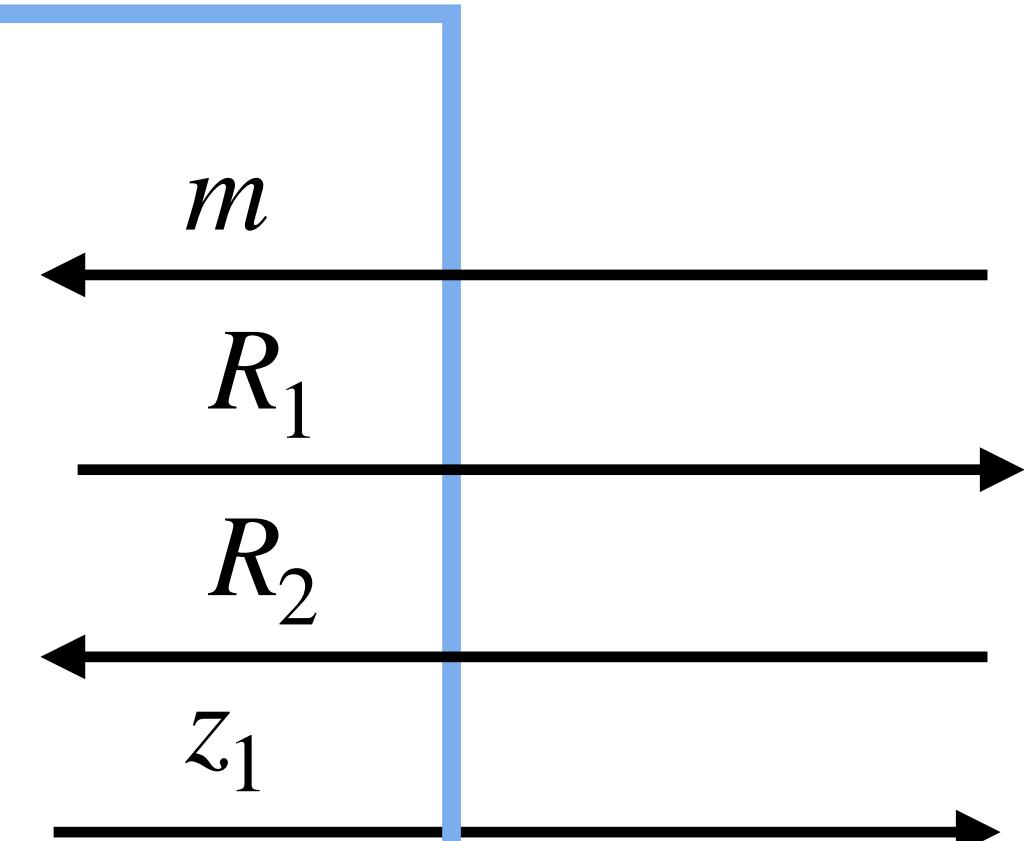
$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R = R_1 R_2$$

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

$$z_1 = r_1 + csk_1$$



$$r_1 \leftarrow H_1(m, sk_1)$$

$$R_1 = g^{r_1}$$

$$R' = R_1 R'_2$$

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

Question: Can we build a real-world (efficient) deterministic threshold signature? ⁷⁹

Corrupt

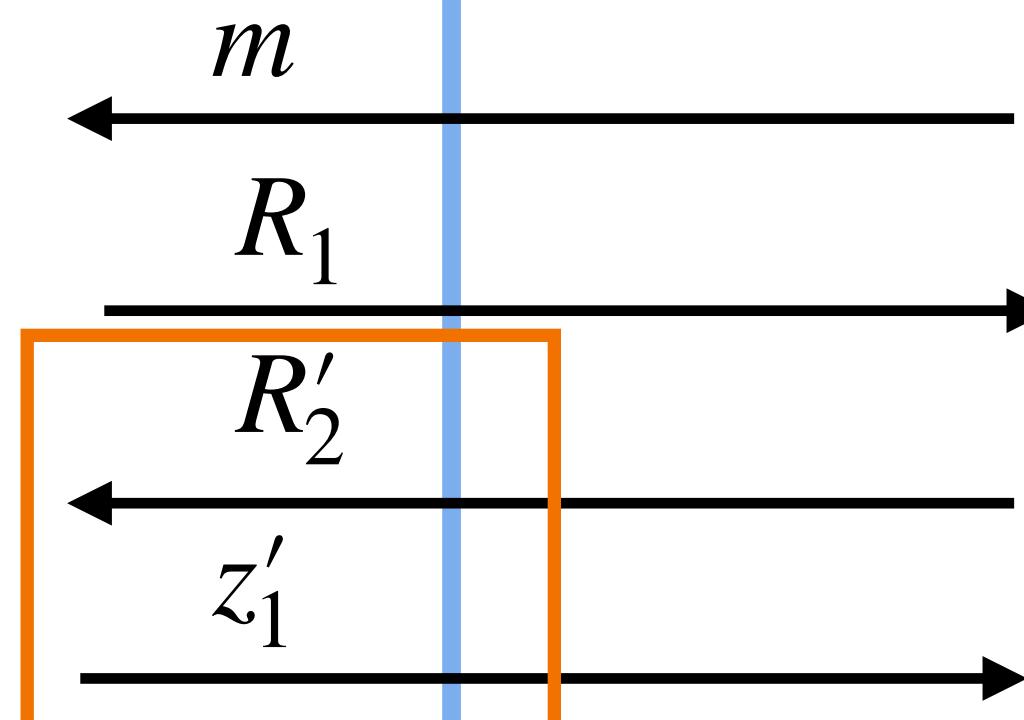


PK_2

$$r_2 \leftarrow H_1(m, sk_2)$$

$$R_2 = g^{r_2}$$

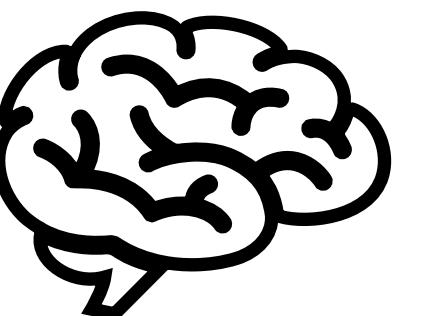
$$\begin{aligned} r'_2 &\xleftarrow{\$} \mathbb{F} \\ R'_2 &= g^{r'_2} \end{aligned}$$



$$sk = \frac{z_v - z'_v}{c - c'}$$



Clarify the Tradeoff Between Efficiency and Security Assumptions for Signing

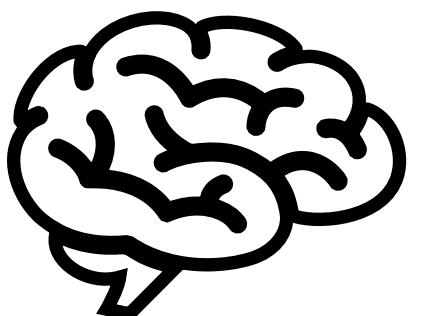


	Scheme	Assumptions	Signing Rounds
Multi-sigs	MuSig [MPSW18, BDN18]	DL+ROM	3
	SimpleMuSig [BDN18, CKM21]		
Threshold	MuSig2 [NRS21]	OMDL+ROM	2
	DWMS [AB21]		
Threshold	SpeedyMuSig [CKM21]		
	Lindell22	Schnorr DL+ROM	3
	Sparkle [CKM23]		
	FROST [KG20, BCKMTZ22]	OMDL+ROM	2
	FROST2 [CKM21]		

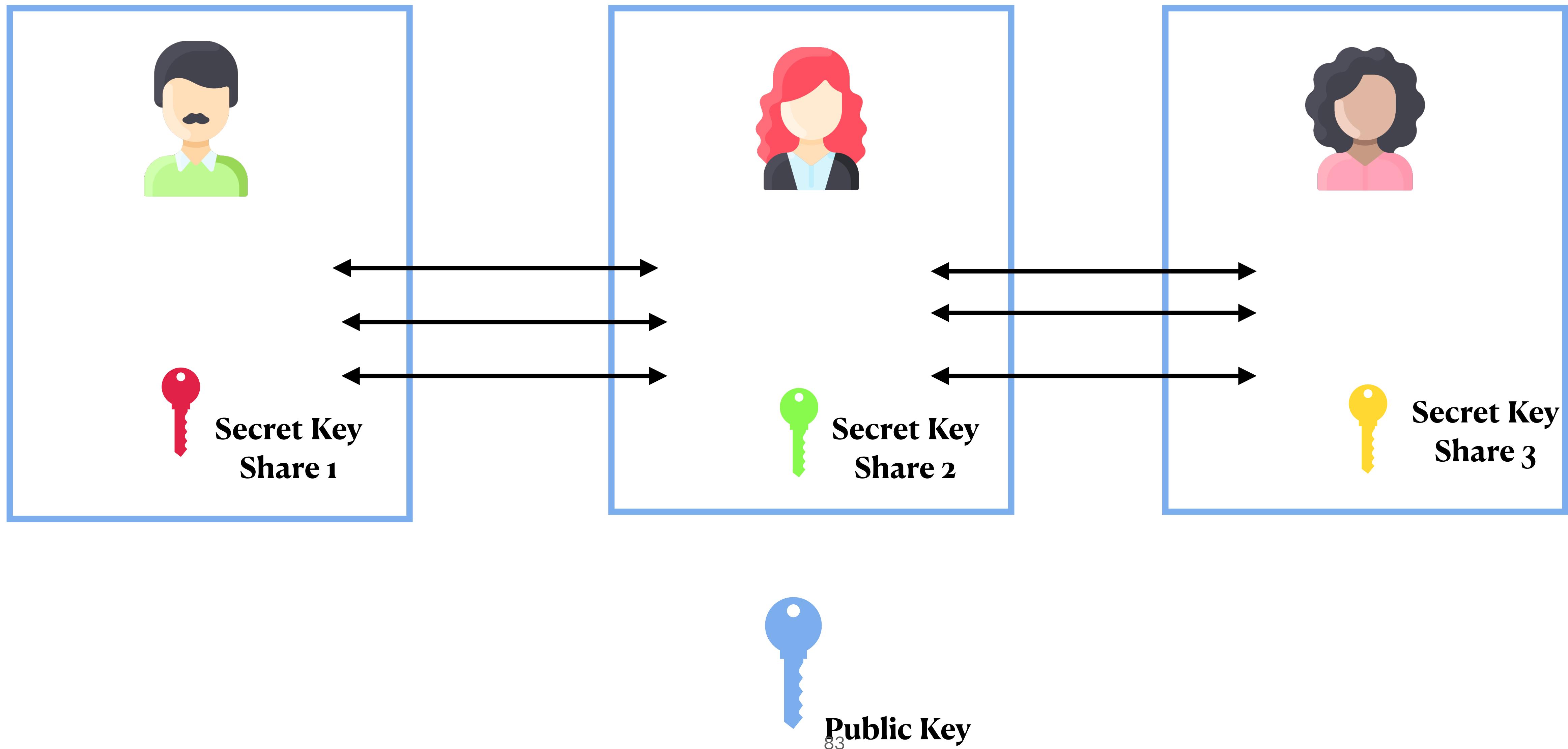
	Scheme	Assumptions	Signing Rounds
Multi-sigs	MuSig [MPSW18, BDN18]	DL+ROM	3
	SimpleMuSig [BDN18, CKM21]		
Threshold	MuSig2 [NRS21]	OMDL+ROM	2
	DWMS [AB21]		
Threshold	SpeedyMuSig [CKM21]		
	Lindell22	Schnorr DL+ROM	3
	Sparkle [CKM23]		
	FROST [KG20, BCKMTZ22]	OMDL+ROM	2
	FROST2 [CKM21]		

Question: Can we prove that two-round, efficient multi-party Schnorr requires stronger assumptions?

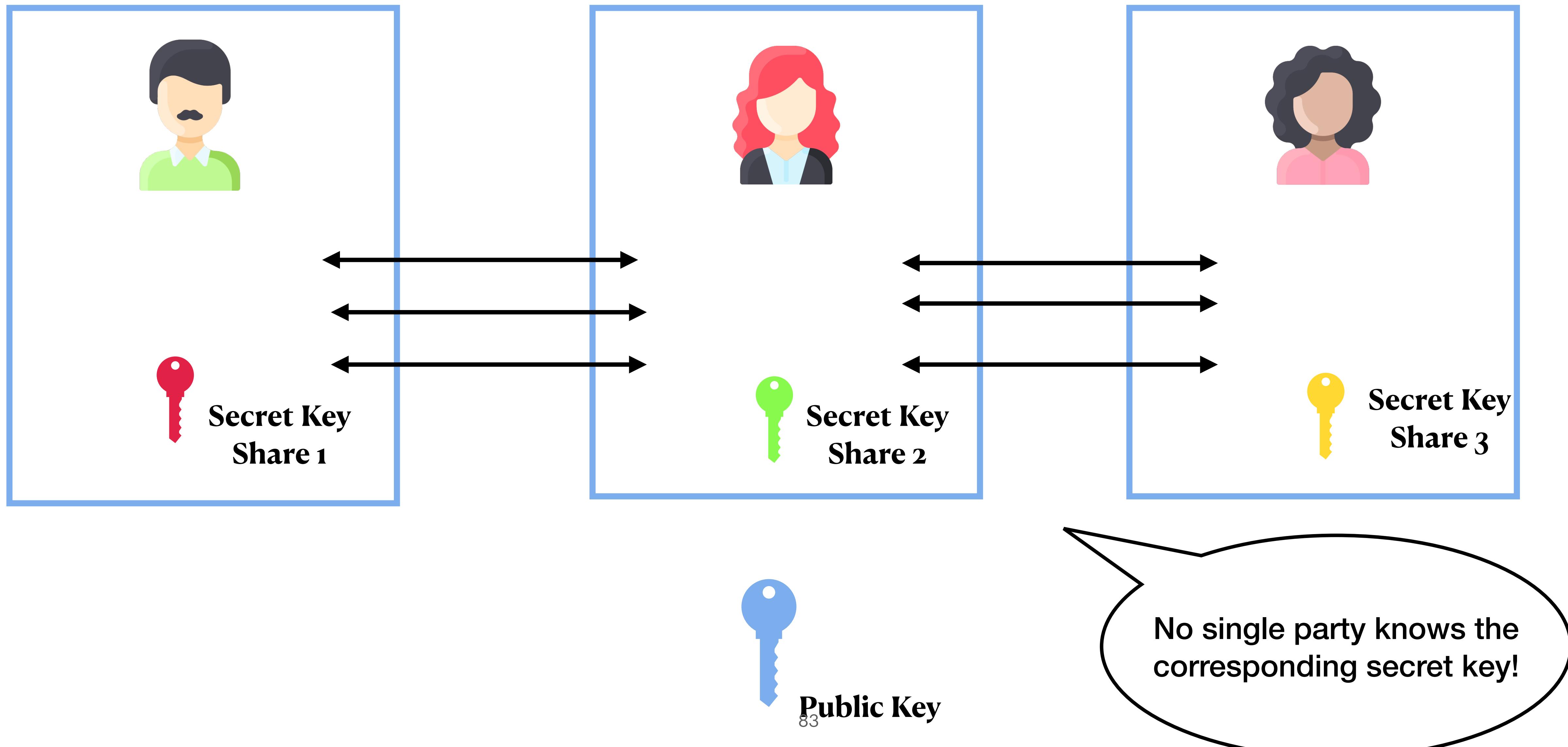
Investigate the Tradeoff Between Efficiency and Security Assumptions for Key Generation



Distributed Key Generation



Distributed Key Generation



Proving the security of DKGs

Proving the security of DKGs

- Two options:

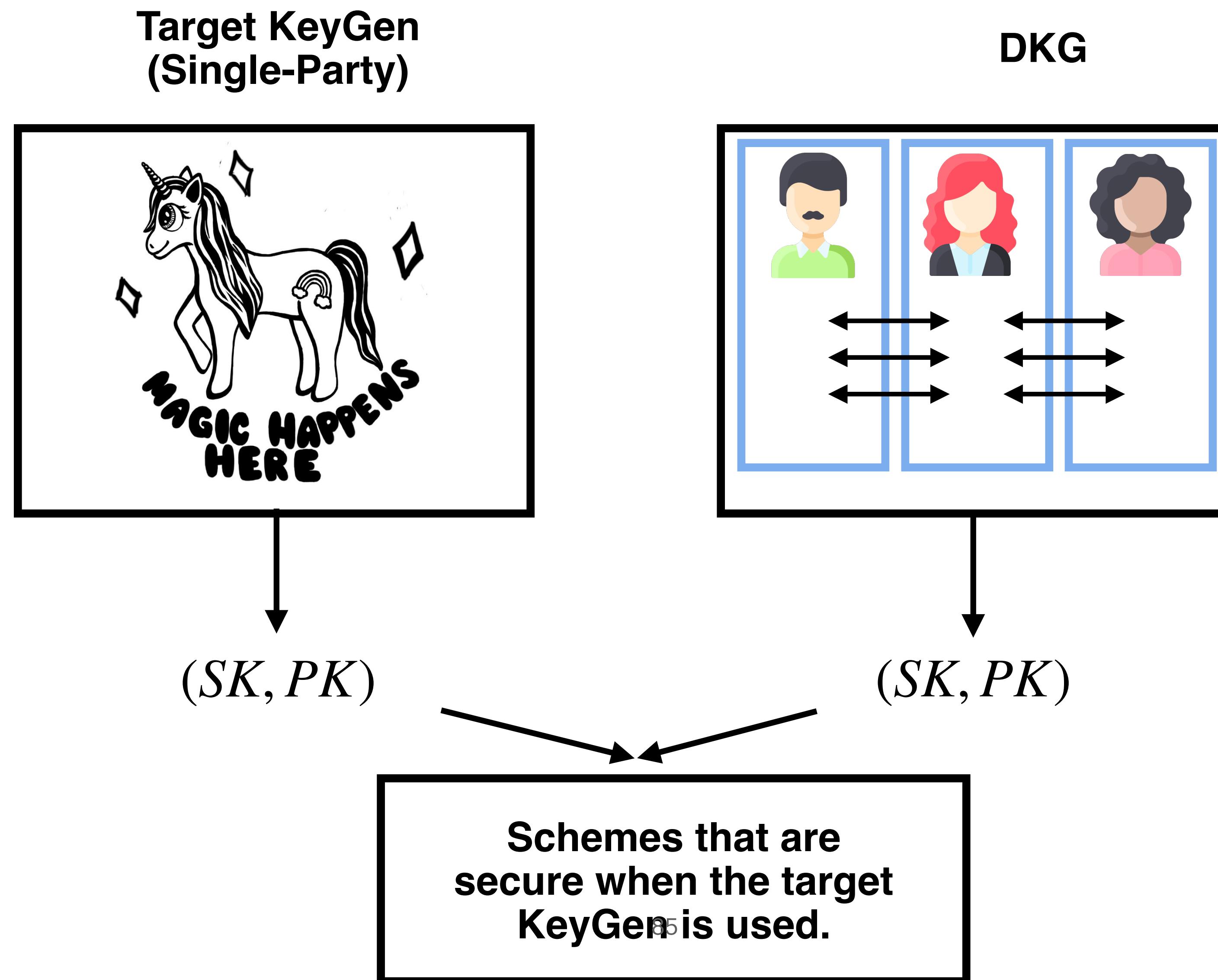
Proving the security of DKGs

- Two options:
 - Prove security of the DKG in the context of a proof for unforgeability for a threshold signature scheme

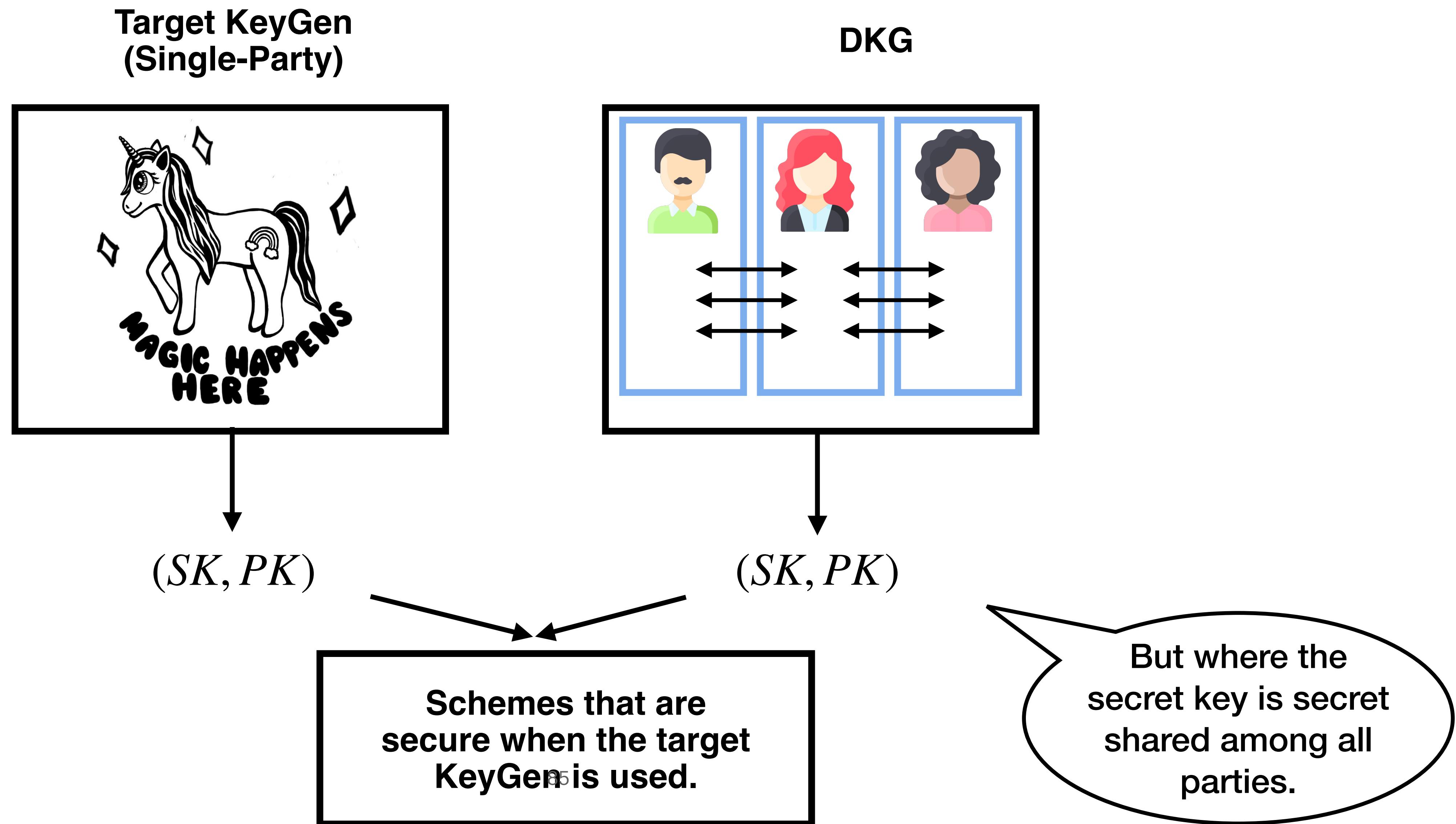
Proving the security of DKGs

- Two options:
 - Prove security of the DKG in the context of a proof for unforgeability for a threshold signature scheme
 - Prove the security of the DKG independently, i.e, via a proof of compositability

Composability of Distributed Key Generation



Composability of Distributed Key Generation



Distributed Key Generation

3-Round (optimistically),
Composable



DL, CDH

standard
assumption

2-Round (total)
Proven for FROST



AGM

stronger
assumption

Distributed Key Generation

3-Round (optimistically),
Composable



2-Round (total)
Proven for FROST



DL, CDH

AGM

standard
assumption

stronger
assumption

Question: Do two-round, efficient and composable DKGs exist?

Takeaways

Takeaways

- Schnorr multi-party signatures are being used in practice today!

Takeaways

- Schnorr multi-party signatures are being used in practice today!
- Many questions and challenges remain, to improving their usability and security.

Takeaways

- Schnorr multi-party signatures are being used in practice today!
- Many questions and challenges remain, to improving their usability and security.
- We would love to collaborate with anyone interested in tackling these problems or using these schemes, so please come talk to us!

Takeaways

- Schnorr multi-party signatures are being used in practice today!
- Many questions and challenges remain, to improving their usability and security.
- We would love to collaborate with anyone interested in tackling these problems or using these schemes, so please come talk to us!
- (t,n) of us will also be involved in the NIST call for threshold schemes, so please let us know if you would like to join forces.

Takeaways

- Schnorr multi-party signatures are being used in practice today!
- Many questions and challenges remain, to improving their usability and security.
- We would love to collaborate with anyone interested in tackling these problems or using these schemes, so please come talk to us!
- (t,n) of us will also be involved in the NIST call for threshold schemes, so please let us know if you would like to join forces.

Thank you!
87

	Scheme	Static Assumptions	Adaptive Assumptions	Signing Rounds
Schnorr	Lindell22	Schnorr	Schnorr	3
	Sparkle [CKM23]	DL+ROM	AOMDL+ROM+AGM	
	FROST [KG20, BCKMTZ22]	OMDL+ROM		2
Non-Schnorr	Twinkle []			

One-More Discrete Log (OMDL):
 - stronger assumption
 + partially non-interactive schemes

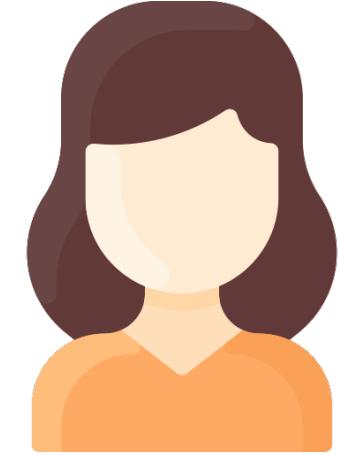
	Scheme	Static Assumptions	Adaptive Assumptions	Signing Rounds
Schnorr	Lindell22	Schnorr	Schnorr	3
	Sparkle [CKM23]	DL+ROM	AOMDL+ROM+AGM	
	FROST [KG20, BCKMTZ22]	OMDL+ROM		2
Non-Schnorr	Twinkle []			

All are concurrently secure ✓

One-More Discrete Log (OMDL):

- stronger assumption
- + partially non-interactive schemes

Case Study: Sparkle



Key Generation:

$(sk_i, PK_i), PK$

Combine / Verify:

Case Study: Sparkle



$$\begin{array}{c} cm_i = H(R_i) \\ \xrightarrow{\hspace{1cm}} \\ R_i \\ \xrightarrow{\hspace{1cm}} \end{array}$$



Key Generation:

$$(sk_i, PK_i), PK$$

Combine / Verify:

Round 1: Sample $r_i = \mathbb{Z}_q$; set $R_i \leftarrow g^{r_i}$

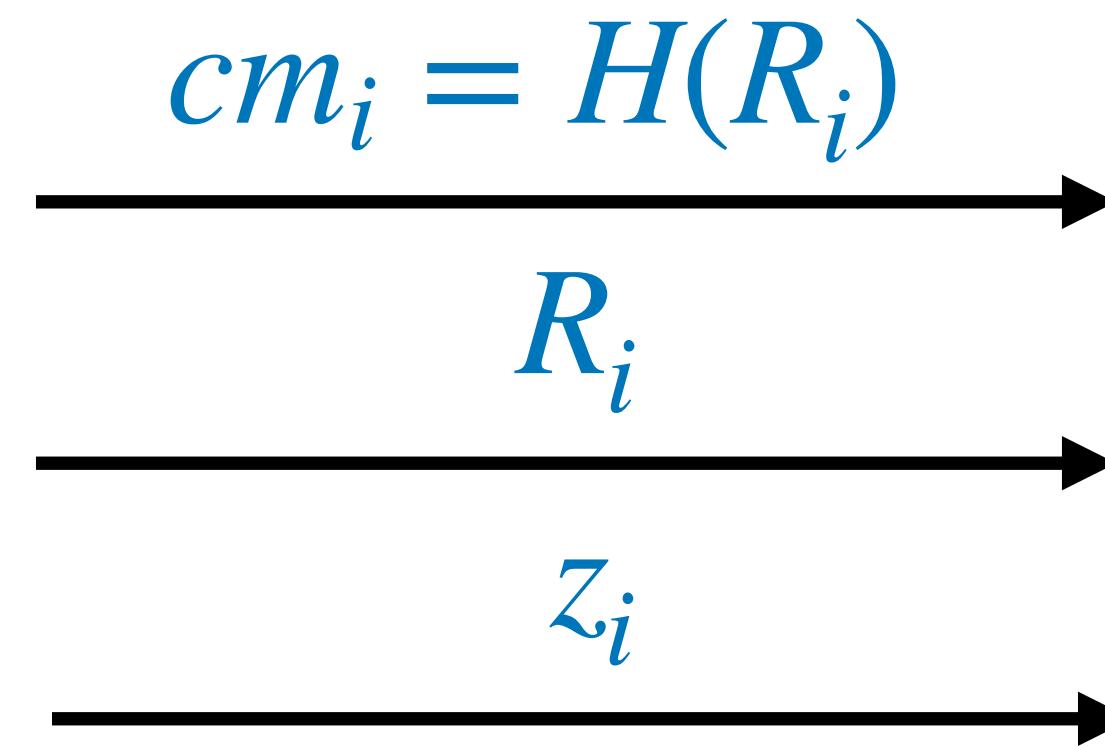
Output $cm_i = H(R_i)$

Round 2: Output R_i

Case Study: Sparkle



Key Generation:
 $(sk_i, PK_i), PK$



Combine / Verify:

Round 1: Sample $r_i = \mathbb{Z}_q$; set $R_i \leftarrow g^{r_i}$
Output $cm_i = H(R_i)$

Round 2: Output R_i

Round 3: $R = \prod_{i=1}^n R_i$
 $c \leftarrow H(PK, m, R)$
Output $z_i \leftarrow r_i + csk_i$

Case Study: Sparkle



Key Generation:

$$(sk_i, PK_i), PK$$

Round 1: Sample $r_i = \mathbb{Z}_q$; set $R_i \leftarrow g^{r_i}$

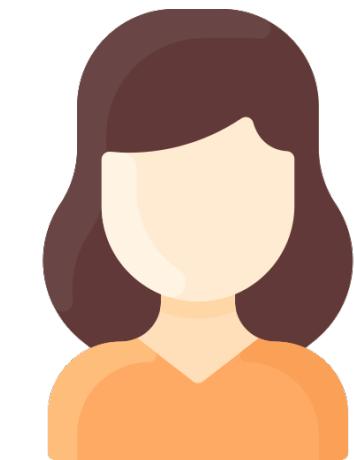
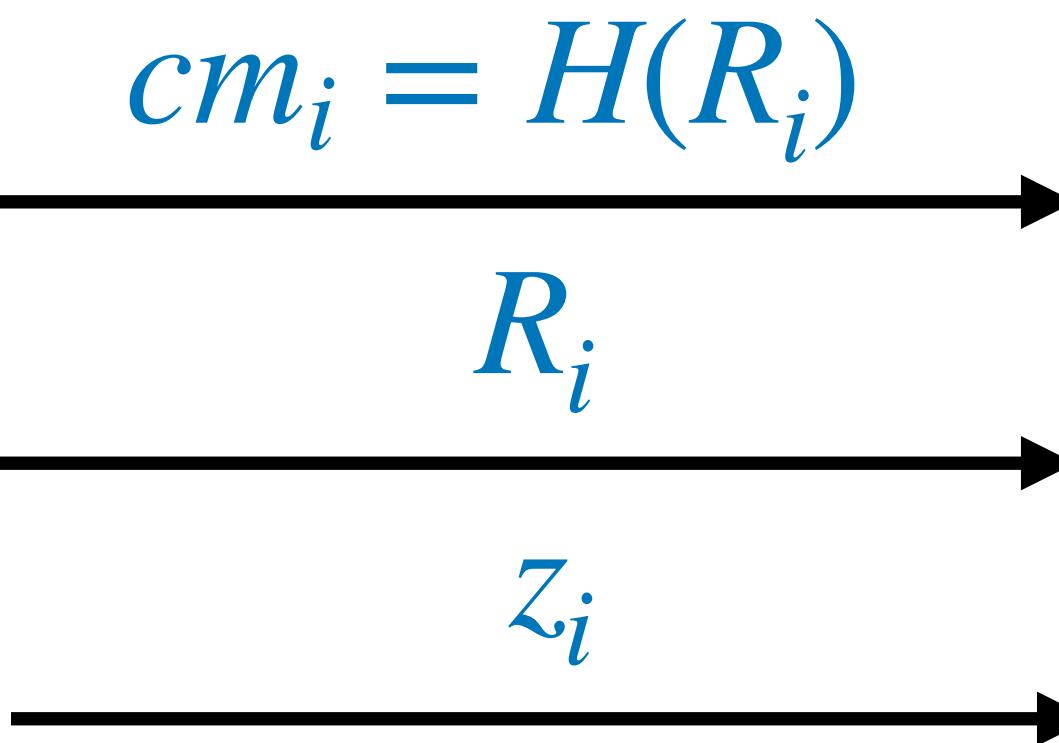
Output $cm_i = H(R_i)$

Round 2: Output R_i

Round 3: $R = \prod_{i=1}^n R_i$

$c \leftarrow H(PK, m, R)$

Output $z_i \leftarrow r_i + csk_i$



Combine / Verify:

$$z = \sum_{i=1}^n z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z \quad \checkmark$$

Case Study: FROST

[KG20]

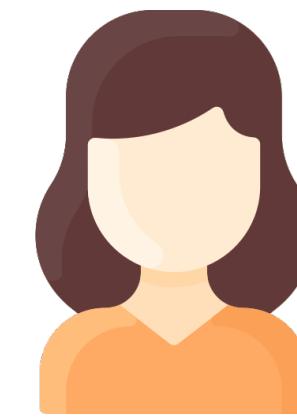
PK_1



PK_2



PK_t



Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$



PK_2



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$
$$R_1 = g^{r_1}$$

PK_2



⋮



PK_t

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



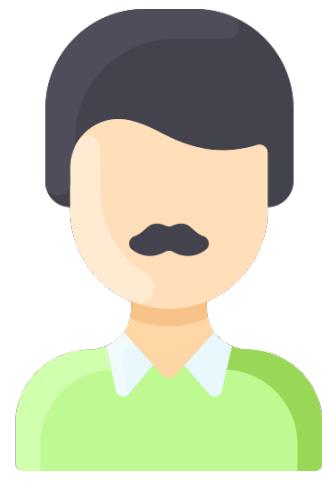
Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

