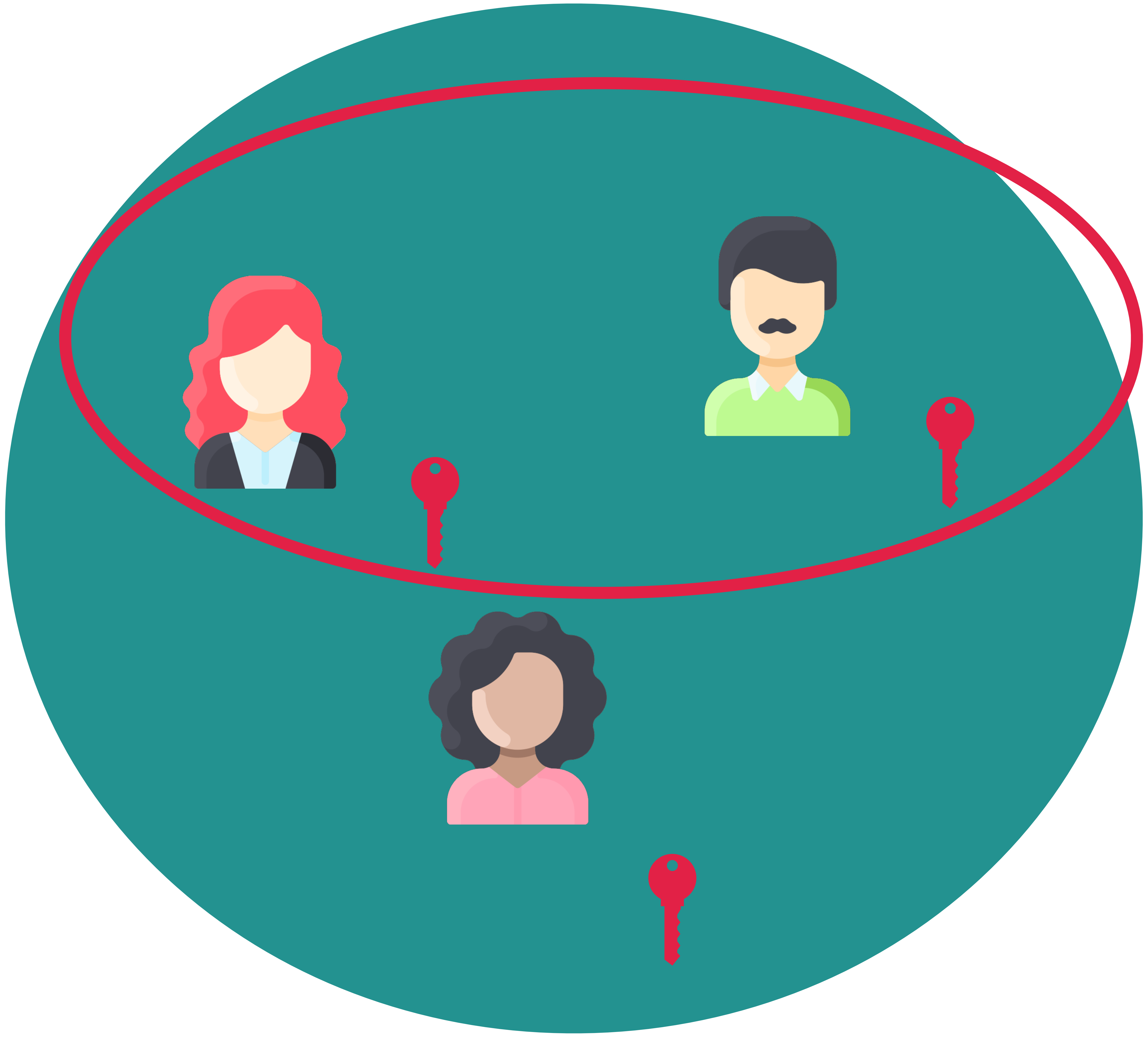
Stateless and Two-Round Threshold Schnorr Signatures

Chelsea Komlo

University of Waterloo, NEAR One

May 15, 2025

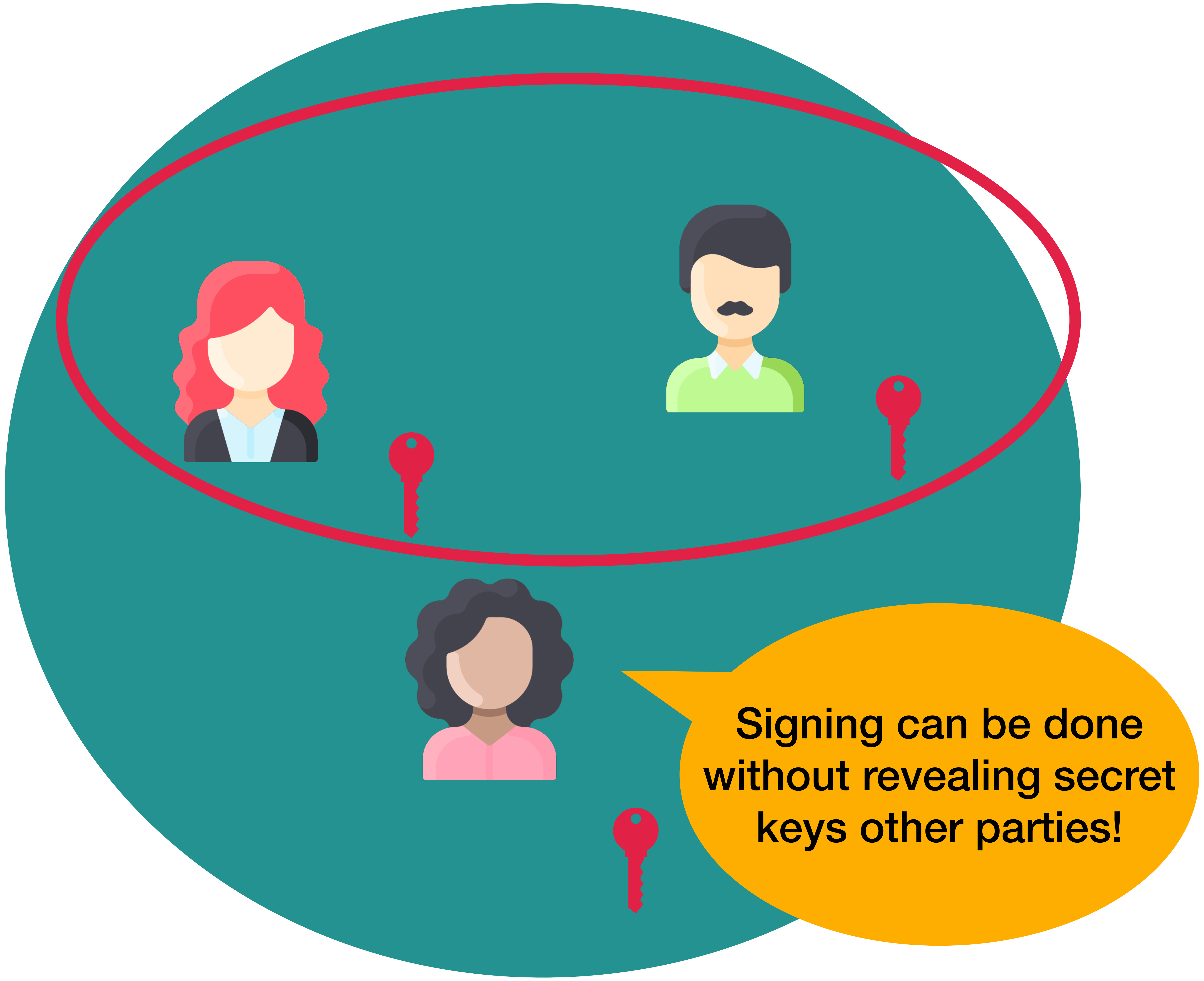
What are Threshold Signatures?



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

(2,3) Example

What are Threshold Signatures?



