Fully Adaptive Schnorr Threshold Signatures

Elizabeth Crites*

University of Edinburgh

Chelsea Komlo

University of Waterloo
Zcash Foundation, Dfns

Mary Maller

Ethereum Foundation
PQShield

Why Schnorr signatures?

Why Schnorr signatures?

- NIST standard signatures: RSA, ECDSA, EdDSA (Aug. 2022)

Why Schnorr signatures?

- NIST standard signatures: RSA, ECDSA, EdDSA (Aug. 2022)
- EdDSA & Schnorr verification are the same

Why Schnorr signatures?

- NIST standard signatures: RSA, ECDSA, EdDSA (Aug. 2022)
- EdDSA & Schnorr verification are the same
- RSA signatures are large (~6x ECDSA/EdDSA)

Why Schnorr signatures?

- NIST standard signatures: RSA, ECDSA, EdDSA (Aug. 2022)
- EdDSA & Schnorr verification are the same
- RSA signatures are large (~6x ECDSA/EdDSA)
- ECDSA requires nonce inversion and other complexities

Why Schnorr signatures?

- NIST standard signatures: RSA, ECDSA, EdDSA (Aug. 2022)
- EdDSA & Schnorr verification are the same
- RSA signatures are large (~6x ECDSA/EdDSA)
- ECDSA requires nonce inversion and other complexities
    - no security reduction like Schnorr -> DL + ROM

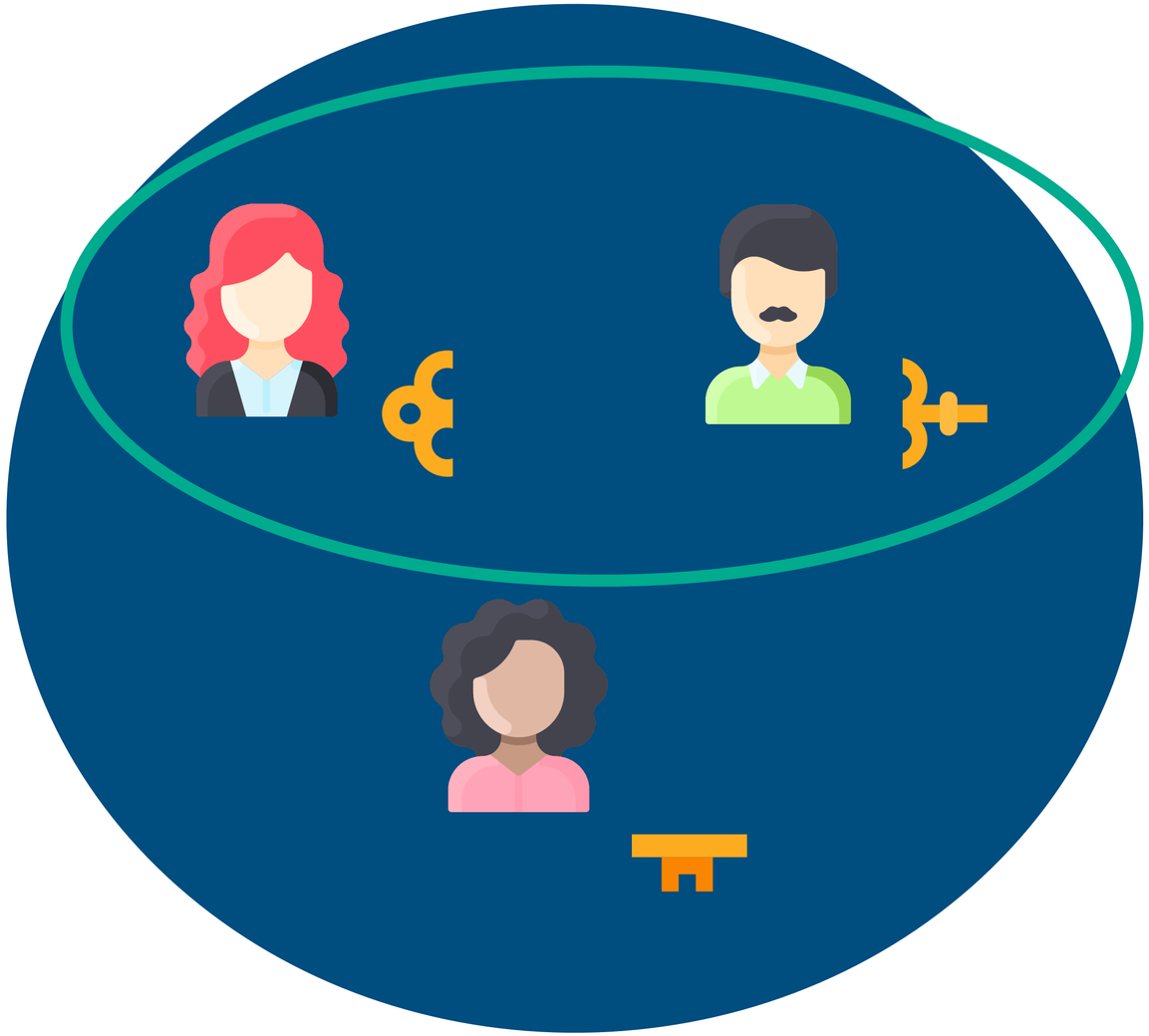
Why Schnorr signatures?

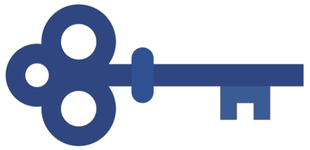
- NIST standard signatures: RSA, ECDSA, EdDSA (Aug. 2022)
- EdDSA & Schnorr verification are the same
- RSA signatures are large (~6x ECDSA/EdDSA)
- ECDSA requires nonce inversion and other complexities
    - no security reduction like Schnorr -> DL + ROM
- BLS requires bilinear pairings (slow to verify, not a NIST standard - yet!)

Why Schnorr signatures?

- NIST standard signatures: RSA, ECDSA, EdDSA (Aug. 2022)
- EdDSA & Schnorr verification are the same
- RSA signatures are large (~6x ECDSA/EdDSA)
- ECDSA requires nonce inversion and other complexities
    - no security reduction like Schnorr -> DL + ROM
- BLS requires bilinear pairings (slow to verify, not a NIST standard - yet!)
    - adaptive security of BLS [BL22]

What are threshold signatures?