PK_2



⋮

PK_t



Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



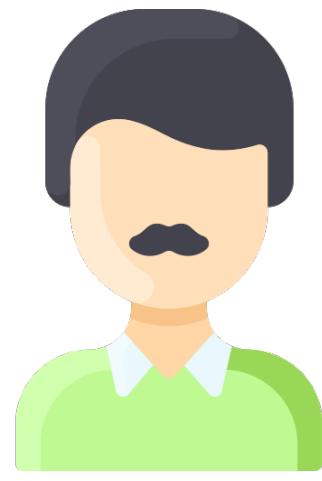
Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

cm_1



PK_2



PK_t

Verification
identical to
Schnorr

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

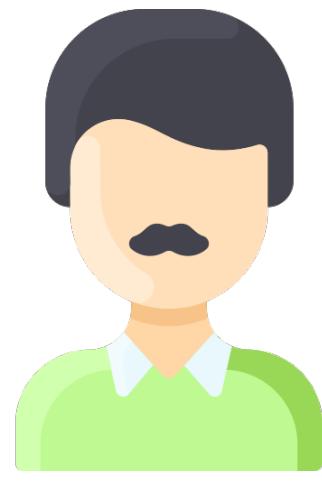


Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

cm_1



PK_2



PK_t

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

90

Verification
identical to
Schnorr

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

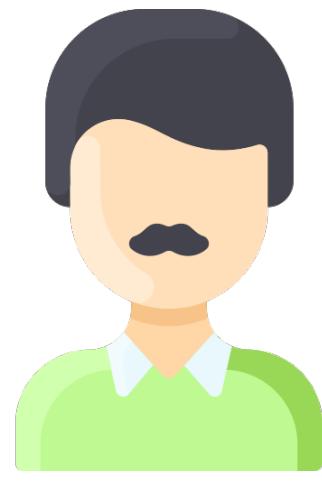
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$cm_1$$

$$\{cm_1, \dots, cm_t\}$$



PK_2



PK_t

Verification
identical to
Schnorr

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

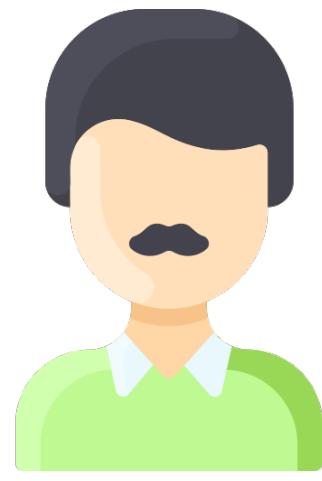


Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$cm_1$$

$$\{cm_1, \dots, cm_t\}$$



PK_2



PK_t

90

Similar Techniques: MuSig [MPSW18], Lindell22, ZeroS [M23]

Verification
identical to
Schnorr

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

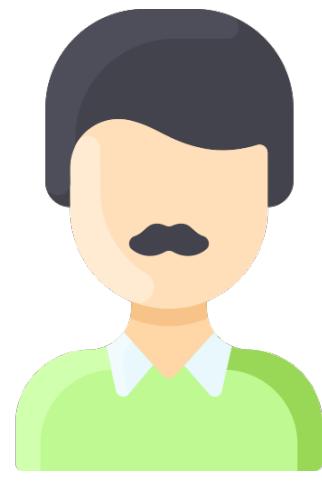
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$cm_1$$

$$\{cm_1, \dots, cm_t\}$$

$$R_1$$



PK_2



PK_t

Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

90

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

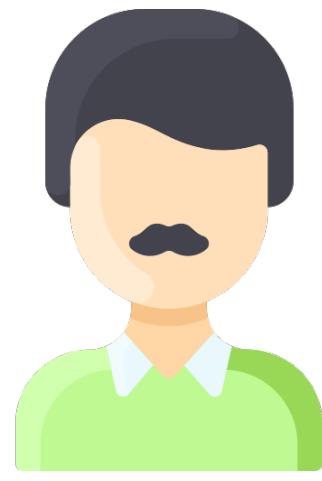
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$cm_1$$

$$\{cm_1, \dots, cm_t\}$$

$$R_1$$



PK_2



PK_t

Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

90

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$cm_1$$

$$\{cm_1, \dots, cm_t\}$$

$$R_1$$

$$\{R_1, \dots, R_t\}$$



PK_2



PK_t

Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

90

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

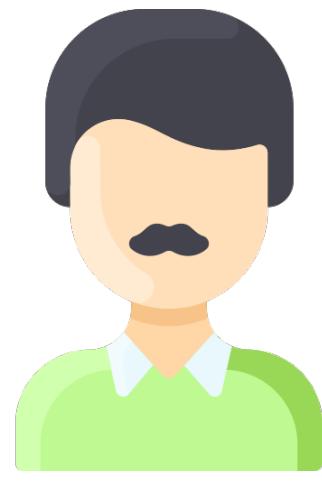
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$cm_1$$

$$\{cm_1, \dots, cm_t\}$$

$$R_1$$

$$\{R_1, \dots, R_t\}$$



PK_2



PK_t

Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

90

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

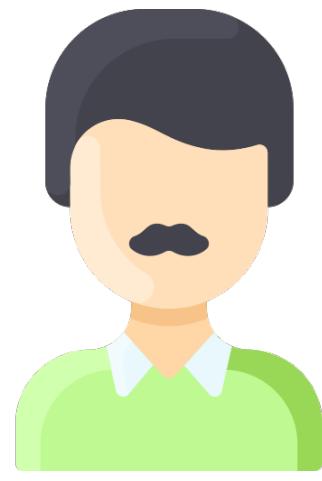
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1

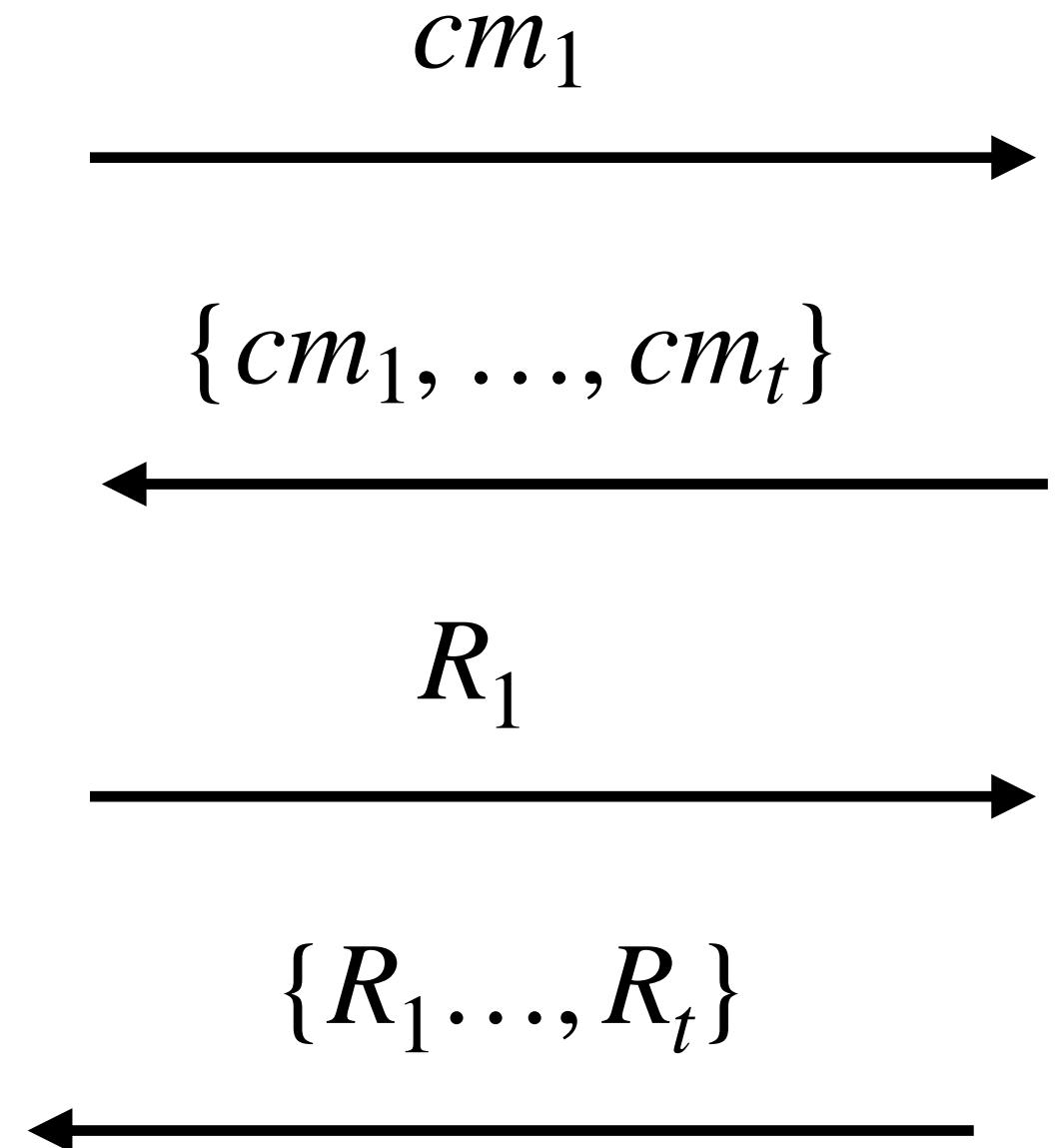


$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$R = \prod_{i=1}^t R_i$$



PK_2



PK_t

Similar Techniques: MuSig [MPSW18], Lindell22, ZeroS [M23]

90

Verification
identical to
Schnorr

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

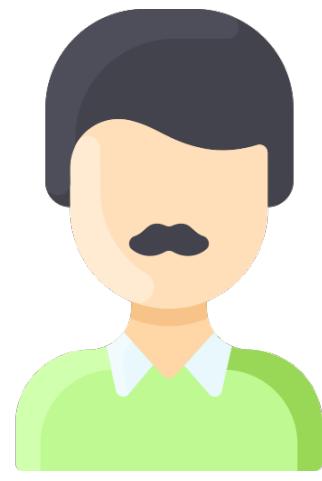
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$R = \prod_{i=1}^t R_i$$

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

$$cm_1$$

$$\{cm_1, \dots, cm_t\}$$

$$R_1$$

$$\{R_1, \dots, R_t\}$$



PK_2



PK_t

Verification
identical to
Schnorr

Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

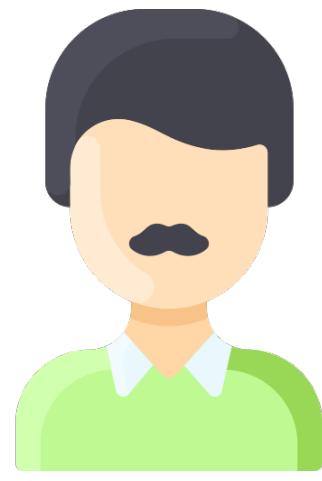
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

$$R = \prod_{i=1}^t R_i$$

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

$$z_1 = r_1 + c\lambda_1 sk_1$$

$$cm_1$$

$$\{cm_1, \dots, cm_t\}$$

$$R_1$$

$$\{R_1, \dots, R_t\}$$



PK_2



PK_t

Verification
identical to
Schnorr

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$

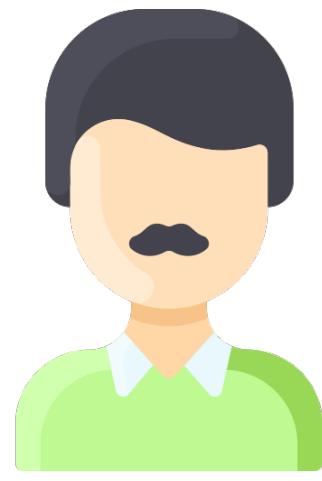


Similar Techniques: MuSig [MPSW18], Lindell22,
ZeroS [M23]

Case Study: FROST

[KG20]

PK_1



$$\begin{aligned} r_1 &\leftarrow \mathbb{Z}_q \\ R_1 &= g^{r_1} \\ cm_i &= H(R_i, m, S = \{1, \dots, t\}) \end{aligned}$$

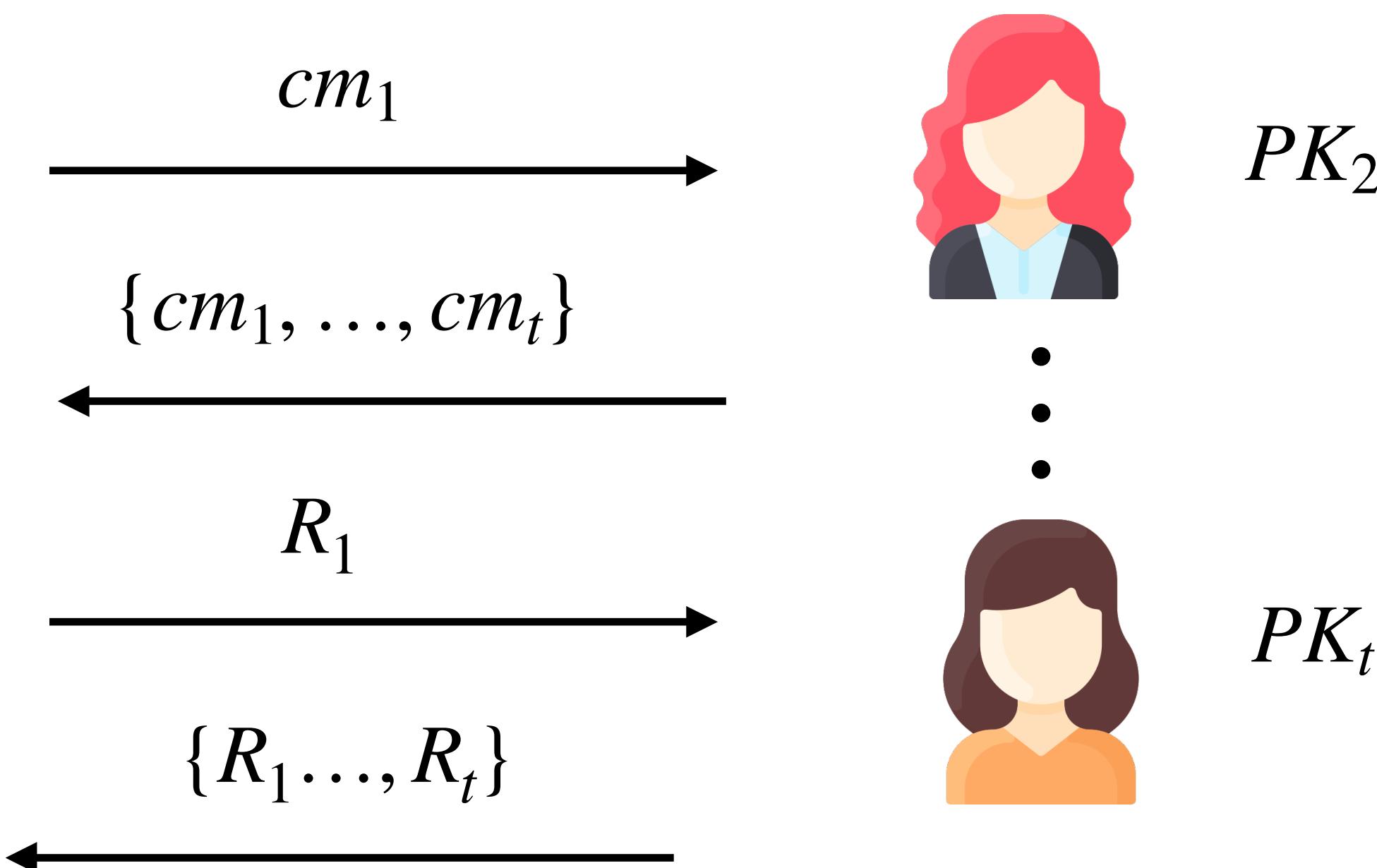
$$R = \prod_{i=1}^t R_i$$

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

$$z_1 = r_1 + c\lambda_1 sk_1$$

Reminder: $sk = \sum_{i=1}^t \lambda_i sk_i$

Similar Techniques: MuSig [MPSW18], Lindell22, ZeroS [M23]



90

Verification
identical to
Schnorr

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

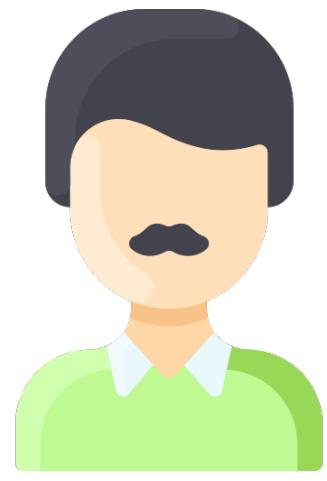
$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

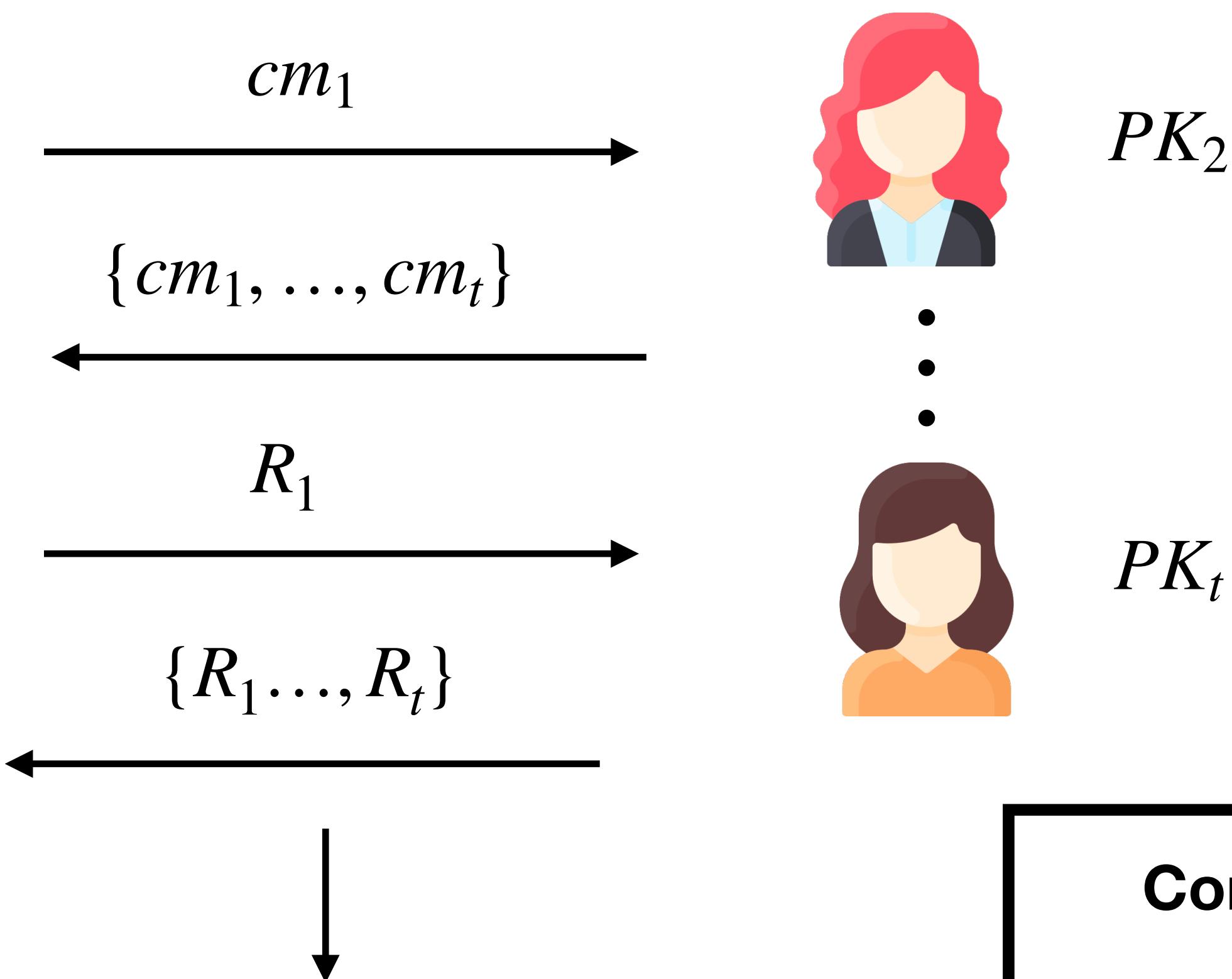
$$R = \prod_{i=1}^t R_i$$

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

$$z_1 = r_1 + c\lambda_1 sk_1$$

Reminder: $sk = \sum_{i=1}^t \lambda_i sk_i$

Similar Techniques: MuSig [MPSW18], Lindell22, ZeroS [M23]



90

Verification
identical to
Schnorr

Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



Case Study: FROST

[KG20]

PK_1



$$r_1 \leftarrow \mathbb{Z}_q$$

$$R_1 = g^{r_1}$$

$$cm_i = H(R_i, m, S = \{1, \dots, t\})$$

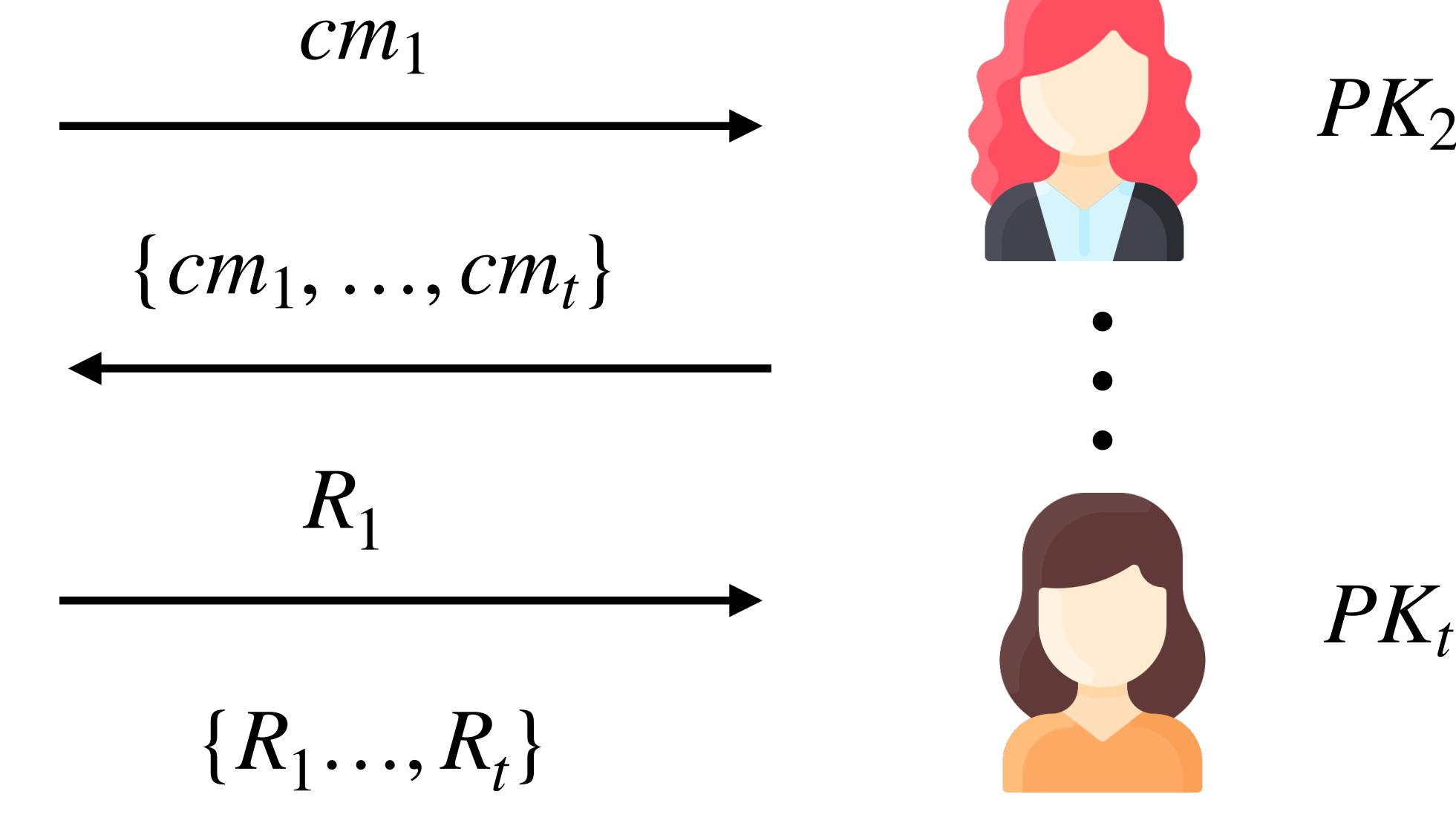
$$R = \prod_{i=1}^t R_i$$

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

$$z_1 = r_1 + c\lambda_1 sk_1$$

Reminder: $sk = \sum_{i=1}^t \lambda_i sk_i$

Similar Techniques: MuSig [MPSW18], Lindell22, ZeroS [M23]



(z_1, \dots, z_t)

Verification
identical to
Schnorr

PK_2



PK_t



Combine/Verify

$$z = \sum_{i=1}^t z_i$$

$$sig = (R, z)$$

$$c \leftarrow H(PK, m, R)$$

$$R \cdot PK^c = g^z$$



Key Uniqueness for Example #1

Public key and shares are generated via an injective map f .

$$f(0, sk) \rightarrow g^{sk} = PK$$

$$f(i, sk_i) \rightarrow g^{sk_i} = PK_i$$

Employed by: FROST/2/3 [KG20, BCKMTZ22CKM21, CGRS23],
Sparkle [CKM23], Lindell22, ZeroS [M23]

Key Uniqueness for Example #2

Public key and shares are not generated via an injective map f .

$$f(0, sk) \rightarrow g^{sk} = PK$$

$$f(i, sk_i) \rightarrow \perp$$

Key Uniqueness for Example #2

Public key and shares are not generated via an injective map f .

$$f(0, sk) \rightarrow g^{sk} = PK$$

$$f(i, sk_i) \rightarrow \perp$$

Tradeoff: No identifiable abort

Employed by: KRT24

Why key-uniqueness?

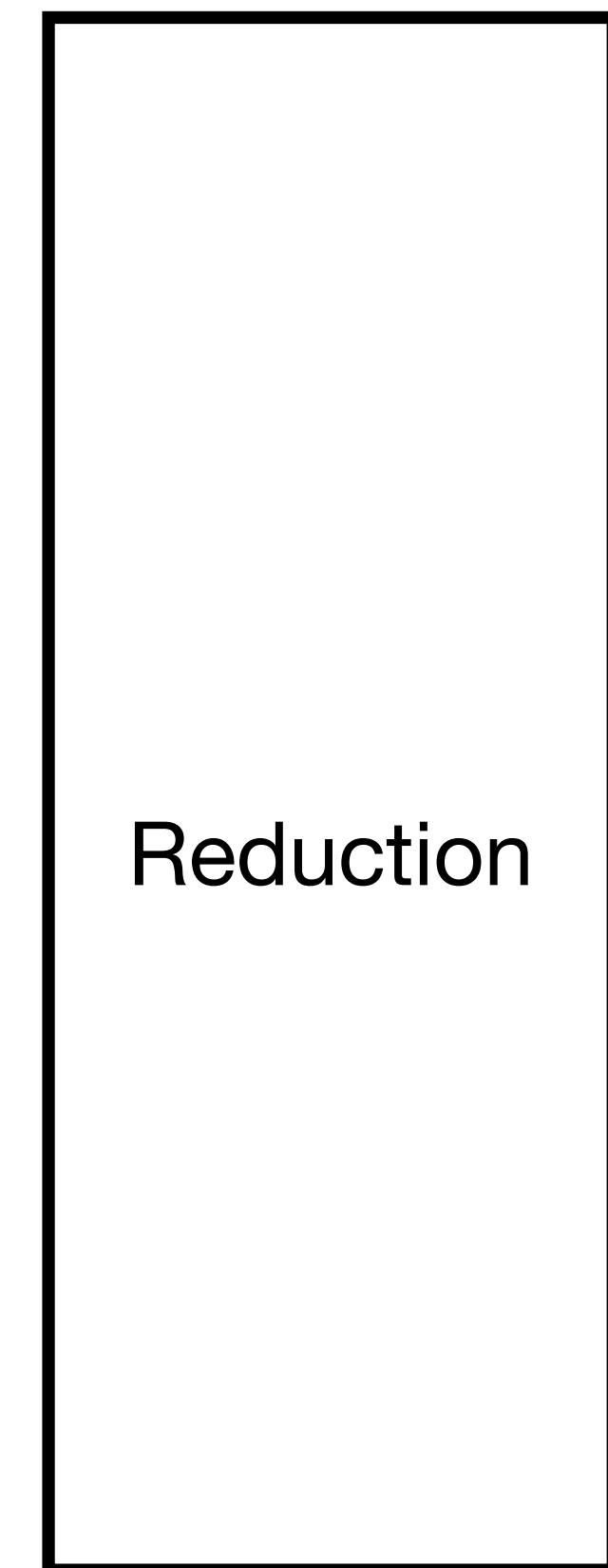
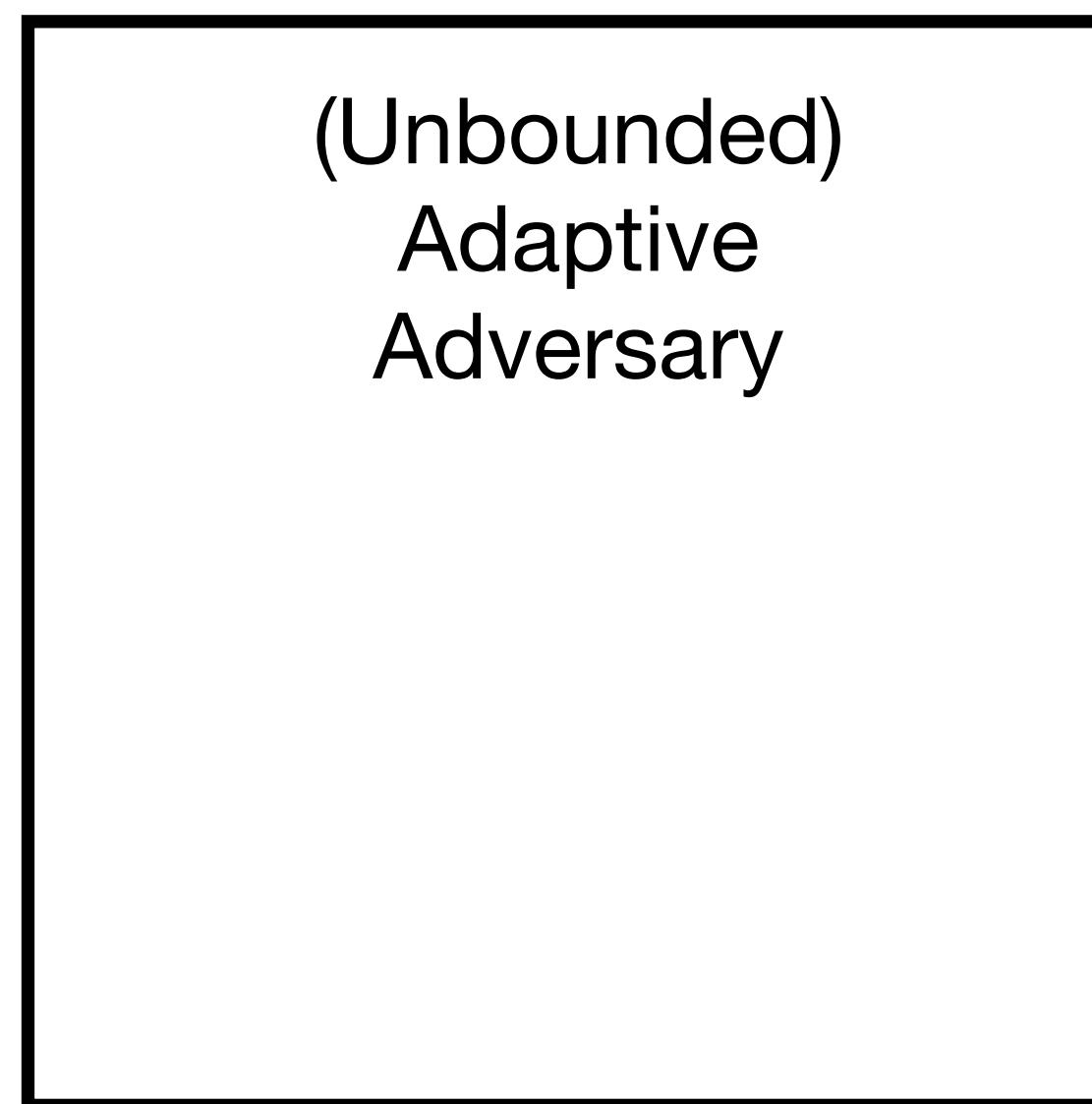
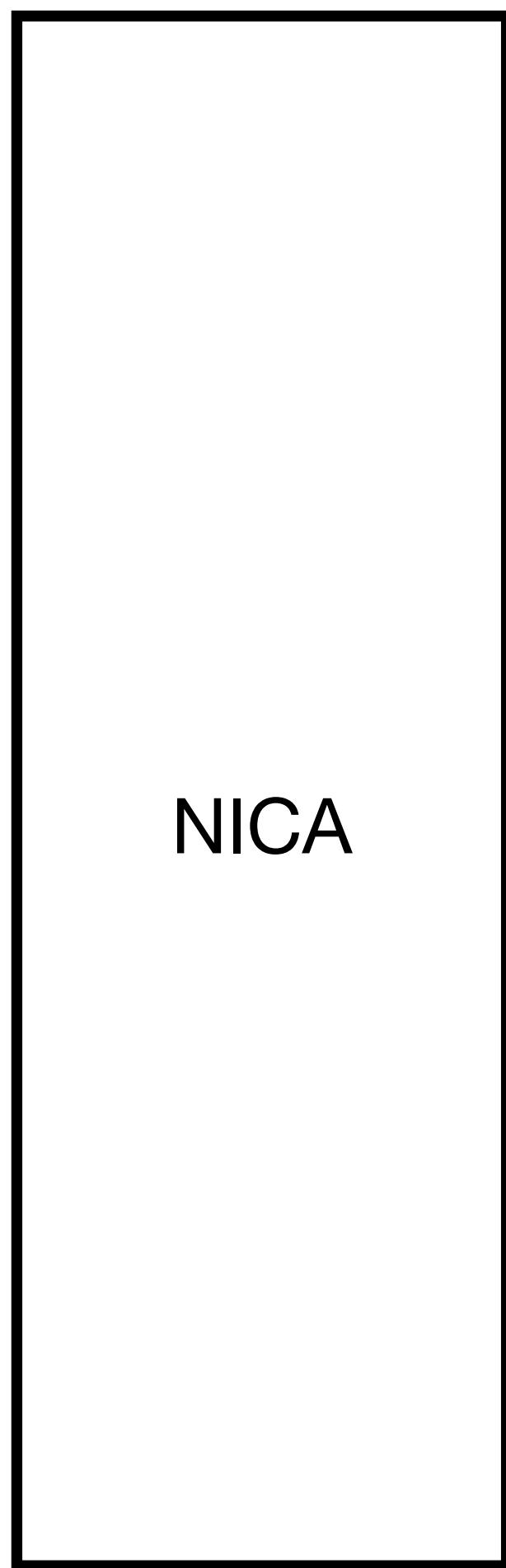
Achieving identifiable abort is straightforward and efficient.

