

Threshold Signing for Standard Hash-Based Signatures is Expensive

Yashvanth Kondi, [Naman Kumar](#), Hernán Vanegas — MPTS '26



SILENCE
LABORATORIES



Motivation & Table of Contents

- Study hash-based signatures from a **theoretical** perspective
- Understand the technical feasibility of **thresholdizing** hash-based signatures

Motivation & Table of Contents

- Study hash-based signatures from a *theoretical* perspective
- Understand the technical feasibility of *thresholdizing* hash-based signatures
- Contents:
 - Introduction, Background
 - Our new ‘extractability’ definition
 - New impossibility results, proofs
 - Conclusions

Review: Hash-Based Signatures

What working definition do we use?

- Idea: Lamport's one-time signature

$$\text{sk} := (s_0, s_1) \leftarrow \{0,1\}^\lambda$$

$$\text{vk} := (H(s_0), H(s_1))$$

$$\text{Sign}(b) = s_b$$

Review: Hash-Based Signatures

What working definition do we use?

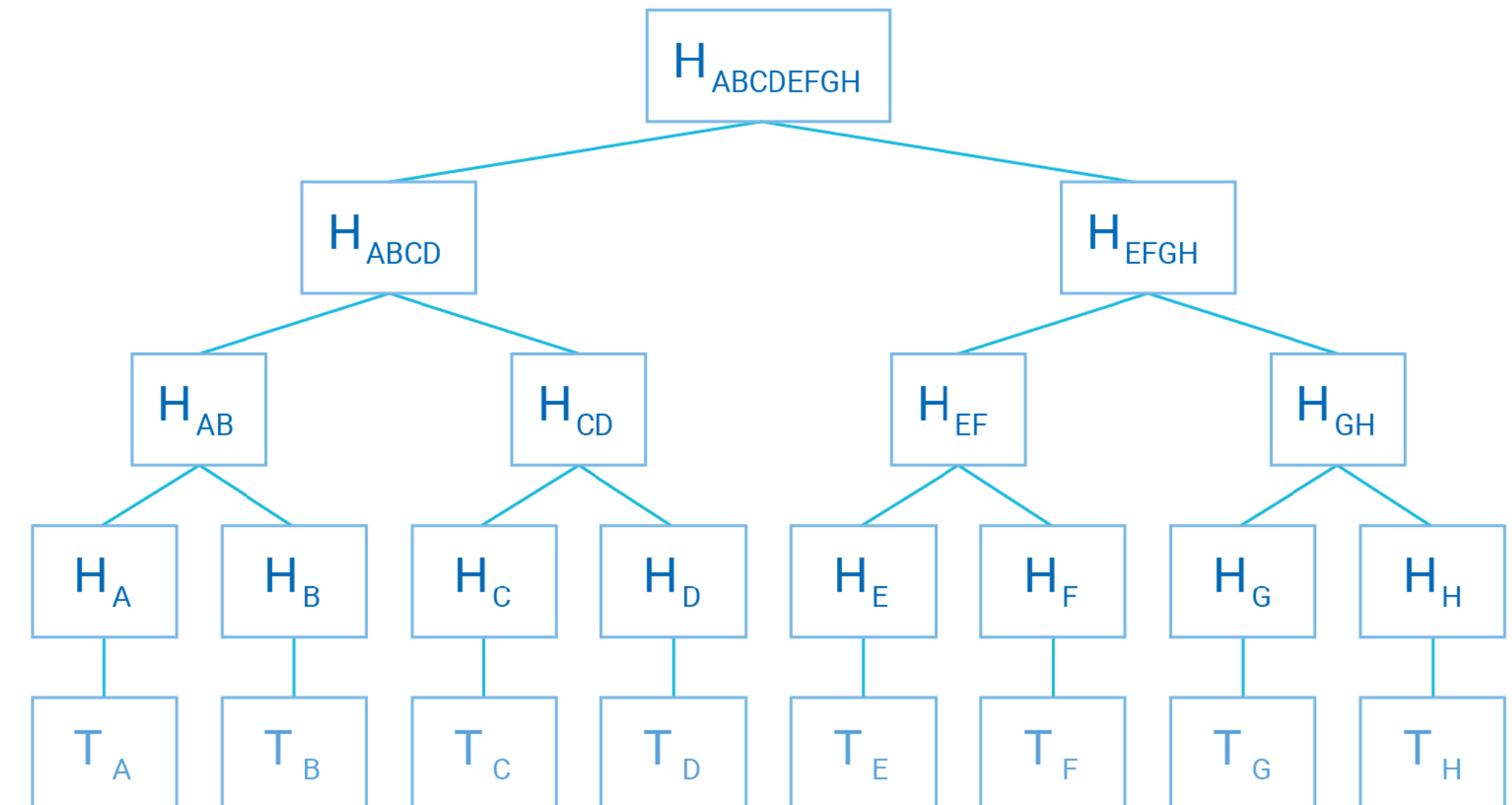
- Idea: Lamport's one-time signature

$$\text{sk} := (s_0, s_1) \leftarrow \{0,1\}^\lambda$$

$$\text{vk} := (H(s_0), H(s_1))$$

$$\text{Sign}(b) = s_b$$

- Hash-based signatures follow the same ideas
- Extend using Merkle-style constructions