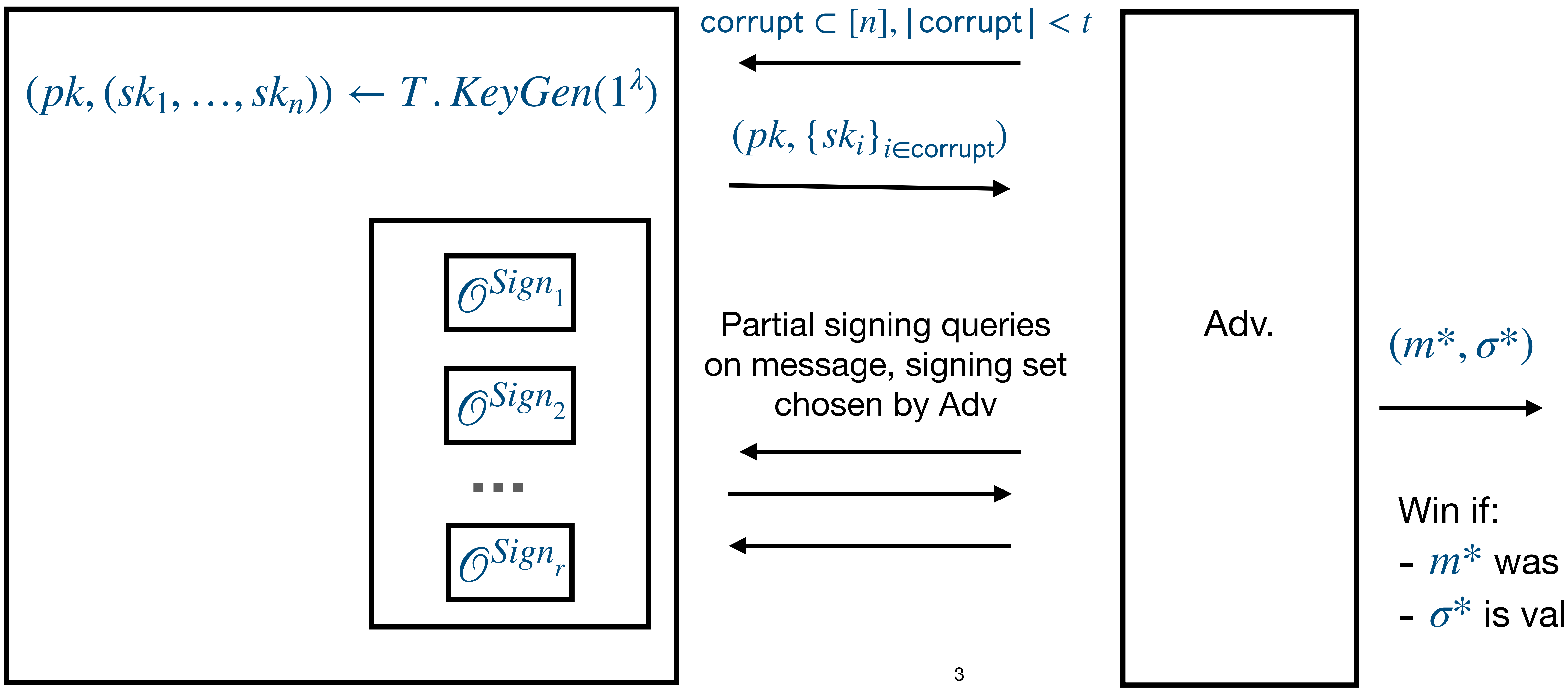
Public Key

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

(2,3) Example

Unforgeability

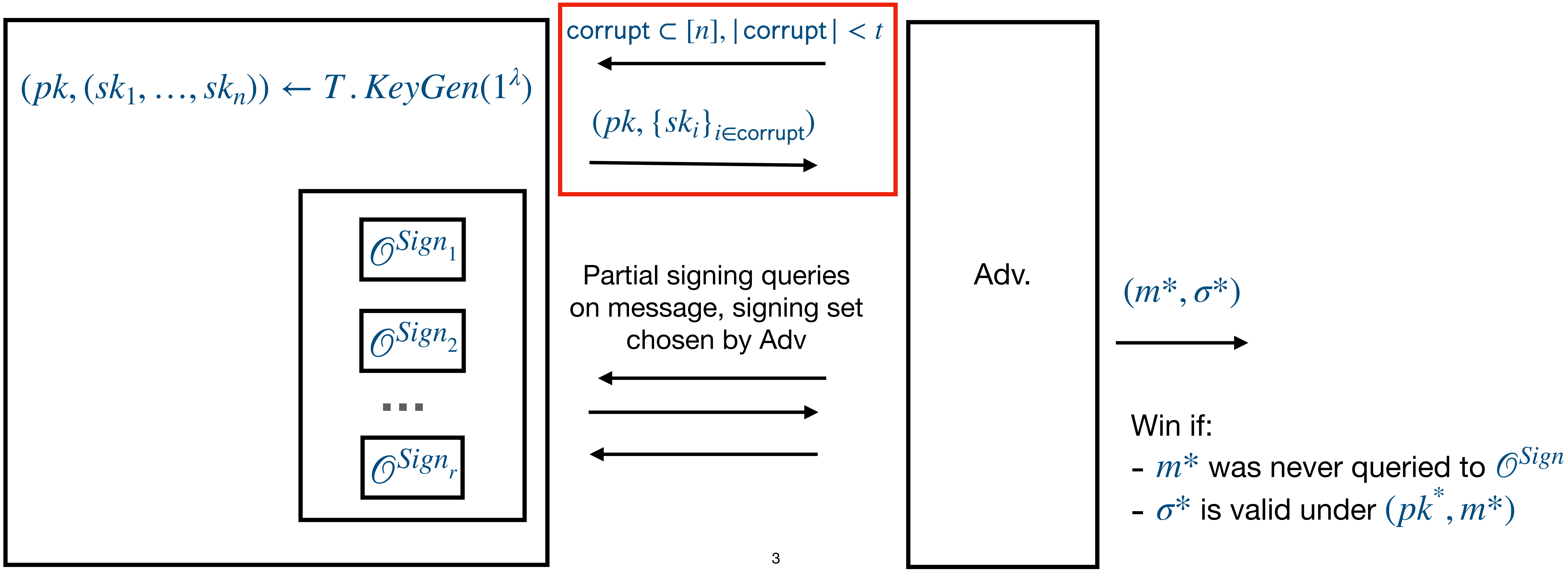
A threshold signature scheme T is secure if no PPT adversary can win the following game with non-negligible advantage:



Win if:
- m^* was never queried to $\mathcal{O}^{Sign}$
- σ^* is valid under (pk^*, m^*)

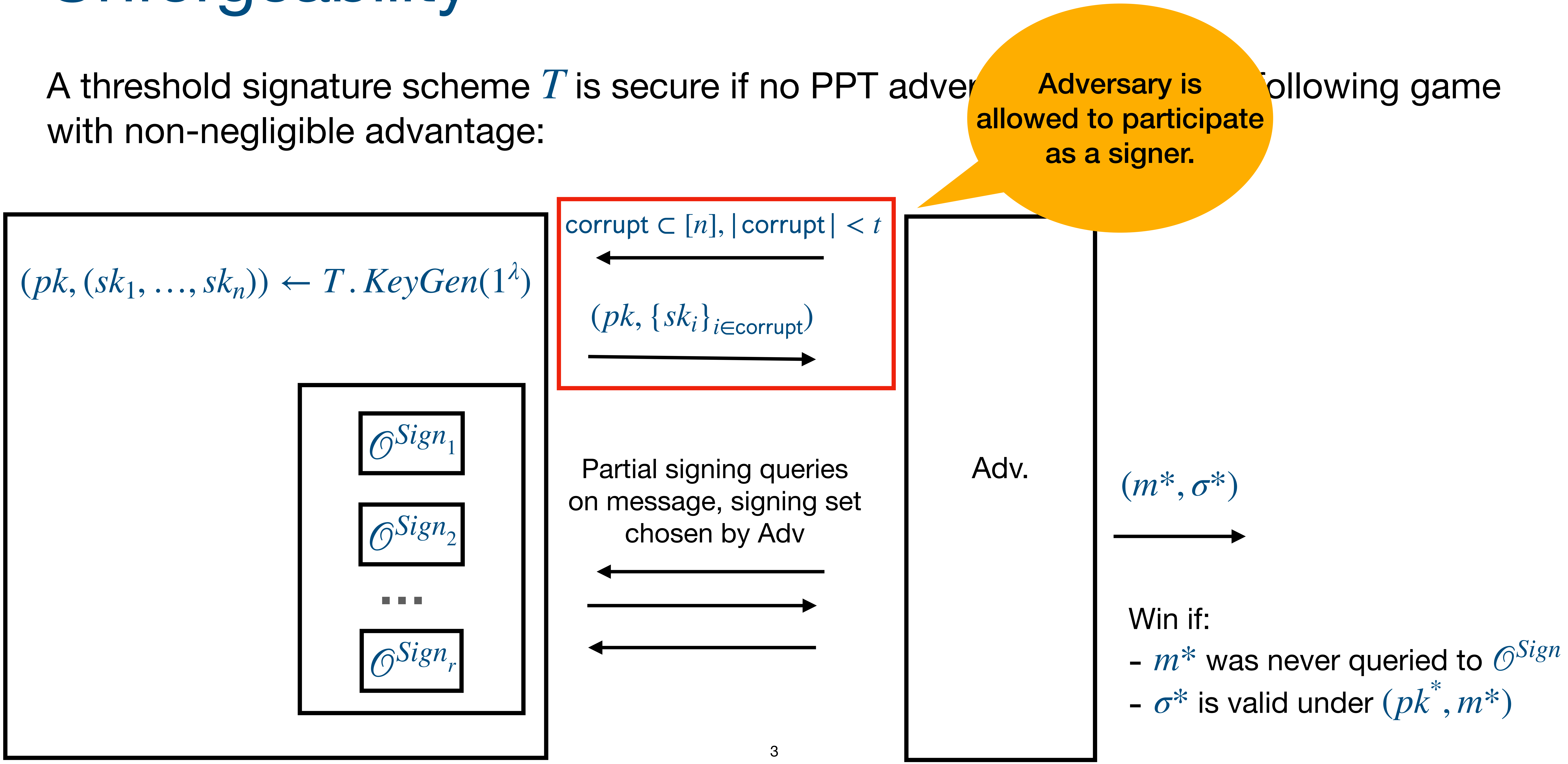
Unforgeability

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



Unforgeability

A threshold signature scheme T is secure if no PPT adver … following game with non-negligible advantage:



Multi-Party Schnorr Signatures

How to share r ?

How to share sk ?

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

$$sig = (R, z)$$

	Scheme	Assumptions	Signing Rounds
Multi-sigs (n-of-n)	MuSig [MPSW18, BDN18] SimpleMuSig [BDN18, C**K**M21]	DL+ROM	3
	MuSig2 [NRS21] DWMS [AB21] SpeedyMuSig [C**K**M21]	OMDL+ROM	2
Threshold (t-of-n)	Lindell22 Sparkle [C**K**M23]	Schnorr DL+ROM	3
	FROST [**K**G20, BC**K**MTZ22] FROST2 [C**K**M21]	OMDL+ROM	2

Concurrently Secure ✓

Randomized (Stateful) ✗

One-More Discrete Log (OMDL)