Signature share $z_i = r_i + csk_i \lambda_i$

Public key share $PK_i = g^{sk_i} = g^{f(i)}$

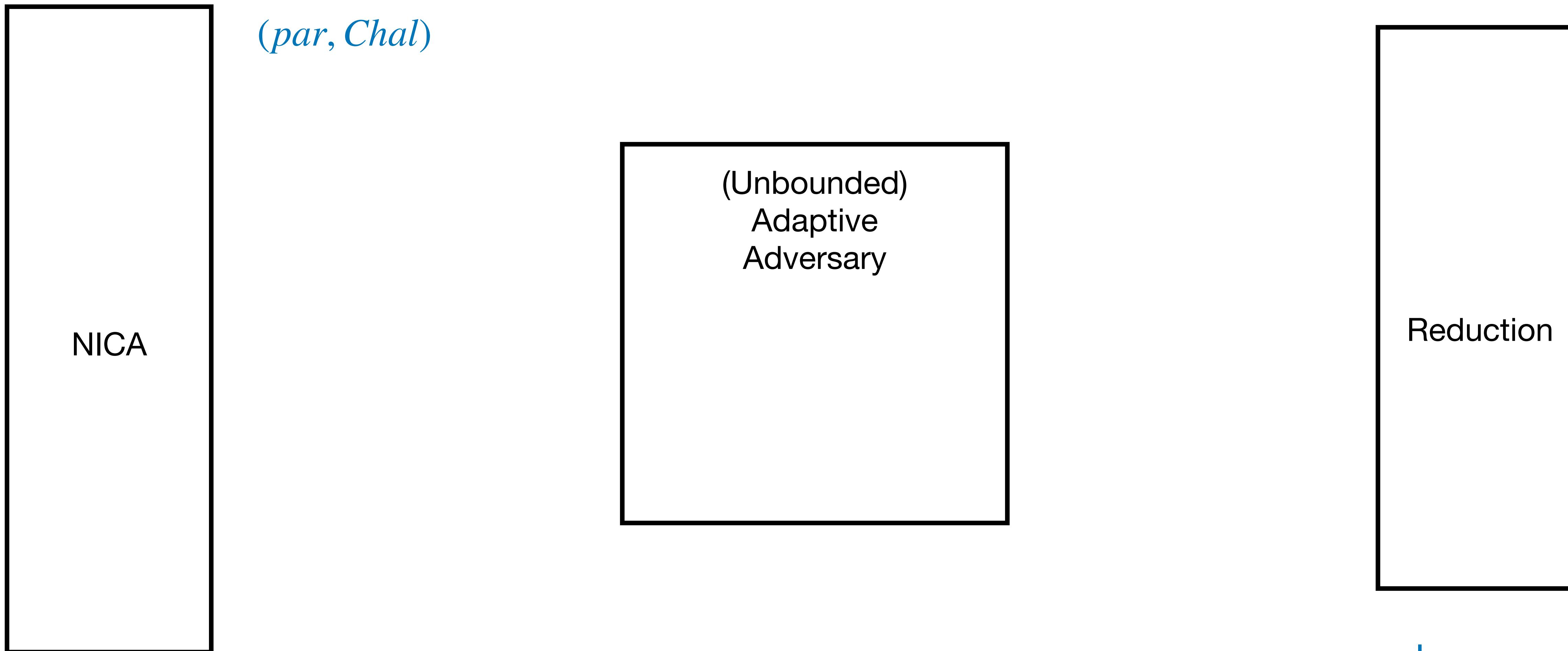
$$g^{z_i} \stackrel{?}{=} R_i \cdot PK_i^{c\lambda_i}$$

Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA

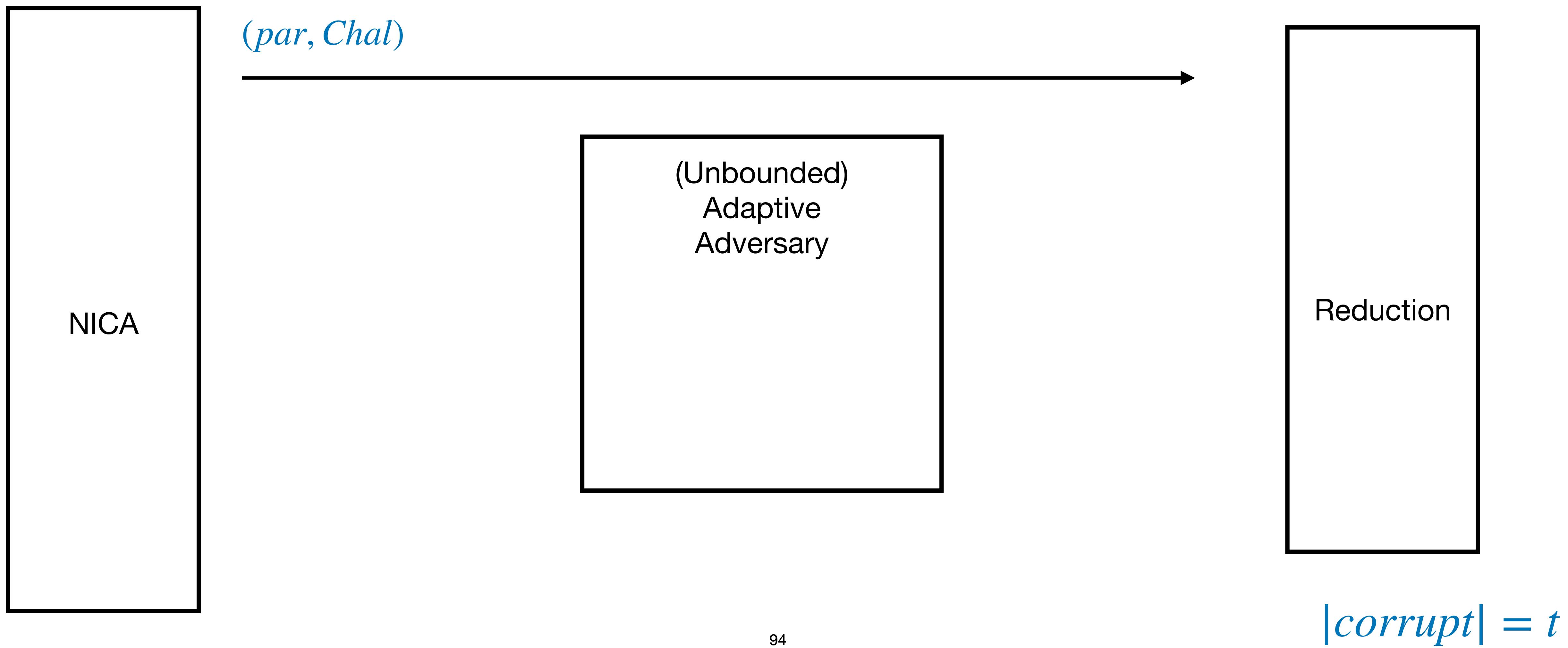


$|corrupt| = t$

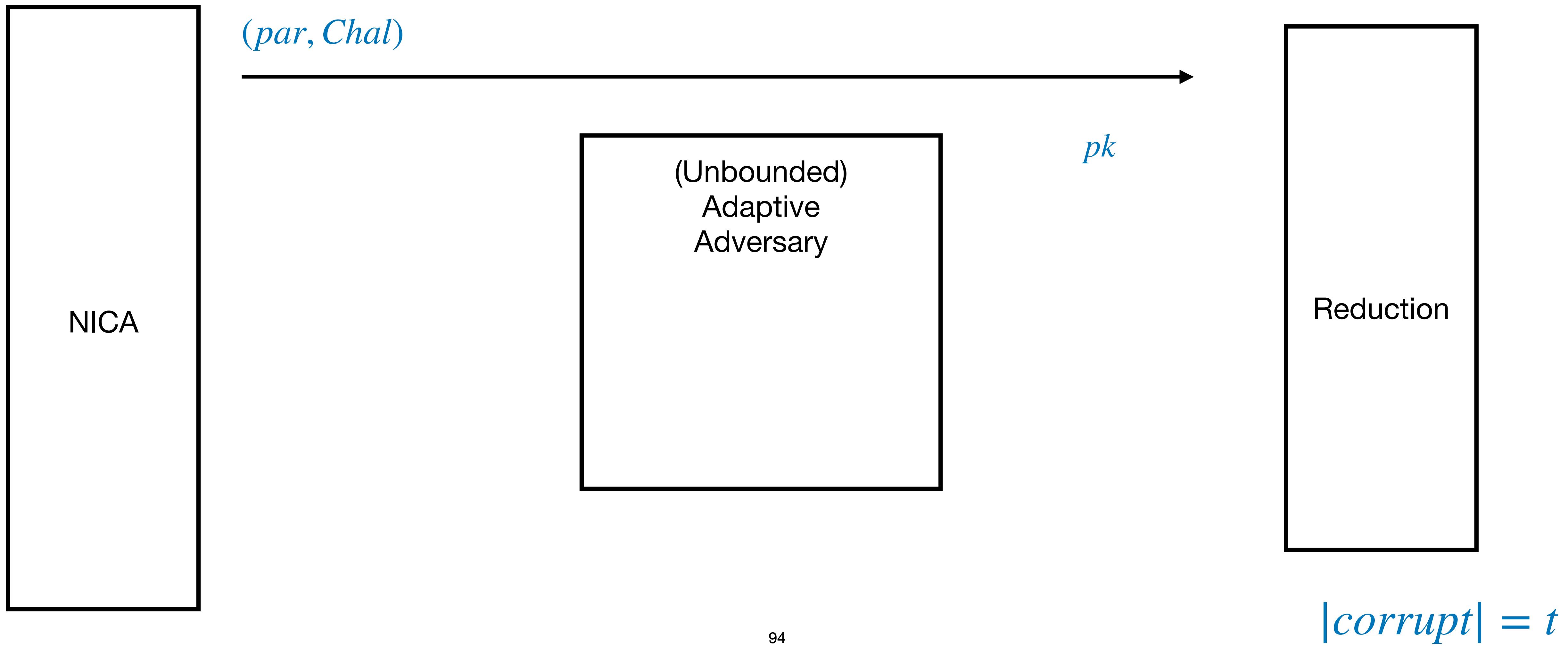
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



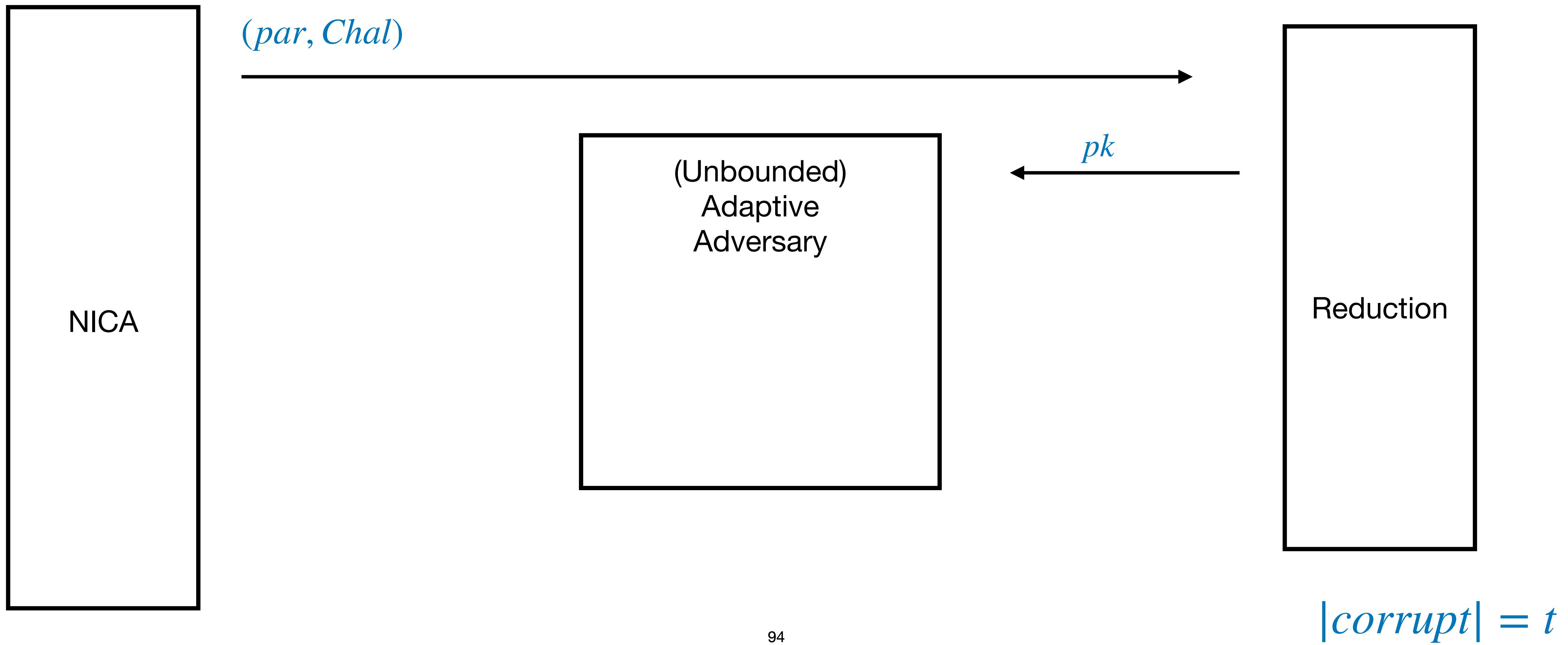
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



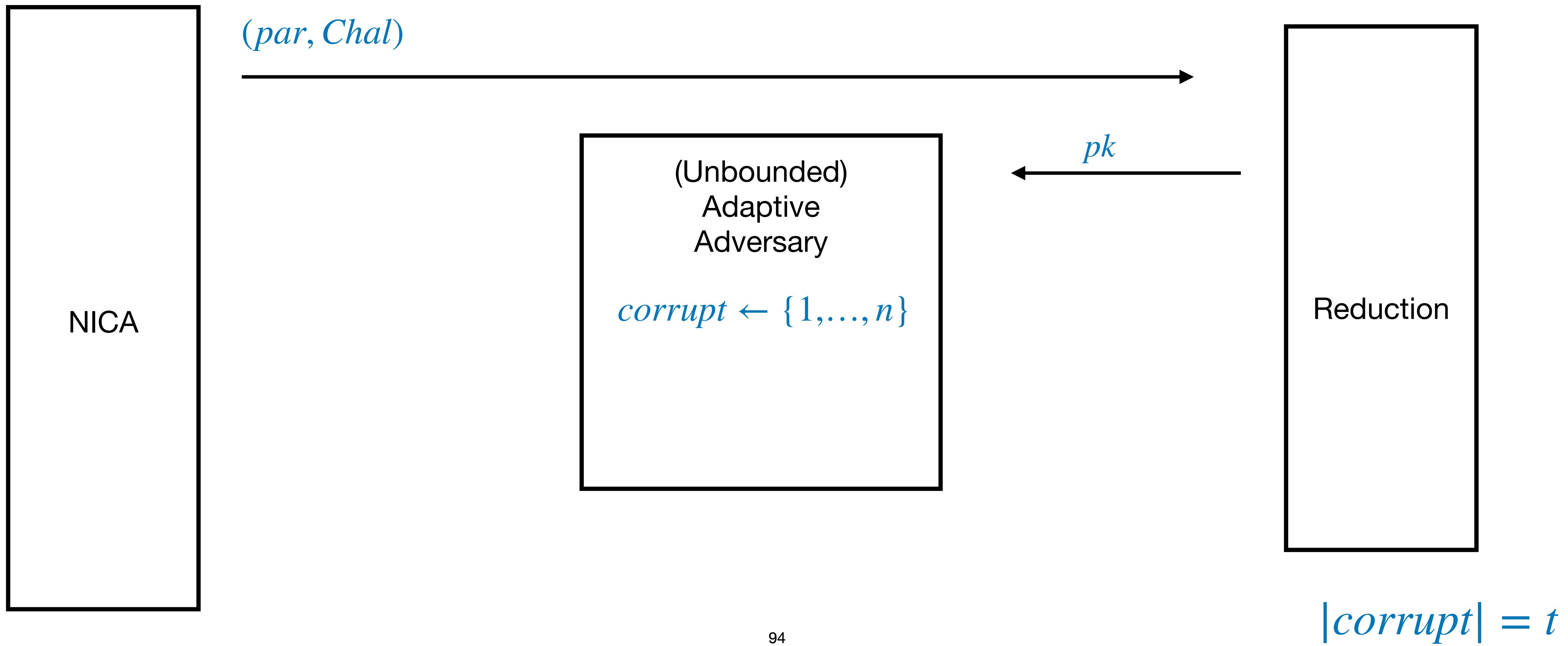
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



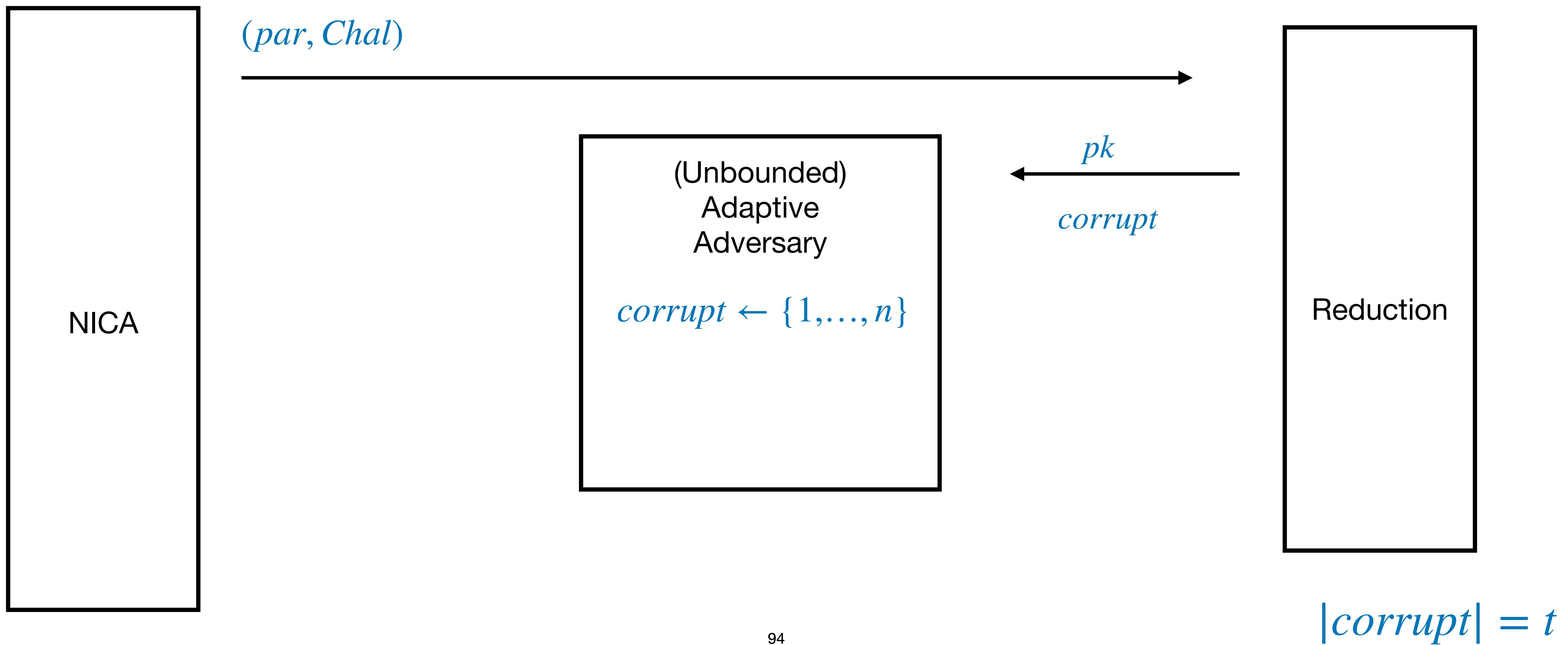
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



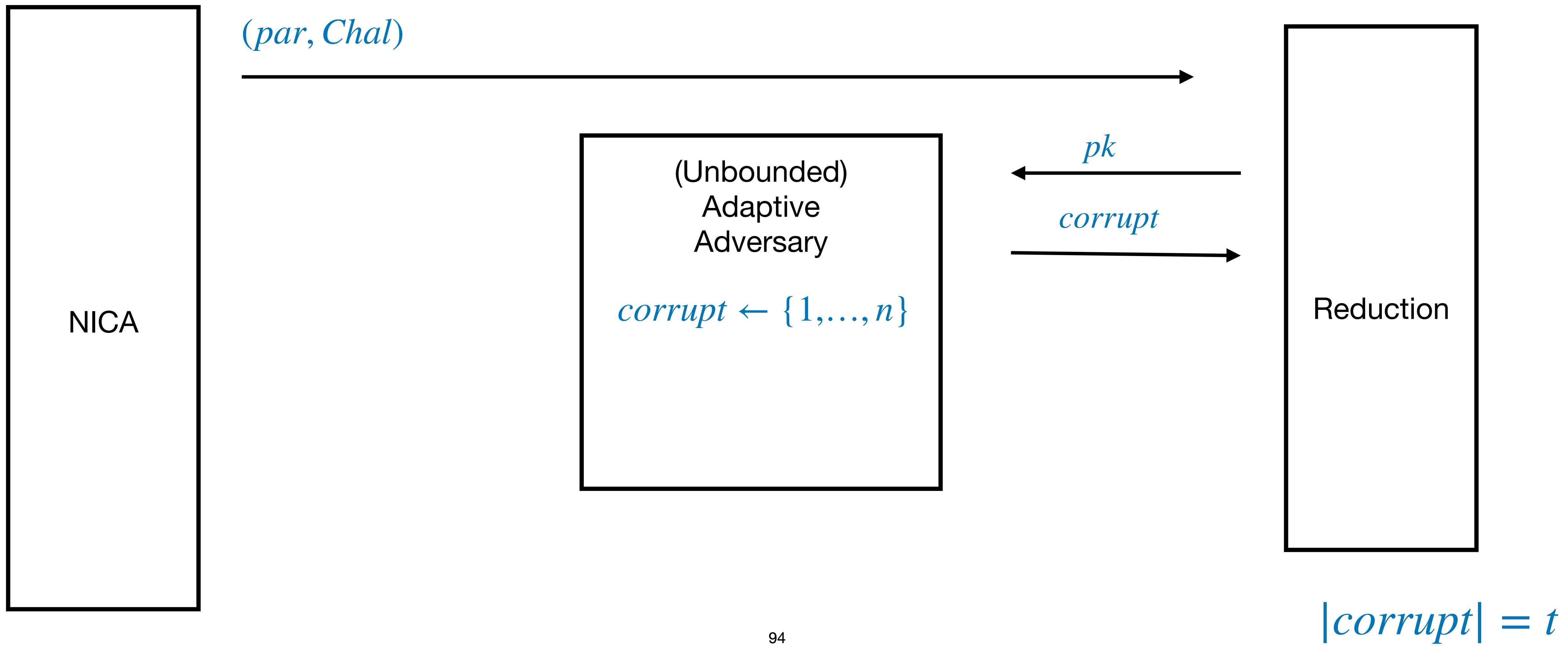
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



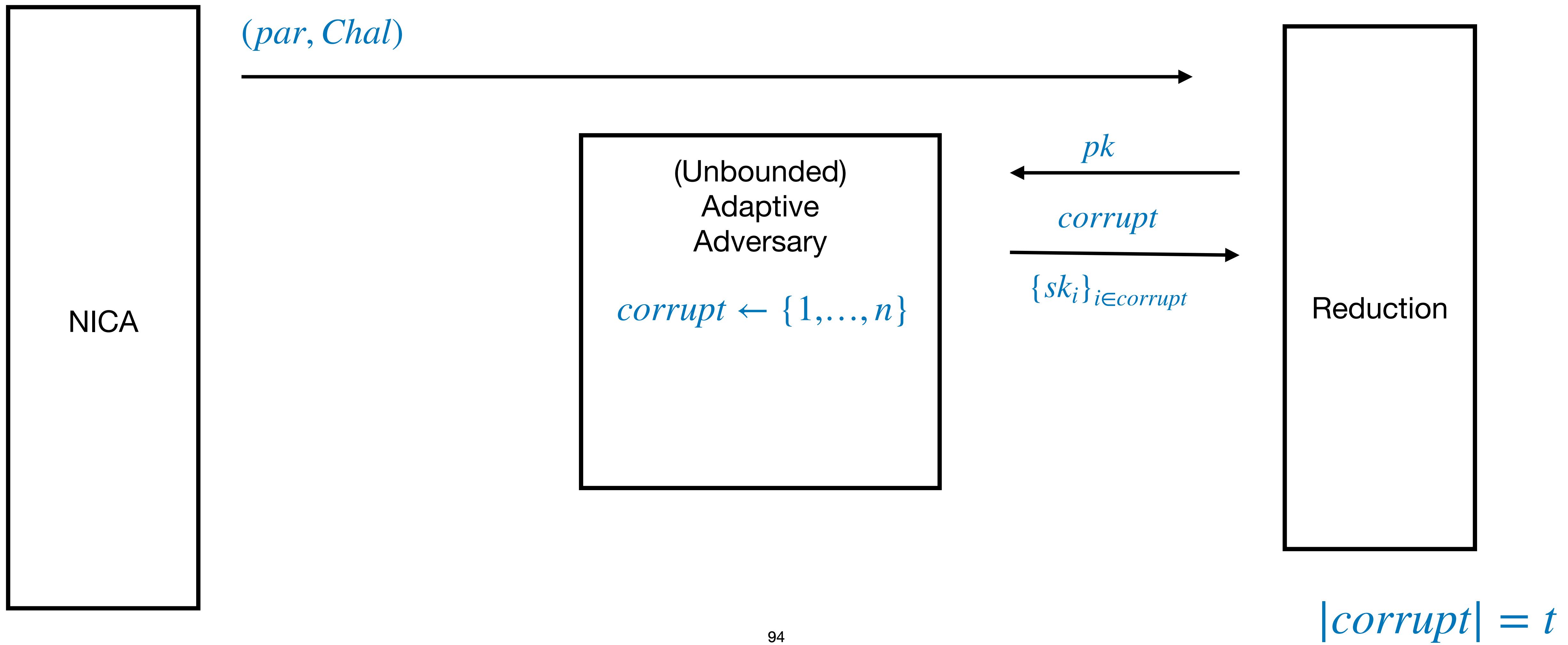
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



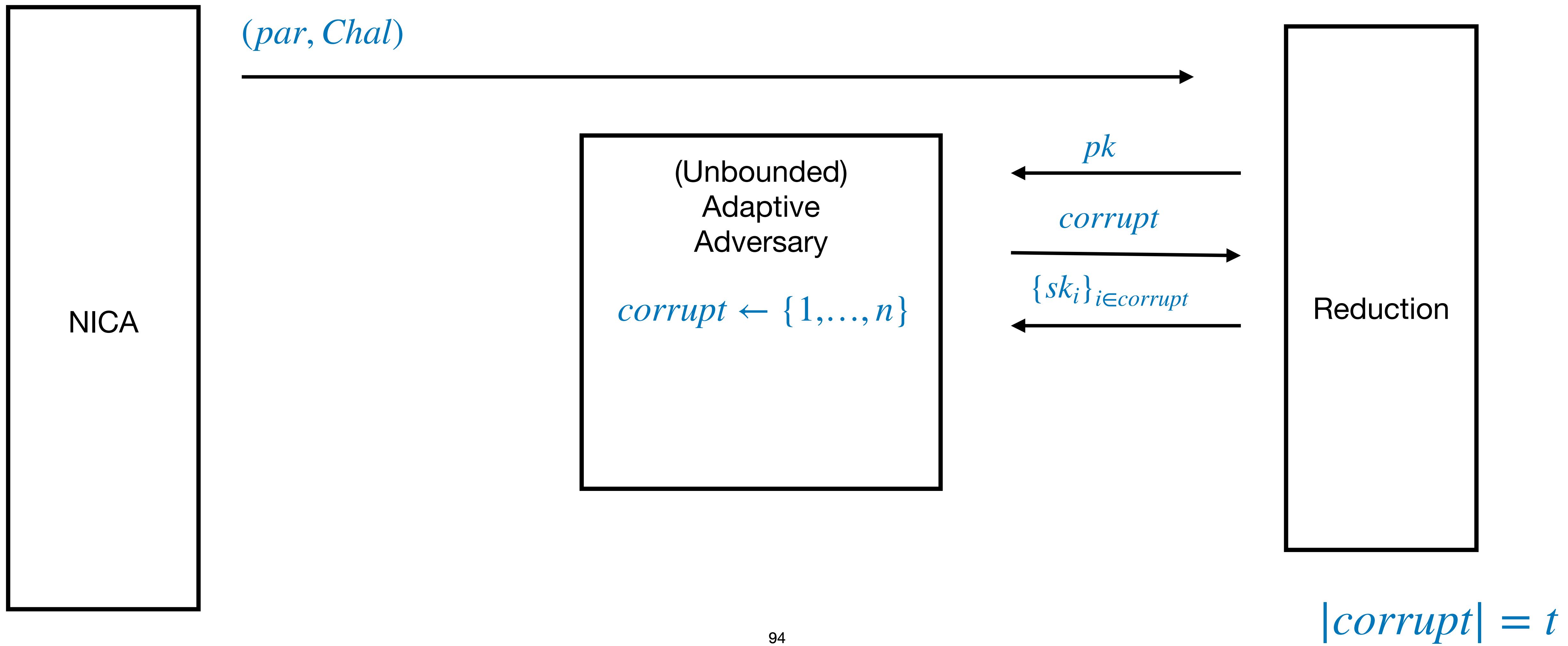
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



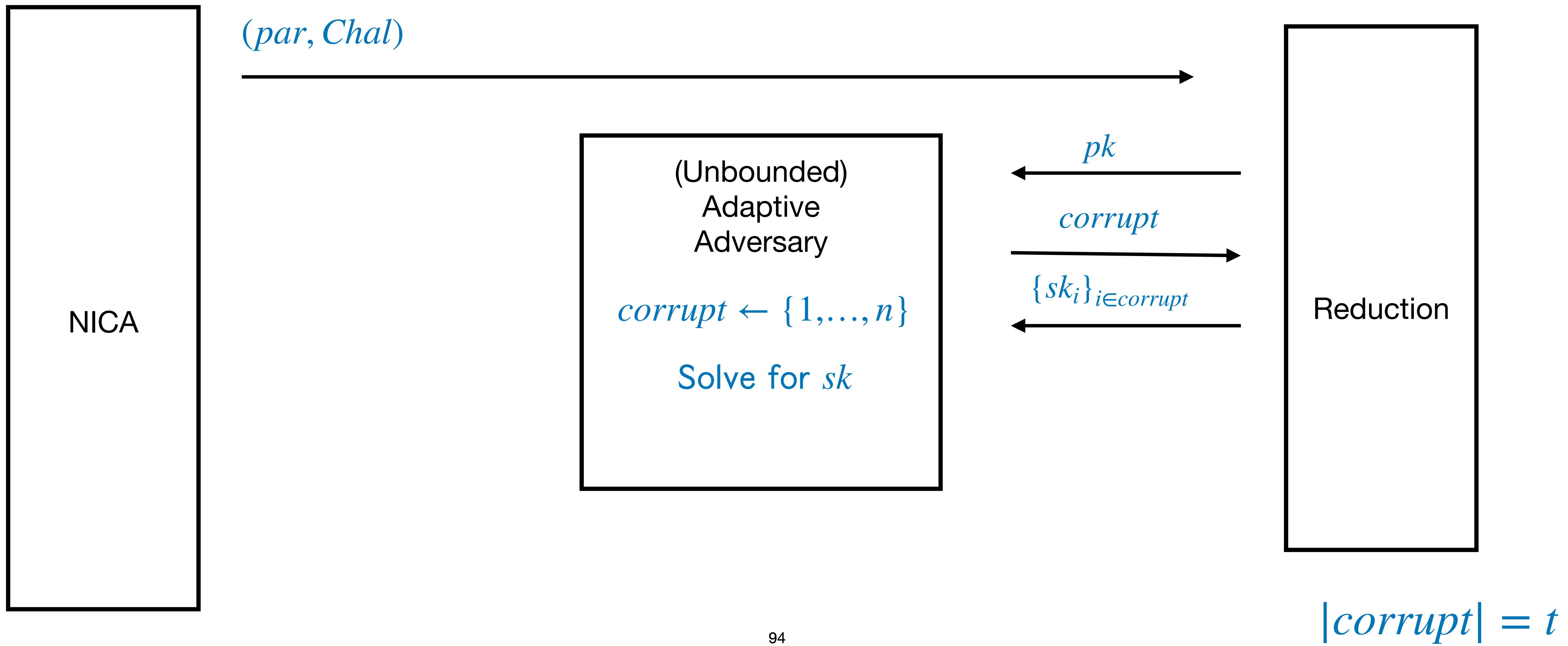
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



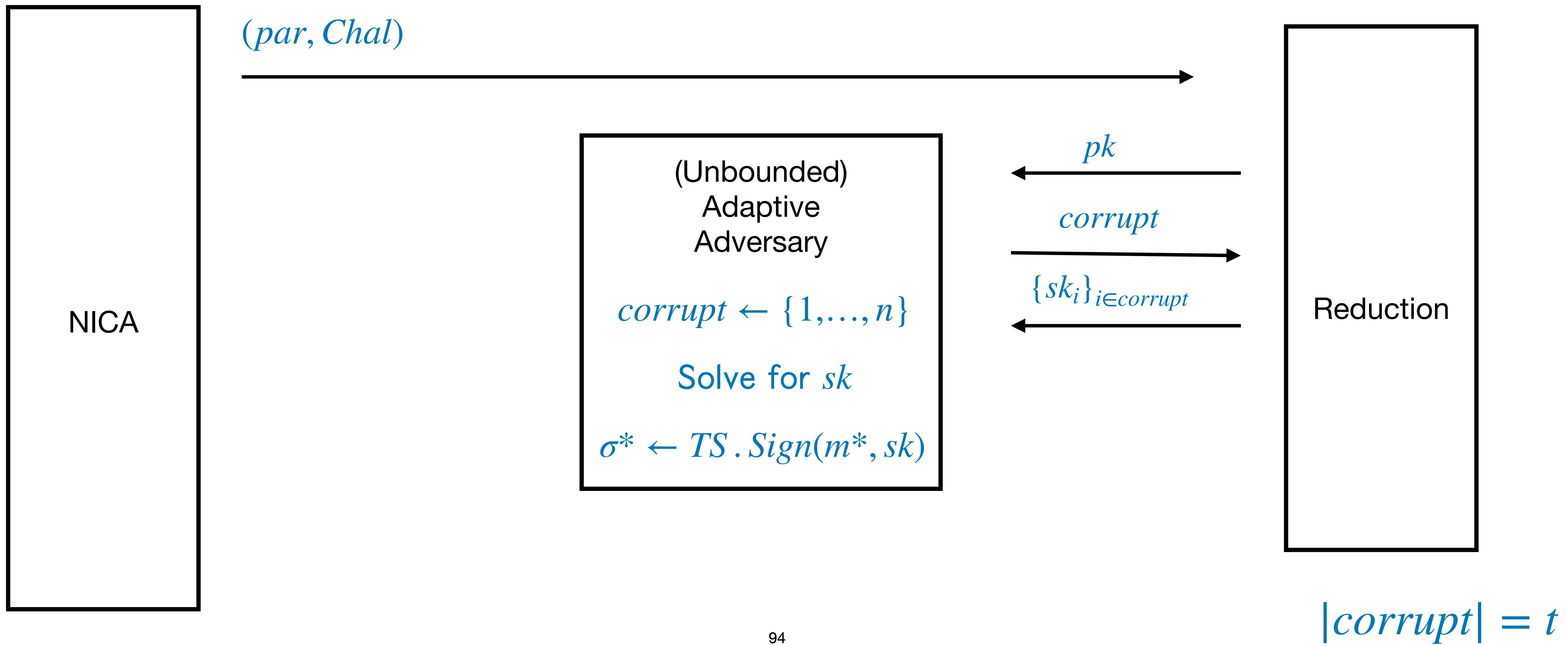
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