Review: Hash-Based Signatures

What working definition do we use?

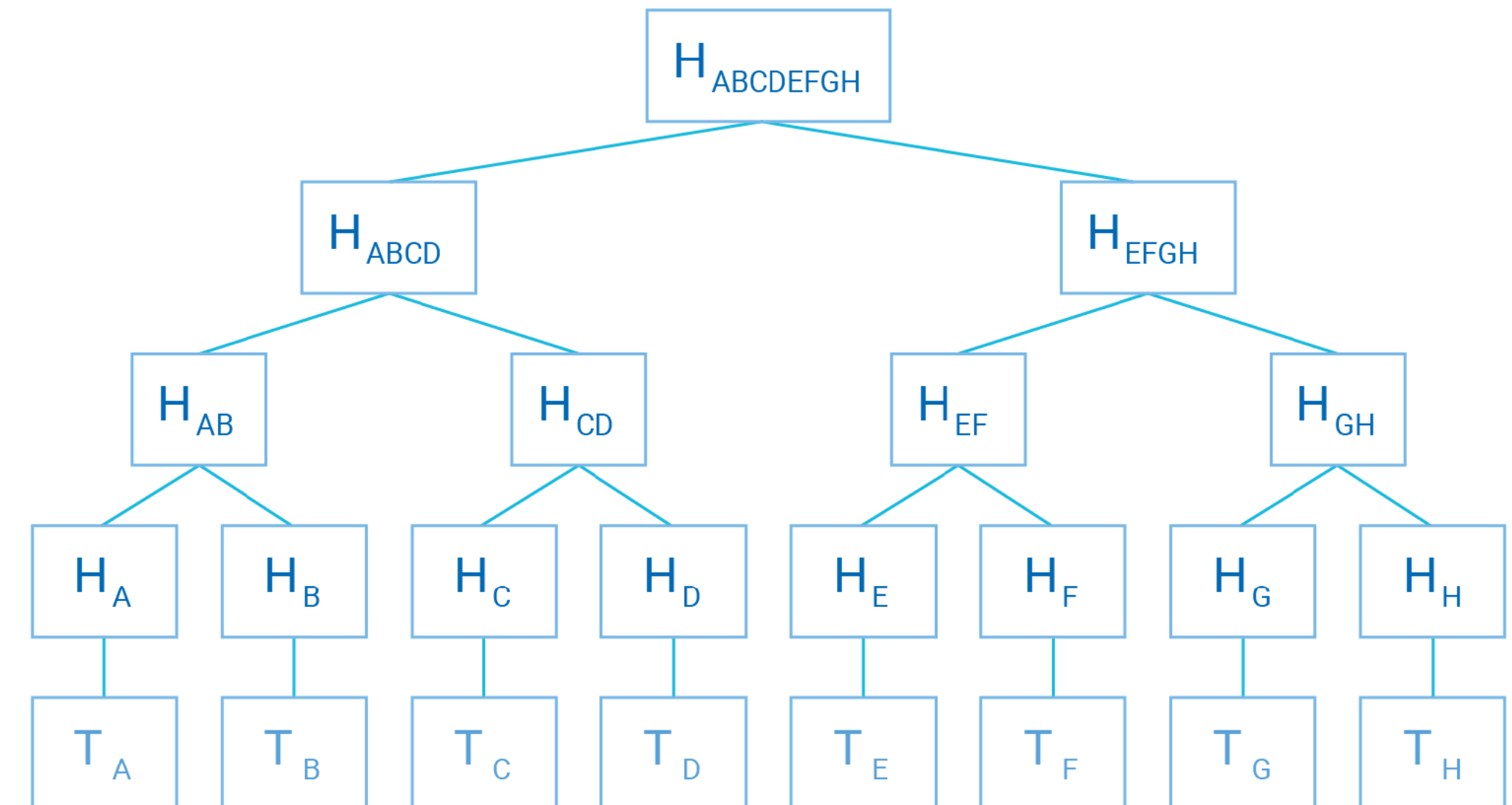
- Idea: Lamport's one-time signature

$$\text{sk} := (s_0, s_1) \leftarrow \{0,1\}^\lambda$$

$$\text{vk} := (H(s_0), H(s_1))$$

$$\text{Sign}(b) = s_b$$

- Hash-based signatures follow the same ideas
- Extend using Merkle-style constructions
- Only one cryptographic object: hash function (SHA)



Examples: XMSS, SPHINCS/SPHINCS+

SLH-DSA: PQ Standardization

Review: Hash-Based Signatures

What working definition do we use?