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

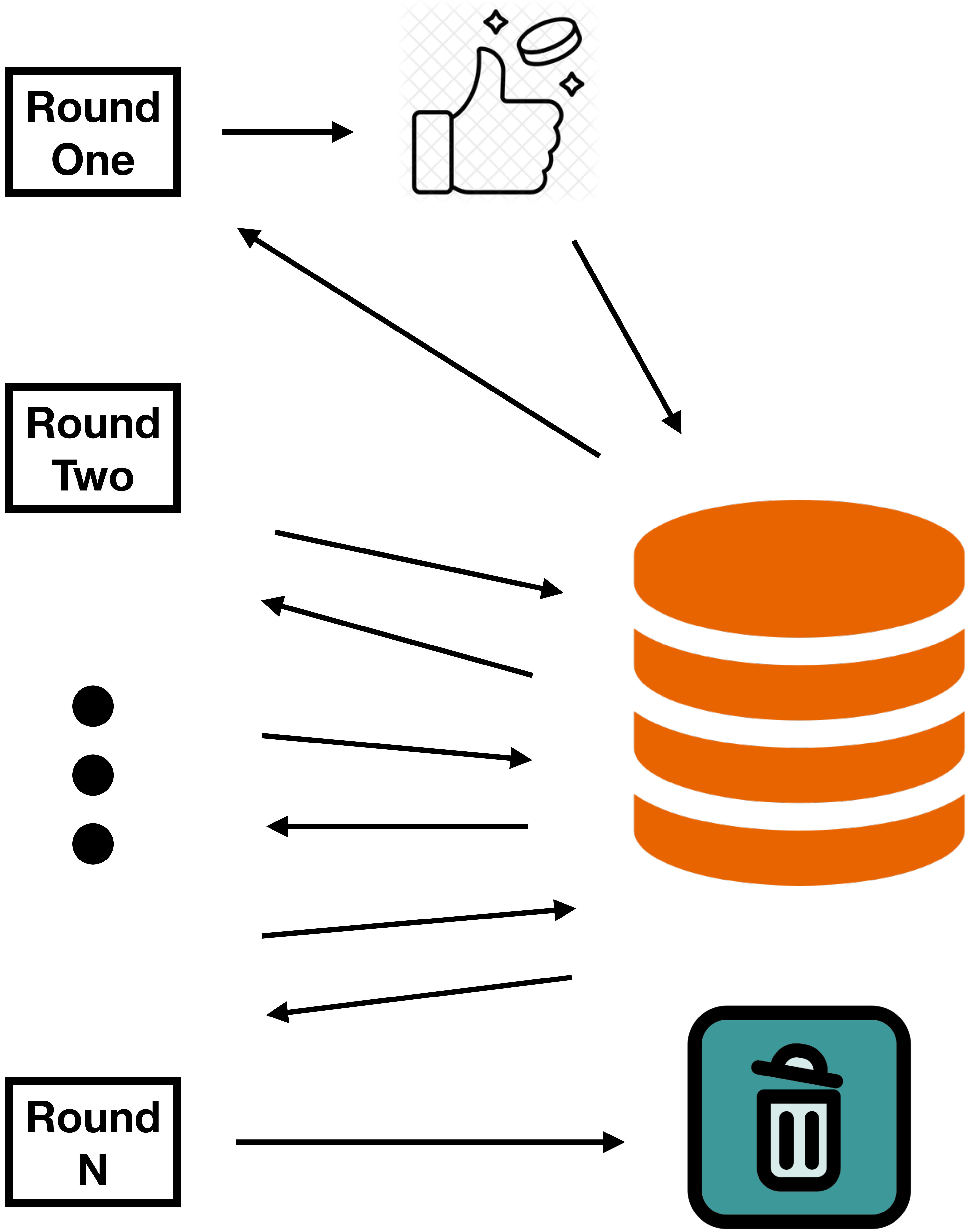
Honest minority: up to (t-1) corrupt; at least one honest (t total).

Concurrently Secure ✓

Randomized (Stateful) ✗

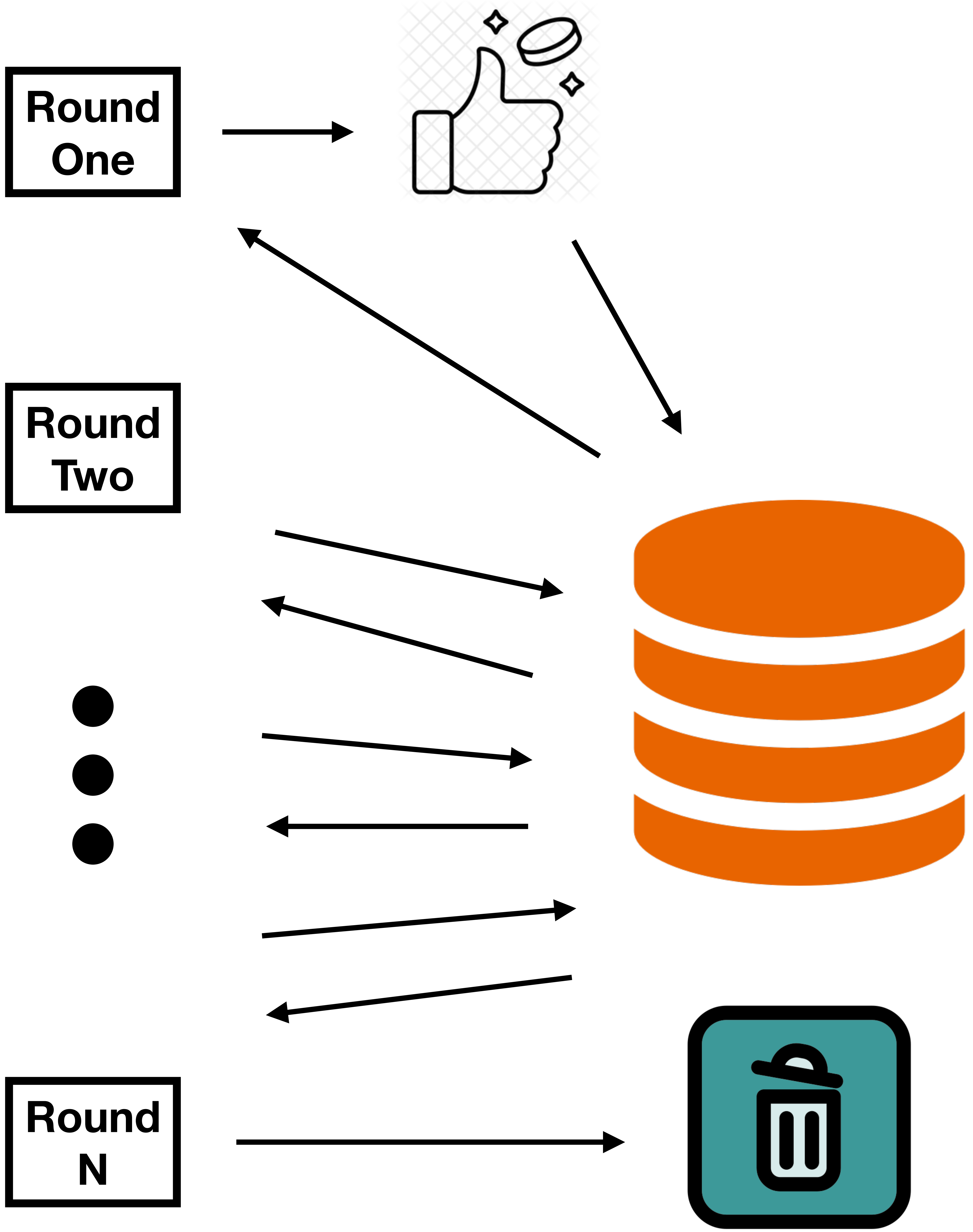
One-More Discrete Log (OMDL)