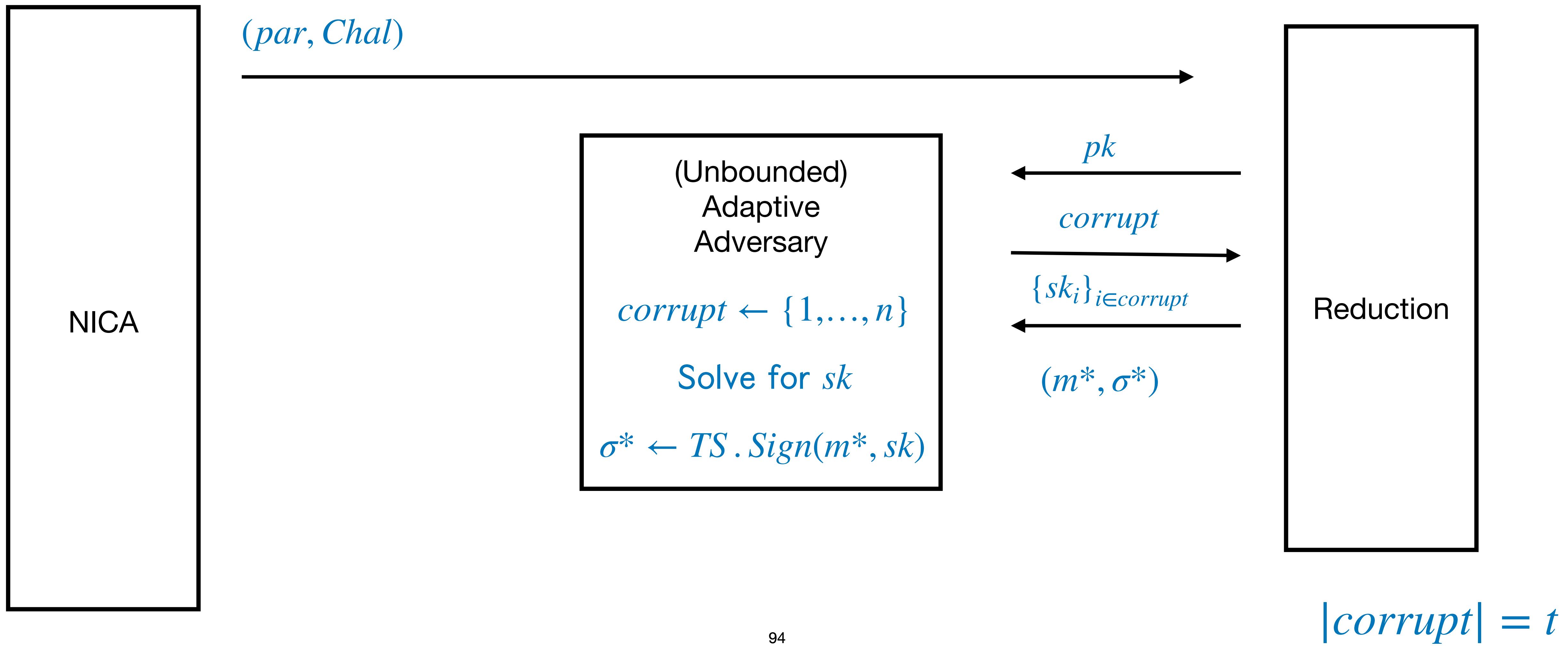
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



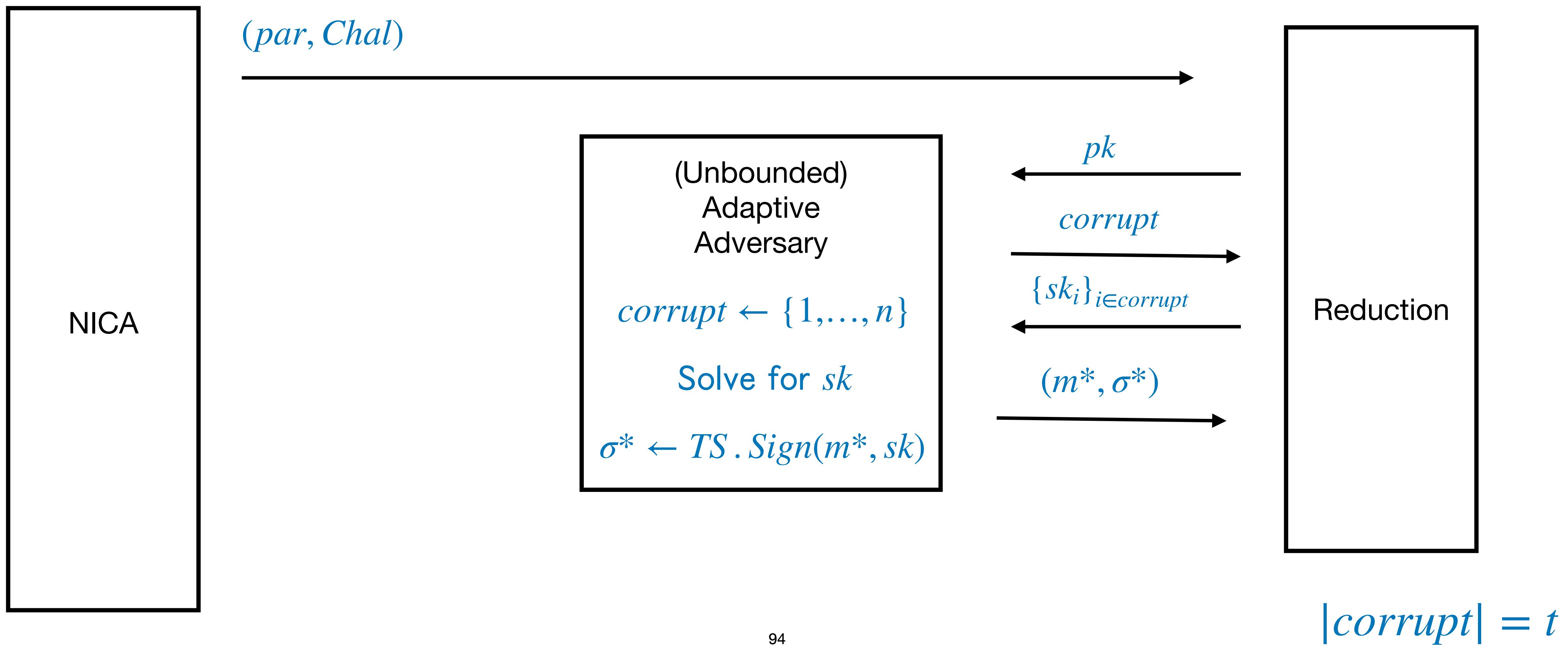
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



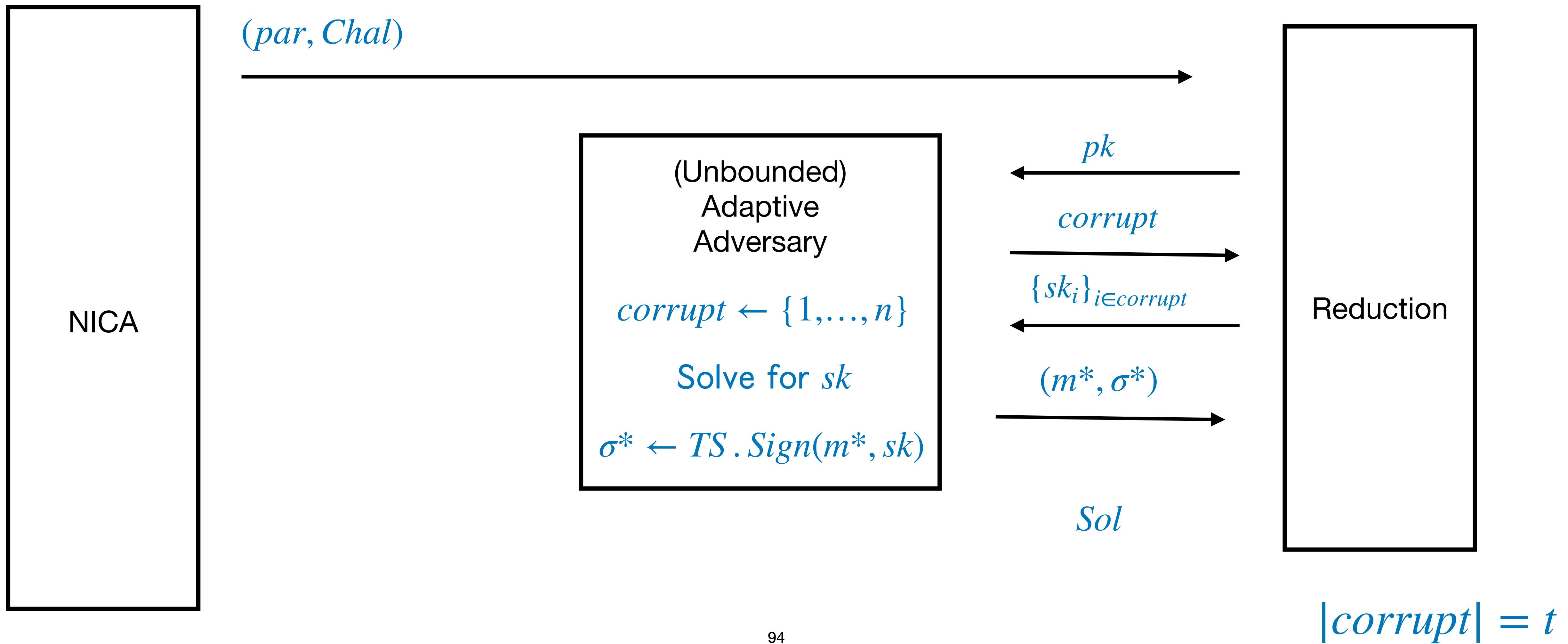
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



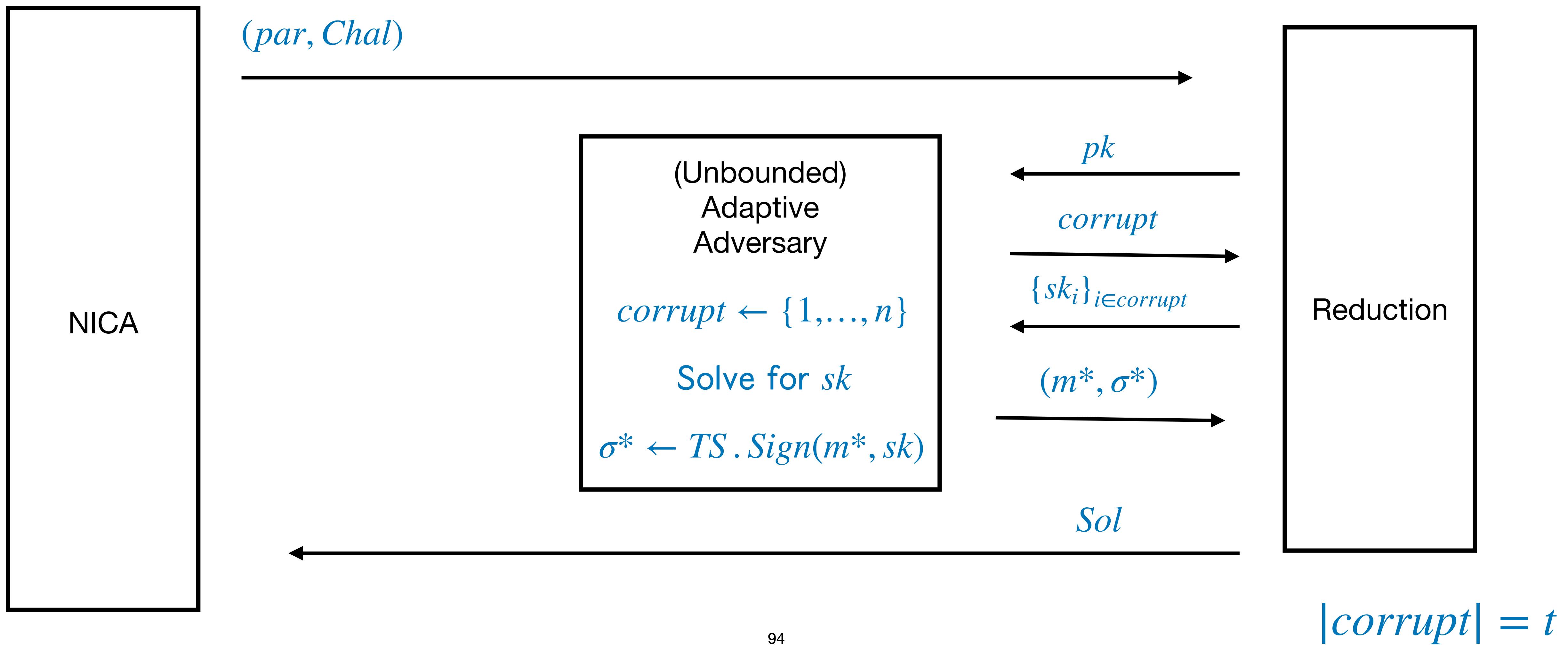
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA

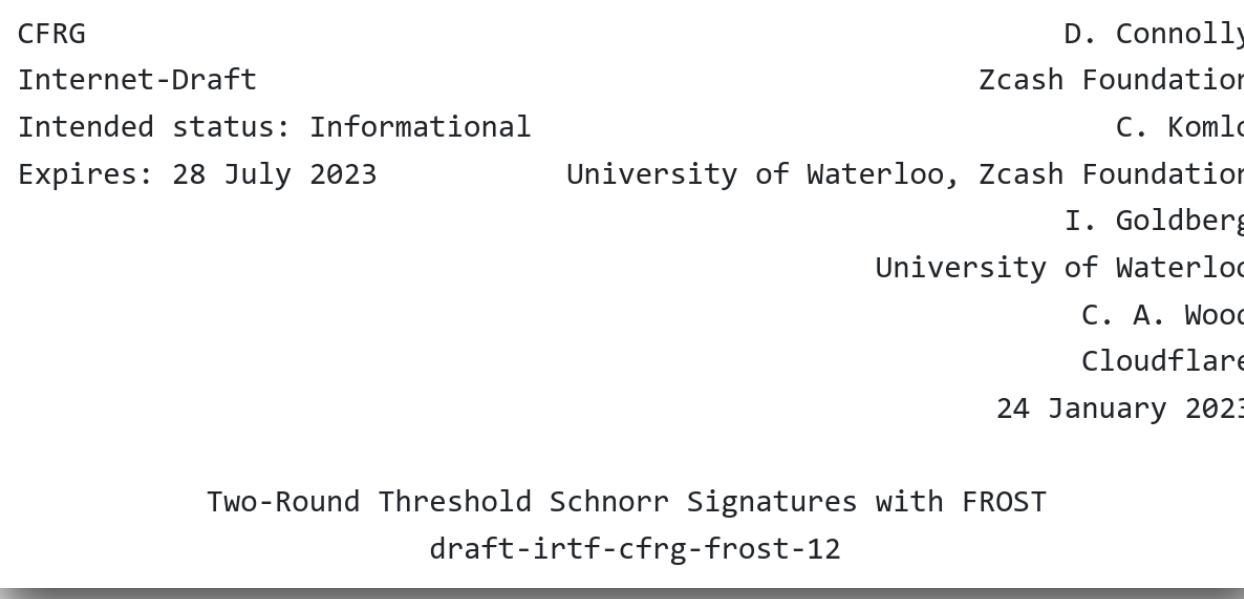


Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



Introduction

- Real-world cryptographer
- At the University of Waterloo and Near One
- Looking for areas to collaborate!



Near

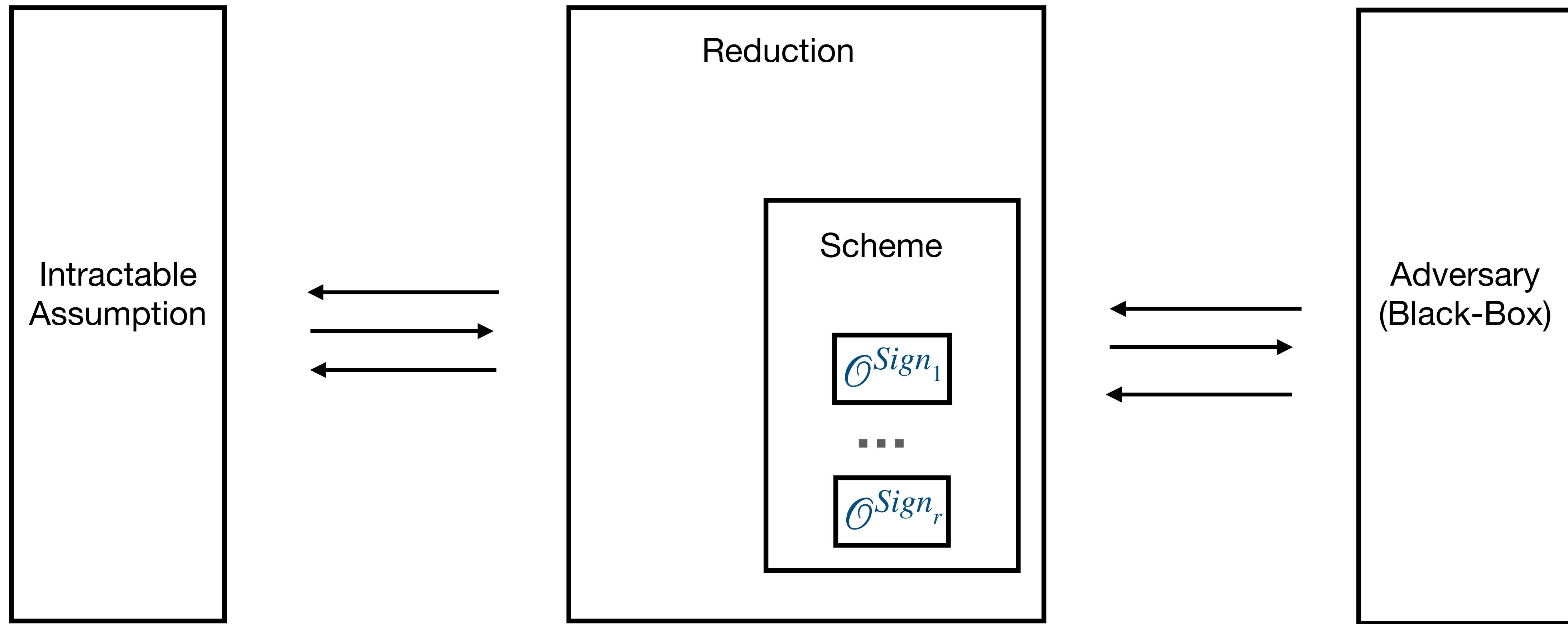
otrv4/otrv4

Off-the-Record Messaging Protocol version 4. -This is a draft- This repository is a mirror of <http://bugs.otr.im/otrv4/otrv4>



AK 17 Contributors ⚡ 7 Issues ★ 190 Stars ⚙ 21 Forks

Reduction: Prove equivalence of two problems



Example #3: Pedersen Commitments



Pedersen commitments are not injective. (Employed by: DR24)

Example #3: Pedersen Commitments



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Pedersen commitments are not injective. (Employed by: DR24)

Example #3: Pedersen Commitments



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_j x^j$

Pedersen commitments are not injective. (Employed by: DR24)

Example #3: Pedersen Commitments



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$ and $\alpha_i = F(i)$

Pedersen commitments are not injective. (Employed by: DR24)

Example #3: Pedersen Commitments



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$ and $\alpha_i = F(i)$

Step 4: Define $PK_i = g^{E(i)} h^{F(i)}$ and $PK = g^{E(0)} = g^{sk}$

Pedersen commitments are not injective. (Employed by: DR24)

Example #3: Pedersen Commitments



Step 1: Sample $(sk, \{a_1, \dots, a_{t-1}\}) \leftarrow \mathbb{Z}_q^t$

Step 2: Define $E(x) = sk + \sum_{j=1}^{t-1} a_j x^j$ and $F(x) = 0 + \sum_{j=1}^{t-1} a_j x^j$

Step 3: Define $sk_i = E(i)$ and $\alpha_i = F(i)$

Step 4: Define $PK_i = g^{E(i)} h^{F(i)}$ and $PK = g^{E(0)} = g^{sk}$

Pedersen commitments are not injective. (Employed by: DR24)

$((i, sk_i, \alpha_i), \{\underline{PK_i}\}_{i=1}^n, PK)$

Implications for Threshold Schnorr Signatures

Scheme	Static Assumptions	(Full) Adaptive Assumptions	Our Results Apply
Lindell22 Classic S [M23]	Schnorr	✗	
Sparkle [CKM23]	DL+ROM	AOMDL+ROM, $t/2$	✓
FROST [KG20, BCKMTZ22]		AOMDL + ROM	
FROST2/3 [CKM21, CGRS23]	(A)OMDL+ROM	+ AGM + LVDR	

Implications

	Scheme	Static Assumptions	Adaptive Assumptions (Full)	Our Results Apply
Schnorr	Lindell22	Schnorr		
	Sparkle [CKM23]	DL+ROM	AOMDL+ROM, $t/2$	
	FROST [KG20, BCKMTZ22]	(A)OMDL+ROM	AOMDL + ROM + AGM + LVDR	
	FROST2/3 [CKM21, CGRS23]			
Non-Schnorr	Twinkle [BLTWZ23]		OMCDH/DDH+ROM	
	Threshold BLS [B03, BL22]	GDH+ROM	OMDL+AGM+ROM	
	Threshold ECDSA [GG18, CGGMP20]	Various		

Examples

Examples

- Discrete Logarithm Assumption (DL)

Examples

- Discrete Logarithm Assumption (DL)
- Computational Diffie Hellman Assumption (CDH)

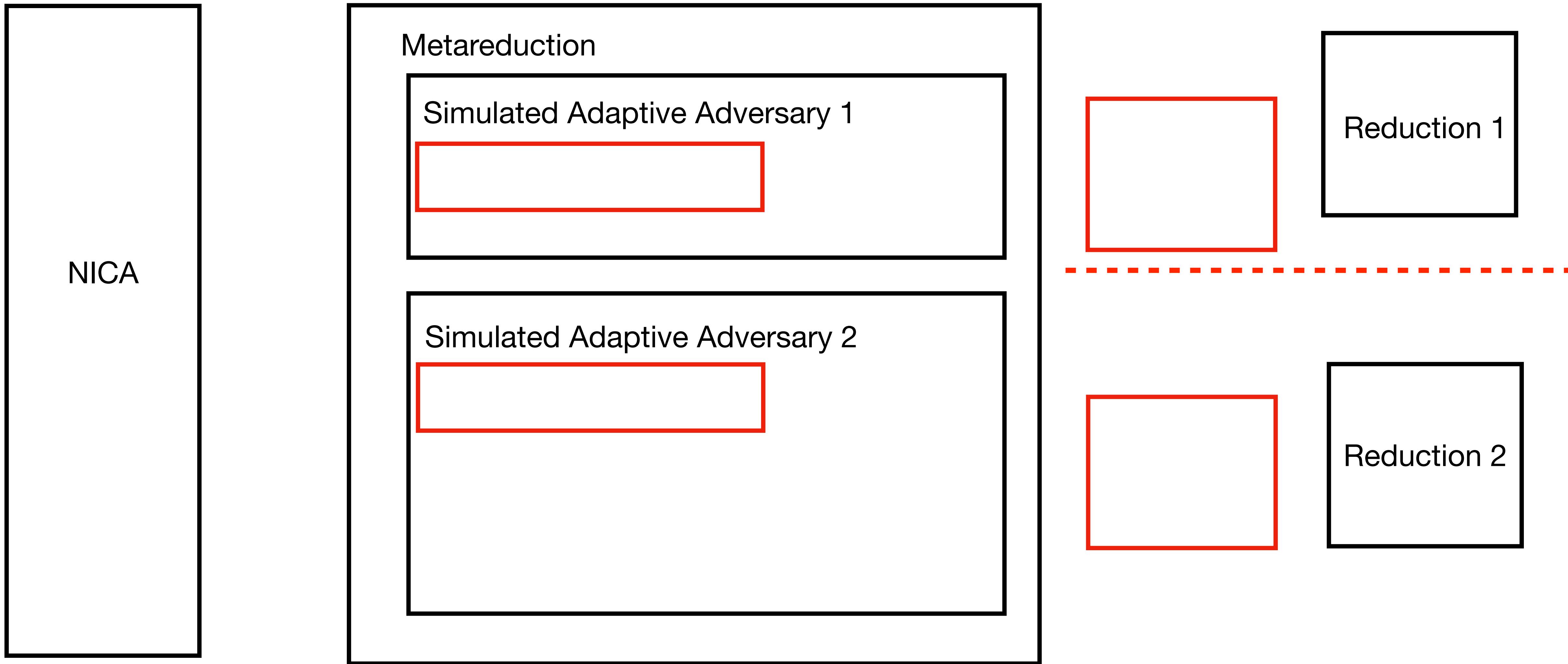
Examples

- Discrete Logarithm Assumption (DL)
- Computational Diffie Hellman Assumption (CDH)
- q-Strong Diffie-Hellman Assumption (q-SDH)

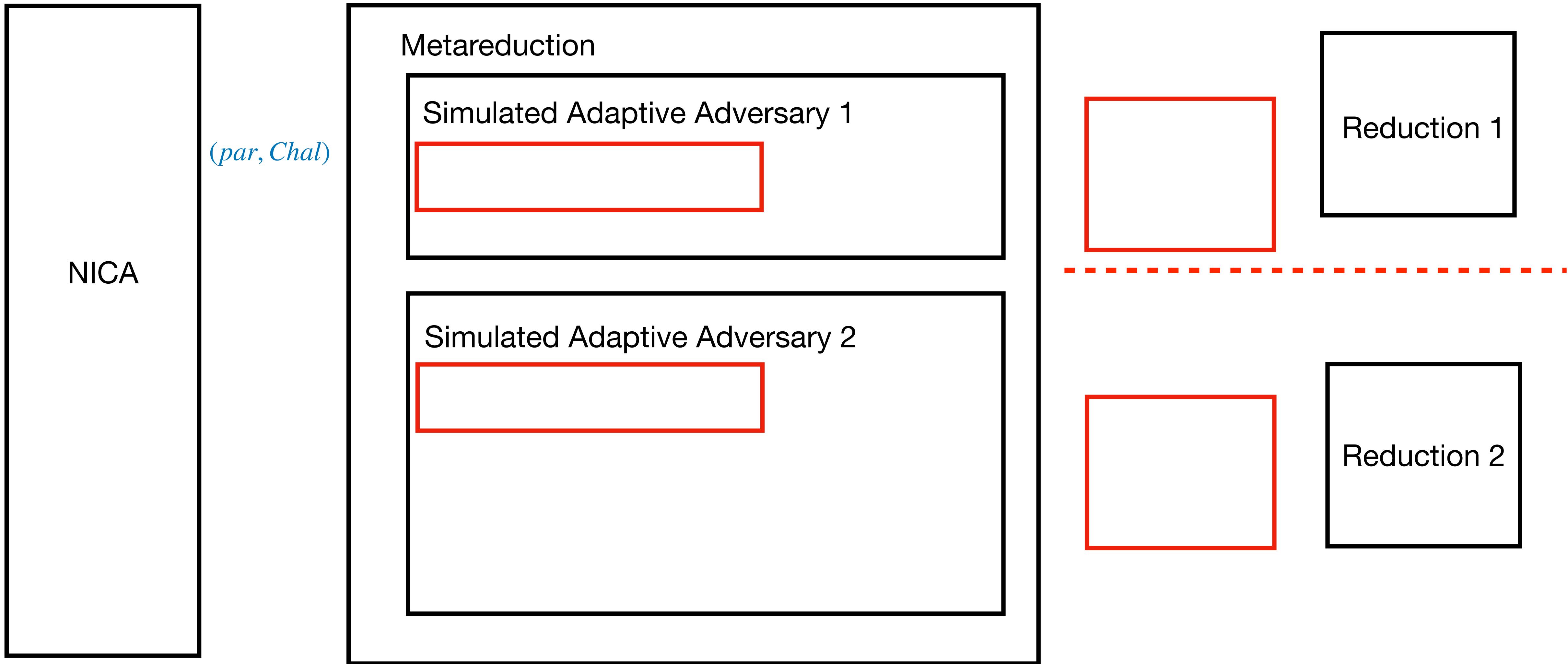
Examples

- Discrete Logarithm Assumption (DL)
- Computational Diffie Hellman Assumption (CDH)
- q-Strong Diffie-Hellman Assumption (q-SDH)
- q-Bilinear Diffie-Hellman Inversion Assumption (q-BDHI)

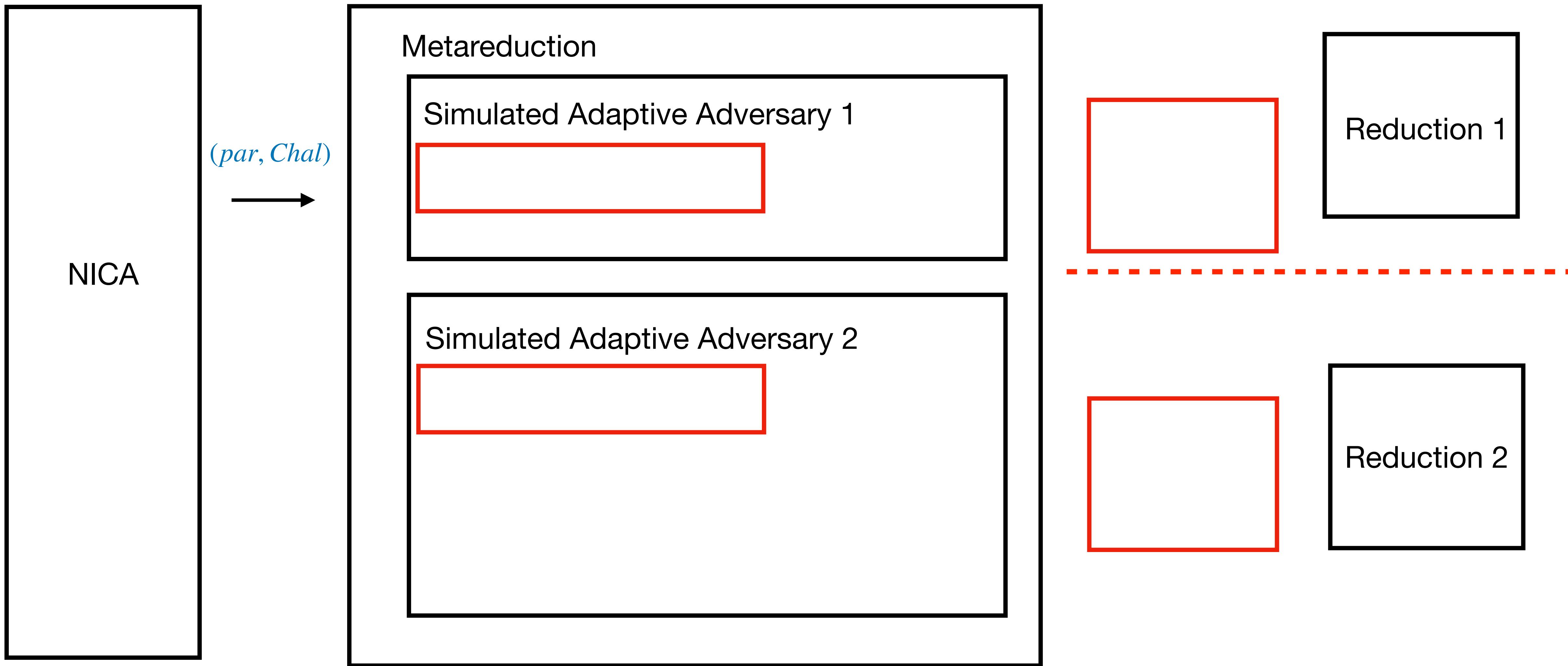
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



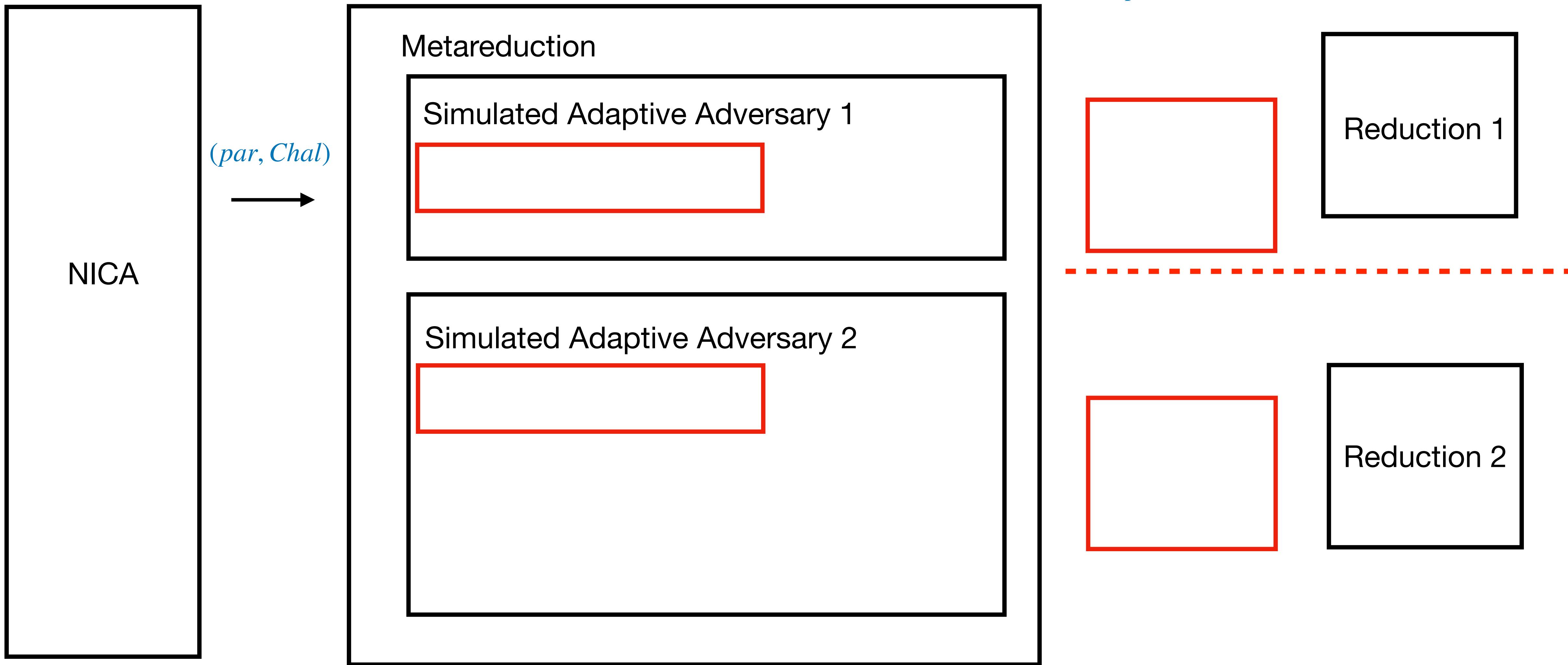
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