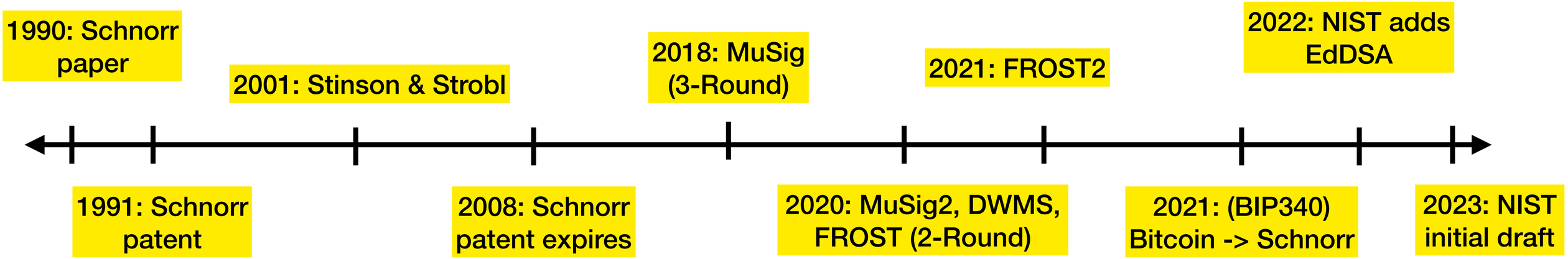


Public Key PK

- t-out-of-n
- trusted key generation or DKG to produce PK

$(2,3)$ Example

Why multi-party Schnorr signatures? Why now?



Standards

NIST COMPUTER SECURITY RESOURCE CENTER CSRC

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

Main Goals

- output signature that verifies like standard, single-party Schnorr signature
- few signing rounds
- reasonable security assumptions
- concurrent security
- adaptive security

(Single-Party) Schnorr Signatures [Sch91]

Signer:

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

(Single-Party) Schnorr Signatures [Sch91]

Signer:

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

To sign a message m:

$r \leftarrow \mathbb{F};\ R \leftarrow g^r$

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

$z \leftarrow r + c \cdot sk$

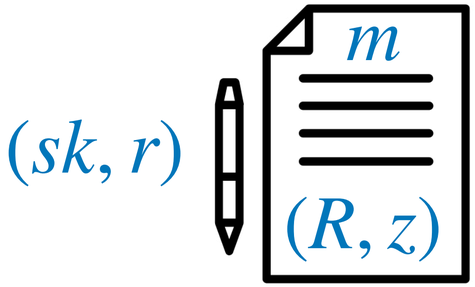
icons by flaticon.com

(Single-Party) Schnorr Signatures [Sch91]

$$\xrightarrow{sig = (R, z)}$$

Signer:

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



To sign a message m:

$r \leftarrow \mathbb{F};\ R \leftarrow g^r$

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

$z \leftarrow r + c \cdot sk$

icons by flaticon.com

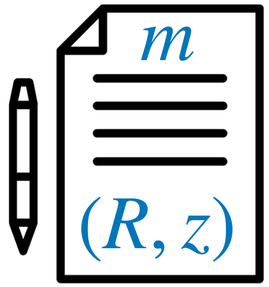
(Single-Party) Schnorr Signatures [Sch91]

$$sig = (R, z) \longrightarrow$$

Signer:

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

(sk, r)



Verifier:

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

$R \cdot PK^c = g^z$ ✓

To sign a message m:

$r \leftarrow \mathbb{F};\ R \leftarrow g^r$

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

$z \leftarrow r + c \cdot sk$

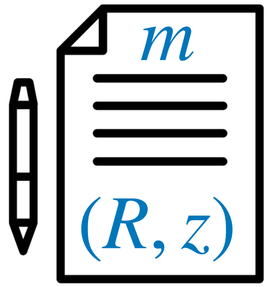
(Single-Party) Schnorr Signatures [Sch91]

$$\xrightarrow{sig = (R, z)}$$

Signer:

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

(sk, r)



Verifier:

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

$R \cdot PK^c = g^z$ ✓

To sign a message m:

$r \leftarrow \mathbb{F};\ R \leftarrow g^r$

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

$z \leftarrow r + c \cdot sk$

Main security property:

unforgeability

icons by flaticon.com

Multi-Party Schnorr Signatures

How to share r ?

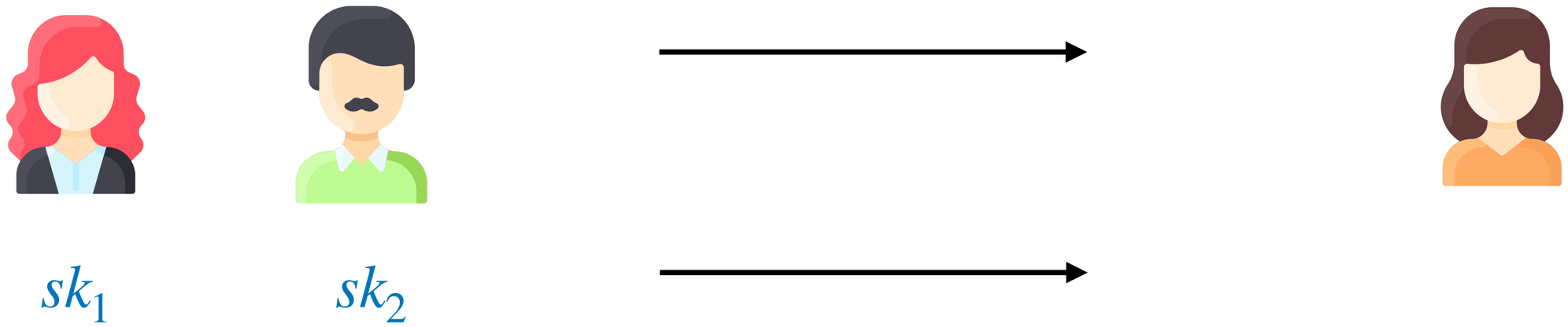
How to share sk ?

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

$$sig = (R, z)$$

2-Round Threshold Scheme

Key Generation: PK



2-Round Threshold Scheme

Key Generation: PK

sk_1

$R_1 \leftarrow g^{r_1}$

sk_2

$R_2 \leftarrow g^{r_2}$

2-Round Threshold Scheme

Key Generation: PK

Round 1:

R_1, R_2

sk_1

sk_2

$R_1 \leftarrow g^{r_1}$

$R_2 \leftarrow g^{r_2}$

2-Round Threshold Scheme

Key Generation: PK

sk_1 $\quad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\quad$ $R_2 \leftarrow g^{r_2}$

$$R = R_1 R_2$$

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

Round 1:

R_1, R_2

2-Round Threshold Scheme

Key Generation: PK

Round 1:

R_1, R_2

sk_1 $\qquad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\qquad$ $R_2 \leftarrow g^{r_2}$

$$R = R_1 R_2$$

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

$z_1 \leftarrow r_1 + c \cdot \lambda_1^{\mathcal{S}} \cdot sk_1$ $\qquad$ $z_2 \leftarrow r_2 + c \cdot \lambda_2^{\mathcal{S}} \cdot sk_2$

2-Round Threshold Scheme

Key Generation: PK

sk_1 $\quad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\quad$ $R_2 \leftarrow g^{r_2}$

$R = R_1 R_2$

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

Round 1:

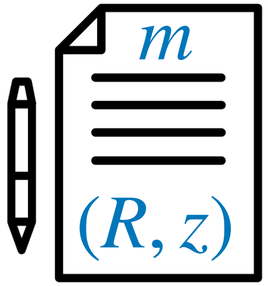
R_1, R_2

Round 2:

z_1, z_2

(sk_1, r_1)