Motivation



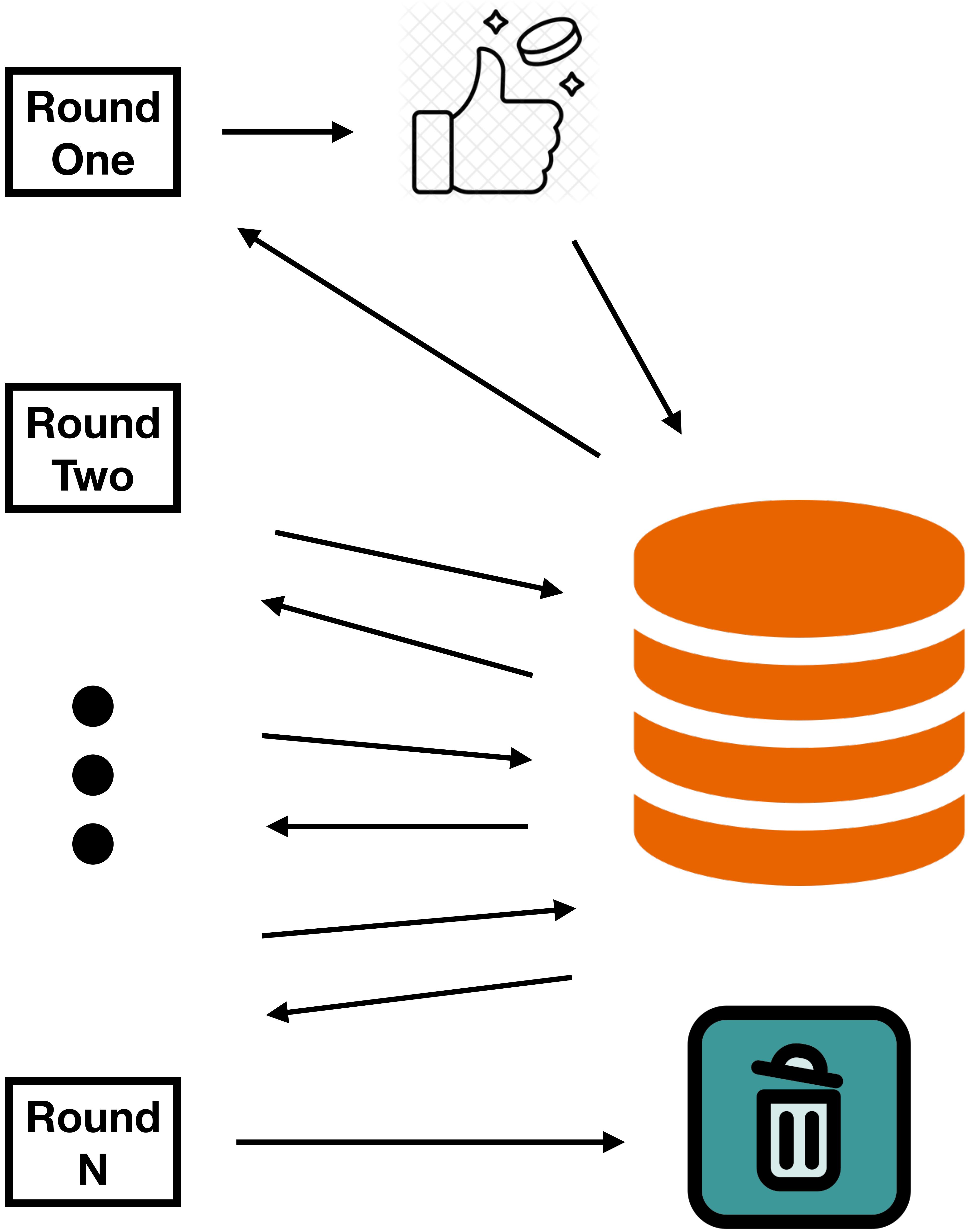
Motivation

- Randomized multi-party schemes require state-keeping between rounds



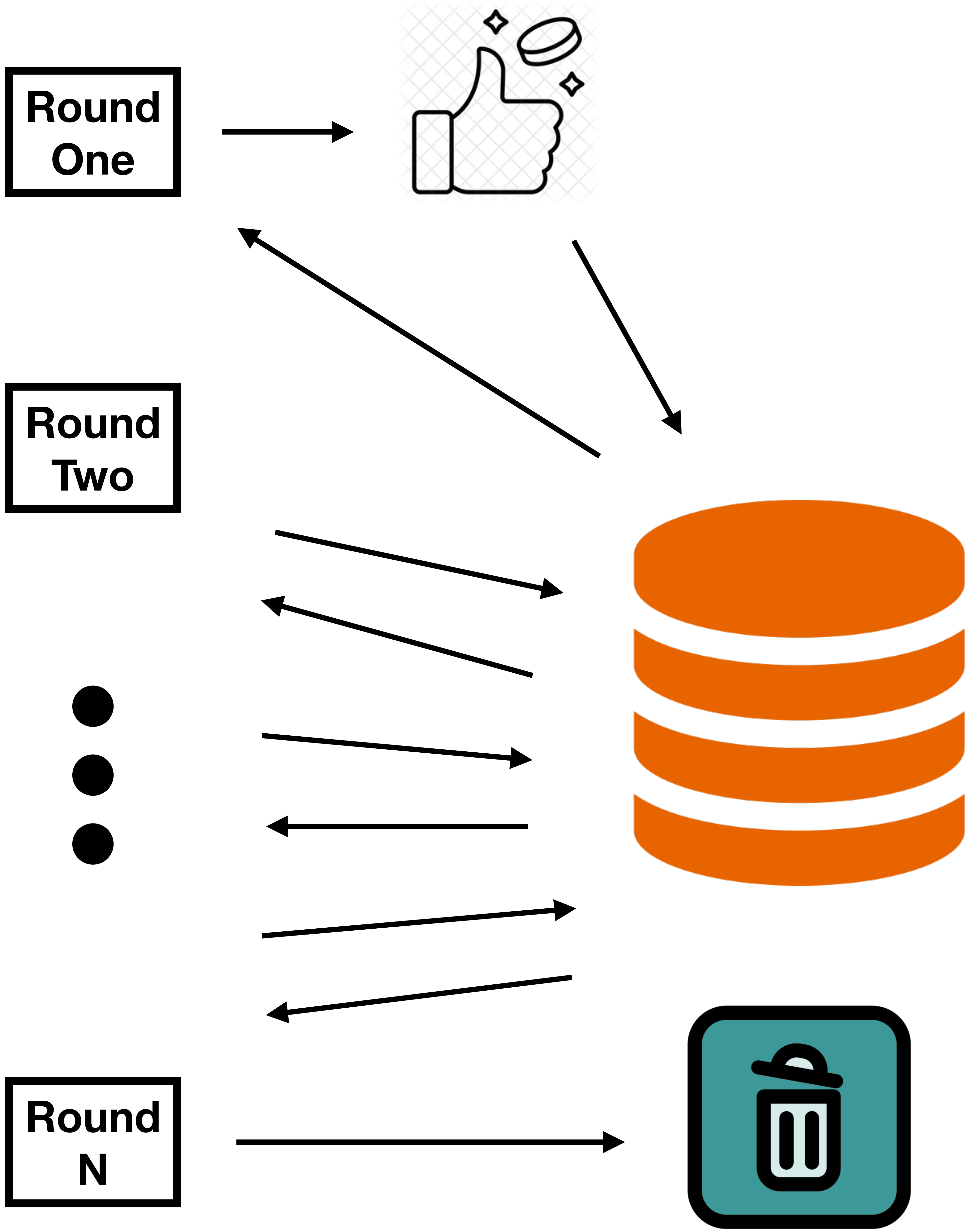
Motivation

- Randomized multi-party schemes require state-keeping between rounds
- Key recovery attacks are possible if state is re-used.



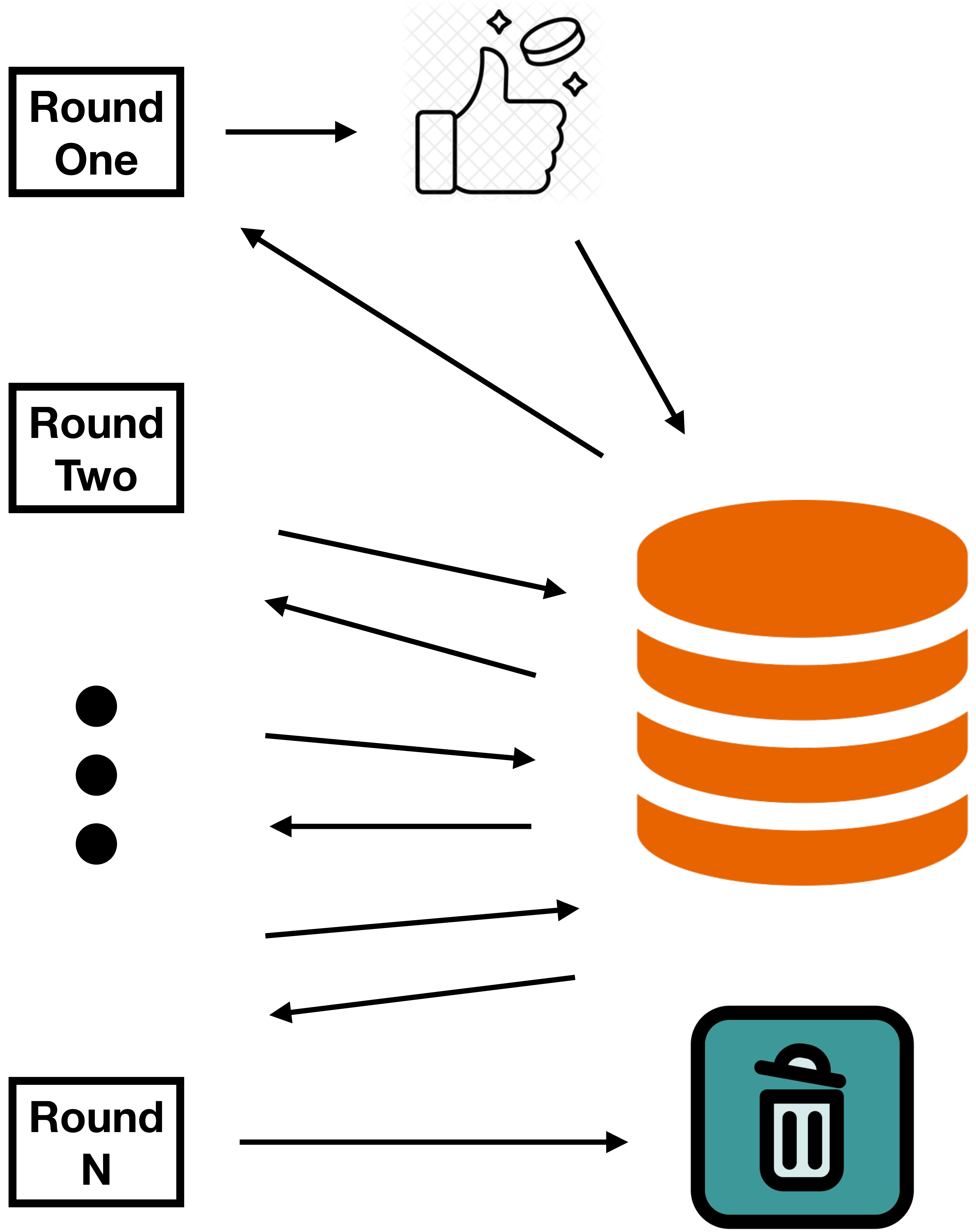
Motivation

- Randomized multi-party schemes require state-keeping between rounds
- Key recovery attacks are possible if state is re-used.
- Requires locks (when concurrent) and careful deletion



Motivation

- Randomized multi-party schemes require state-keeping between rounds
- Key recovery attacks are possible if state is re-used.
- Requires locks (when concurrent) and careful deletion
- Determinism is a means to achieve statelessness



(Single-Party) Schnorr Signatures

To generate a key pair:

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

(Single-Party) Schnorr Signatures

To generate a key pair:

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

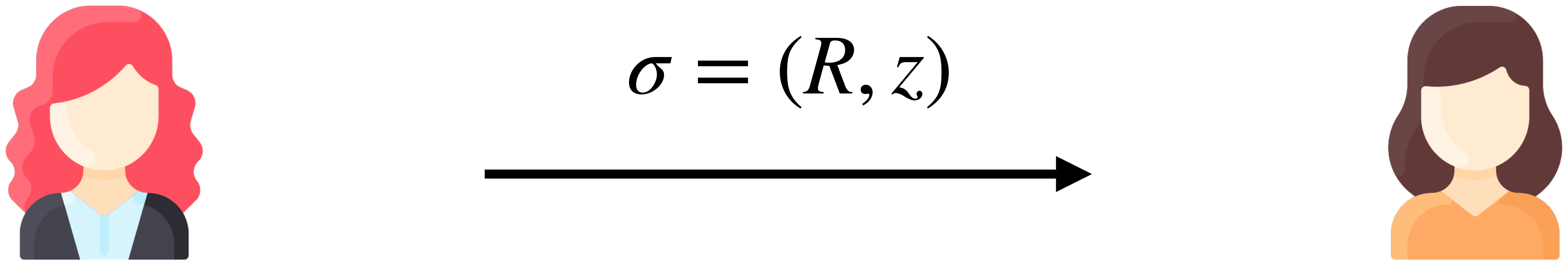
To sign a message m:

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

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

$$z \leftarrow r + csk$$

(Single-Party) Schnorr Signatures



To generate a key pair:

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

To sign a message m:

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

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

$$z \leftarrow r + csk$$

(Single-Party) Schnorr Signatures

$\sigma = (R, z)$

To generate a key pair:

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

To sign a message m:

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

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

$$z \leftarrow r + csk$$

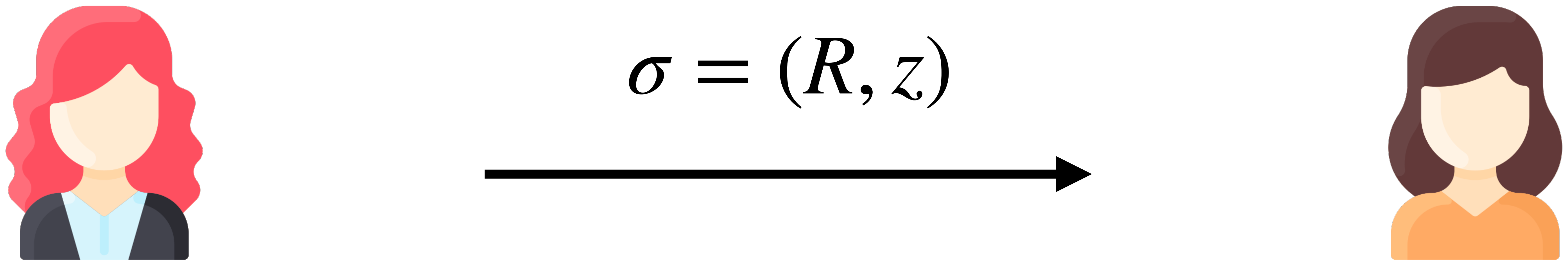
To verify (PK, σ, m):