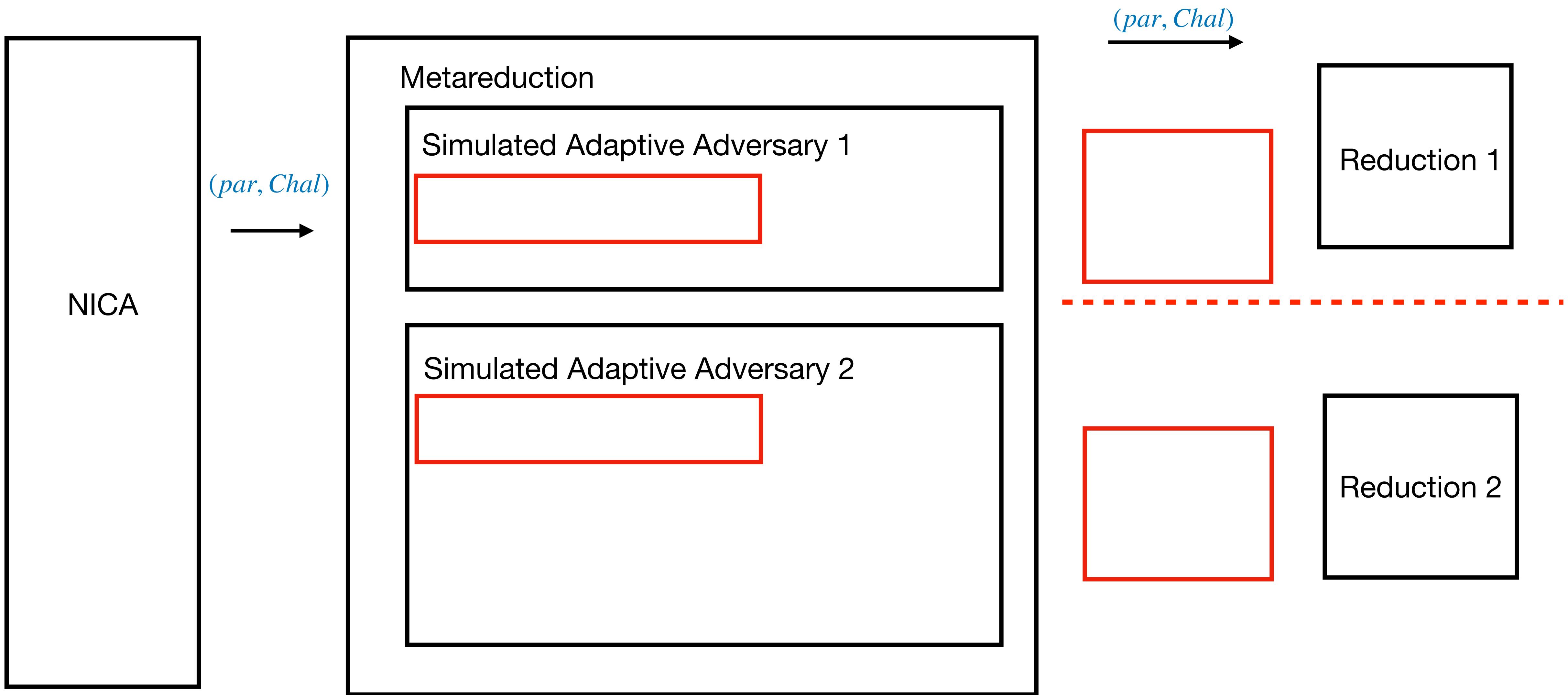
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



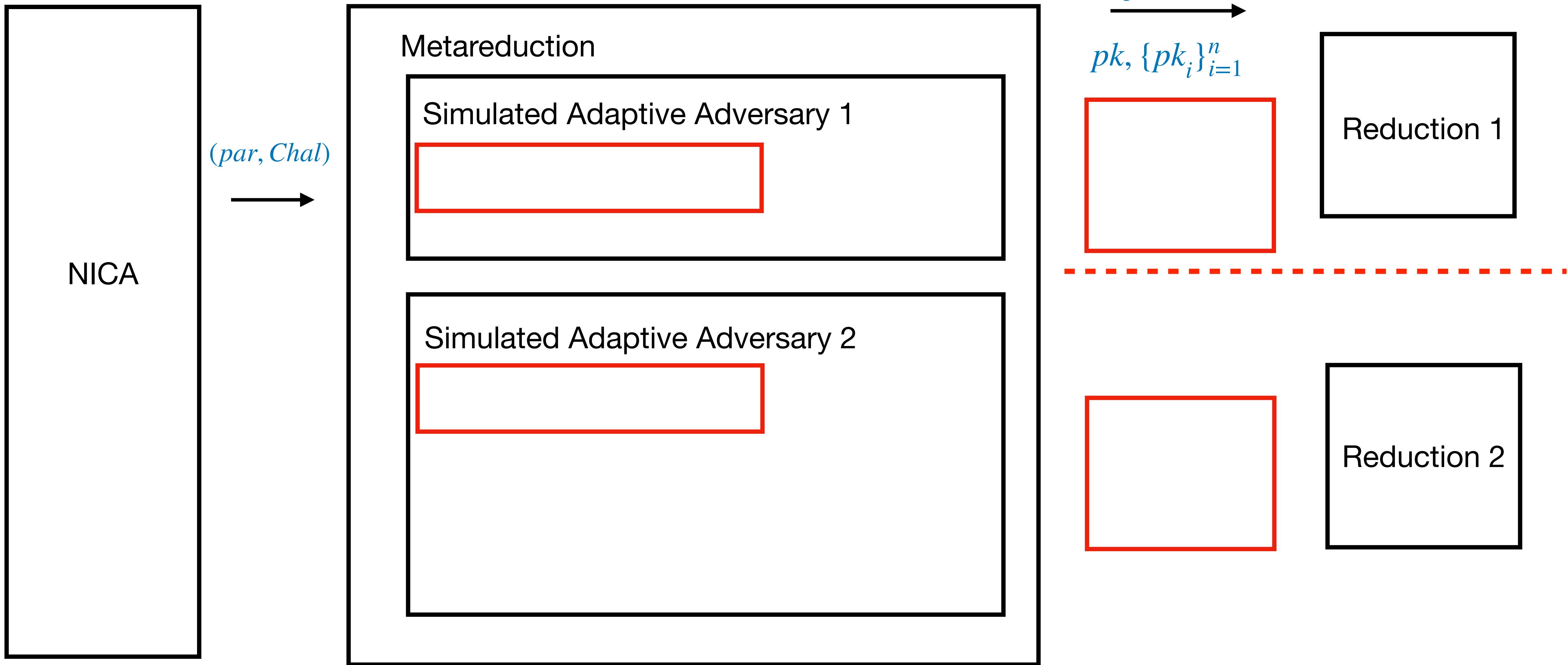
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



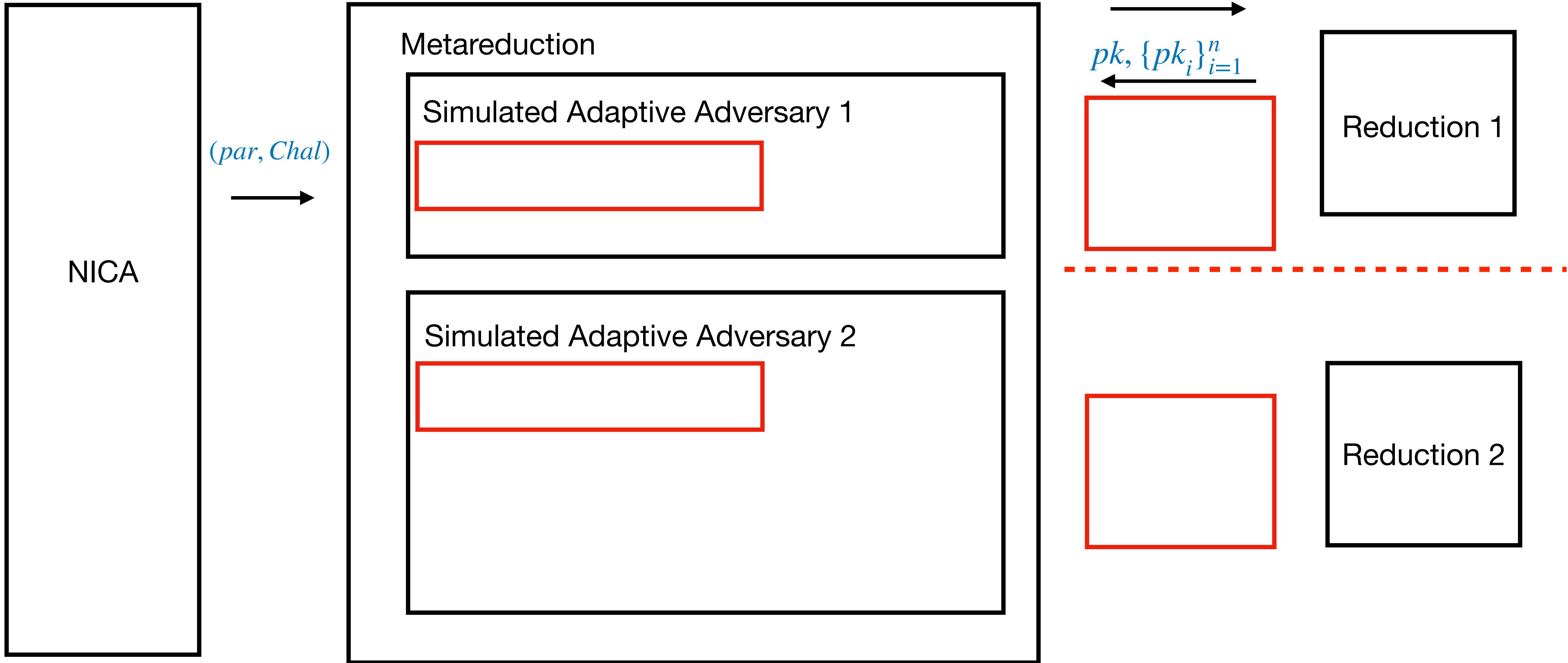
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



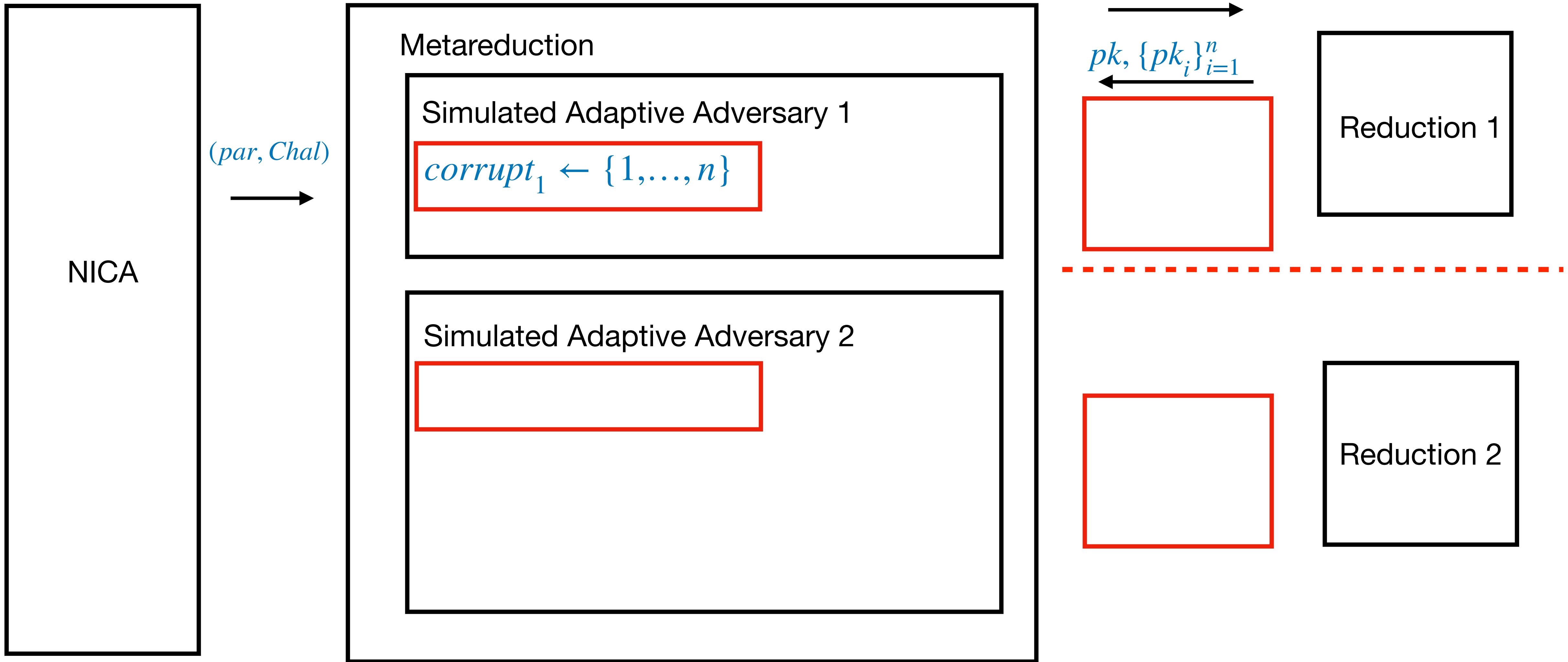
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



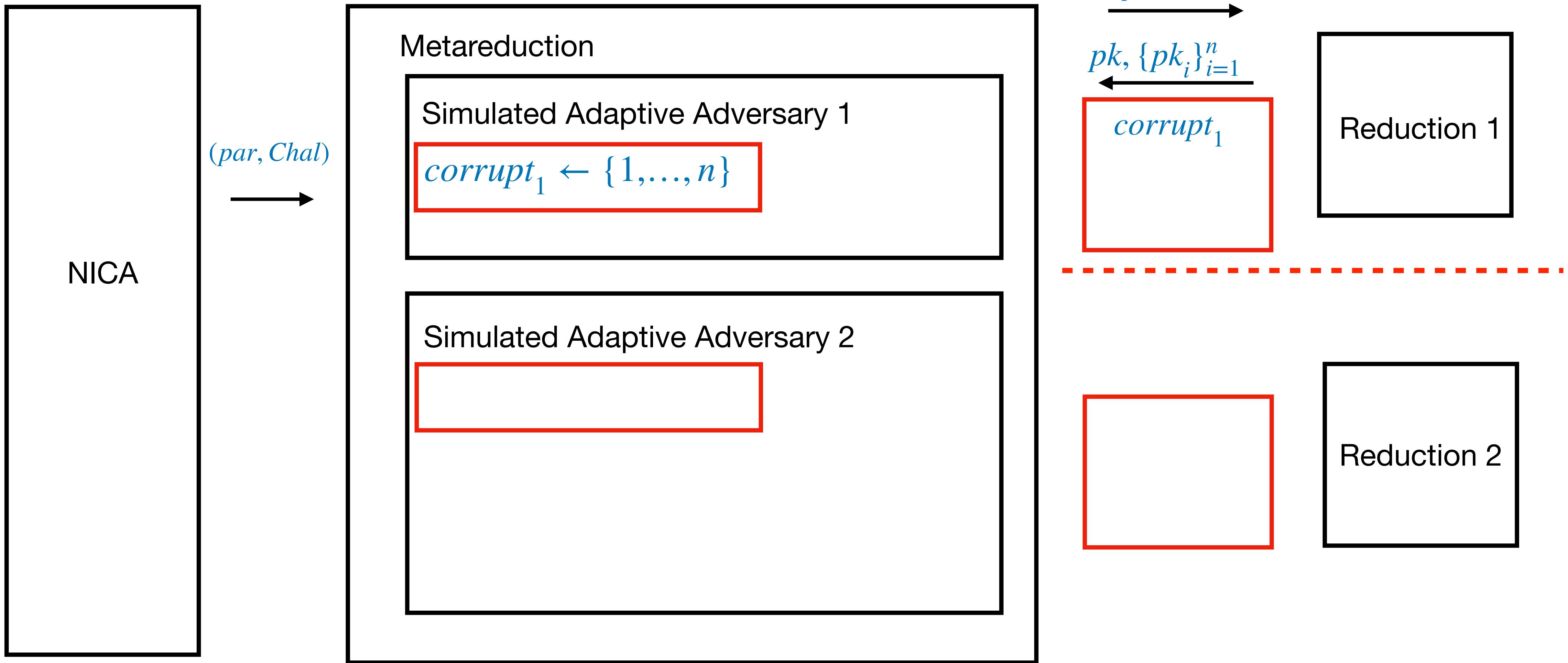
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



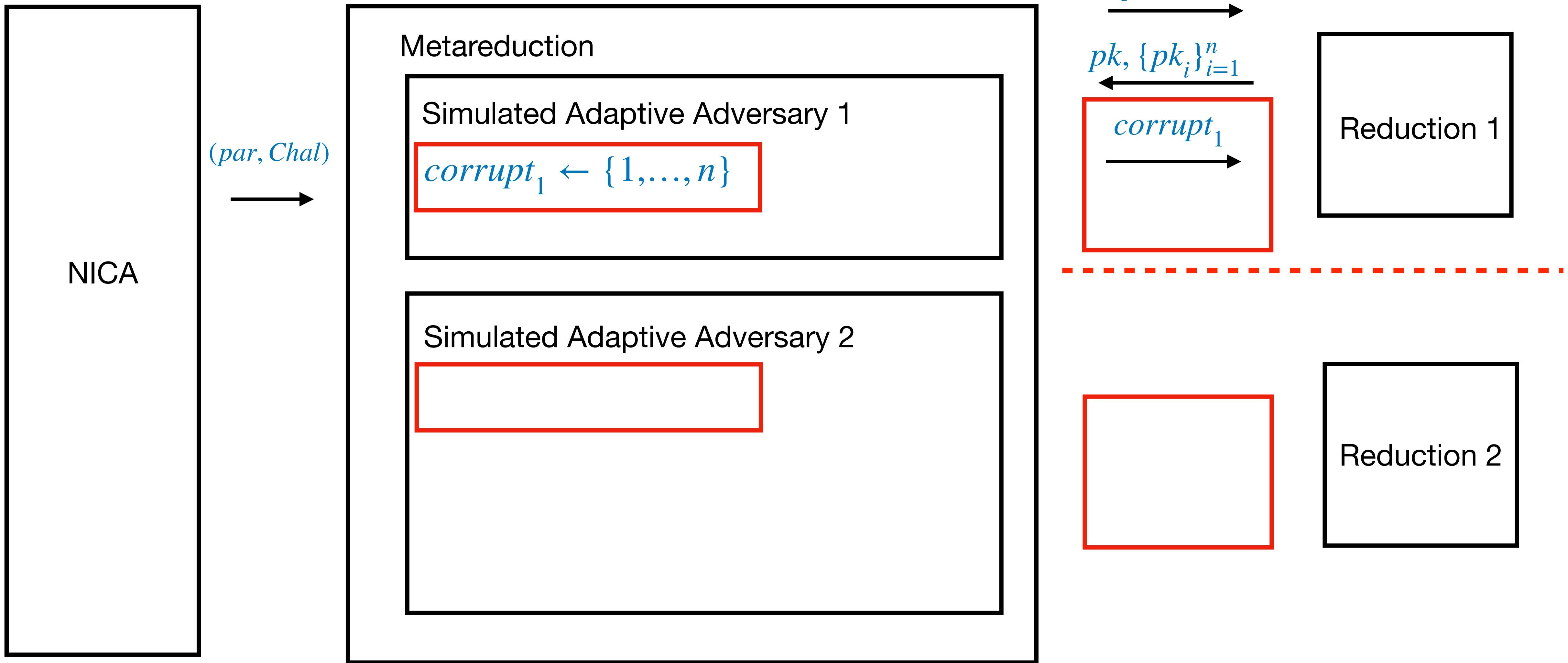
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



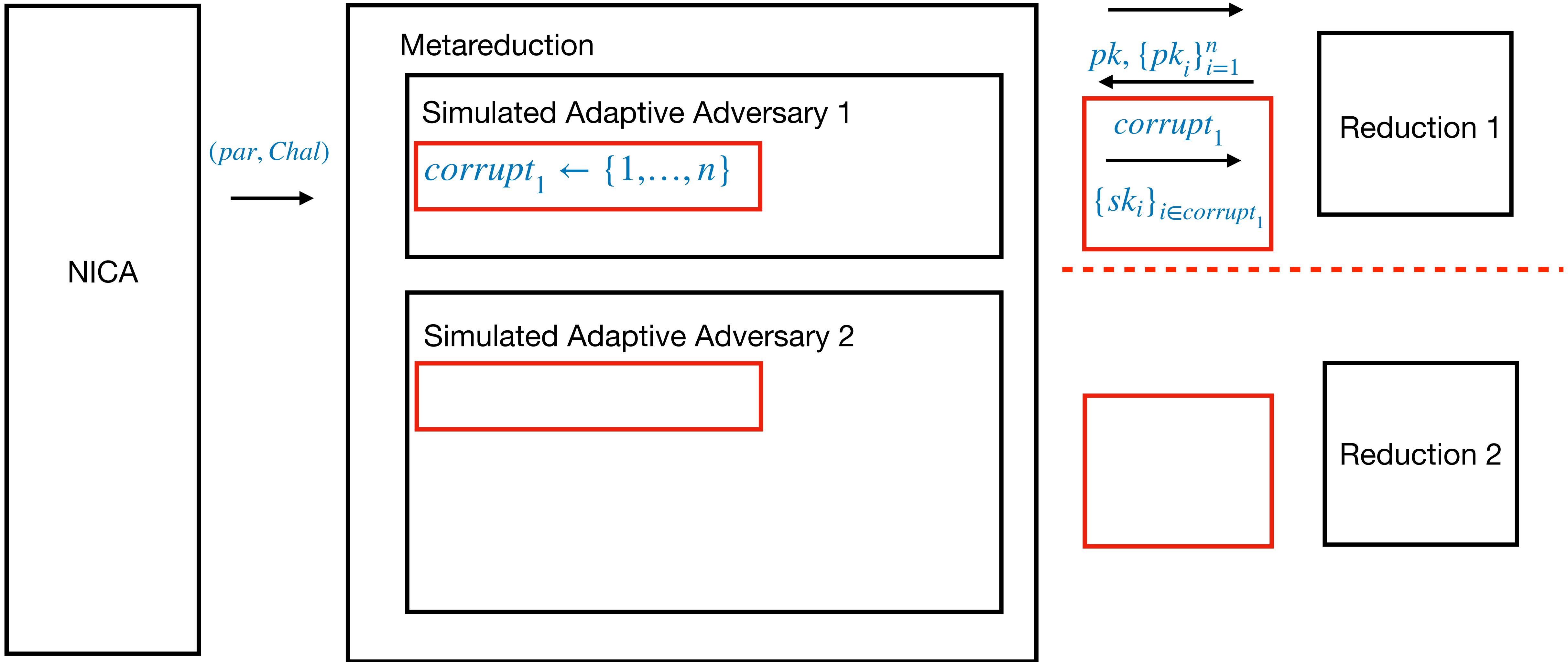
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



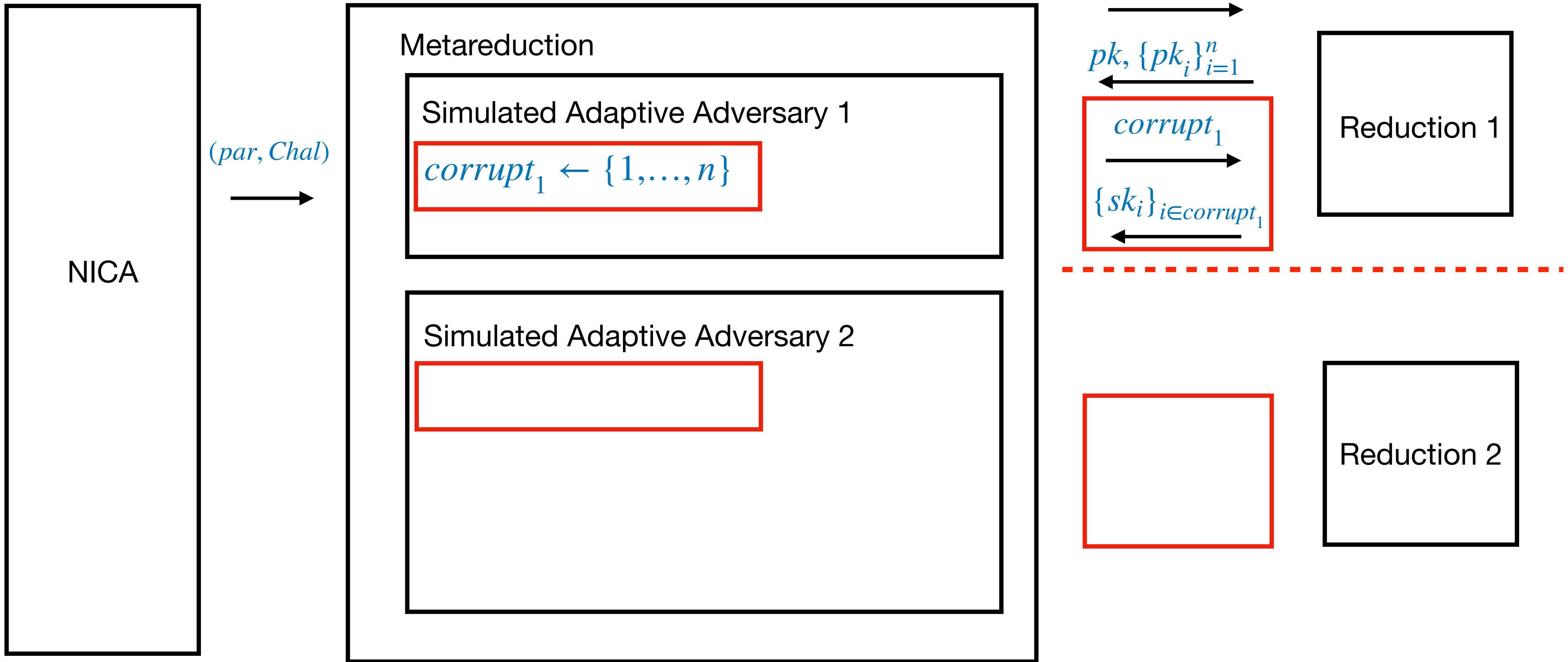
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



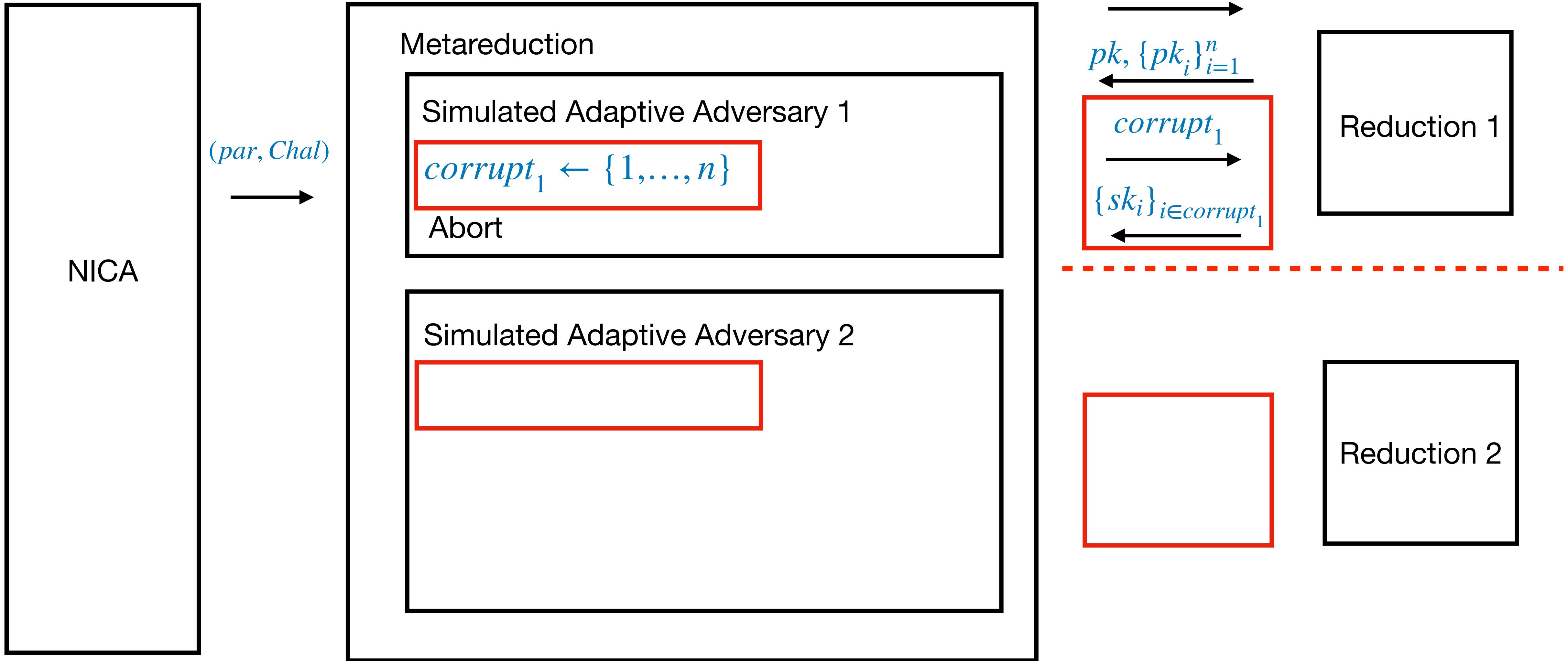
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



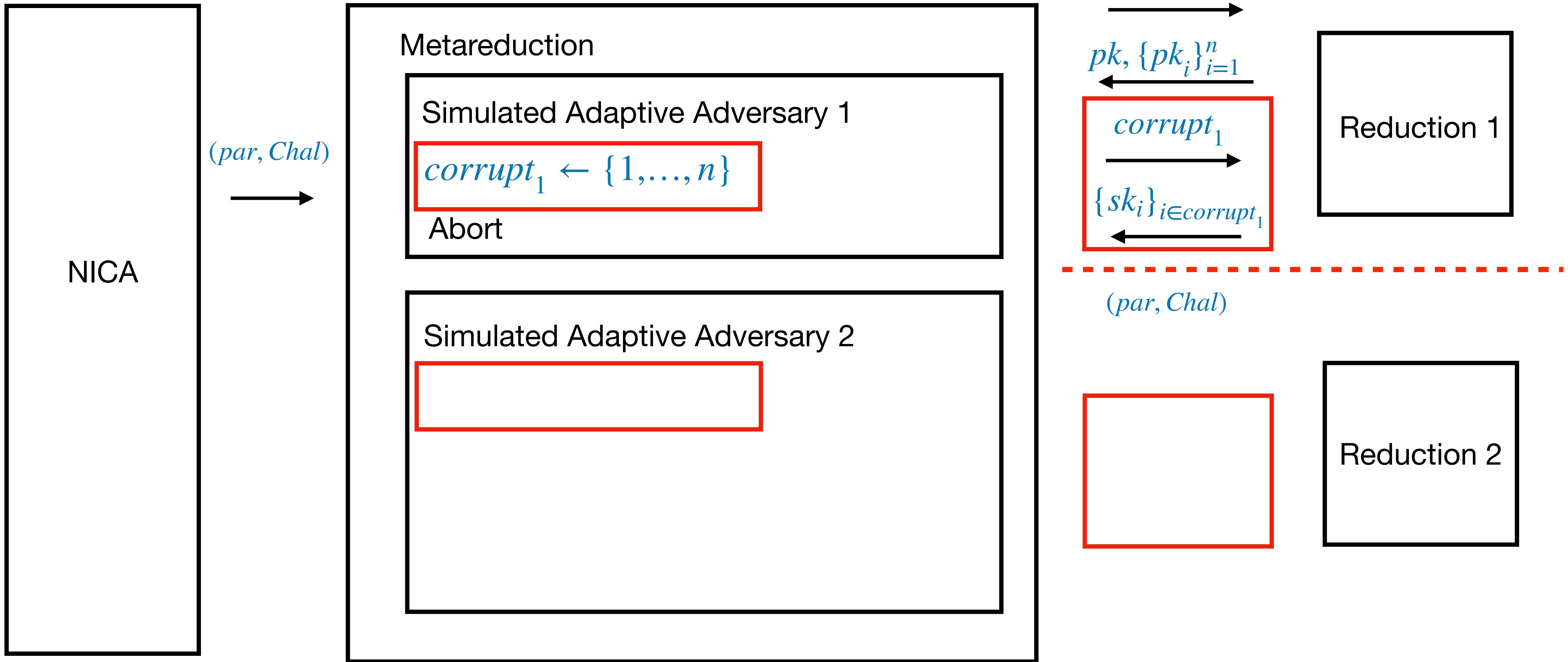
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



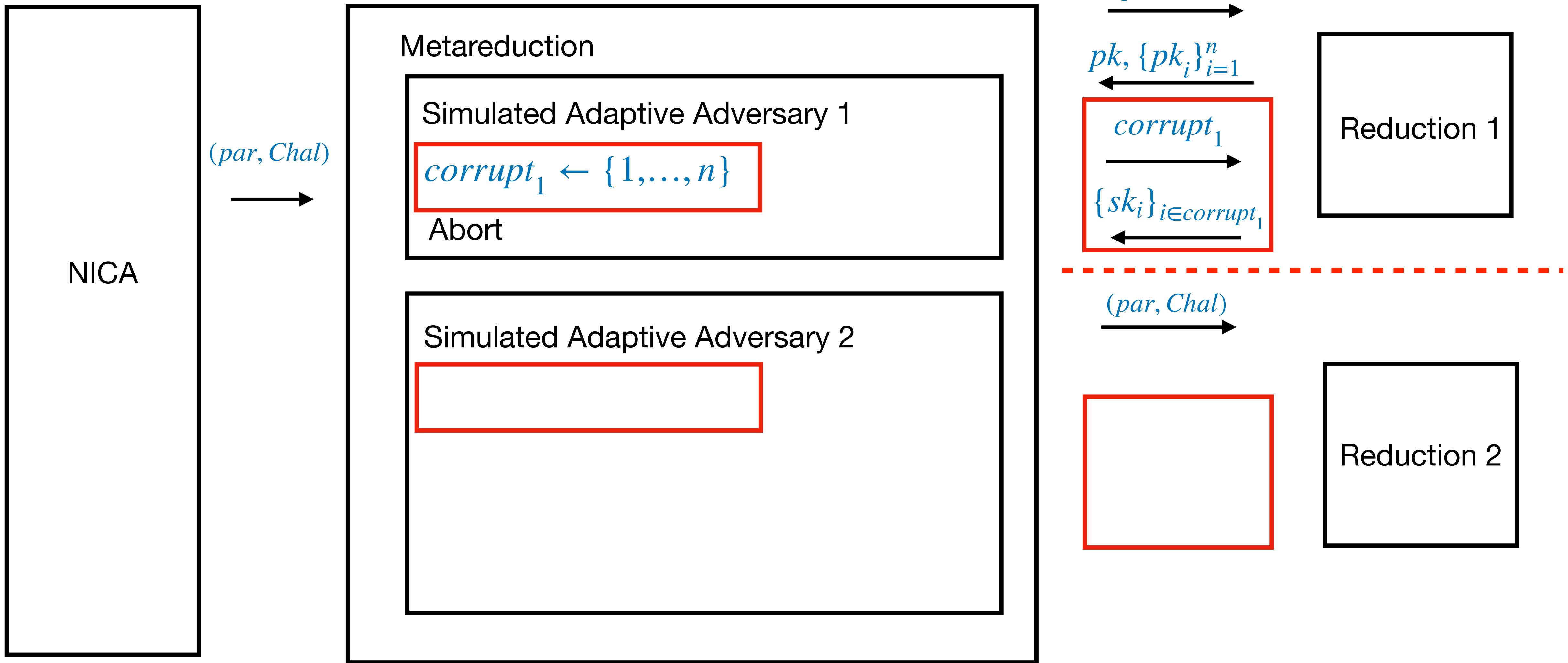
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