(sk_2, r_2)



$z_1 \leftarrow r_1 + c \cdot \lambda_1^{\mathcal{S}} \cdot sk_1$ $\quad$ $z_2 \leftarrow r_2 + c \cdot \lambda_2^{\mathcal{S}} \cdot sk_2$

2-Round Threshold Scheme

Key Generation: PK

sk_1 $\quad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\quad$ $R_2 \leftarrow g^{r_2}$

$R = R_1 R_2$

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

Round 1:

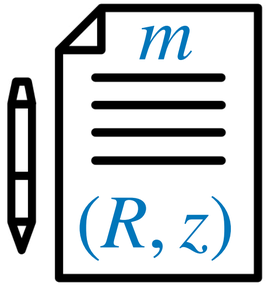
R_1, R_2

Round 2:

z_1, z_2

(sk_1, r_1)

(sk_2, r_2)



Combine / Verify:

$z \leftarrow z_1 + z_2$

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

$R \cdot PK^c = g^z$ ✓

$z_1 \leftarrow r_1 + c \cdot \lambda_1^{\mathcal{S}} \cdot sk_1$ $\quad$ $z_2 \leftarrow r_2 + c \cdot \lambda_2^{\mathcal{S}} \cdot sk_2$

2-Round Threshold Scheme

Key Generation: PK

sk_1 $\quad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\quad$ $R_2 \leftarrow g^{r_2}$

$R = R_1 R_2$

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

$z_1 \leftarrow r_1 + c \cdot \lambda_1^{\mathcal{S}} \cdot sk_1$ $\quad$ $z_2 \leftarrow r_2 + c \cdot \lambda_2^{\mathcal{S}} \cdot sk_2$

Round 1:

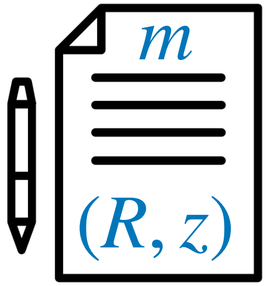
R_1, R_2 $\longrightarrow$

Round 2:

z_1, z_2 $\longrightarrow$

(sk_1, r_1)

(sk_2, r_2)



Combine / Verify:

$z \leftarrow z_1 + z_2$

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

$R \cdot PK^c = g^z$ ✓

NOT concurrently secure ✗

Concurrent Security: ROS Attacks

[NKDM03, DEFKLNS19, BLLOR21]

Session 1

Session k

sk_1

$\cdots$

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

$R_1^{(1)}$

$R_2^{(1)}$

$\cdots$

Session k

$R_1^{(k)}$

$R_2^{(k)}$

sk_2

Concurrent Security: ROS Attacks

[NKDM03, DEFKLNS19, BLLOR21]

Session 1

sk_1

$R_1^{(1)}$

$R_2^{(1)}$

$\cdots$

Session k

$R_1^{(k)}$

$R_2^{(k)}$

sk_2

Can forge!

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

Can forge!

Affected:

- multi-signatures
- threshold signatures
- blind signatures

Solution: Force adversary to commit to its nonces…

Sparkle ✨

Key Generation: PK

sk_1

sk_2

Sparkle ✨

Key Generation: PK

sk_1

$R_1 \leftarrow g^{r_1}$

sk_2

$R_2 \leftarrow g^{r_2}$

Sparkle ✨

Key Generation: PK

sk_1

$R_1 \leftarrow g^{r_1}$

sk_2

$R_2 \leftarrow g^{r_2}$

Round 1:

$H'(R_1, m, \mathcal{S}), H'(R_2, m, \mathcal{S})$

Sparkle

Key Generation: PK

sk_1

$R_1 \leftarrow g^{r_1}$

sk_2

$R_2 \leftarrow g^{r_2}$

Round 1:

$H'(R_1, m, \mathcal{S}), H'(R_2, m, \mathcal{S})$

$\longrightarrow$

Round 2:

R_1, R_2

$\longrightarrow$

$\longrightarrow$

Sparkle

Key Generation: PK

sk_1 $\quad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\quad$ $R_2 \leftarrow g^{r_2}$

$R = R_1 R_2$

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

Round 1:

$H'(R_1, m, \mathcal{S}), H'(R_2, m, \mathcal{S})$

$\longrightarrow$

Round 2:

R_1, R_2

$\longrightarrow$

$\longrightarrow$

Sparkle

Key Generation: PK

Round 1:

$$H'(R_1, m, \mathcal{S}), H'(R_2, m, \mathcal{S}) \longrightarrow$$

Round 2:

$$R_1, R_2 \longrightarrow$$

sk_1 $\qquad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\qquad$ $R_2 \leftarrow g^{r_2}$

$$R = R_1 R_2 \longrightarrow$$

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

$z_1 \leftarrow r_1 + c \cdot \lambda_1^{\mathcal{S}} \cdot sk_1$ $\qquad$ $z_2 \leftarrow r_2 + c \cdot \lambda_2^{\mathcal{S}} \cdot sk_2$

Sparkle

Key Generation: PK

sk_1 $\qquad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\qquad$ $R_2 \leftarrow g^{r_2}$

$R = R_1 R_2$

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

Round 1:

$H'(R_1, m, \mathcal{S}), H'(R_2, m, \mathcal{S})$

$\longrightarrow$

Round 2:

R_1, R_2

$\longrightarrow$

Round 3:

z_1, z_2

$\longrightarrow$

$z_1 \leftarrow r_1 + c \cdot \lambda_1^{\mathcal{S}} \cdot sk_1$ $\qquad$ $z_2 \leftarrow r_2 + c \cdot \lambda_2^{\mathcal{S}} \cdot sk_2$

Sparkle ✨

Key Generation: PK

sk_1 $\quad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\quad$ $R_2 \leftarrow g^{r_2}$

$R = R_1 R_2$

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

$z_1 \leftarrow r_1 + c \cdot \lambda_1^{\mathcal{S}} \cdot sk_1$ $\quad$ $z_2 \leftarrow r_2 + c \cdot \lambda_2^{\mathcal{S}} \cdot sk_2$

Round 1:

$H'(R_1, m, \mathcal{S}), H'(R_2, m, \mathcal{S})$ $\longrightarrow$

Round 2:

R_1, R_2 $\longrightarrow$

Round 3:

z_1, z_2 $\longrightarrow$

Combine / Verify:

$z \leftarrow z_1 + z_2$

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

$R \cdot PK^c = g^z$ ✓

Sparkle ✨

Key Generation: PK

sk_1 $\quad$ sk_2

$R_1 \leftarrow g^{r_1}$ $\quad$ $R_2 \leftarrow g^{r_2}$

$R = R_1 R_2$

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

$z_1 \leftarrow r_1 + c \cdot \lambda_1^{\mathcal{S}} \cdot sk_1$ $\quad$ $z_2 \leftarrow r_2 + c \cdot \lambda_2^{\mathcal{S}} \cdot sk_2$