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

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

output accept/reject

(Single-Party) EdDSA Signatures



To generate a key pair:

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

To sign a message m:

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

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

$$z \leftarrow r + csk$$

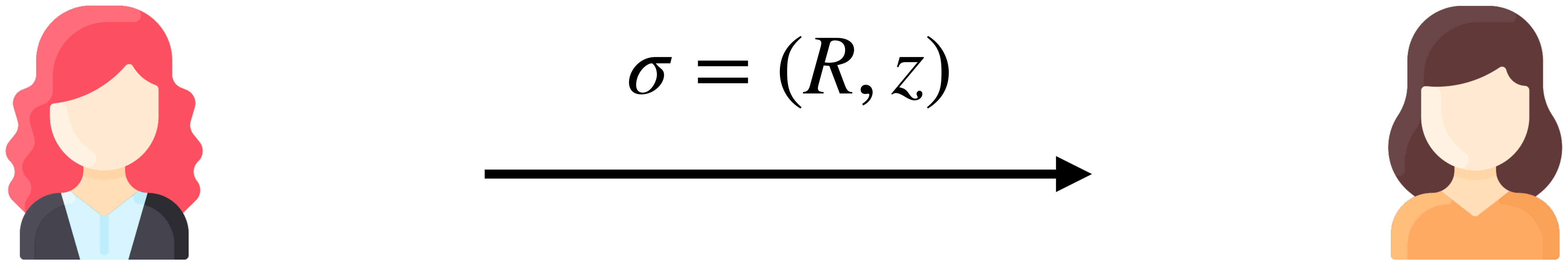
To verify (PK, σ, m):

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

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

output accept/reject

(Single-Party) EdDSA Signatures



To generate a key pair:

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

To sign a message m:

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

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

$$z \leftarrow r + csk$$

To verify (PK, σ, m):

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

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

output accept/reject

(Single-Party) EdDSA Signatures

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

To generate a key pair:

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

To sign a message m:

$$\boxed{r \leftarrow H(m, sk)} \; ; \; R \leftarrow g^{r}$$

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

$$z \leftarrow r + csk$$

Prevents issues from bad randomness.

To verify (PK, σ, m):

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

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

output accept/reject

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

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

Summary: EdDSA-style determinism is **not** publicly verifiable; Adversary can pick its nonce randomly without detection

Towards Multi-Party Deterministic Threshold Schnorr

Towards Multi-Party Deterministic Threshold Schnorr

- Strategy: All parties must prove they generated their nonces honestly.

Towards Multi-Party Deterministic Threshold Schnorr

- Strategy: All parties must prove they generated their nonces honestly.
- Prior approaches:

Towards Multi-Party Deterministic Threshold Schnorr

- Strategy: All parties must prove they generated their nonces honestly.
- Prior approaches:
  - Generic SNARKs: MuSig-DN [GKMN21]

Towards Multi-Party Deterministic Threshold Schnorr

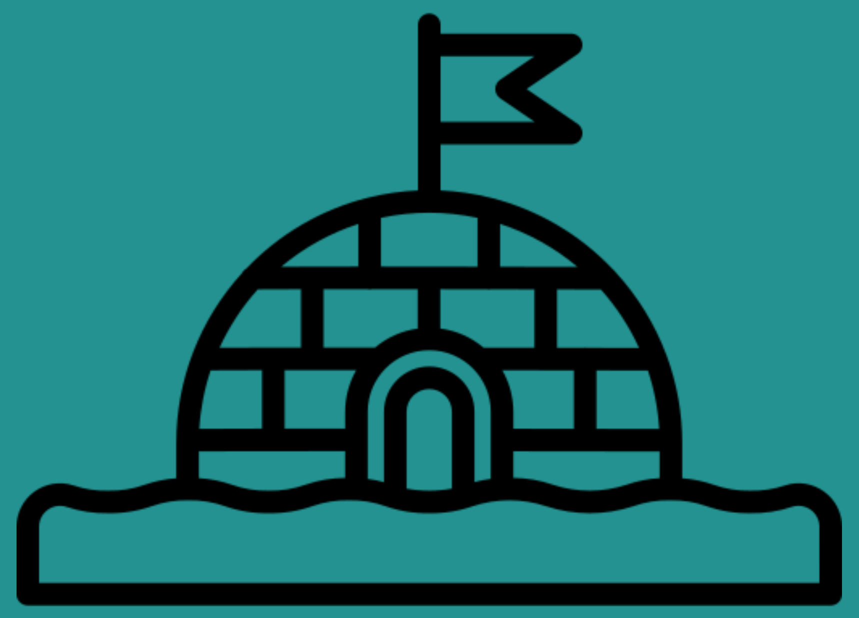
- Strategy: All parties must prove they generated their nonces honestly.
- Prior approaches:
  - Generic SNARKs: MuSig-DN [GKMN21]
  - Generic MPC [NRSW20]

Towards Multi-Party Deterministic Threshold Schnorr

- Strategy: All parties must prove they generated their nonces honestly.
- Prior approaches:
  - Generic SNARKs: MuSig-DN [GKMN21]
  - Generic MPC [NRSW20]

Goal of this work: To design a practical (efficient, simple) deterministic threshold signature.

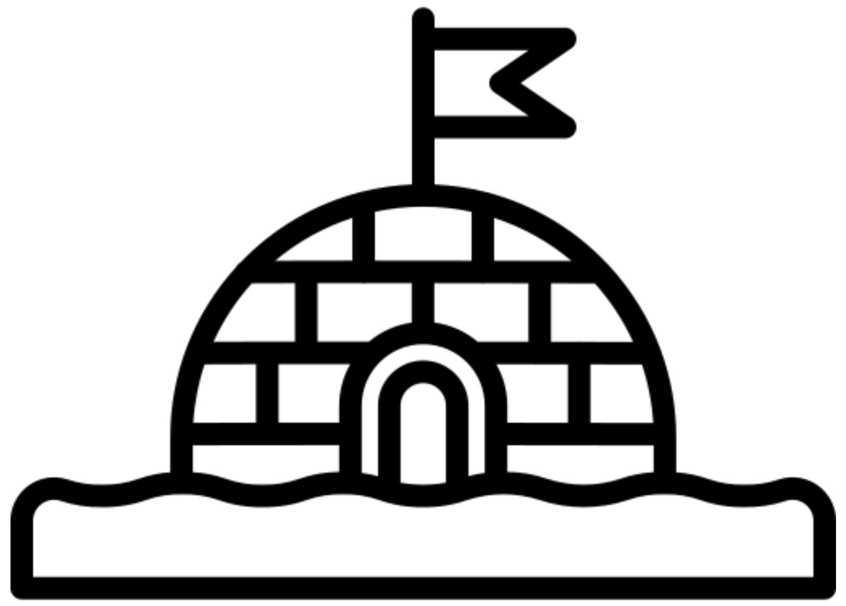
Arctic: A Two-Round Stateless Threshold Schnorr Signature



Stateless Threshold Signatures, with Tradeoffs

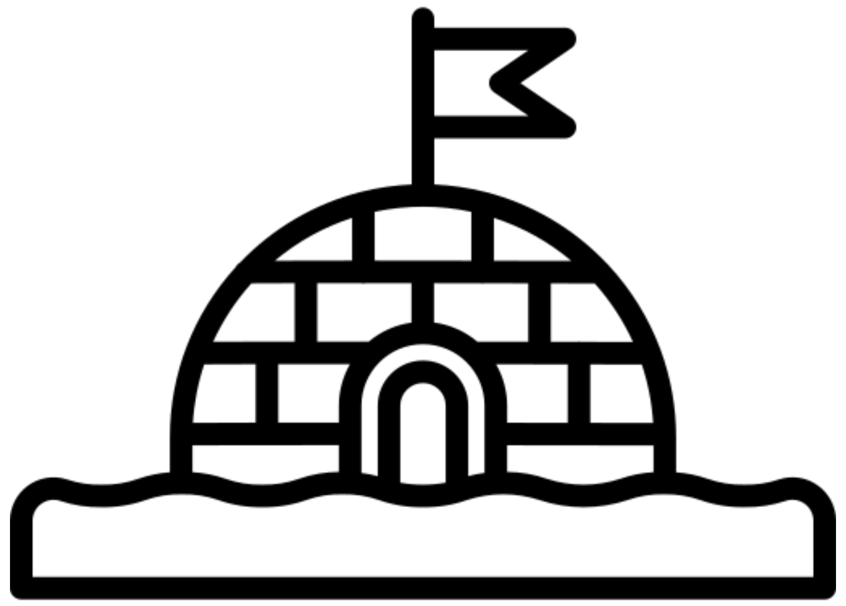
Stateless Threshold Signatures, with Tradeoffs

- We define Arctic, a two-round deterministic threshold Schnorr signature scheme.



Stateless Threshold Signatures, with Tradeoffs

- We define Arctic, a two-round deterministic threshold Schnorr signature scheme.
  - Does not require generic MPC or SNARKS. ✓

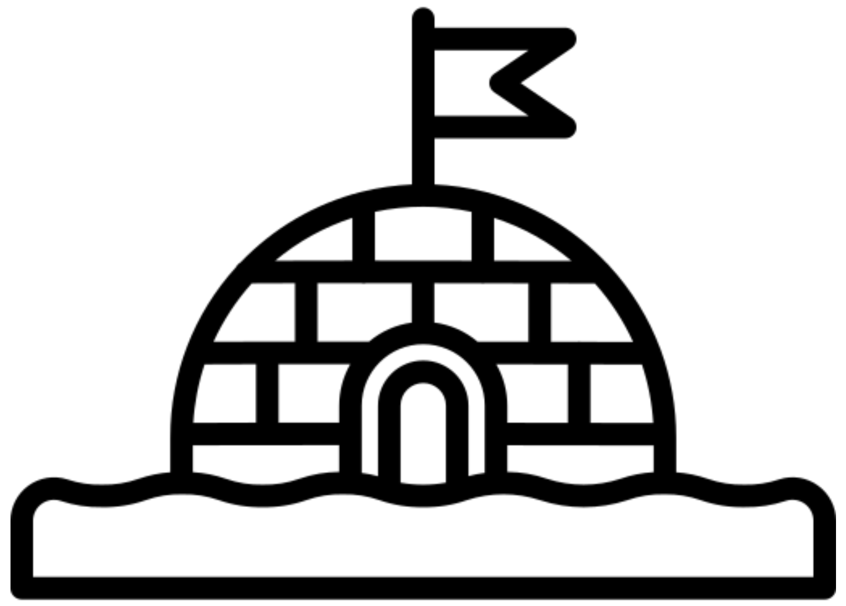


Stateless Threshold Signatures, with Tradeoffs

- We define Arctic, a two-round deterministic threshold Schnorr signature scheme.
  - Does not require generic MPC or SNARKS. ✓
  - Assumption of honest majority (minimum (2t-1) signers). ✗