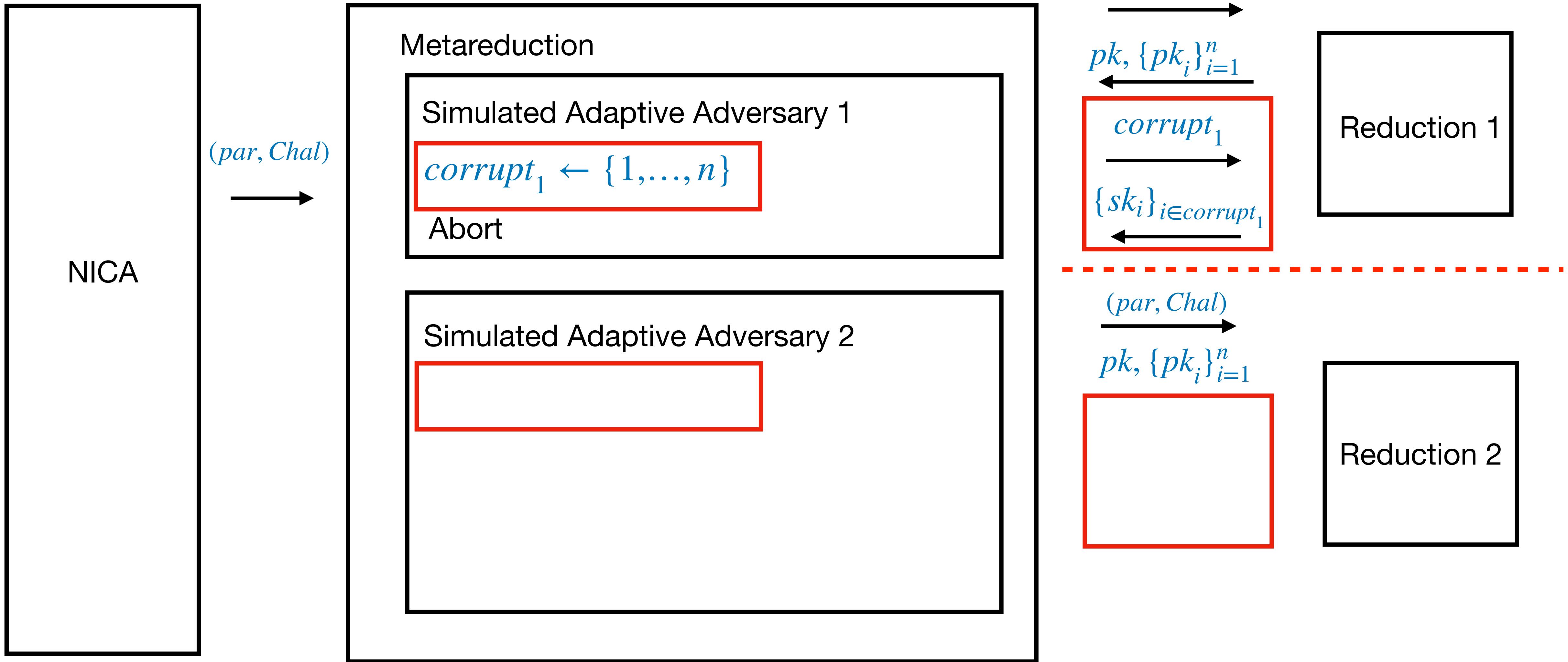
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



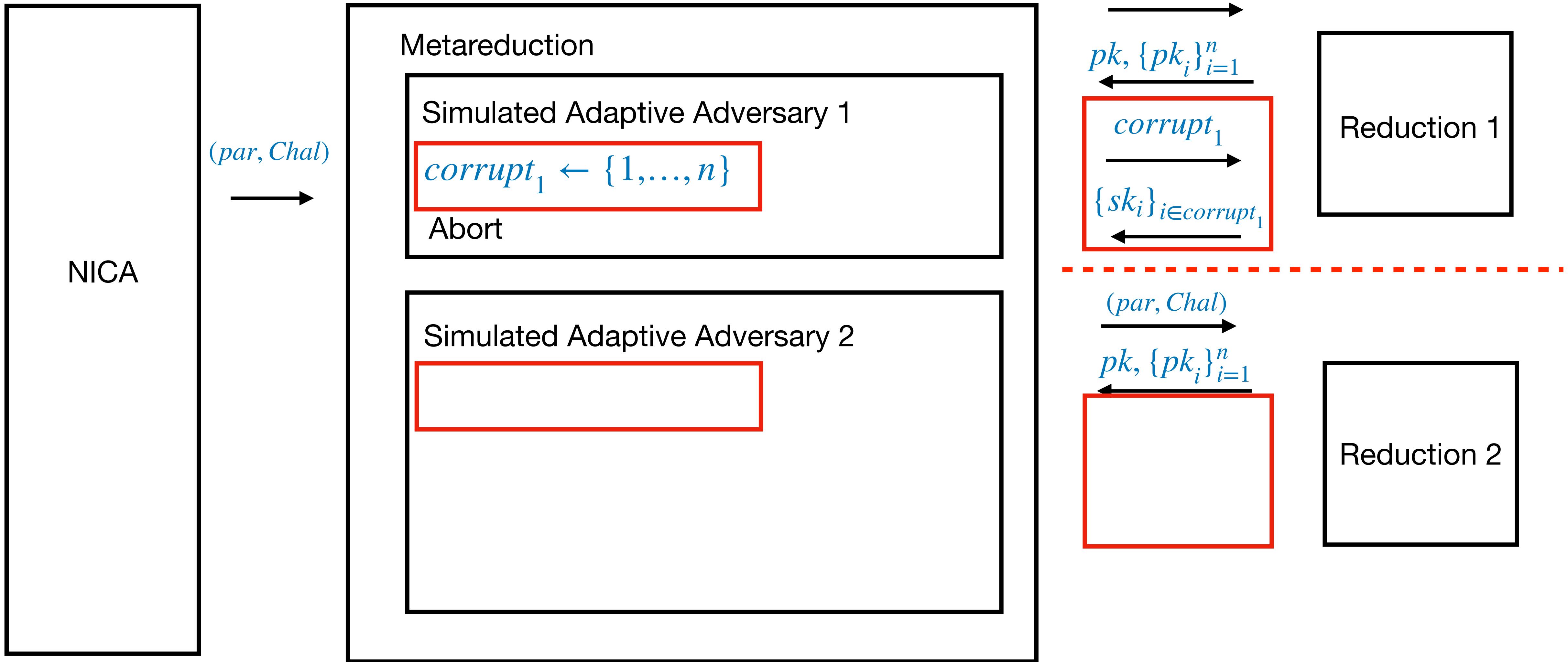
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



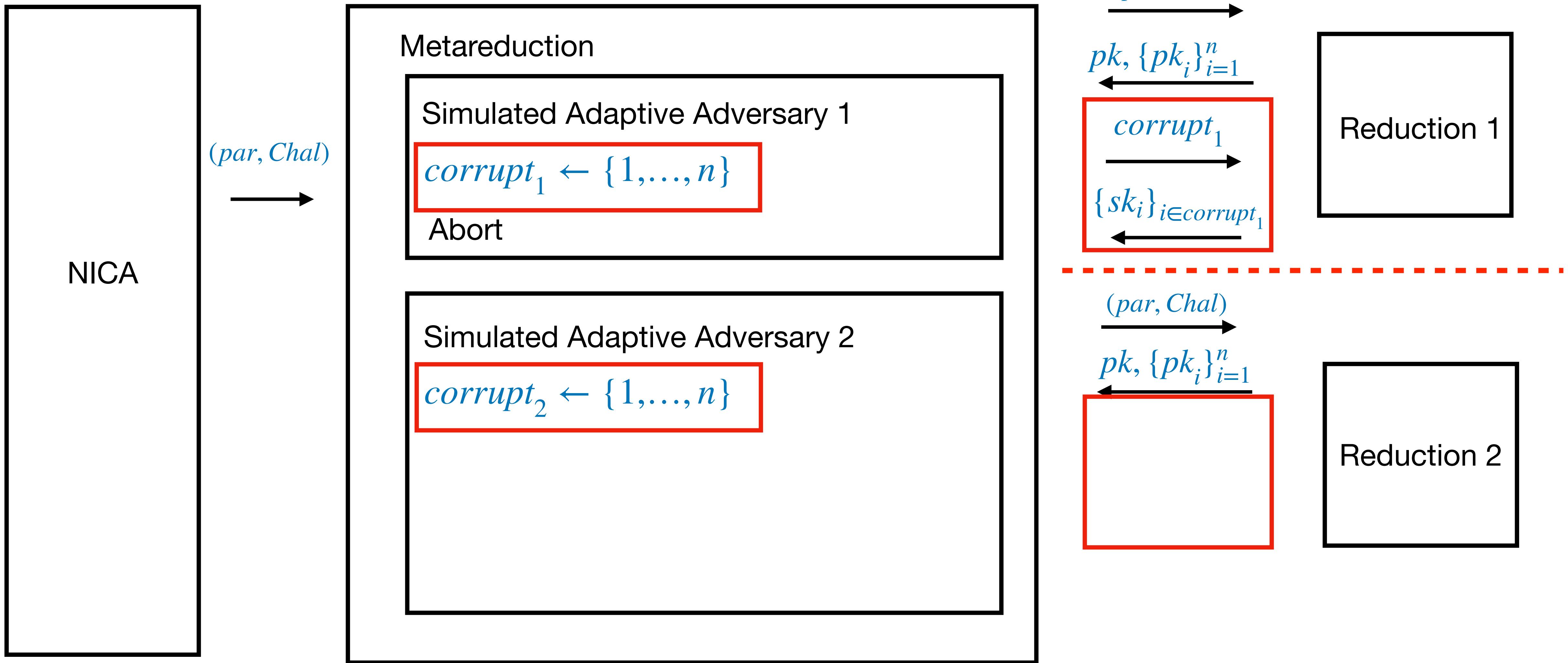
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



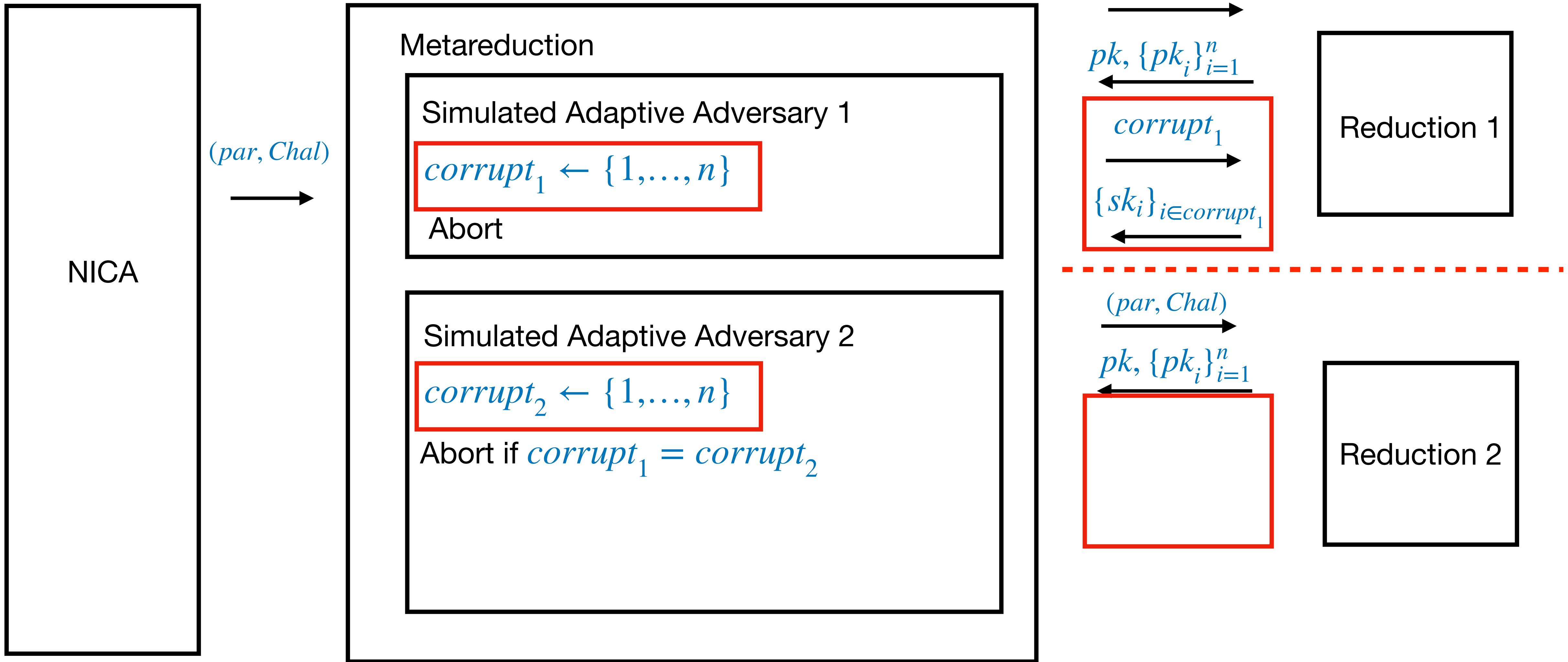
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



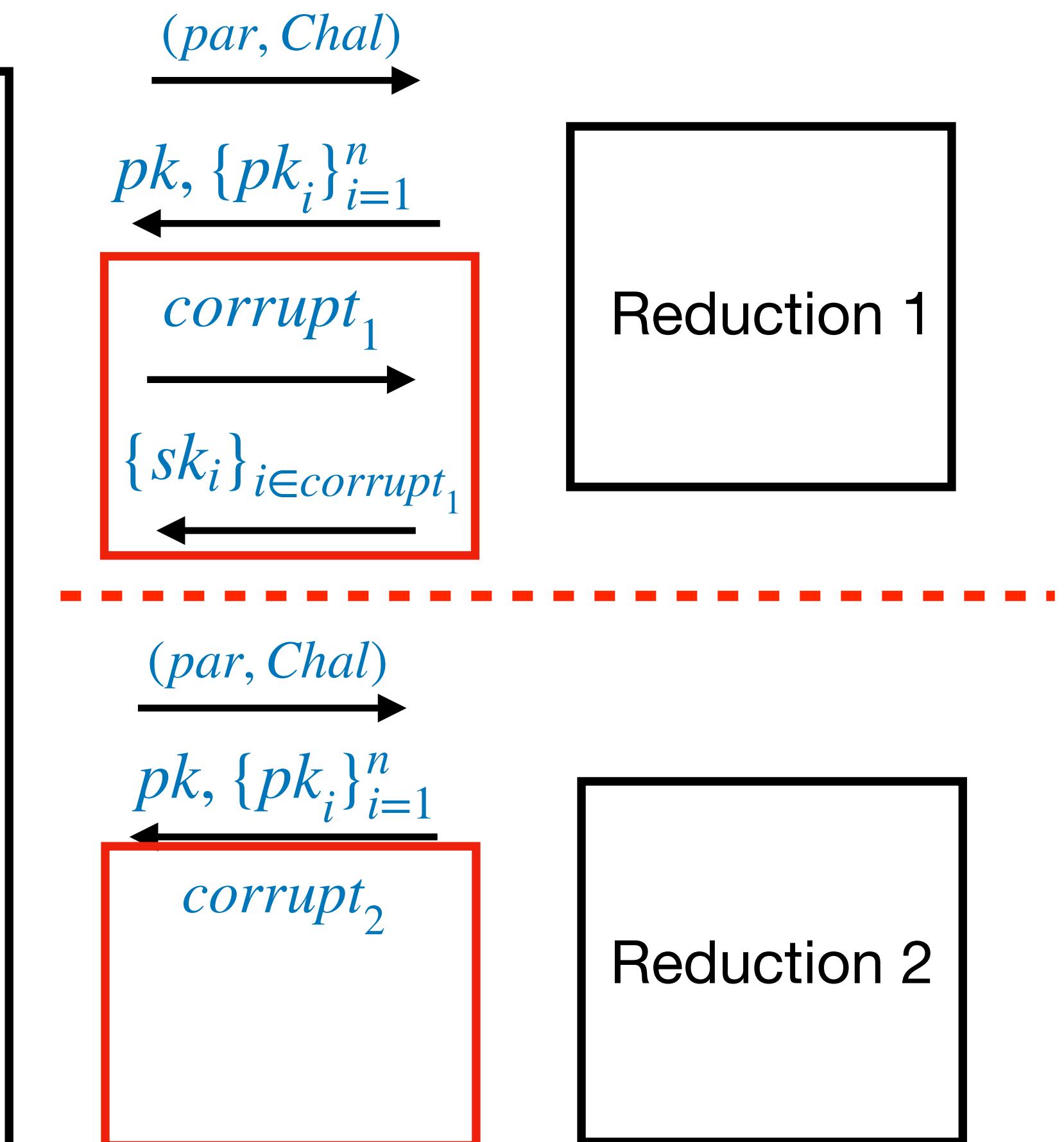
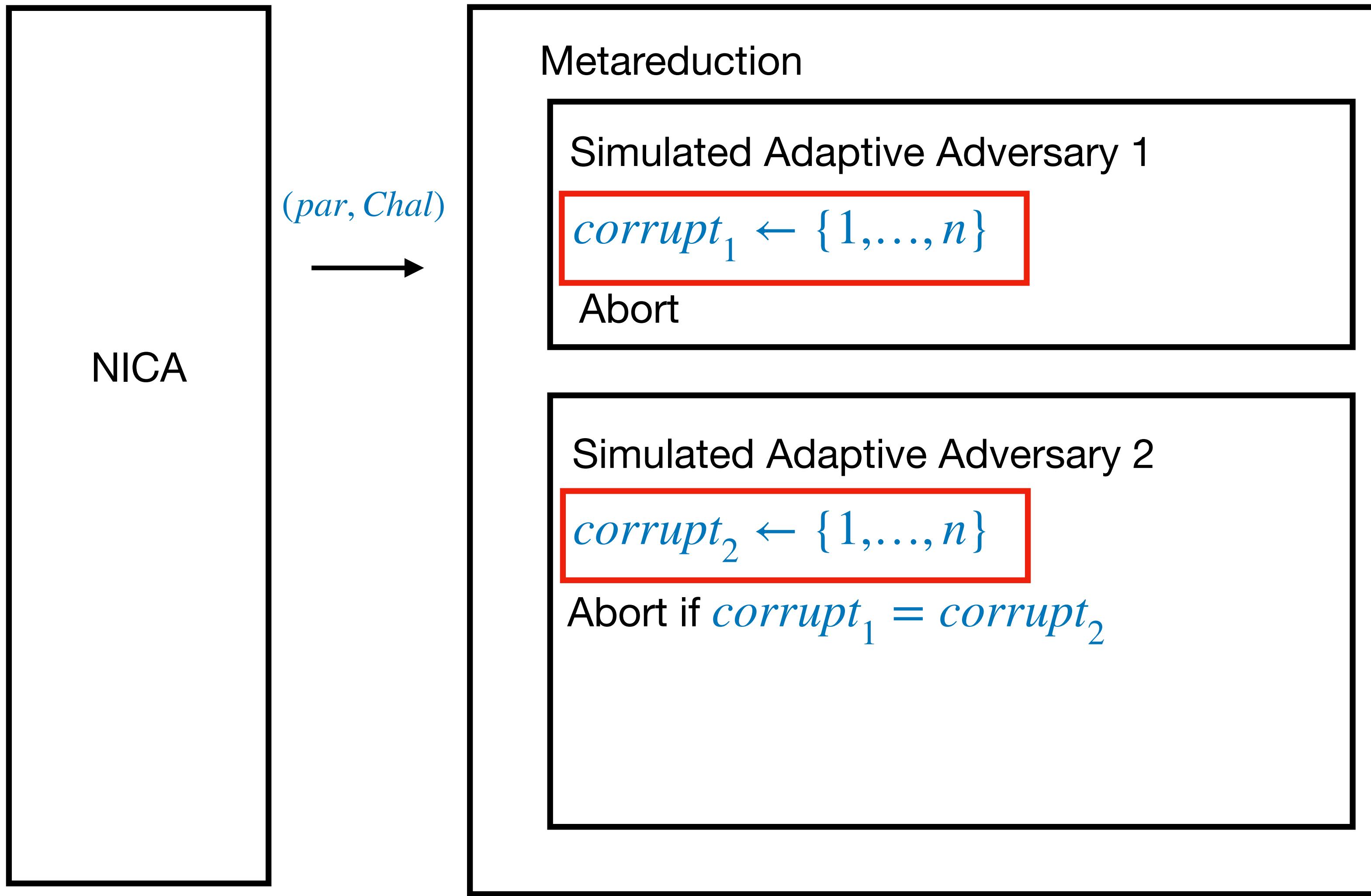
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



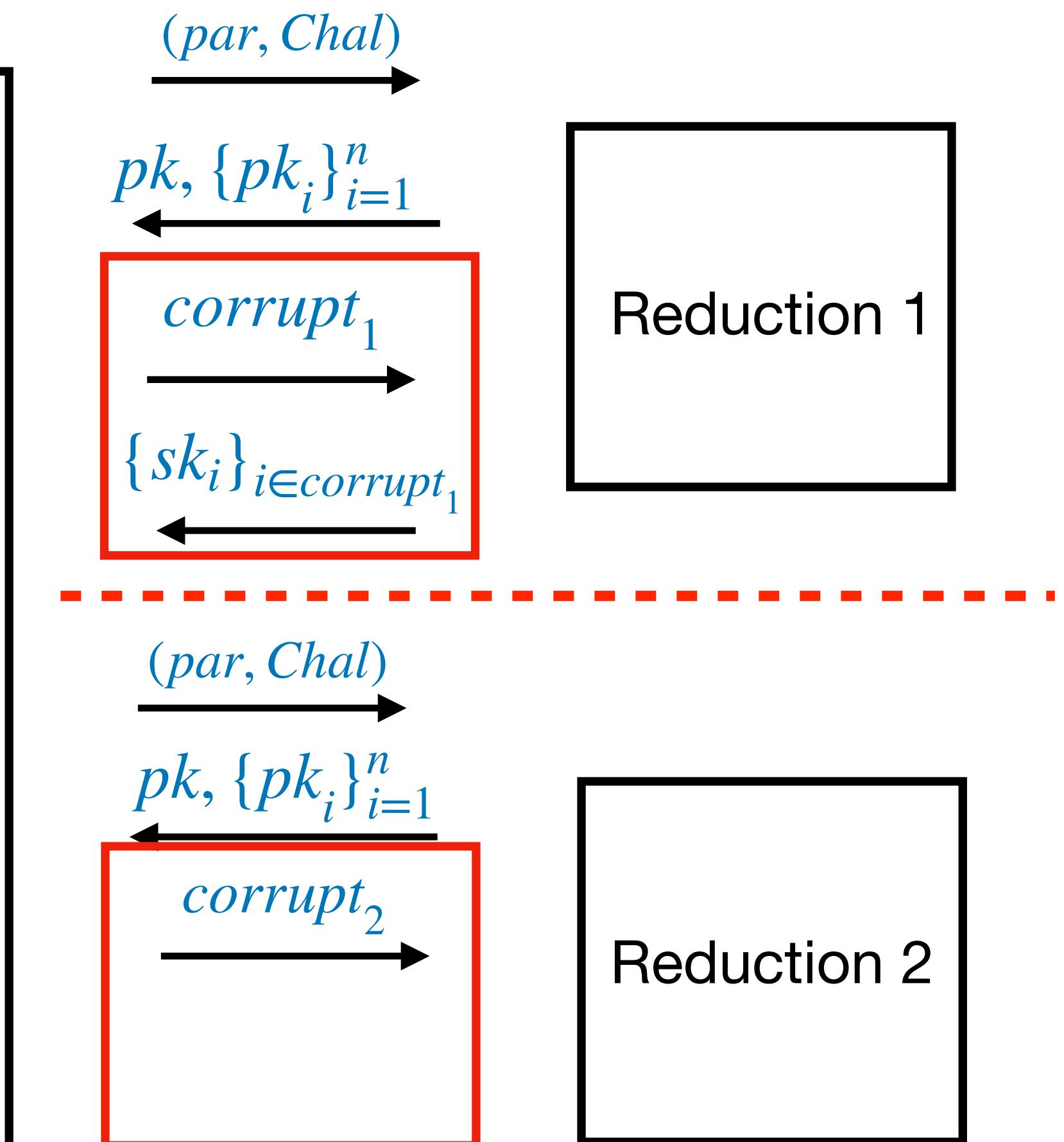
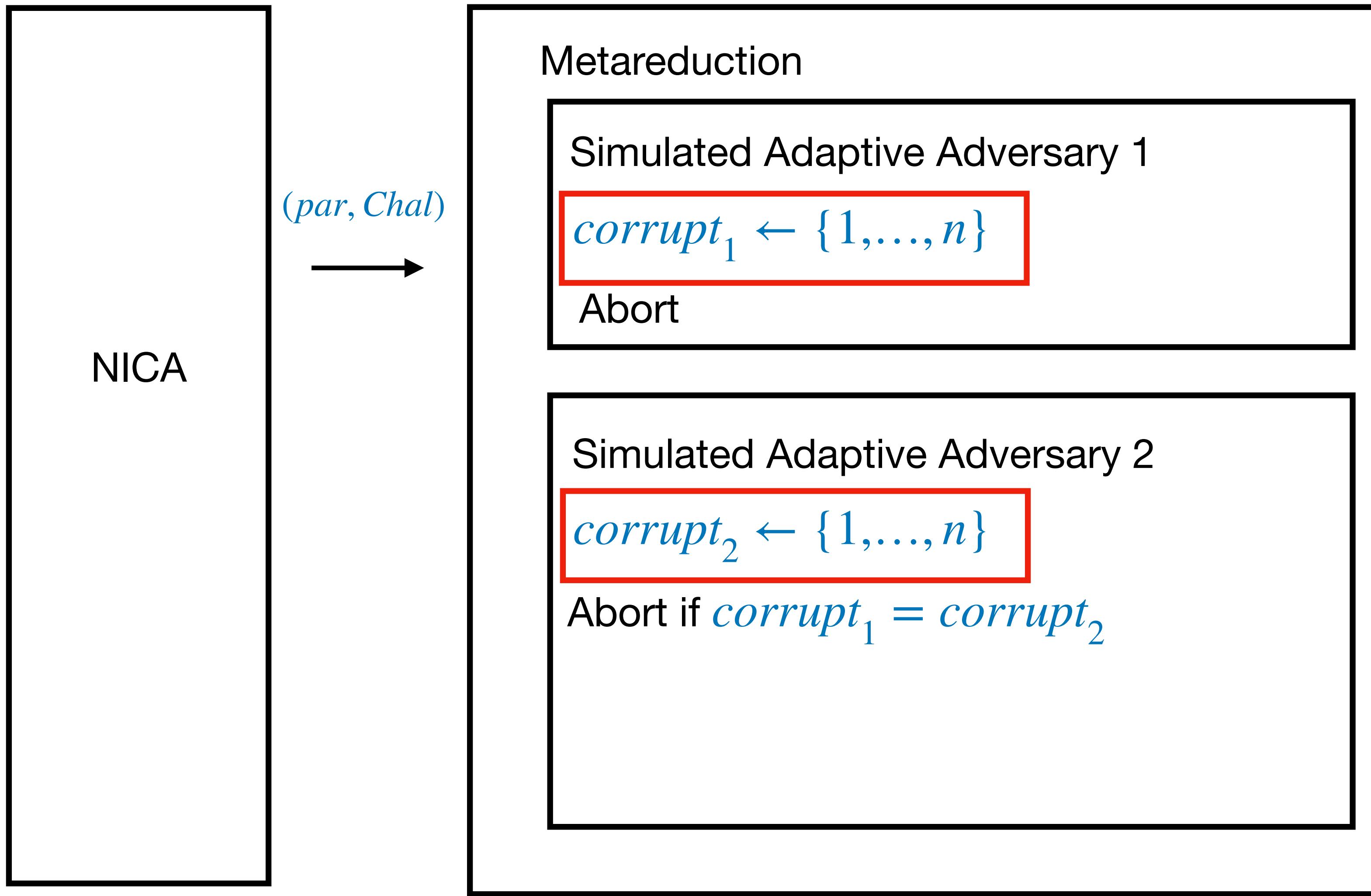
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



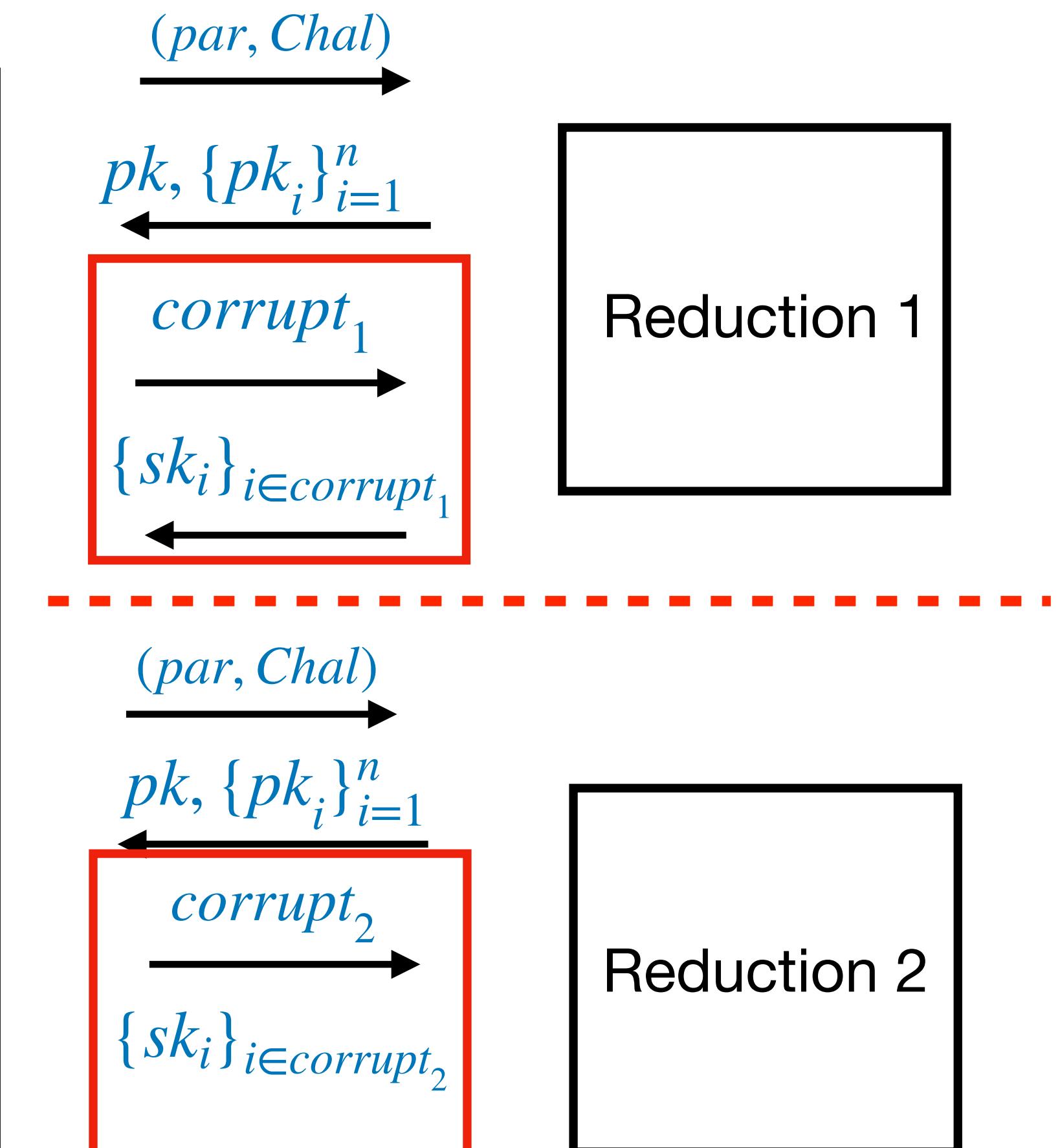
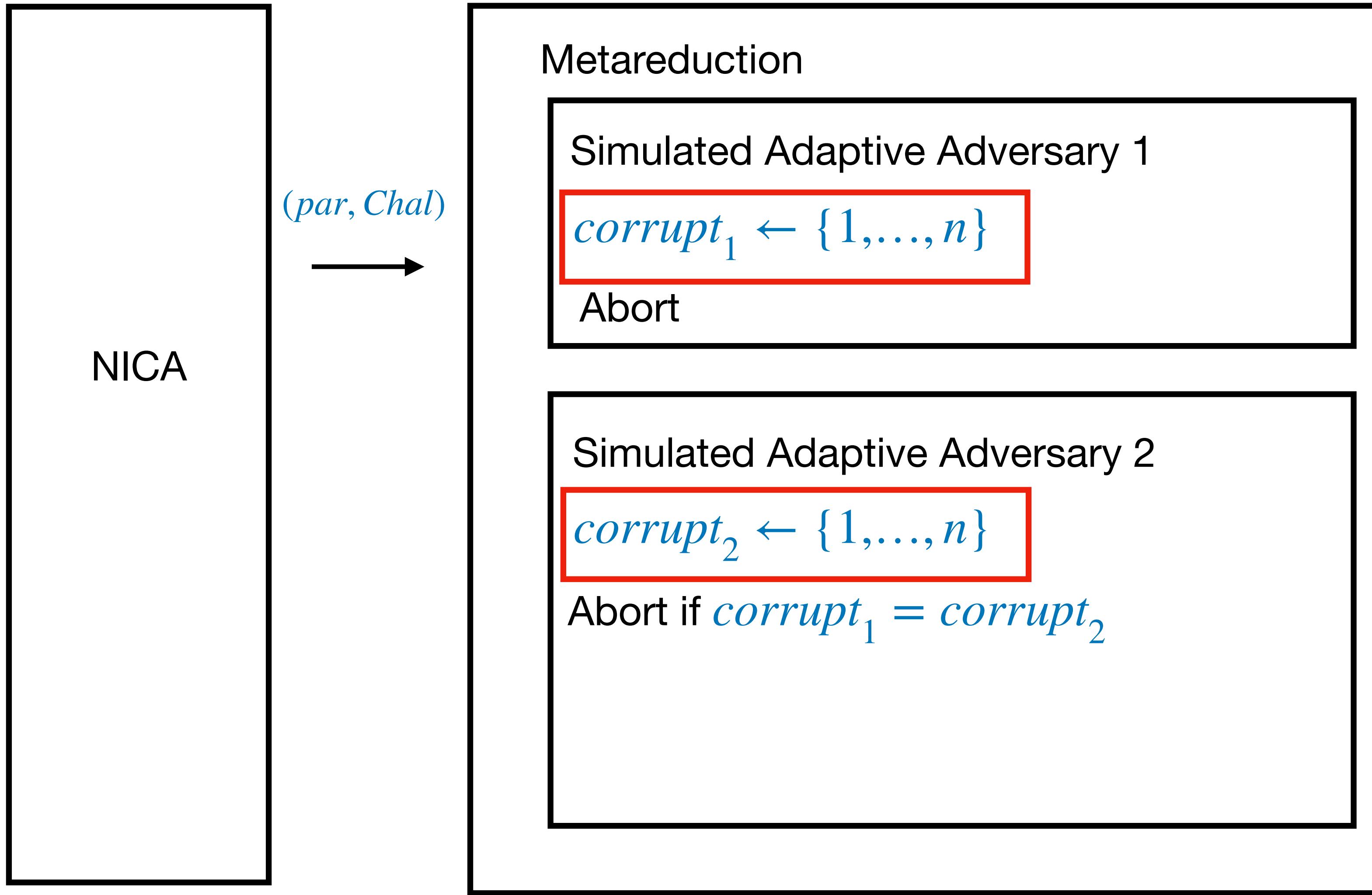
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