Round 1:

$H'(R_1, m, \mathcal{S}), H'(R_2, m, \mathcal{S})$ $\longrightarrow$

Round 2:

$(m, \mathcal{S}) \longrightarrow R_1, R_2$ $\longrightarrow$

Round 3:

z_1, z_2 $\longrightarrow$

Combine / Verify:

$z \leftarrow z_1 + z_2$

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

$R \cdot PK^c = g^z$ ✓

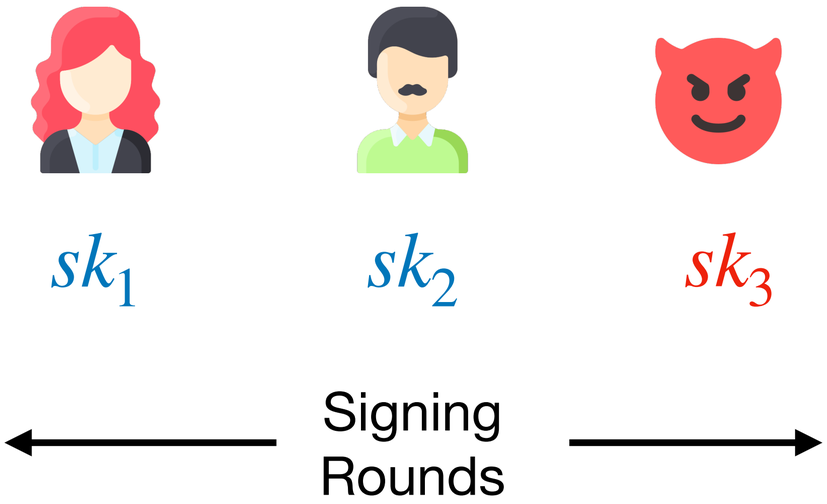
Concurrently secure ✓
(even when $(m, \mathcal{S})$ delayed to Round 2)

Recent Schnorr Threshold Signatures

Scheme	Signing Rounds	*Static* Assumptions	Concurrent Security	Adaptive Security
FROST [KG20, BCKMTZ22] FROST2 [CKM21, BCKMTZ22]	2	OMDL + ROM	✓	✗
Lindell22 Classic Schnorr [Mak22] Sparkle	3	Schnorr Threshold DL + ROM		Exp. loss ✓ (Tight)