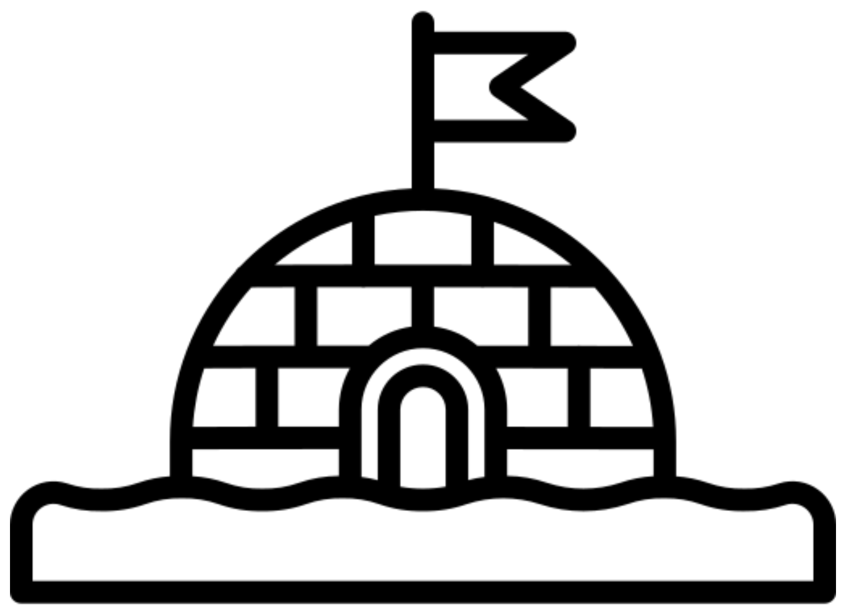
Stateless Threshold Signatures, with Tradeoffs

- We define Arctic, a two-round deterministic threshold Schnorr signature scheme.
  - Does not require generic MPC or SNARKS. ✓
  - Assumption of honest majority (minimum (2t-1) signers). ✗
    - Tolerates t-1 corruptions, assumes t honest signers



Stateless Threshold Signatures, with Tradeoffs

- We define Arctic, a two-round deterministic threshold Schnorr signature scheme.
  - Does not require generic MPC or SNARKS. ✓
  - Assumption of honest majority (minimum (2t-1) signers). ✗
    - Tolerates t-1 corruptions, assumes t honest signers
  - Efficient for moderately-sized groups (i.e., less than 25). ✓



Arctic

Arctic

$\underline{\text{KeyGen}(1^\lambda)}$

1. Derive
   $sk^s \leftarrow \mathbb{Z}_q; PK \leftarrow g^{sk^s}$
2. Shamir secret share sk into
   $(sk_1^s, \ldots, sk_n^s)$
3. Generate VPSS keys
   $(sk_1^v, \ldots, sk_n^v)$
4. Send key shares (sk_i^v, sk_i^s)
   to all parties.

Arctic

KeyGen(1^λ)

1. Derive
   $sk^s \leftarrow \mathbb{Z}_q; PK \leftarrow g^{sk^s}$
2. Shamir secret share sk into
   $(sk_1^s, \ldots, sk_n^s)$
3. Generate VPSS keys
   $(sk_1^v, \ldots, sk_n^v)$
4. Send key shares (sk_i^v, sk_i^s)
   to all parties.

Sign$_1(sk_i^v, m, C)$

$(r_i, R_i) \leftarrow \text{VPSS.Gen}(sk_i^v, m, C)$

Output R_i

Arctic

$\underline{\text{KeyGen}(1^\lambda)}$

1. Derive
   $sk^s \leftarrow \mathbb{Z}_q; PK \leftarrow g^{sk^s}$
2. Shamir secret share sk into
   $(sk_1^s, \ldots, sk_n^s)$
3. Generate VPSS keys
   $(sk_1^v, \ldots, sk_n^v)$
4. Send key shares (sk_i^v, sk_i^s)
   to all parties.

$\underline{\text{Sign}_1(sk_i^v, m, C)}$

$(r_i, R_i) \leftarrow \text{VPSS.Gen}(sk_i^v, m, C)$

Output R_i

$\underline{\text{Sign}_2(sk_i^v, sk_i^s, m, C, \{R_i\}_{i \in C})}$

if $\text{VPSS.Verify}(i, C, \{R_i\}_{i \in C}) \neq 1$

   Output $\perp$

$(r_i, R_i) \leftarrow \text{VPSS.Gen}(sk_i^v, m, C)$

$$R \leftarrow \prod_{i \in C} R_i^{\lambda_i}$$

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

$z_i \leftarrow r_i + (c \cdot sk_i^s)$

Output z_i

Arctic

KeyGen(1^λ)

1. Derive
   $sk^s \leftarrow \mathbb{Z}_q; PK \leftarrow g^{sk^s}$
2. Shamir secret share sk into
   $(sk_1^s, \ldots, sk_n^s)$
3. Generate VPSS keys
   $(sk_1^v, \ldots, sk_n^v)$
4. Send key shares (sk_i^v, sk_i^s)
   to all parties.

Sign$_1(sk_i^v, m, C)$

$(r_i, R_i) \leftarrow$ VPSS . Gen(sk_i^v, m, C)

Output R_i

Sign$_2(sk_i^v, sk_i^s, m, C, \{R_i\}_{i\in C})$

if VPSS . Verify$(i, C, \{R_i\}_{i\in C}) \neq 1$

  Output $\perp$

$(r_i, R_i) \leftarrow$ VPSS . Gen(sk_i^v, m, C)

$R \leftarrow \prod_{i\in C} R_i^{\lambda_i}$

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

$z_i \leftarrow r_i + (c \cdot sk_i^s)$

Output z_i

Combine$(R, \{z_i\}_{i\in C})$

$z \leftarrow \sum_{i\in C} z_i \cdot \lambda_i$

$\sigma = (R, z)$

Output (m, σ)

Arctic

KeyGen(1^λ)

1. Derive
   $sk^s \leftarrow \mathbb{Z}_q; PK \leftarrow g^{sk^s}$
2. Shamir secret share sk into
   $(sk^s_1, \ldots, sk^s_n)$
3. Generate VPSS keys
   $(sk^v_1, \ldots, sk^v_n)$
4. Send key shares (sk^v_i, sk^s_i)
   to all parties.

Sign$_1(sk^v_i, m, C)$

$(r_i, R_i) \leftarrow$ VPSS . Gen(sk^v_i, m, C)

Output R_i

Sign$_2(sk^v_i, sk^s_i, m, C, \{R_i\}_{i \in C})$

if VPSS . Verify$(i, C, \{R_i\}_{i \in C}) \neq 1$

Output $\perp$

$(r_i, R_i) \leftarrow$ VPSS . Gen(sk^v_i, m, C)

$$R \leftarrow \prod_{i \in C} R_i^{\lambda_i}$$

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

$z_i \leftarrow r_i + (c \cdot sk^s_i)$

Output z_i

Combine$(R, \{z_i\}_{i \in C})$

$$z \leftarrow \sum_{i \in C} z_i \cdot \lambda_i$$

$\sigma = (R, z)$

Output (m, σ)

Verify(PK, m, σ)

Identical to single-party Schnorr.

Arctic

$\mathsf{KeyGen}(1^\lambda)$

1. Derive $sk^s \leftarrow \mathbb{Z}_q; PK \leftarrow g^{sk^s}$
2. Shamir secret share sk into $(sk_1^s, \ldots, sk_n^s)$
3. Generate VPSS keys $(sk_1^v, \ldots, sk_n^v)$
4. Send key shares (sk_i^v, sk_i^s) to all parties.

$\mathsf{Sign}_1(sk_i^v, m, C)$

$(r_i, R_i) \leftarrow \mathsf{VPSS.Gen}(sk_i^v, m, C)$

Output R_i

$\mathsf{Sign}_2(sk_i^v, sk_i^s, m, C, \{R_i\}_{i \in C})$

if $\mathsf{VPSS.Verify}(i, C, \{R_i\}_{i \in C}) \neq 1$

Output $\perp$

$(r_i, R_i) \leftarrow \mathsf{VPSS.Gen}(sk_i^v, m, C)$

$R \leftarrow \prod_{i \in C} R_i^{\lambda_i}$

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

$z_i \leftarrow r_i + (c \cdot sk_i^s)$

Output z_i

$\mathsf{Combine}(R, \{z_i\}_{i \in C})$

$z \leftarrow \sum_{i \in C} z_i \cdot \lambda_i$

$\sigma = (R, z)$

Output (m, σ)

$\mathsf{Verify}(PK, m, \sigma)$

Identical to single-party Schnorr.

Correctness: $r = \sum_{i \in C} r_i \lambda_i$ and $sk^s = \sum_{i \in C} sk_i^s \lambda_i$

Verifiable Pseudorandom Secret Sharing

Verifiable Pseudorandom Secret Sharing

- Akin to a secret-shared PRF.

Verifiable Pseudorandom Secret Sharing

- Akin to a secret-shared PRF.
- Builds on pseudorandom secret sharing scheme by Cramer et al.[CDI05], but with an additional Verify algorithm.

Verifiable Pseudorandom Secret Sharing

- Akin to a secret-shared PRF.
- Builds on pseudorandom secret sharing scheme by Cramer et al.[CDI05], but with an additional Verify algorithm.
- Verification ensures each party followed the protocol honestly.