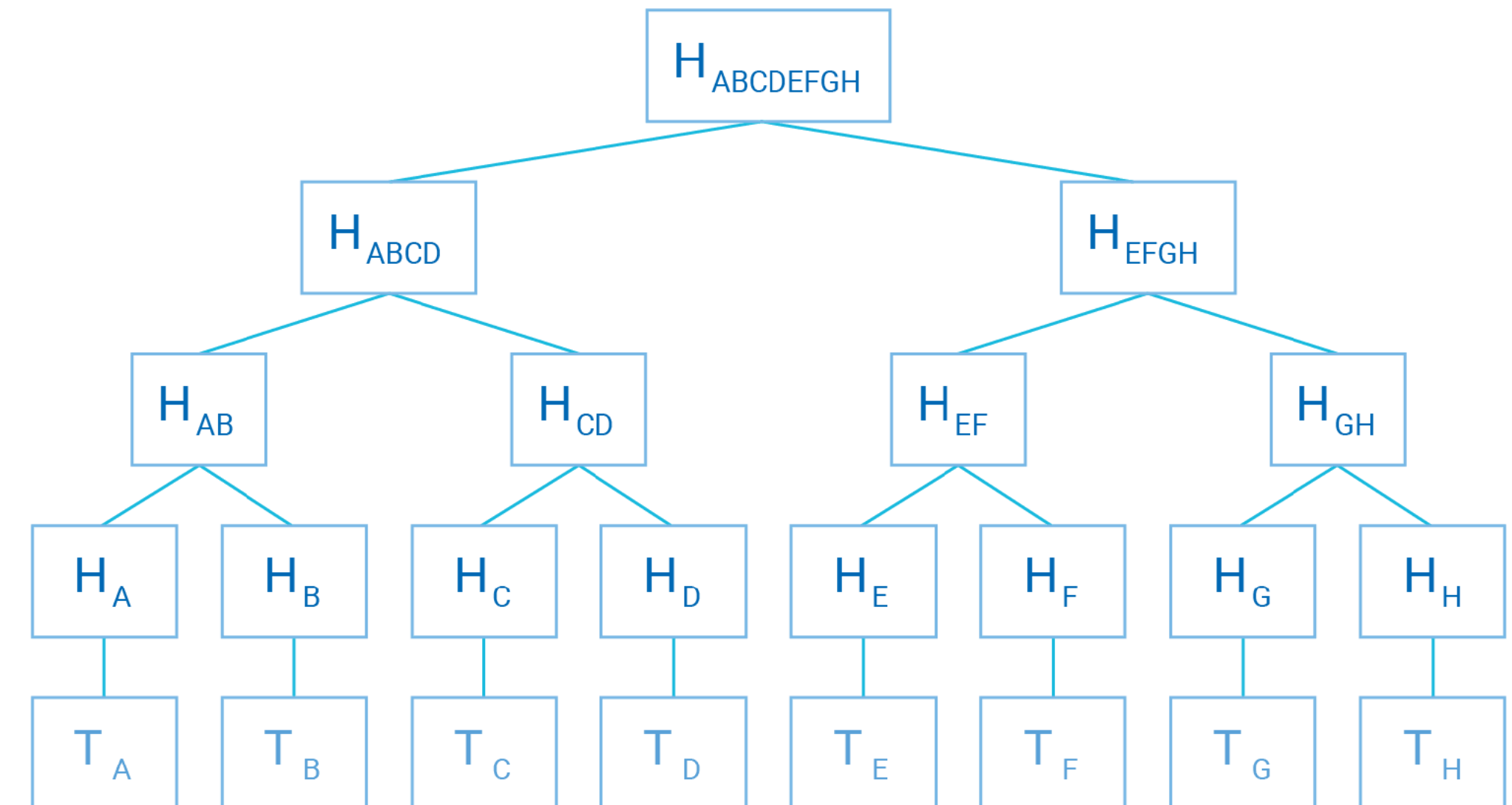
- Idea: Lamport's one-time signature

$$\text{sk} := (s_0, s_1) \leftarrow \{0,1\}^\lambda$$

$$\text{vk} := (H(s_0), H(s_1))$$

$$\text{Sign}(b) = s_b$$

- Hash-based signatures follow the same ideas
- Extend using Merkle-style constructions
- Only one cryptographic object: hash function (SHA)
- **Crucially:** Security of signature (unforgeability) reduces only to **security of underlying hash function!**



Natural Idea: Random Oracle Model

Review: Hash-Based Signatures

What working definition do we use?