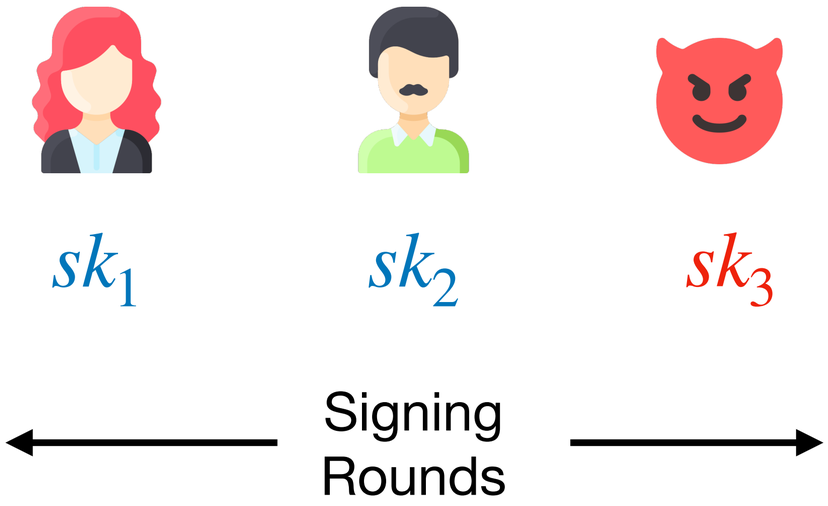
Adaptive Security

Static Corruption

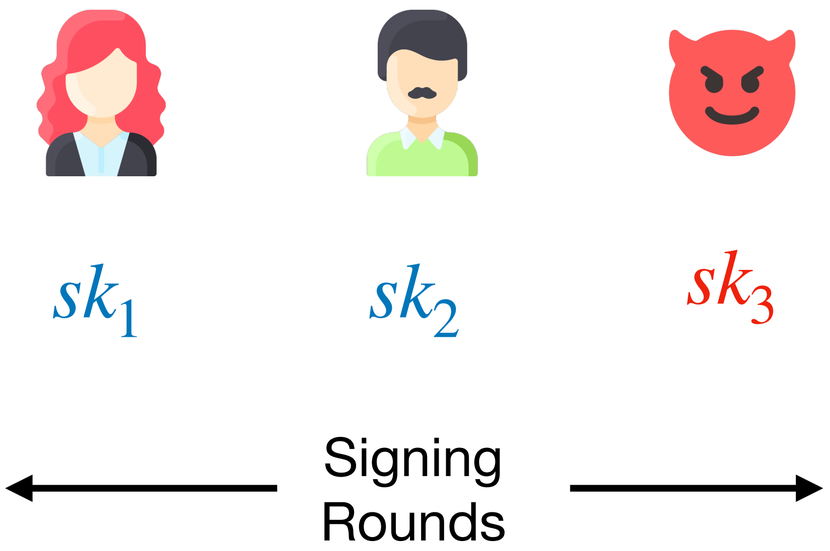


Adaptive Security

Static Corruption

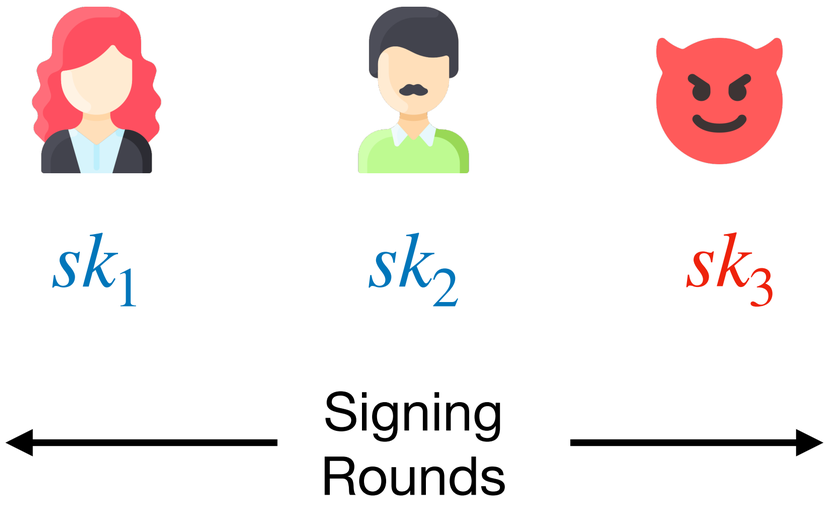


Adaptive Corruption

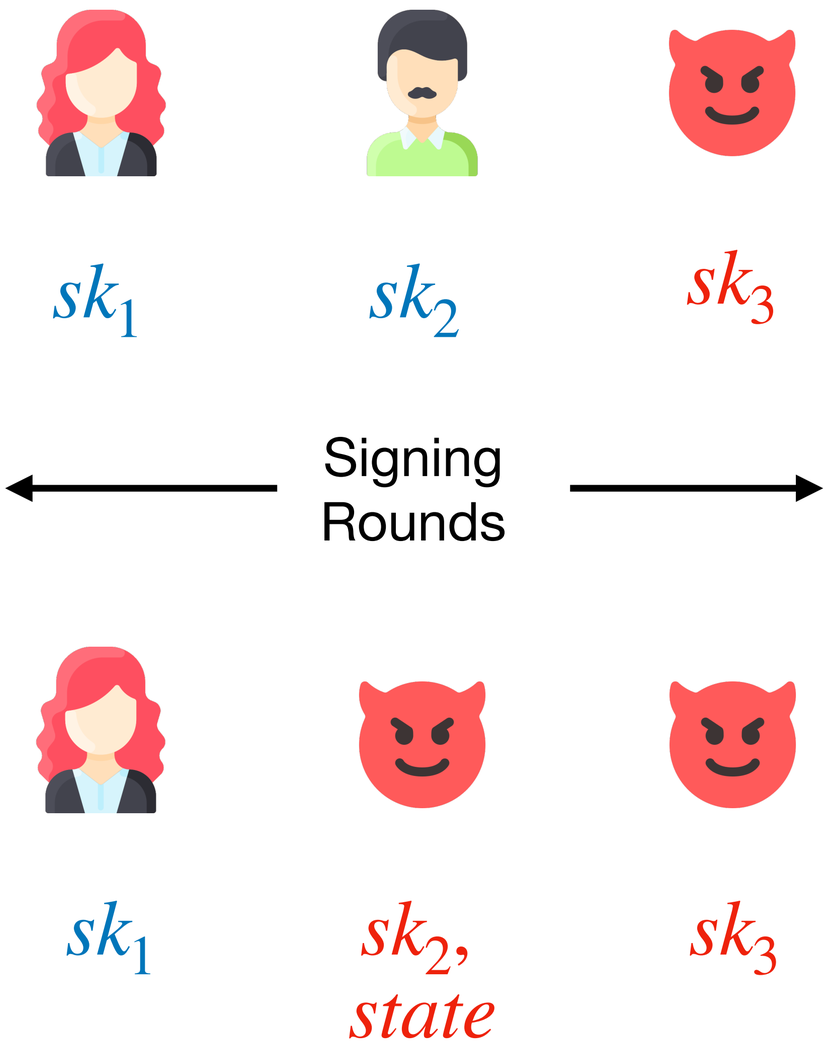


Adaptive Security

Static Corruption



Adaptive Corruption



Concurrent Adaptive Security

	Session 1			Session 2		
						
Round 1:	sk_1	sk_2	sk_3	sk_2	sk_3	sk_4
Round 2:	cm_1		cm_3	cm_2	cm_3	cm_4
Round 3:				R_2	R_3	

Concurrent Adaptive Security

	Session 1			Session 2		
						
Round 1:	sk_1	sk_2	sk_3	sk_2	sk_3	sk_4
Round 2:	cm_1		cm_3	cm_2	cm_3	cm_4
Round 3:				R_2	R_3	

Adaptive Security is Challenging

Adaptive Security is Challenging

- easy when n is small: reduction guesses corrupt parties ahead of time and aborts if wrong

Adaptive Security is Challenging

- easy when n is small: reduction guesses corrupt parties ahead of time and aborts if wrong
    - incurs exponential loss

Adaptive Security is Challenging

- easy when n is small: reduction guesses corrupt parties ahead of time and aborts if wrong
    - incurs exponential loss
- heavyweight tools, like non-committing encryption

Adaptive Security is Challenging

- easy when n is small: reduction guesses corrupt parties ahead of time and aborts if wrong
    - incurs exponential loss
- heavyweight tools, like non-committing encryption
- secure erasure of secret state

Adaptive Security is Challenging

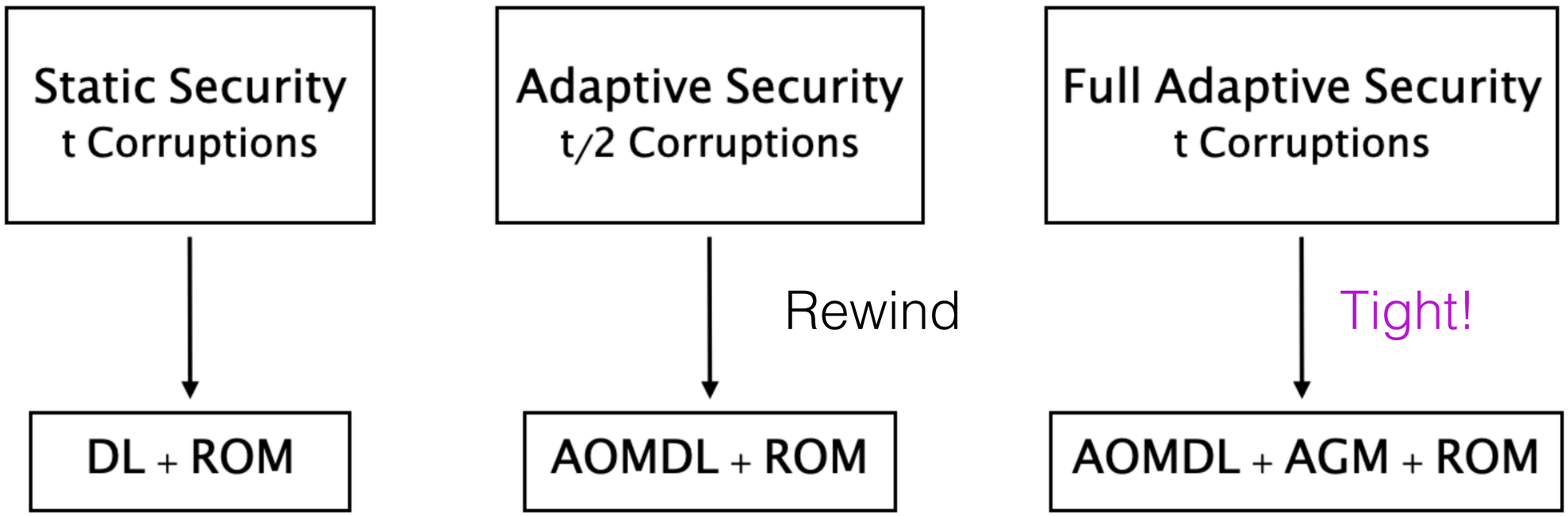
- easy when n is small: reduction guesses corrupt parties ahead of time and aborts if wrong
    - incurs exponential loss
- heavyweight tools, like non-committing encryption
- secure erasure of secret state
- hard when n = number of parties is large, i.e., $n > 1024$