Replicated Secret Sharing: Example

$a_1 = (2,3,4)$

$a_2 = (1,3,4)$

$a_3 = (1,2,4)$

$a_4 = (1,2,3)$

corruption threshold t=2
minimum signers=3
total signers n=4

Replicated Secret Sharing: Example

ϕ_1	ϕ_2	ϕ_3	ϕ_4
$a_1 = (2,3,4)$	$a_2 = (1,3,4)$	$a_3 = (1,2,4)$	$a_4 = (1,2,3)$

Where $sk^v = \phi_1 + \phi_2 + \phi_3 + \phi_4$

corruption threshold t=2
minimum signers=3
total signers n=4

Replicated Secret Sharing: Example

ϕ_1	ϕ_2	ϕ_3	ϕ_4
$a_1 = (2,3,4)$	$a_2 = (1,3,4)$	$a_3 = (1,2,4)$	$a_4 = (1,2,3)$

Where $sk^v = \phi_1 + \phi_2 + \phi_3 + \phi_4$

Set $sk^v_j \leftarrow \{\phi_i\}$ for each $i : j \in a_i$

corruption threshold t=2
minimum signers=3
total signers n=4

Replicated Secret Sharing: Example

ϕ_1	ϕ_2	ϕ_3	ϕ_4
$a_1 = (2,3,4)$	$a_2 = (1,3,4)$	$a_3 = (1,2,4)$	$a_4 = (1,2,3)$

Where $sk^v = \phi_1 + \phi_2 + \phi_3 + \phi_4$

Set $sk^v_j \leftarrow \{\phi_i\}$ for each $i : j \in a_i$

Intuition: sk^v is information-theoretically hidden;
each (t-1) corrupt parties lack exactly one ϕ_i.

corruption threshold t=2
minimum signers=3
total signers n=4

Verifiable Pseudorandom Secret Sharing in Arctic

Verifiable Pseudorandom Secret Sharing in Arctic

To derive Arctic nonces:

$$r_k \leftarrow \sum_{i=1}^{\binom{n-1}{t-1}} H(\phi_i, m) \cdot L_{a_i}(k), \text{ for each } \phi_i \in sk_i^v$$

Verifiable Pseudorandom Secret Sharing in Arctic

To derive Arctic nonces:

$$r_k \leftarrow \sum_{i=1}^{\binom{n-1}{t-1}} H(\phi_i, m) \cdot L_{a_i}(k), \text{ for each } \phi_i \in sk_i^v$$

To derive joint Arctic nonce:

$$r = \sum_{j \in C} r_i \cdot \lambda_i = f'(0) \text{ for } C \subset [n]$$

Verifiable Pseudorandom Secret Sharing in Arctic

To derive Arctic nonces:

$$r_k \leftarrow \sum_{i=1}^{\binom{n-1}{t-1}} H(\phi_i, m) \cdot L_{a_i}(k), \text{ for each } \phi_i \in sk_i^v$$

To derive joint Arctic nonce:

$$r = \boxed{\sum_{j \in C} r_i \cdot \lambda_i = f'(0)} \text{ for } C \subset [n]$$

Verifiable Pseudorandom Secret Sharing in Arctic

To derive Arctic nonces:

$$r_k \leftarrow \sum_{i=1}^{\binom{n-1}{t-1}} H(\phi_i, m) \cdot L_{a_i}(k), \text{ for each } \phi_i \in sk_i^v$$

To derive joint Arctic nonce:

$$r = \sum_{j \in C} r_i \cdot \lambda_i = f'(0) \text{ for } C \subset [n]$$

Interpolate to
the constant term of an
unknown degree t-1
polynomial f'.

Security of Arctic

Security of Arctic

- Unforgeable, assuming:

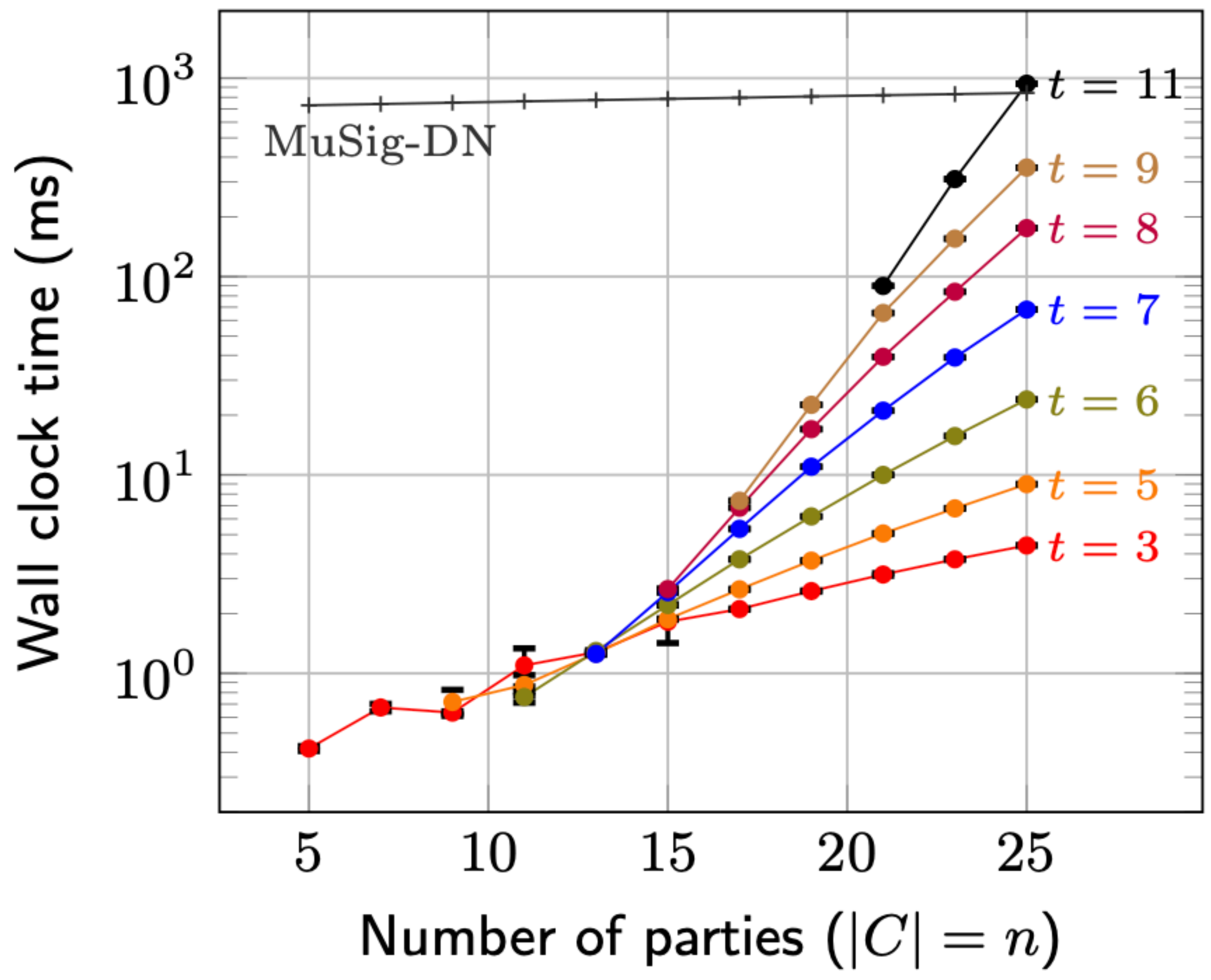
Security of Arctic

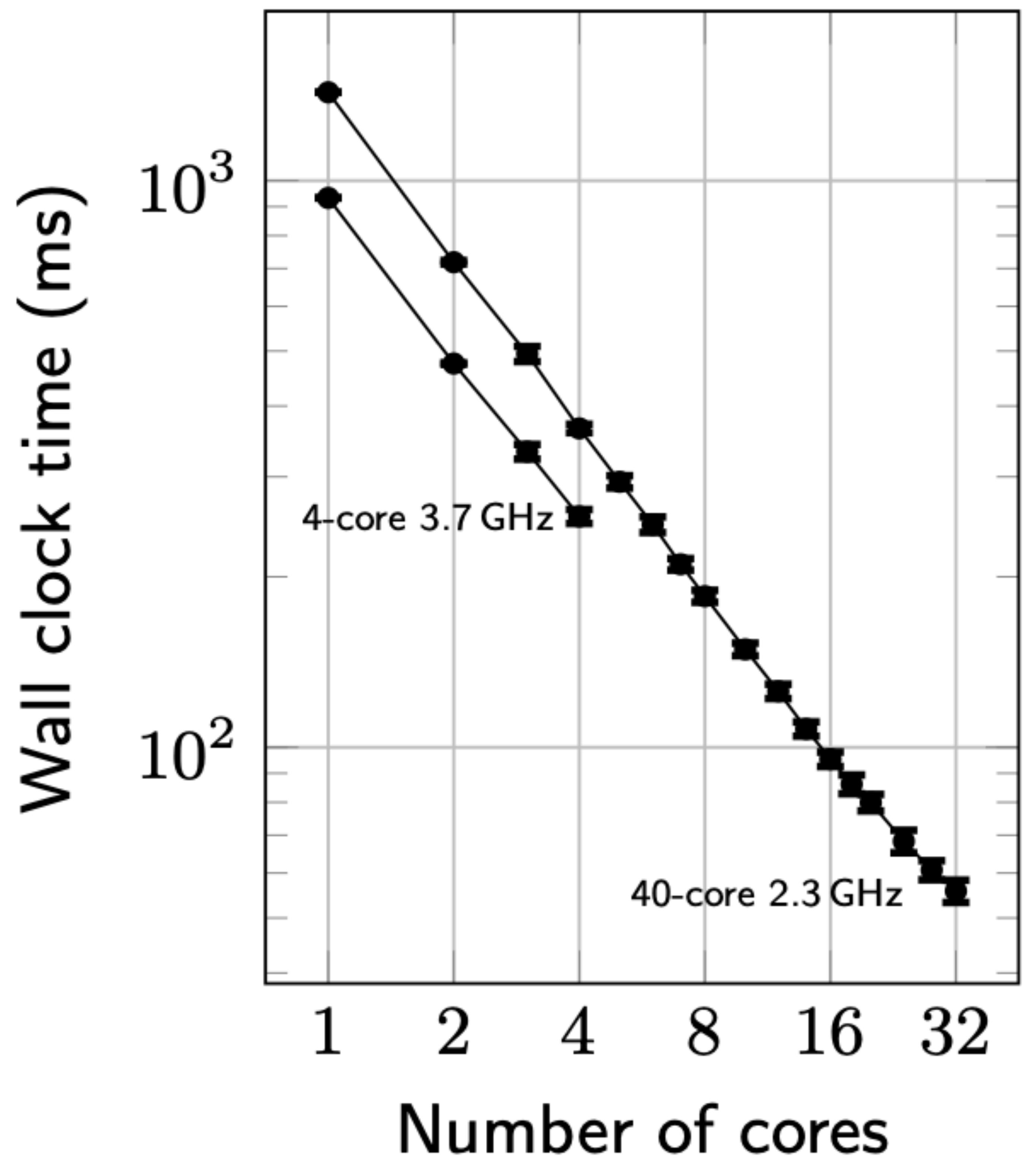
- Unforgeable, assuming:
  - Discrete Logarithm + Random Oracle Model