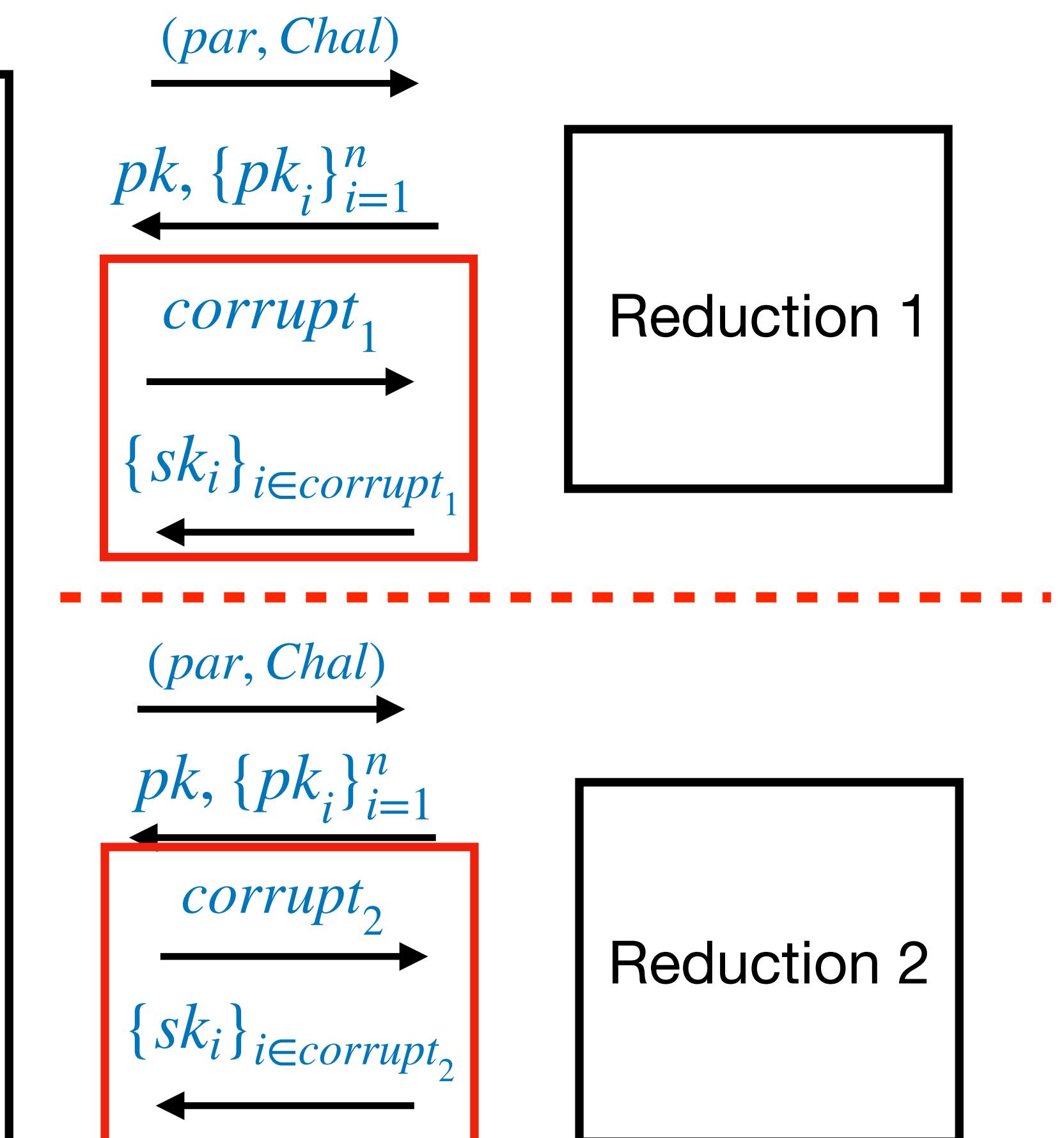
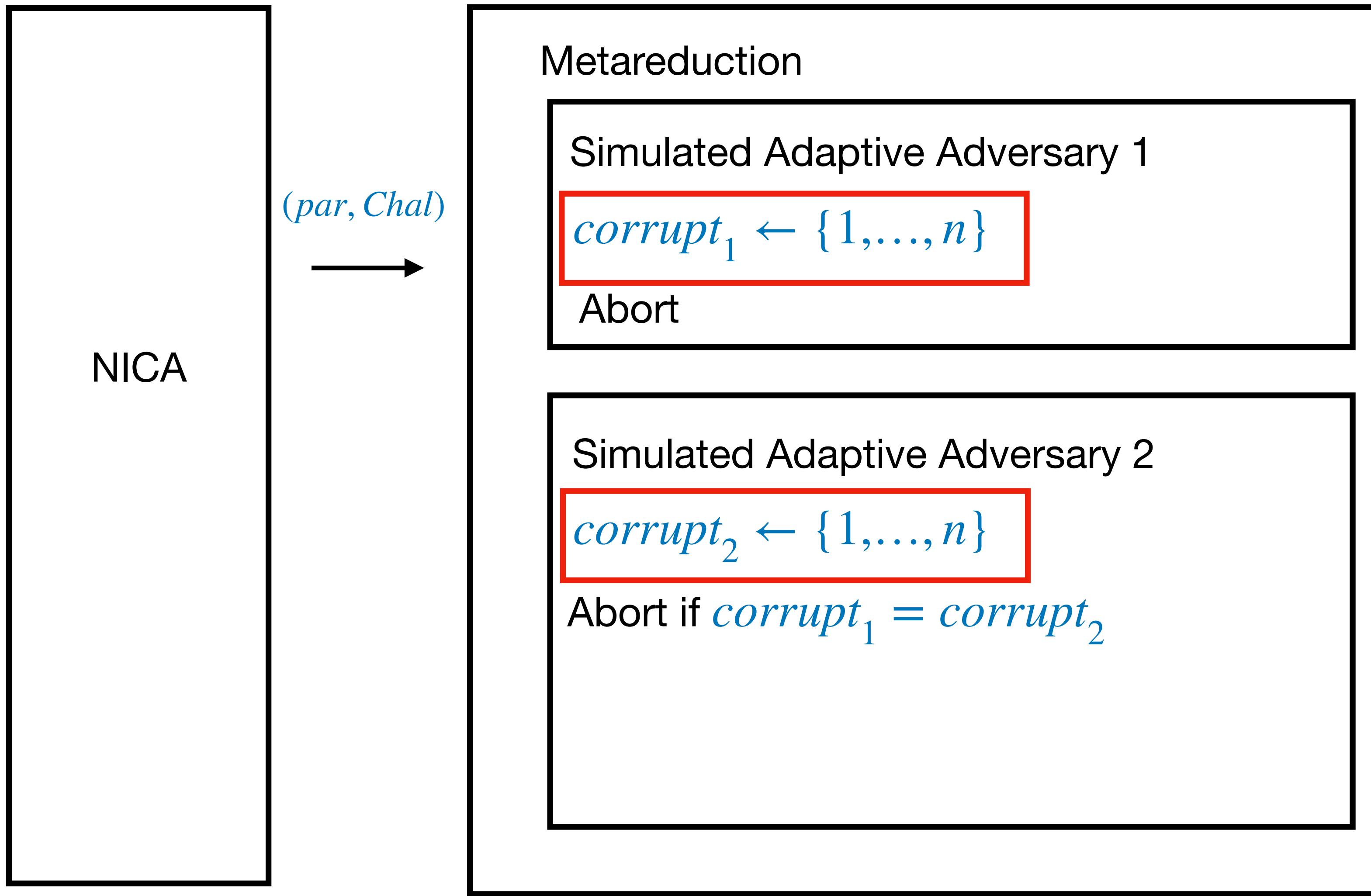
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



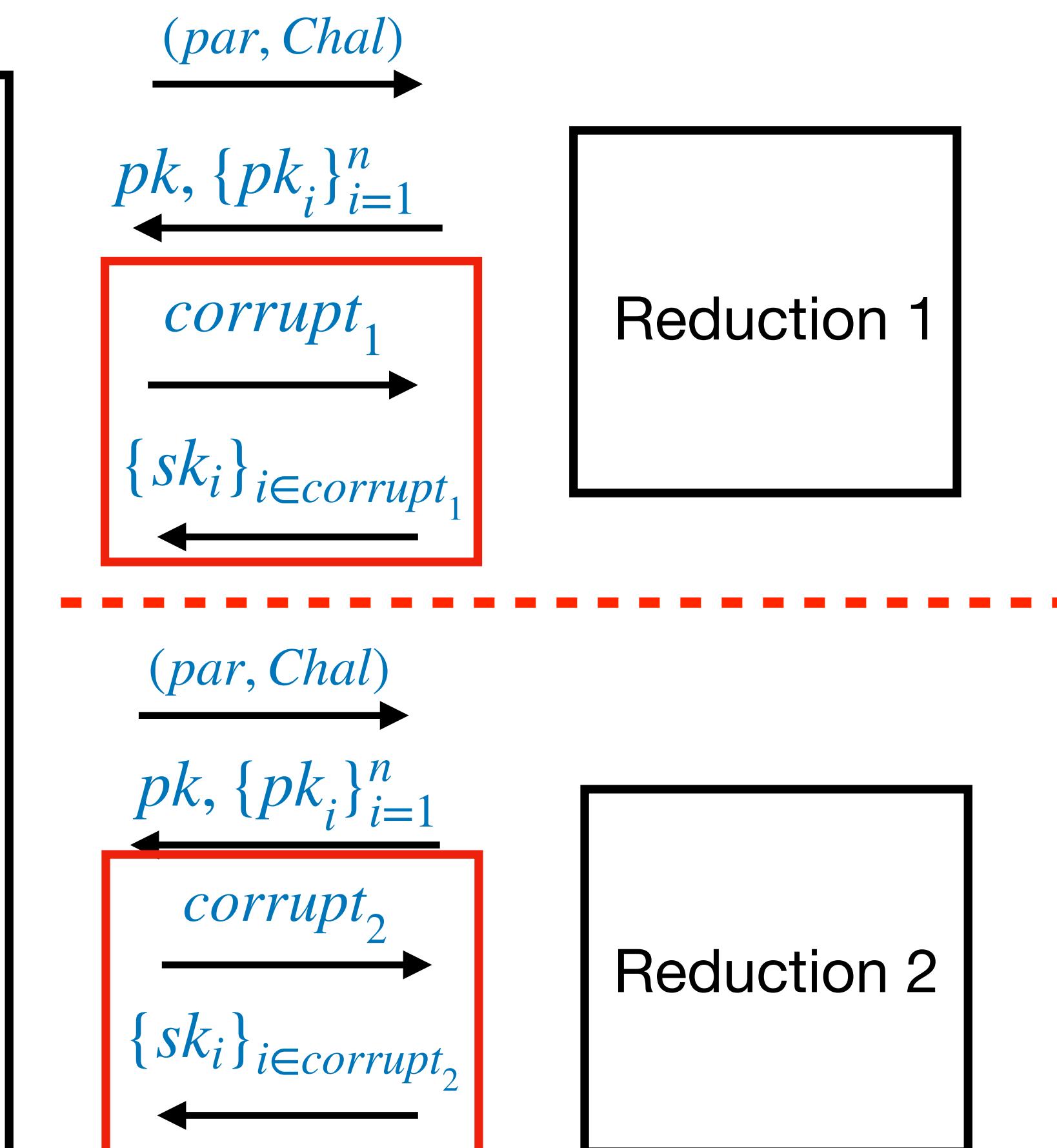
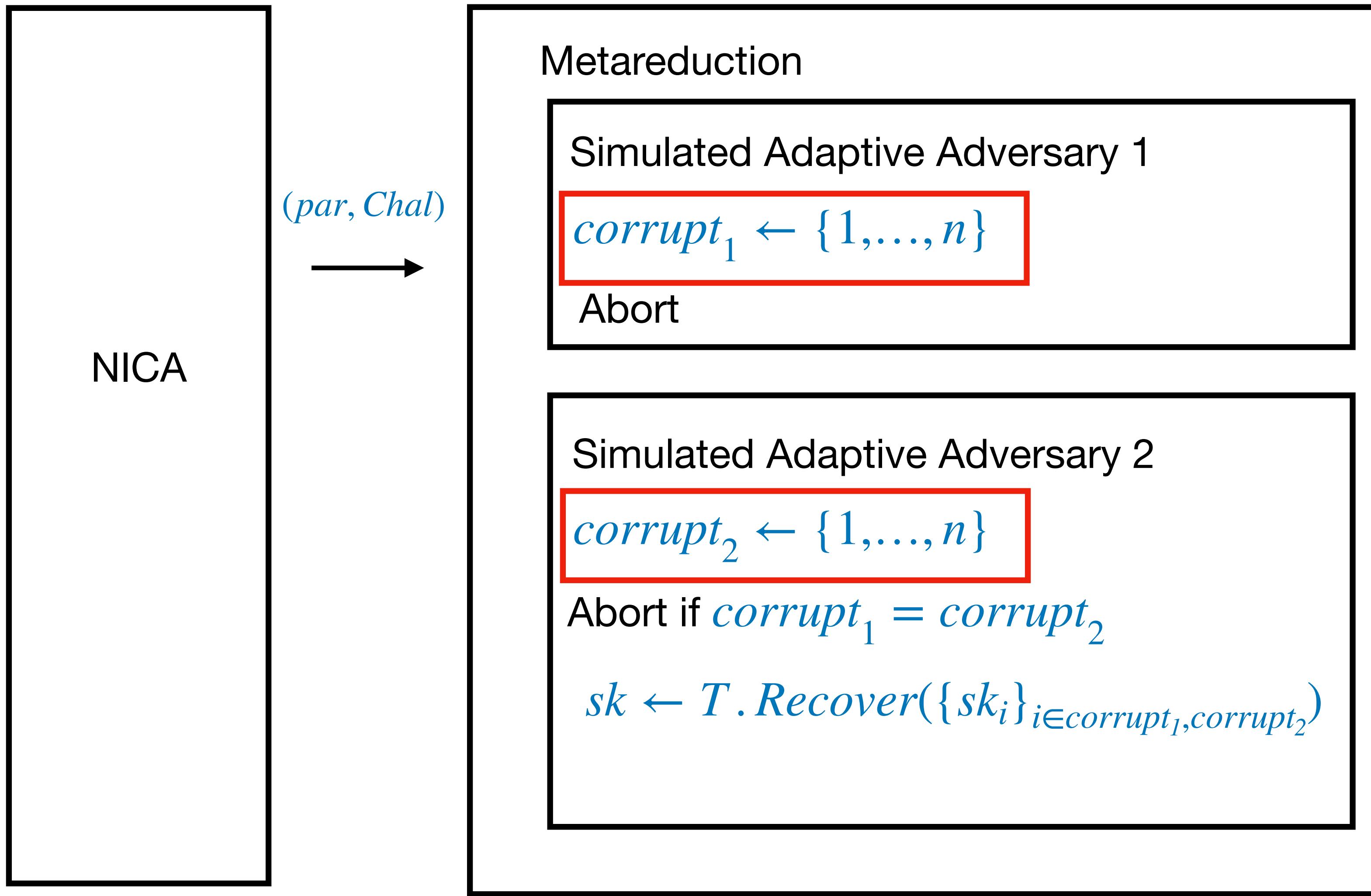
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



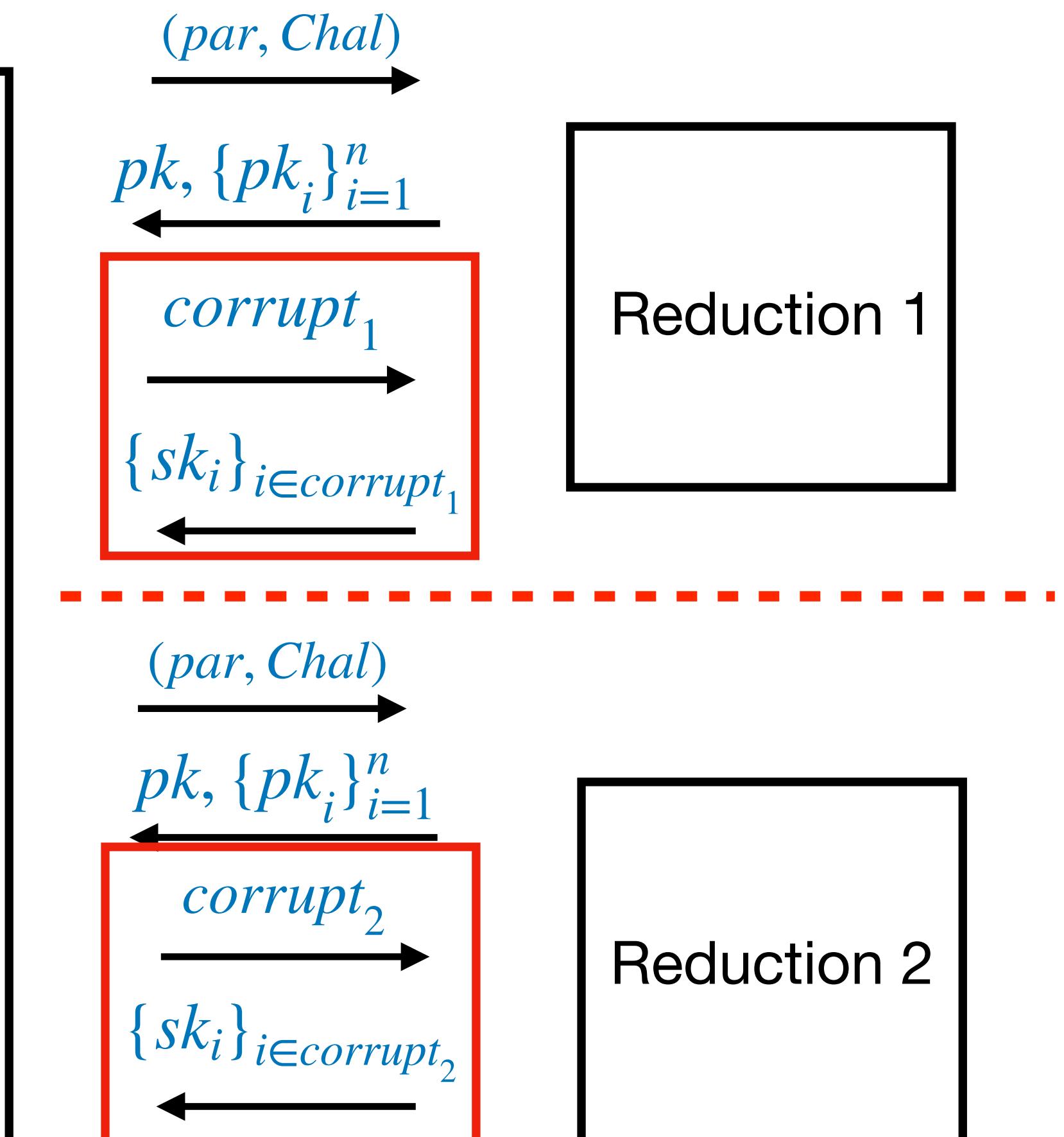
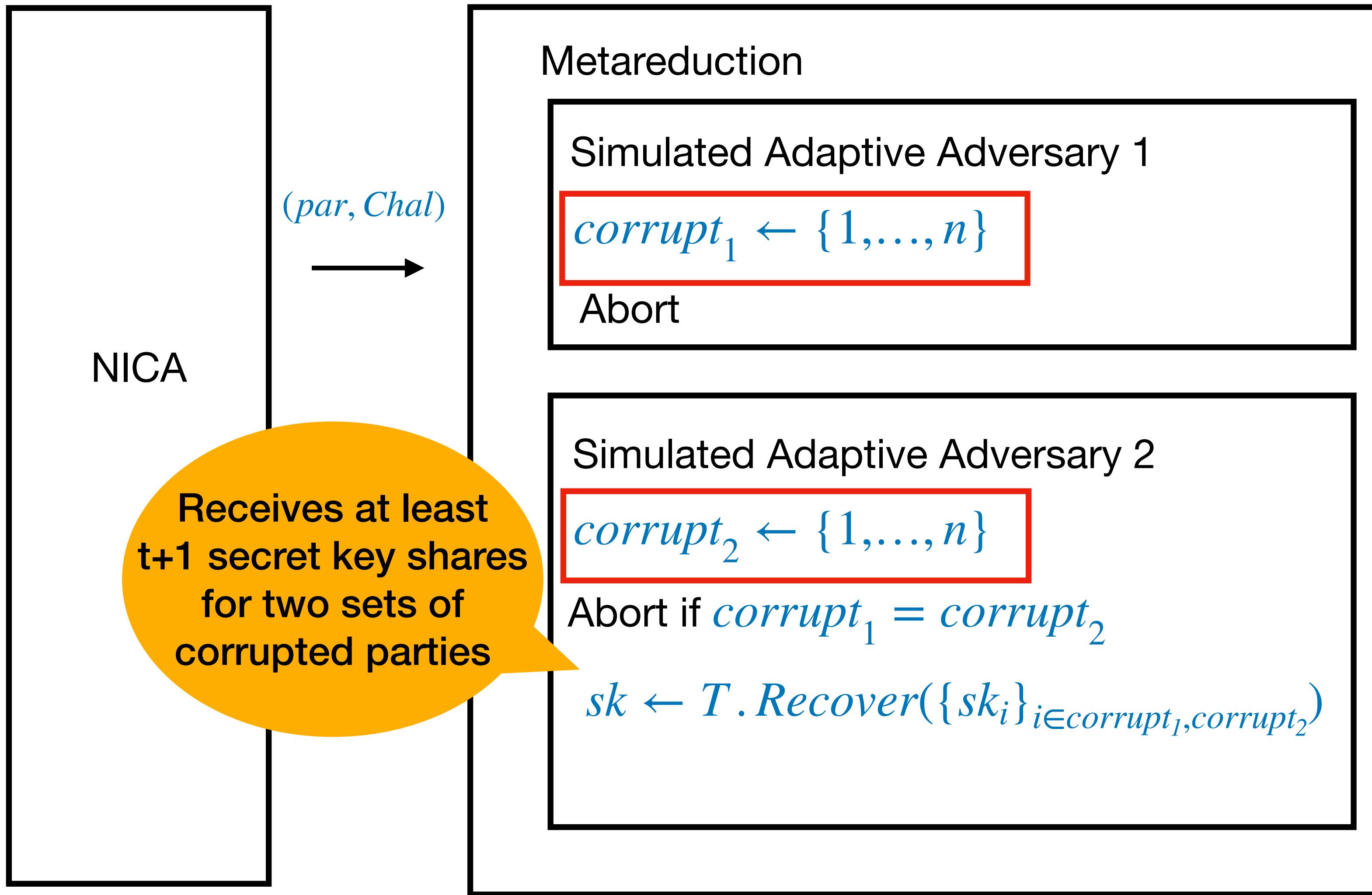
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



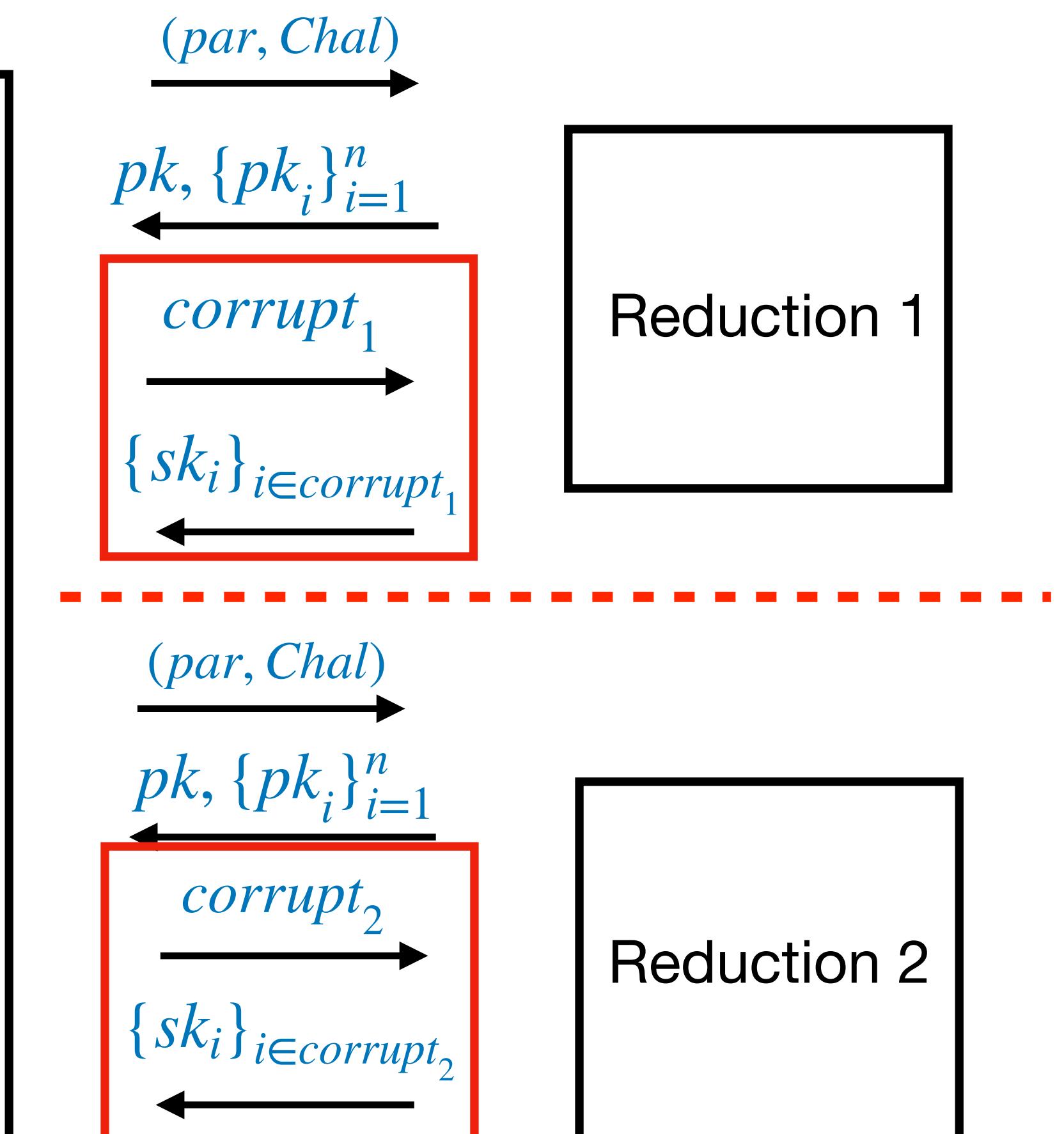
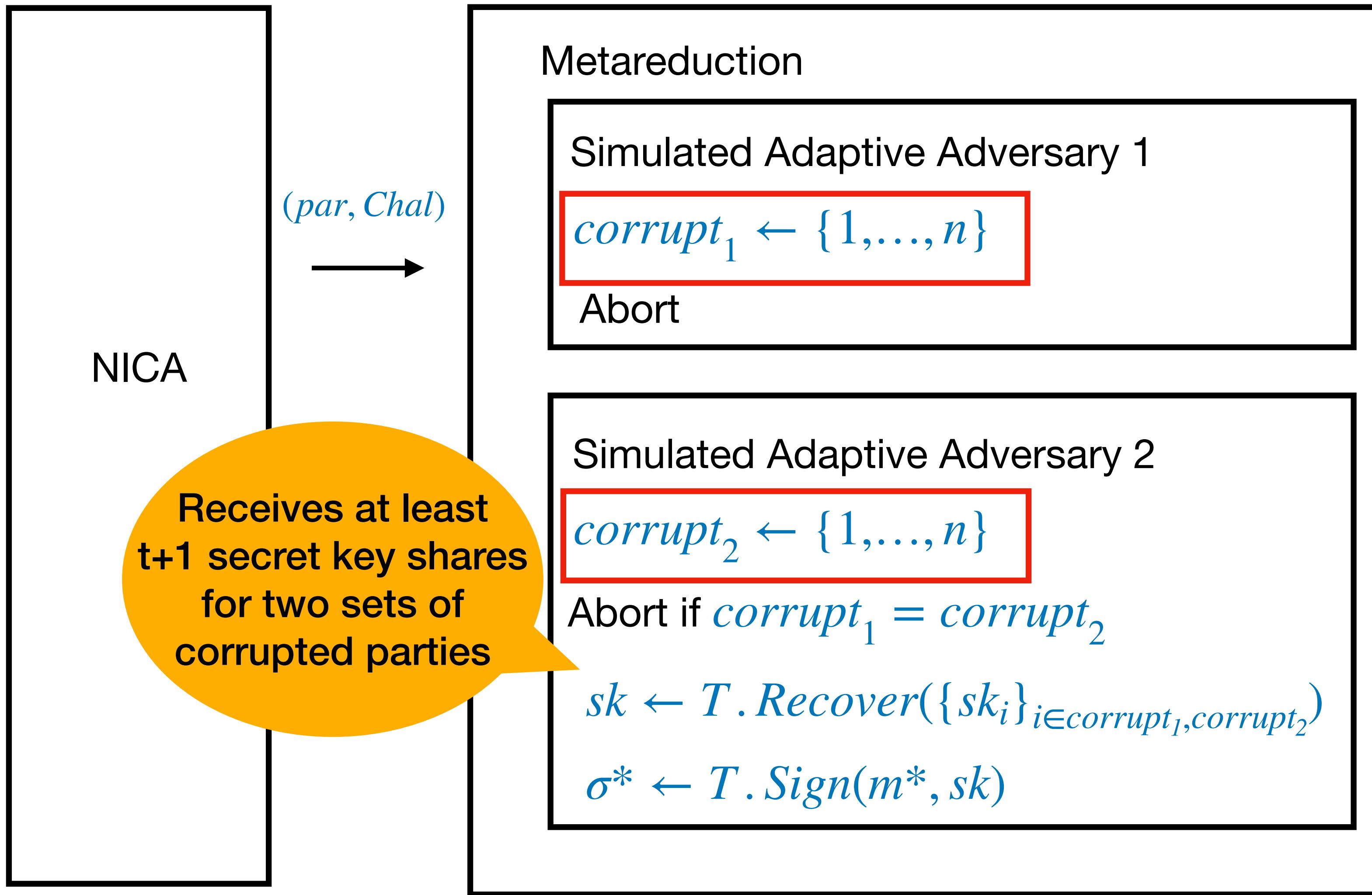
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