Adaptive Security is Challenging

- easy when n is small: reduction guesses corrupt parties ahead of time and aborts if wrong
    - incurs exponential loss
- heavyweight tools, like non-committing encryption
- secure erasure of secret state
- hard when n = number of parties is large, i.e., $n > 1024$
    - large n important to prevent adversary from corrupting majority

Our Results

Sparkle



Same security loss as Schnorr

(Threshold = $t + 1$)

Our Results

Sparkle ✨

Static Security t Corruptions	Adaptive Security t/2 Corruptions	Full Adaptive Security t Corruptions
↓	↓ Rewind	↓ Tight!
DL + ROM	AOMDL + ROM	AOMDL + AGM + ROM
	Same security loss as Schnorr	

And concurrently secure! ✓

(Threshold = $t + 1$)

Adaptive Security under (A)OMDL

DL Oracle

Reduction

Adversary

Adaptive Security under (A)OMDL

OMDL Challenge

$(X_0, X_1, \ldots, X_t)$

DL Oracle

Reduction

Adversary

Adaptive Security under (A)OMDL

OMDL Challenge

$(X_0, X_1, \ldots, X_t)$

DL Oracle

Reduction

$PK, \{PK_i\}_{i \in n}$

Adversary

Adaptive Security under (A)OMDL

OMDL Challenge

$(X_0, X_1, \ldots, X_t)$

DL Oracle

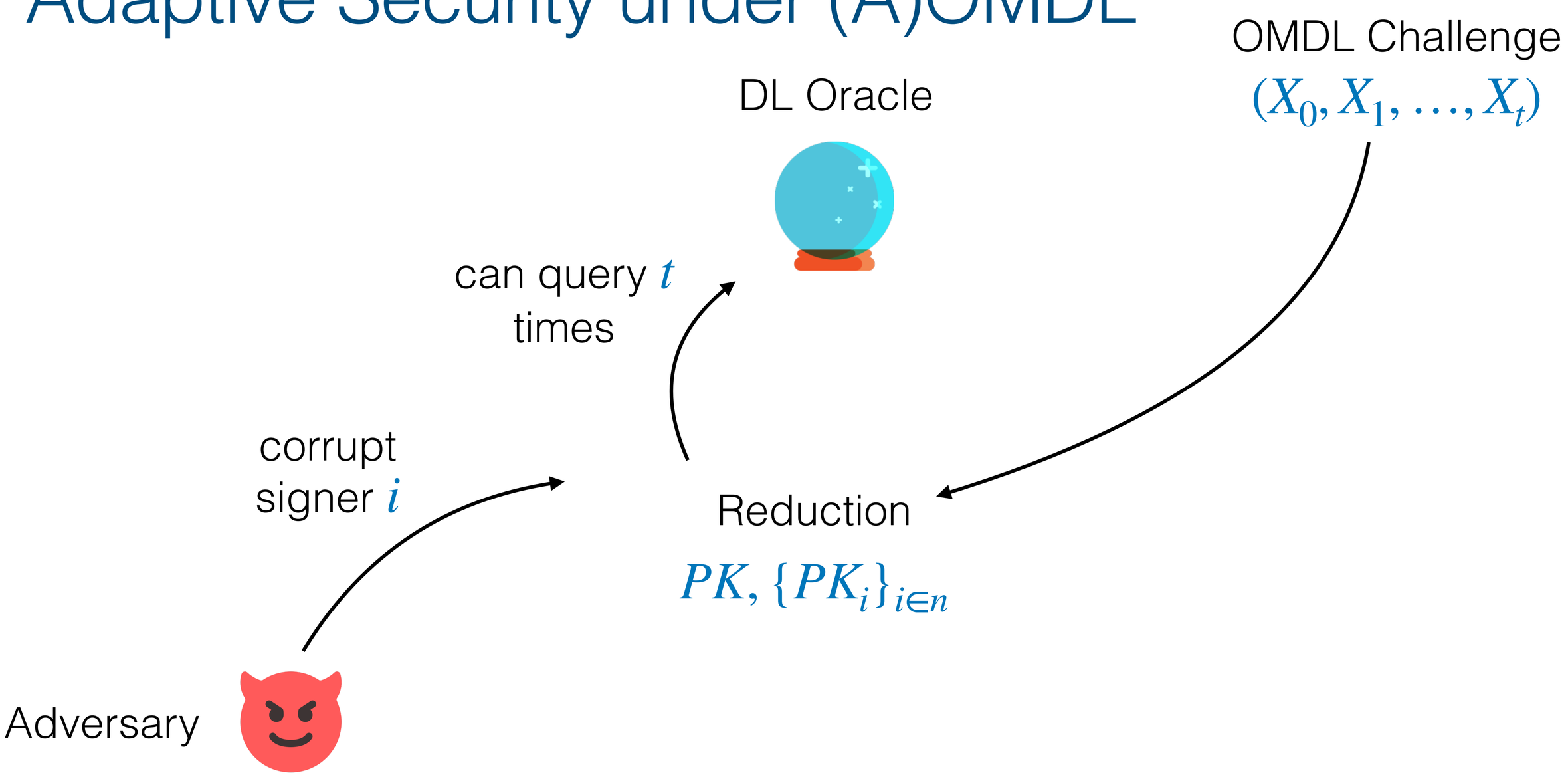
corrupt
signer i

Reduction

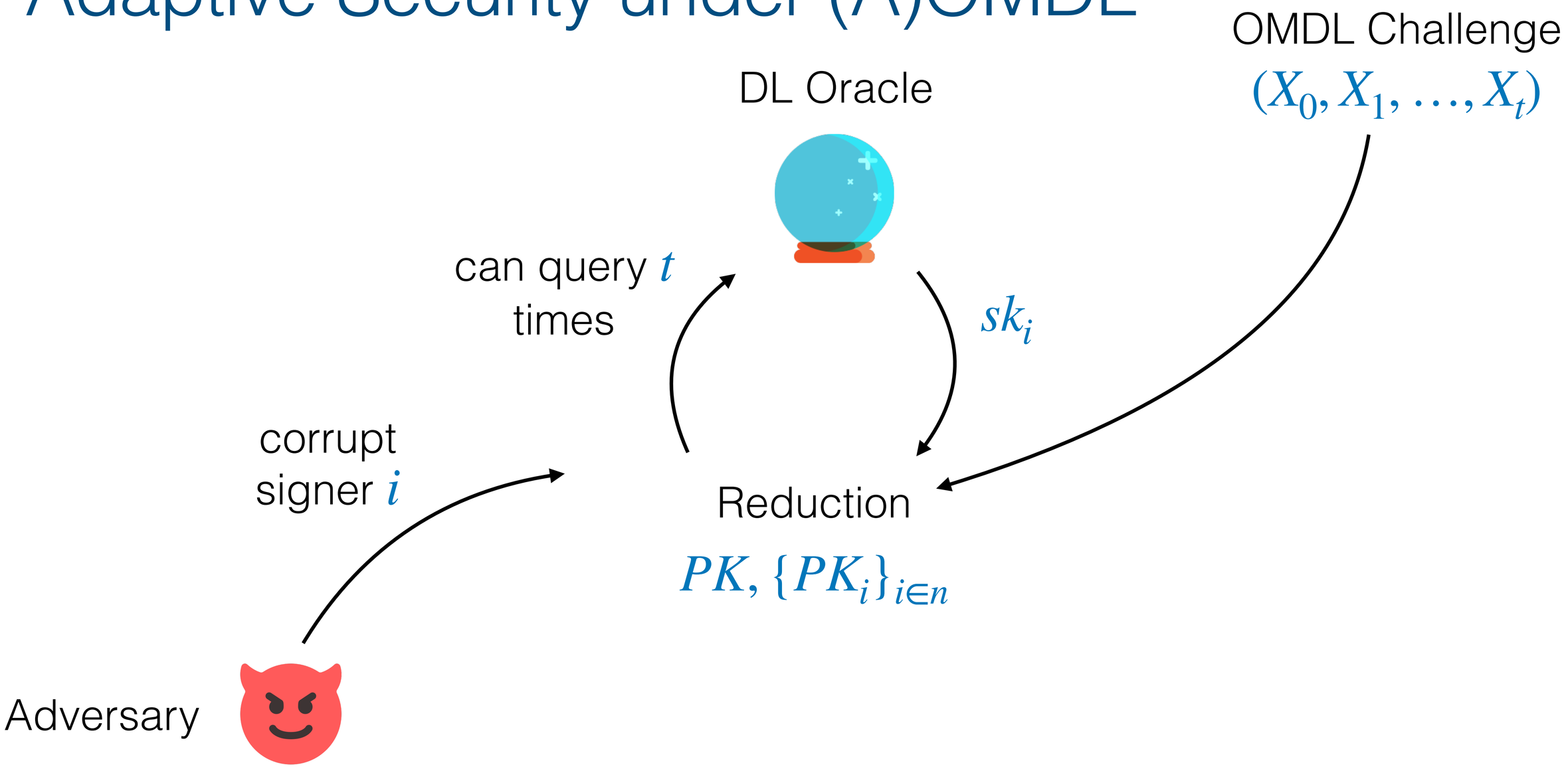
$PK, \{PK_i\}_{i \in n}$

Adversary

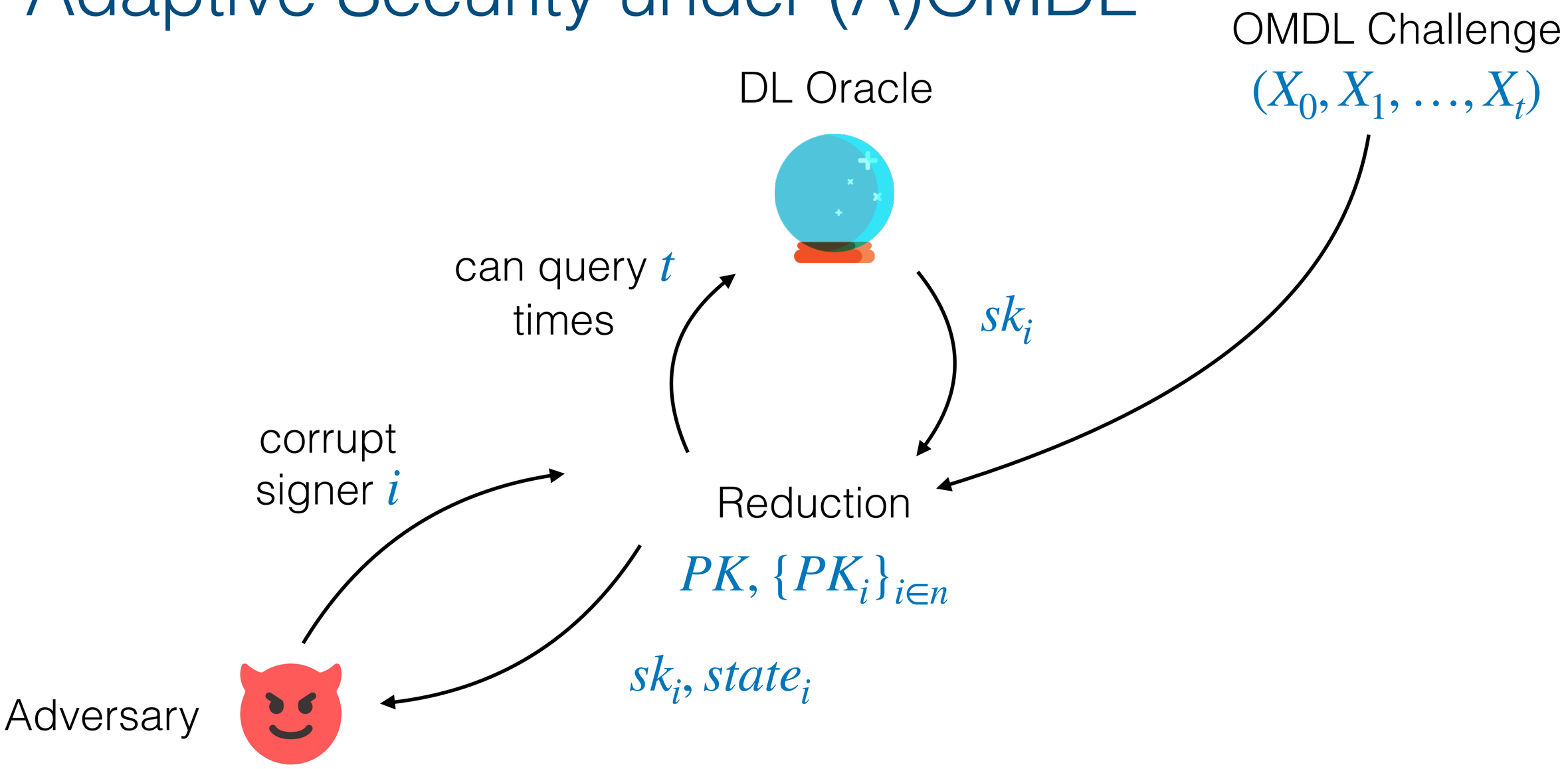
Adaptive Security under (A)OMDL



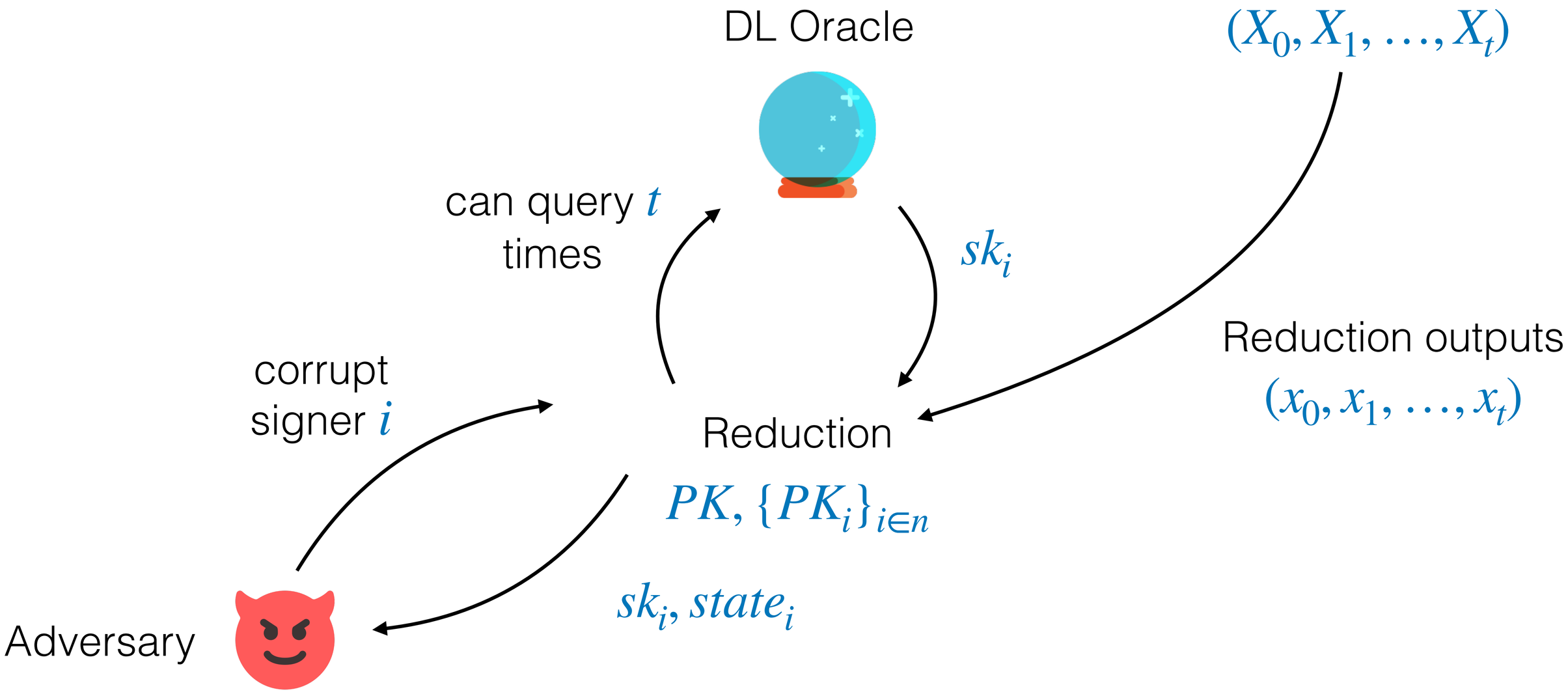
Adaptive Security under (A)OMDL



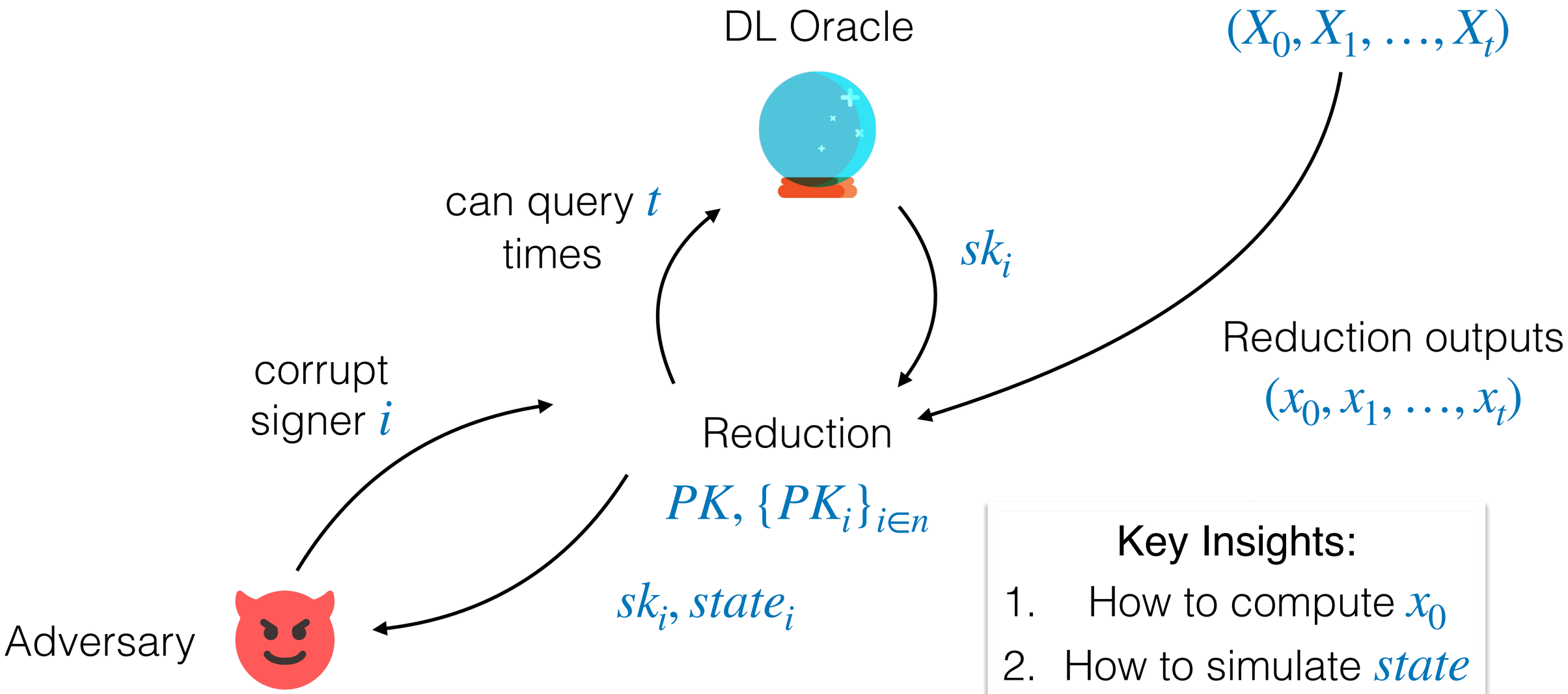
Adaptive Security under (A)OMDL



Adaptive Security under (A)OMDL



Adaptive Security under (A)OMDL



Key Insights:

1. How to compute x_0
2. How to simulate $state$

Key Takeaways

- Adaptive security is important, as threshold signatures are being deployed
- First fully adaptive security proof for threshold Schnorr signatures
- Challenging to achieve:
    - multi-party
    - multi-round
    - concurrently secure
    - adaptively secure
    - and looks like a standard, single-party Schnorr signature!

Key Takeaways

- Adaptive security is important, as threshold signatures are being deployed
- First fully adaptive security proof for threshold Schnorr signatures
- Challenging to achieve:
  - multi-party
  - multi-round
  - concurrently secure
  - adaptively secure
  - and looks like a standard, single-party Schnorr signature!

Coming Soon: Adaptive security of FROST

Thank you!