Security of Arctic

- Unforgeable, assuming:
  - Discrete Logarithm + Random Oracle Model
  - Honest Majority

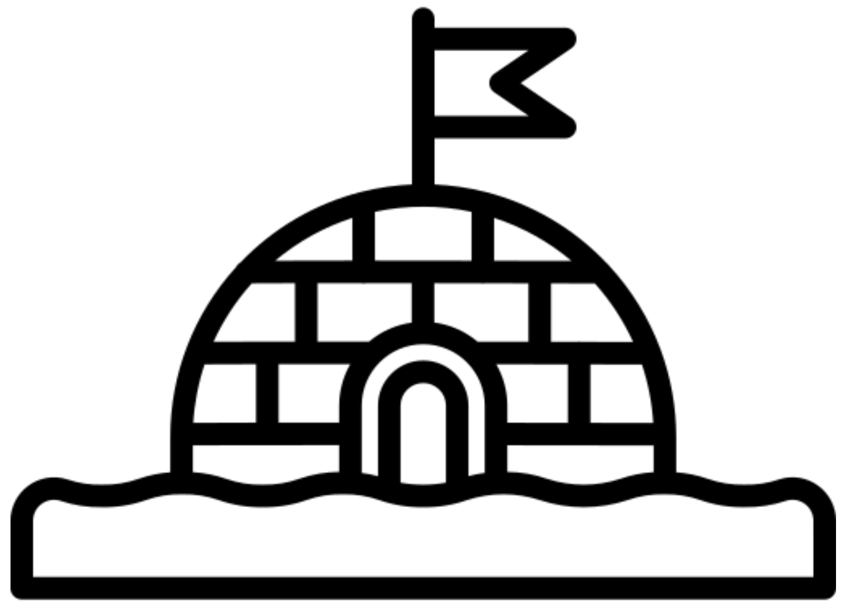
Performance of Arctic





MuSig-DN uses Bulletproofs to prove
a party generated their nonce honestly

Takeaways

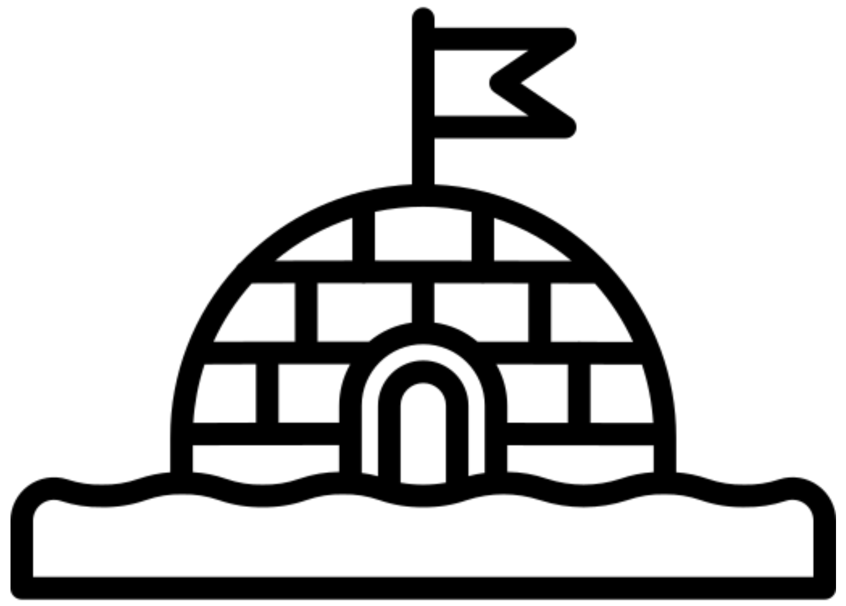


Takeaways

- Statelessness is a desirable property for multi-party schemes

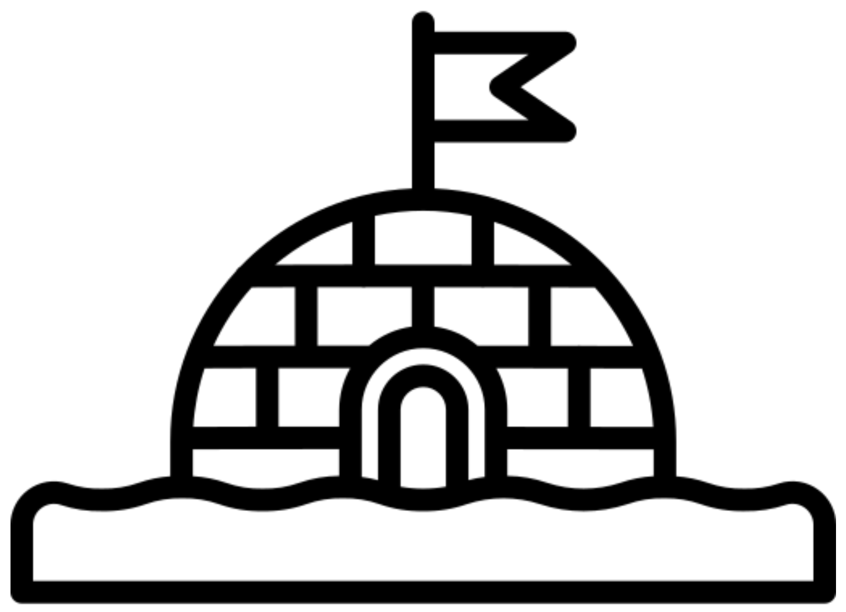
Takeaways

- Statelessness is a desirable property for multi-party schemes
- Arctic is an efficient stateless threshold Schnorr signature scheme



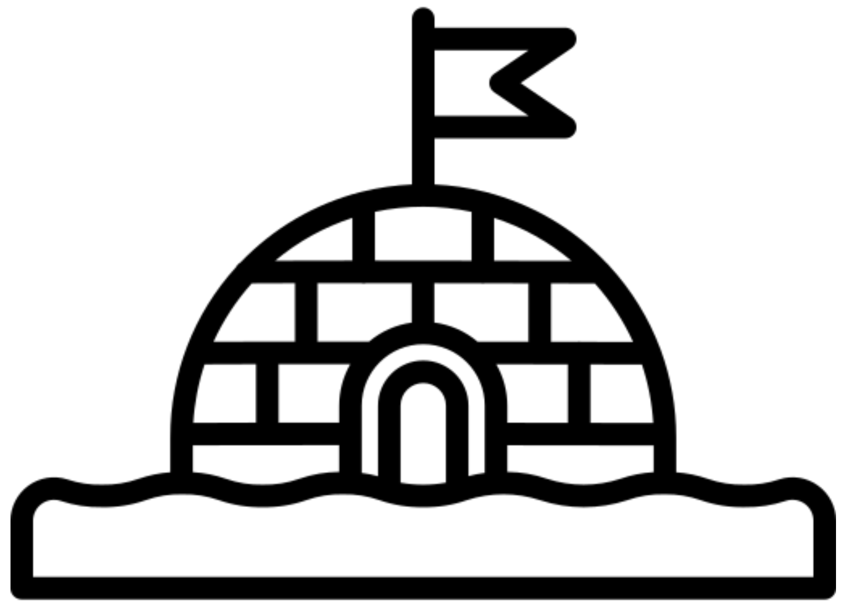
Takeaways

- Statelessness is a desirable property for multi-party schemes
- Arctic is an efficient stateless threshold Schnorr signature scheme
- Builds on verifiable pseudorandom secret sharing



Takeaways

- Statelessness is a desirable property for multi-party schemes
- Arctic is an efficient stateless threshold Schnorr signature scheme
- Builds on verifiable pseudorandom secret sharing
- Requires honest majority, efficient for small signing sets (less than 25)



VPSS Verification

VPSS Verification

- Verifying parties honestly followed the protocol can be done *collectively*.

VPSS Verification

- Verifying parties honestly followed the protocol can be done *collectively*.
- Example where the coalition of signers $|C| = n$

VPSS Verification

- Verifying parties honestly followed the protocol can be done *collectively*.
- Example where the coalition of signers $|C| = n$

Step 1: Let $(r_1, \ldots, r_n)$ be the outputs from each party.

VPSS Verification

- Verifying parties honestly followed the protocol can be done *collectively*.
- Example where the coalition of signers $|C| = n$

Step 1: Let $(r_1, \ldots, r_n)$ be the outputs from each party.

Step 2: Define $b_i = \sum_{j=1}^{n-1} r_j \cdot L_j[i]$

VPSS Verification

- Verifying parties honestly followed the protocol can be done *collectively*.
- Example where the coalition of signers $|C| = n$

Step 1: Let $(r_1, \ldots, r_n)$ be the outputs from each party.

Step 2: Define $b_i = \sum_{j=1}^{n-1} r_j \cdot L_j[i]$

Step 3: Define $f(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$

VPSS Verification

- Verifying parties honestly followed the protocol can be done *collectively*.
- Example where the coalition of signers $|C| = n$

Step 1: Let $(r_1, \ldots, r_n)$ be the outputs from each party.

Step 2: Define $b_i = \sum_{j=1}^{n-1} r_j \cdot L_j[i]$

Step 3: Define $f(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$

Step 4: Verify $f(x)$ is of degree t-1 by checking the top-most coefficients $b_t = 0, \ldots, b_{n-1} = 0$

VPSS Verification

- Verifying parties honestly followed the protocol can be done *collectively*.
- Example where the coalition of signers $|C| = n$

Step 1: Let $(r_1, \ldots, r_n)$ be the outputs from each party.

Step 2: Define $b_i = \sum_{j=1}^{n-1} r_j \cdot L_j[i]$

Step 3: Define $f(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$

Step 4: Verify $f(x)$ is of degree t-1 by checking the top-most coefficients $b_t = 0, \ldots, b_{n-1} = 0$

Outputs from t honest parties completely define a polynomial of degree t-1.

VPSS Verification

- Verifying parties honestly followed the protocol can be done *collectively*.
- Example where the coalition of signers $|C| = n$

Step 1: Let $(r_1, \ldots, r_n)$ be the outputs from each party.

Step 2: Define $b_i = \sum_{j=1}^{n-1} r_j \cdot L_j[i]$

Step 3: Define $f(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$

Outputs from t honest parties completely define a polynomial of degree t-1.

Step 4: Verify $f(x)$ is of degree t-1 by checking the top-most coefficients $b_t = 0, \ldots, b_{n-1} = 0$

Publicly verifiable when performed over commitments $(R_i)_{i \in C}$