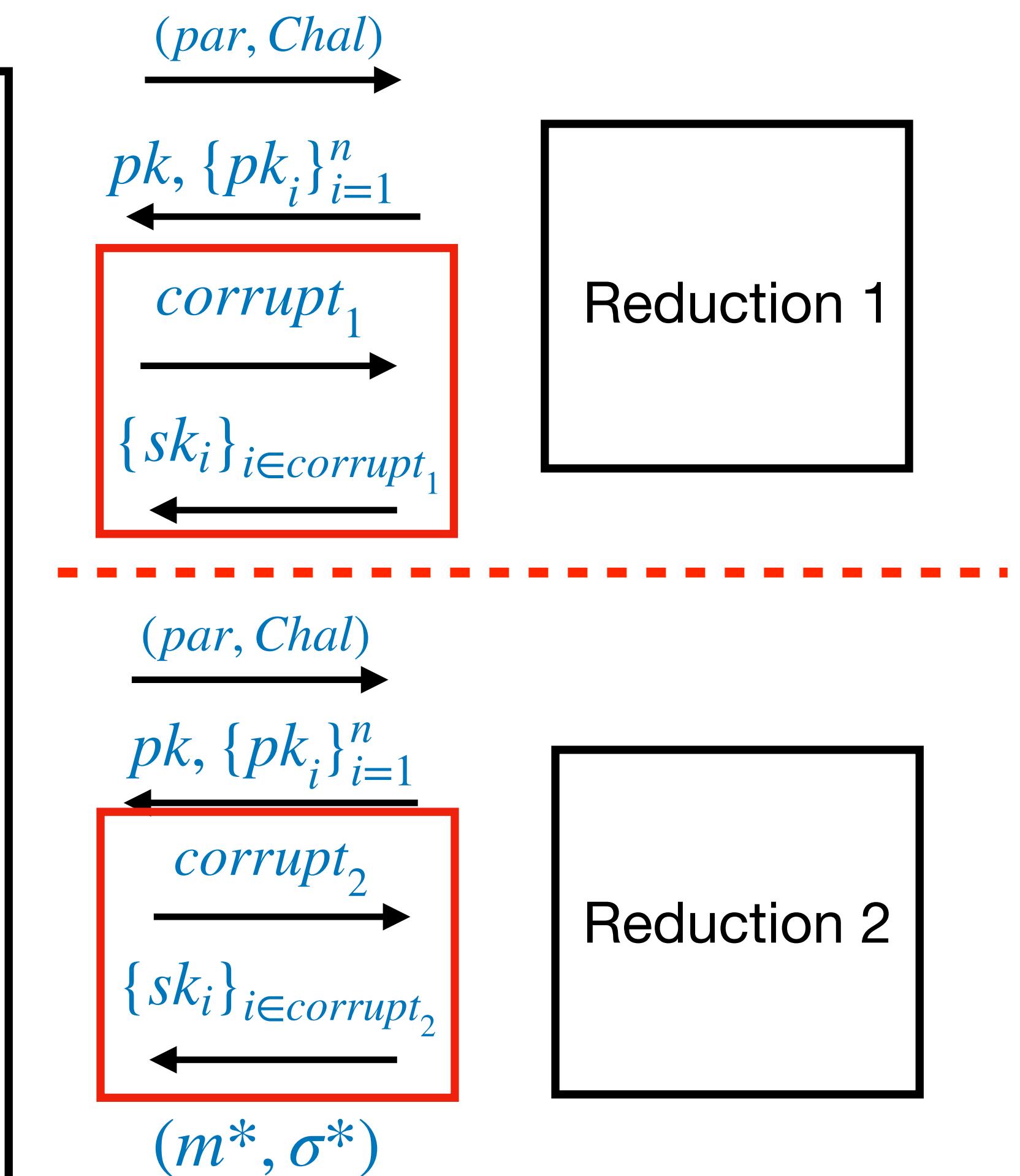
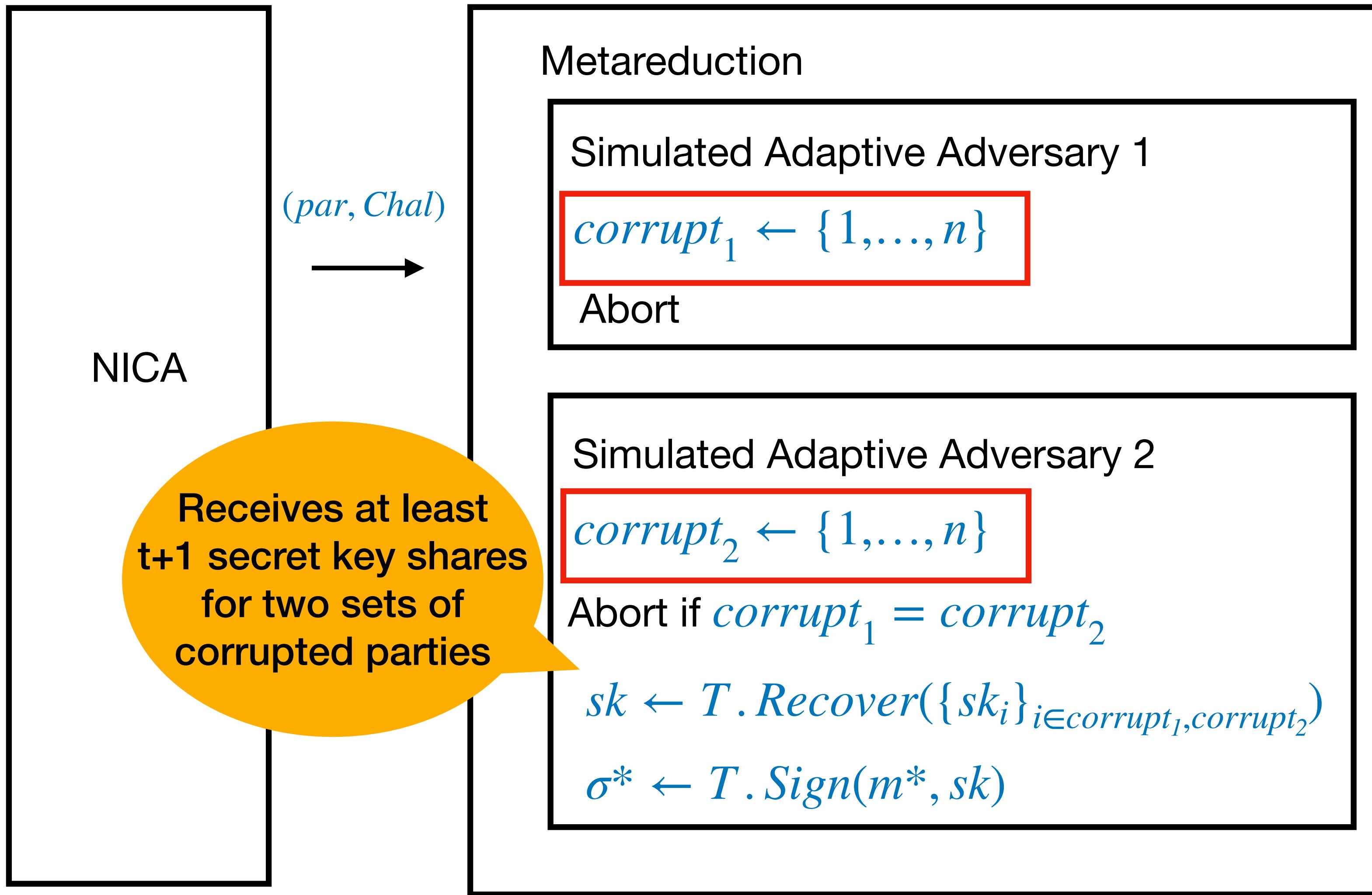
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



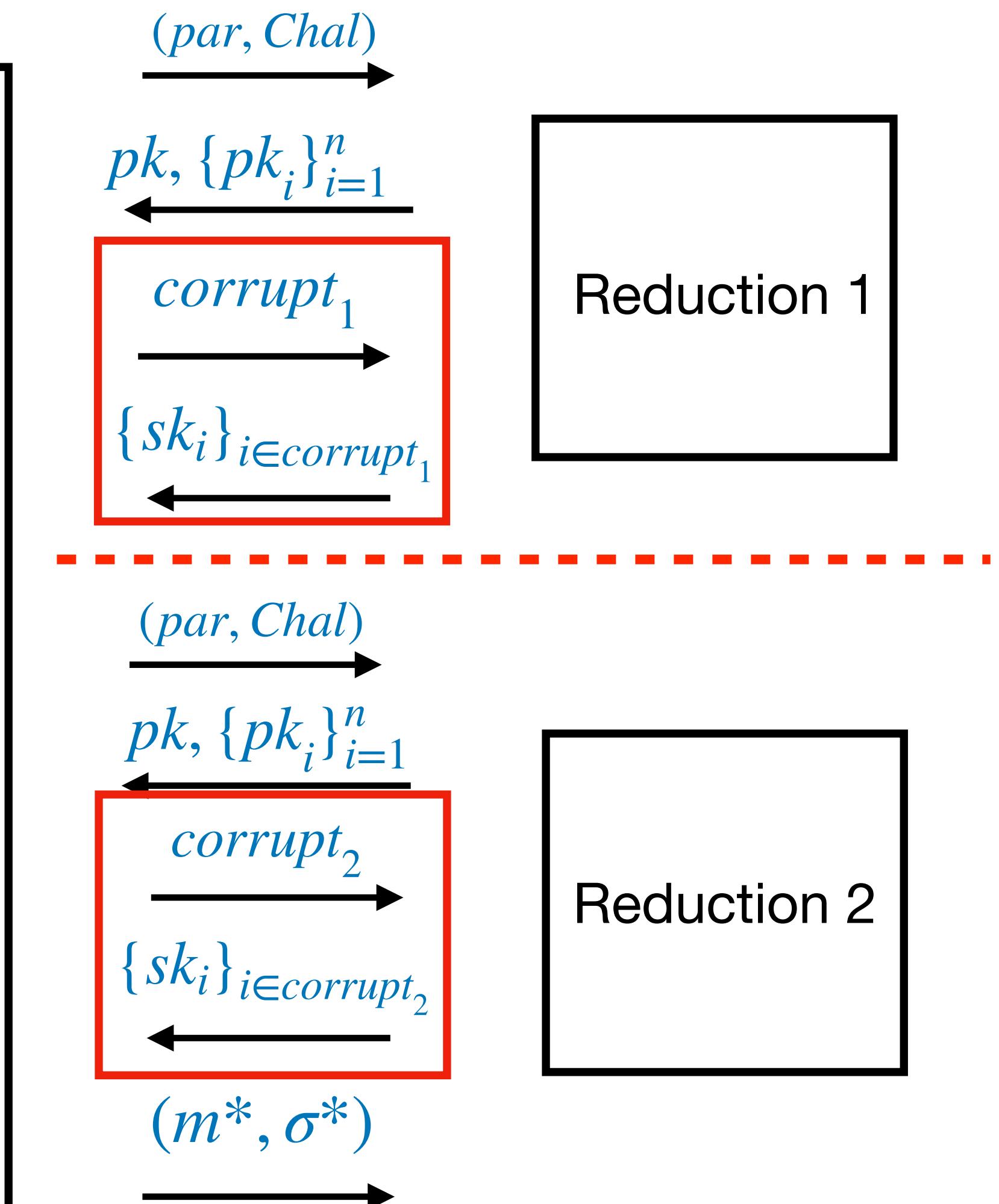
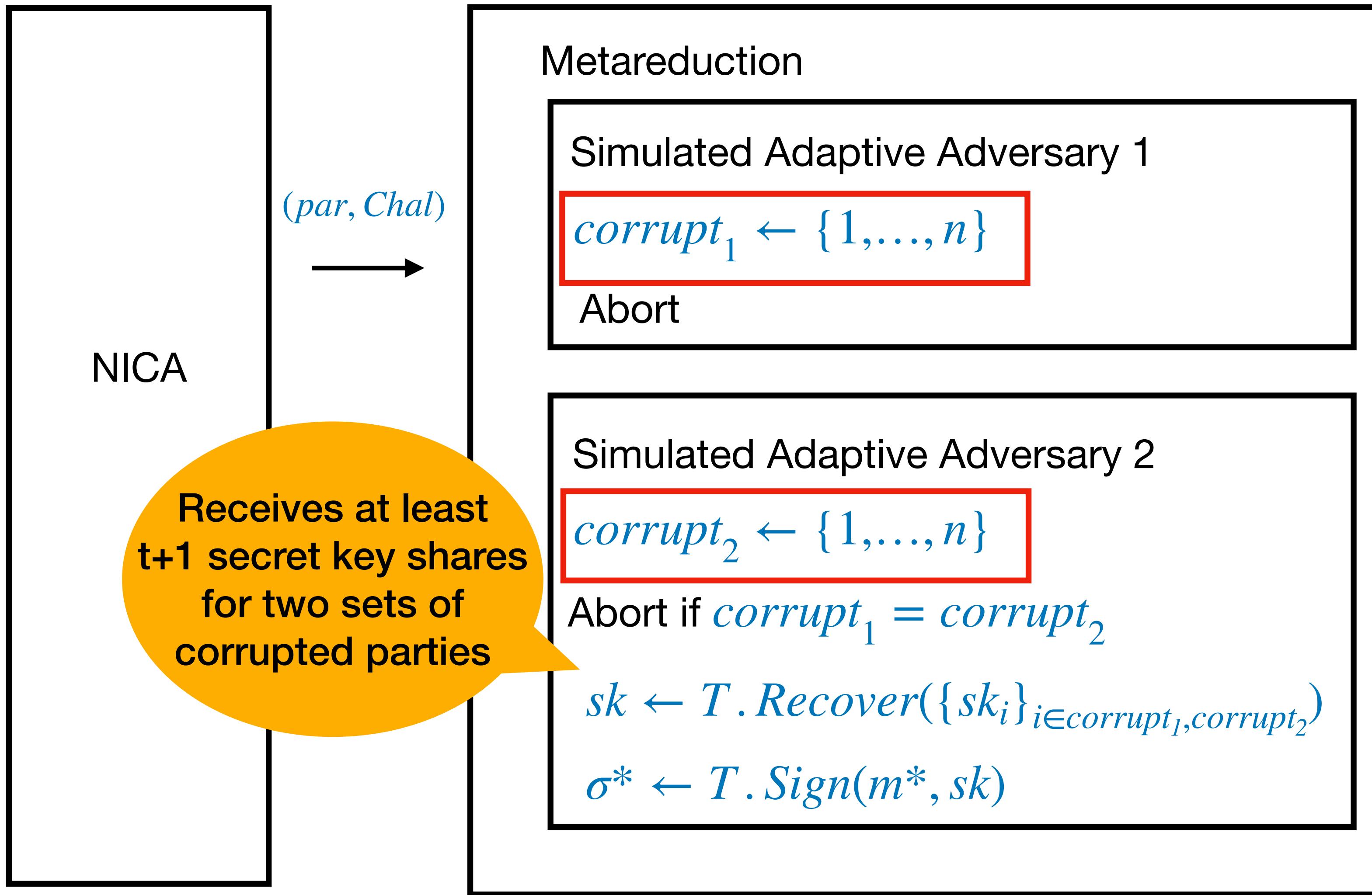
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



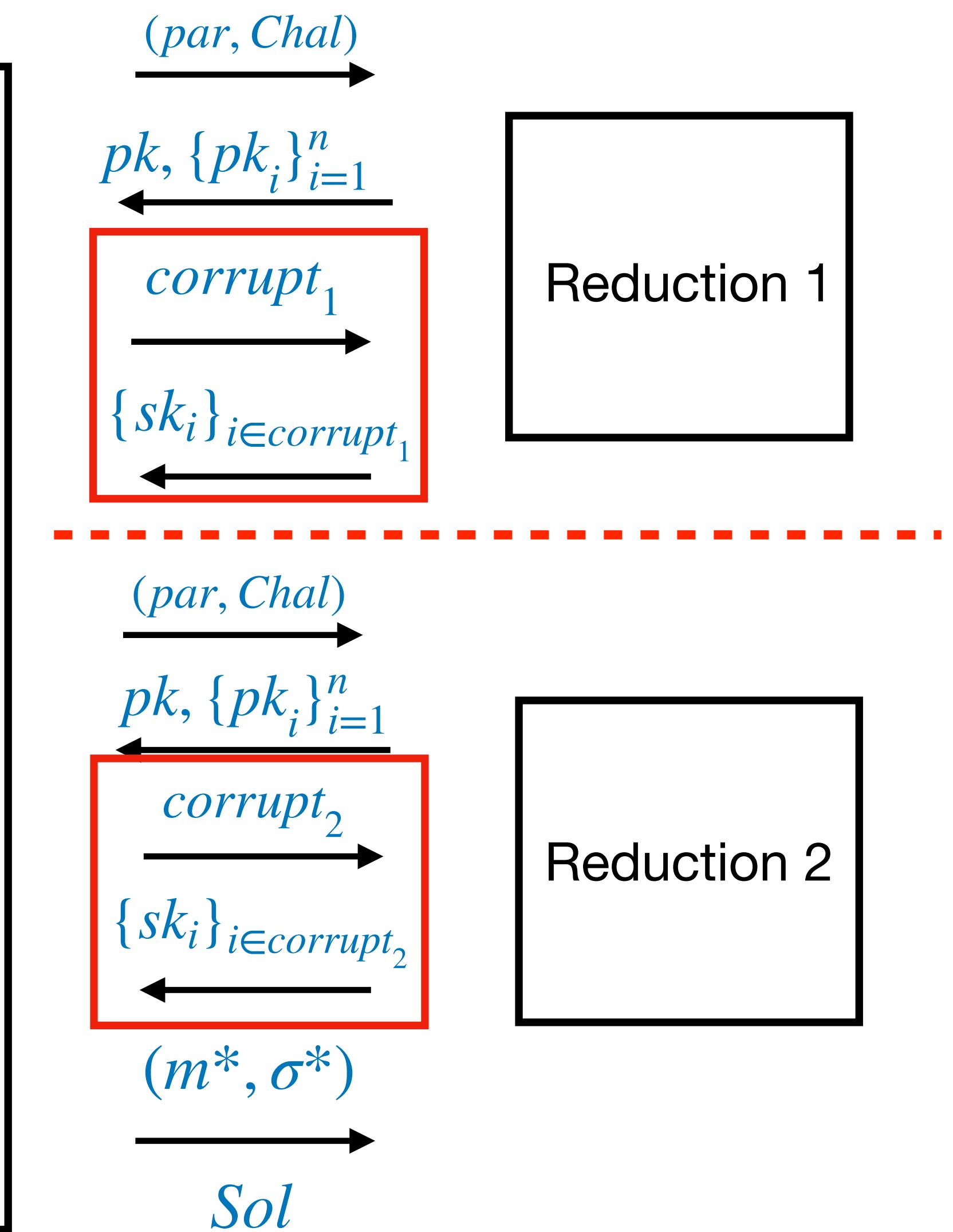
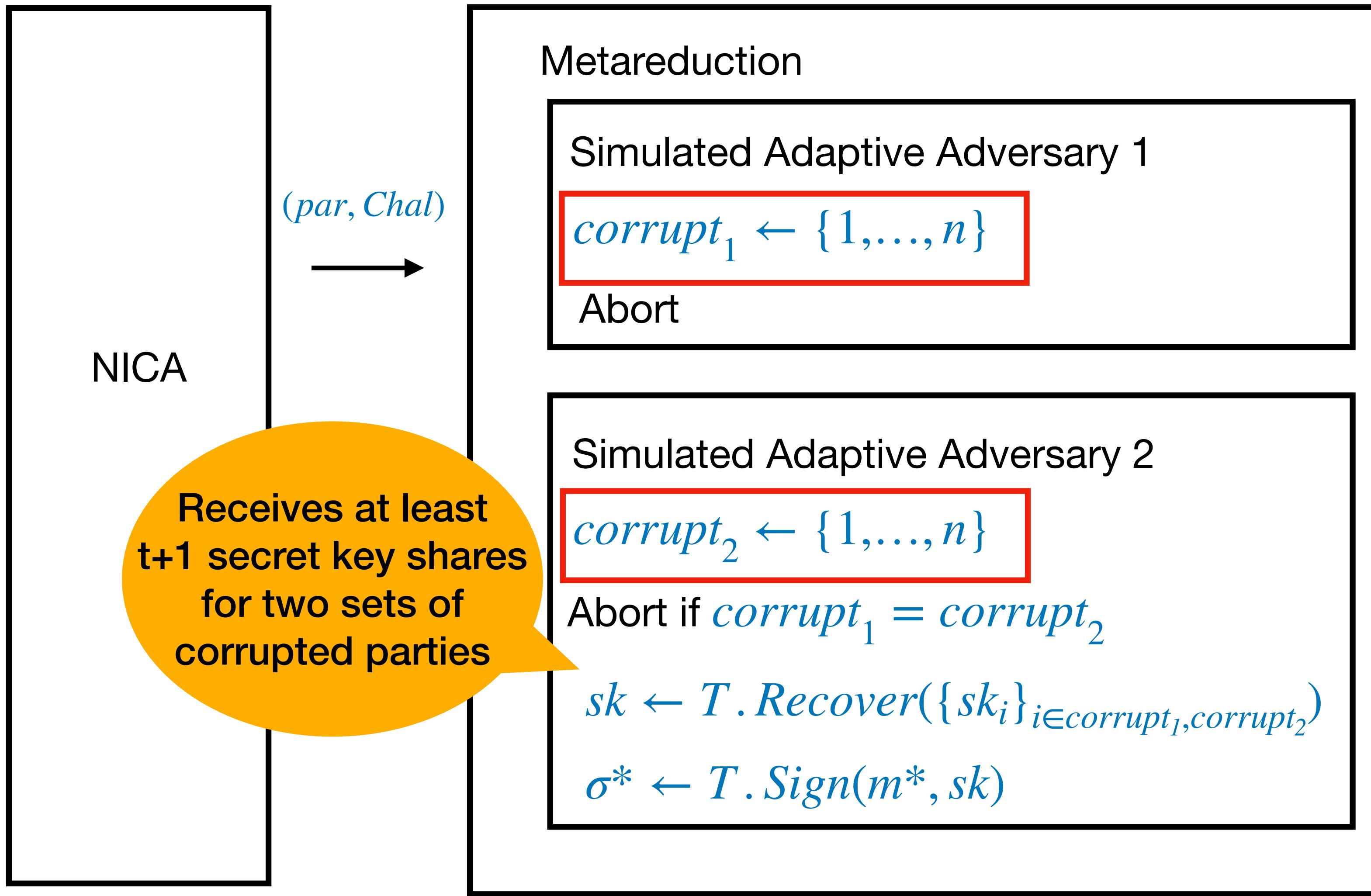
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



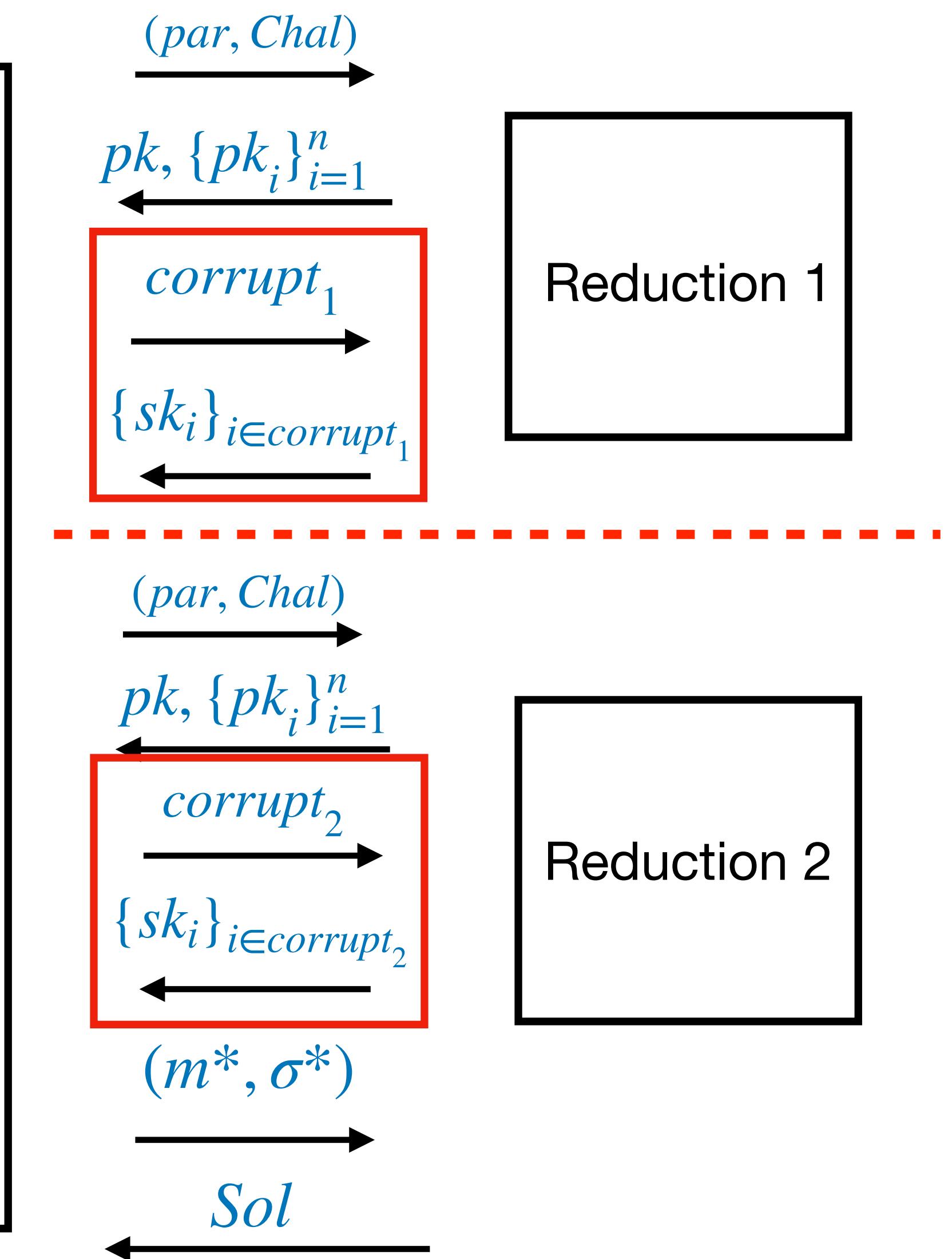
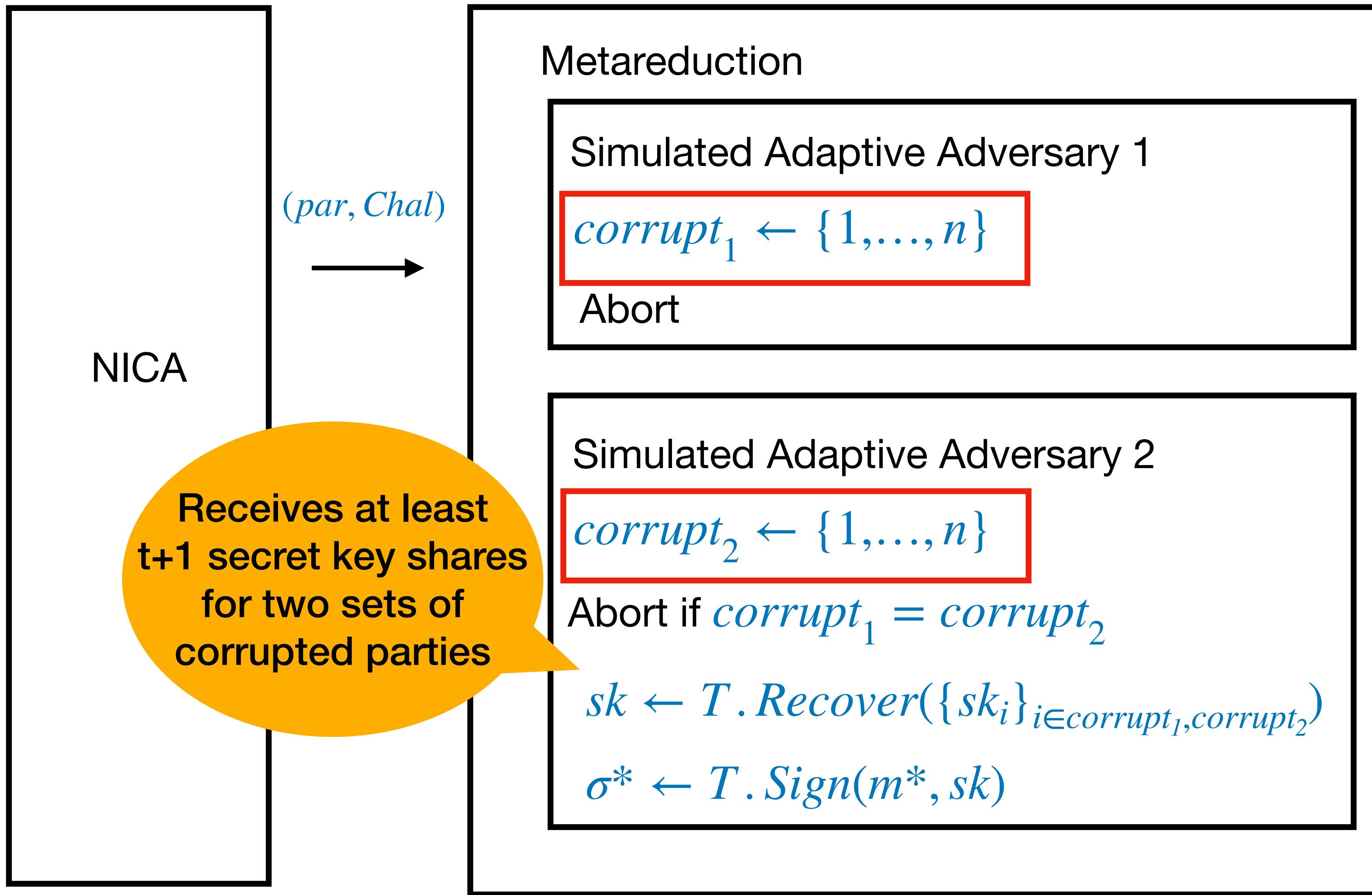
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



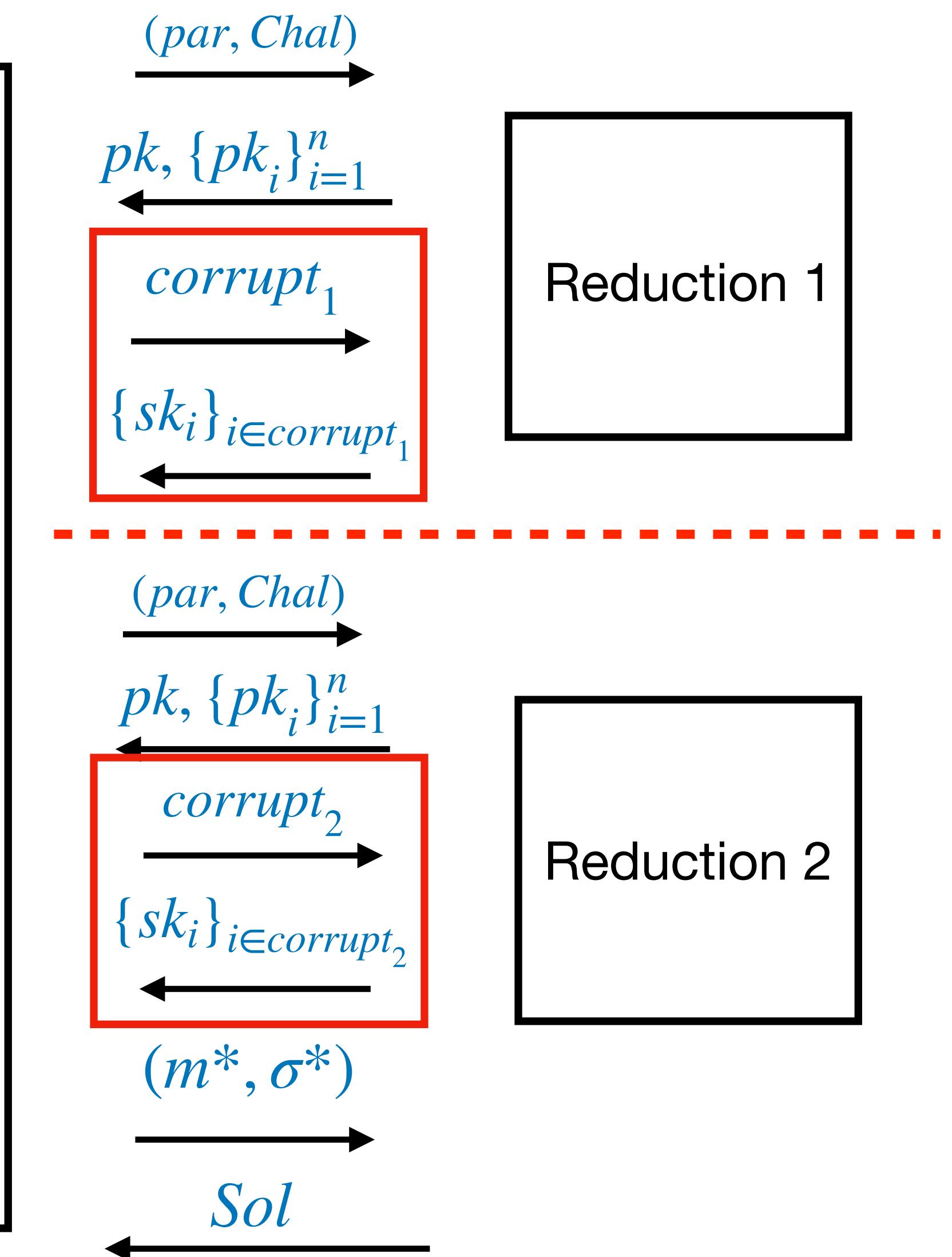
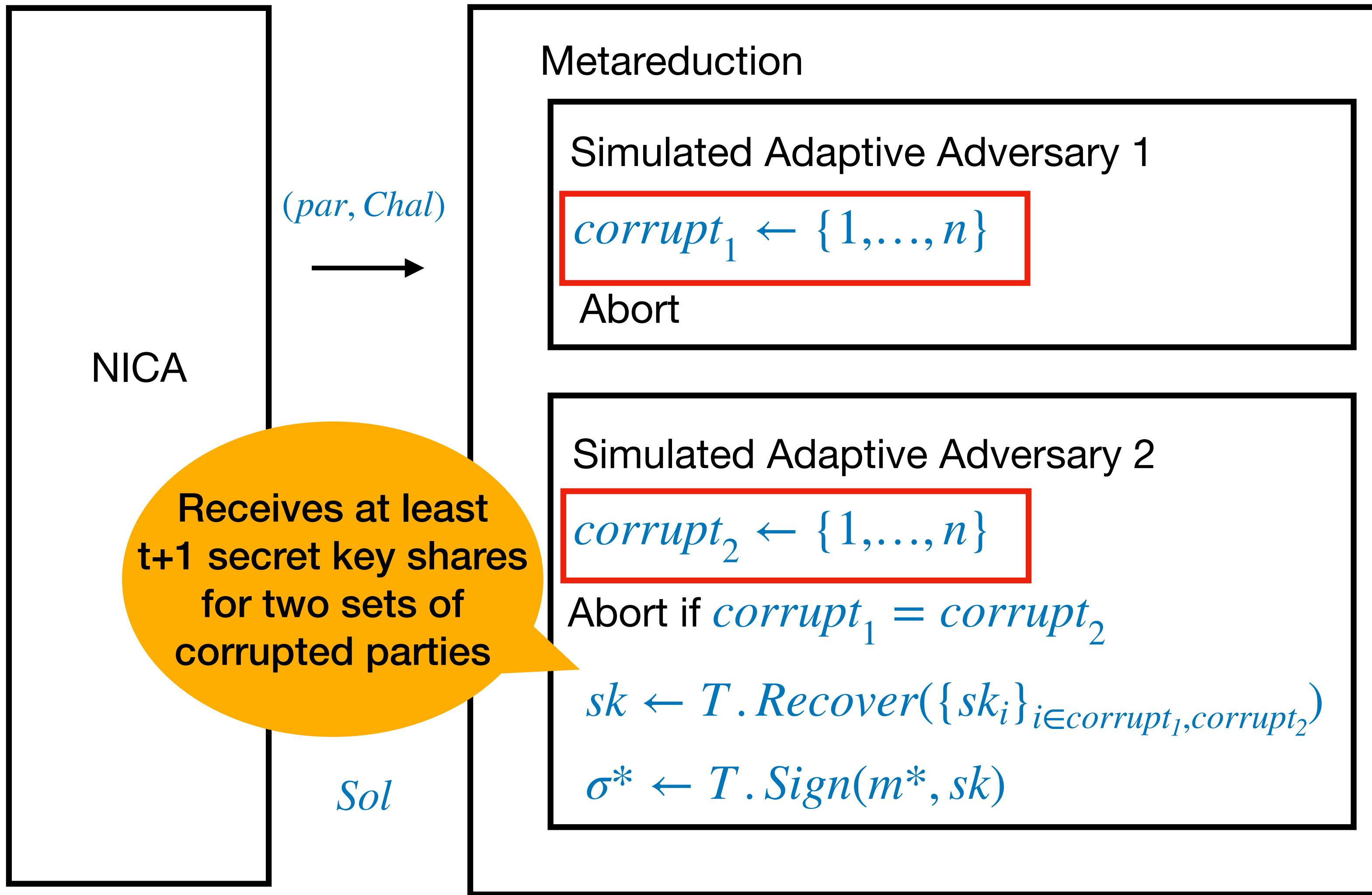
Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA



Impossibility of Key-Unique Adaptive Threshold Signatures Under NICA