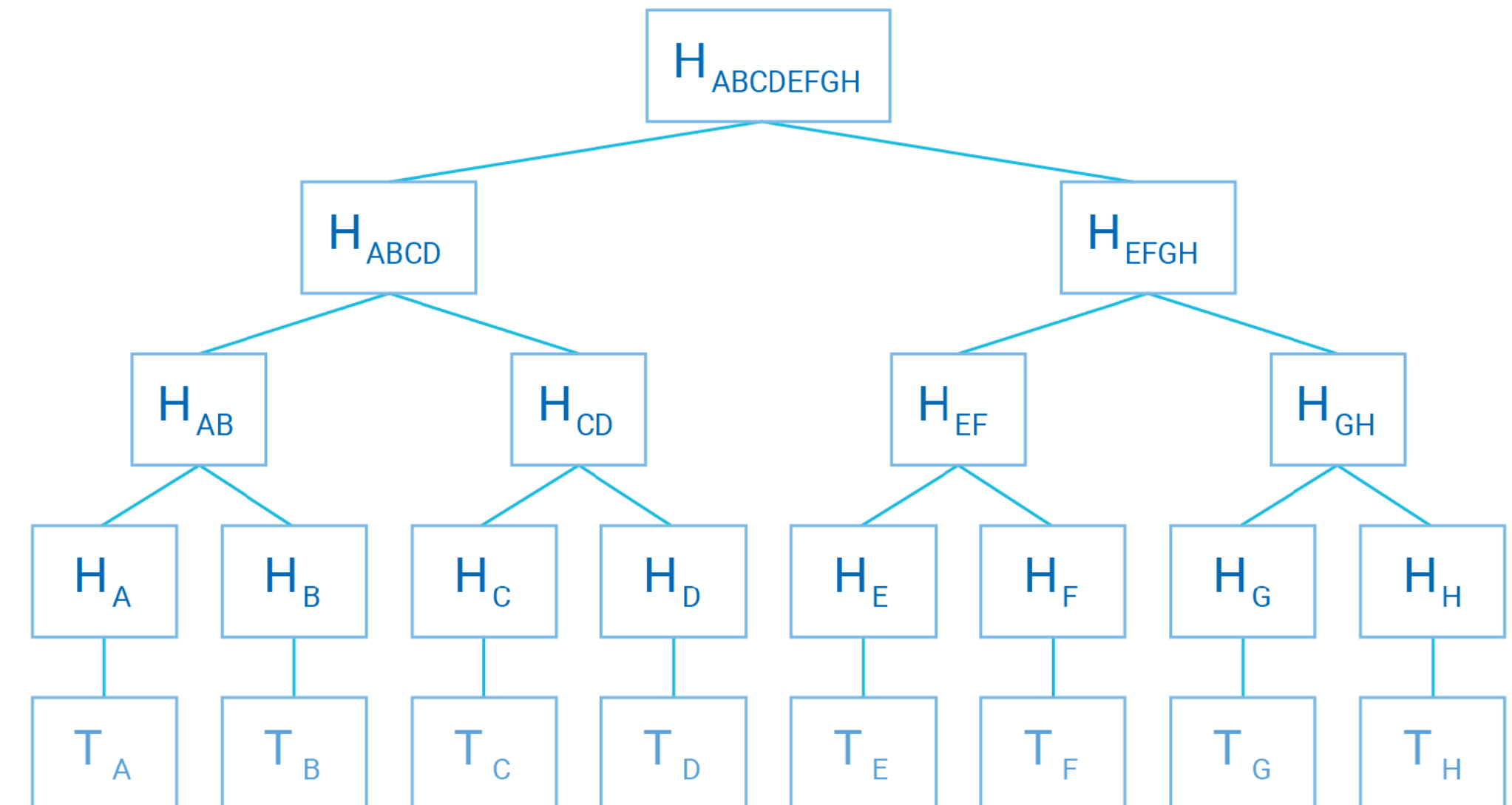
- Idea: Lamport's one-time signature

$$\text{sk} := (s_0, s_1) \leftarrow \{0,1\}^\lambda$$

$$\text{vk} := (H(s_0), H(s_1))$$

$$\text{Sign}(b) = s_b$$

- Hash-based signatures follow the same ideas
- Extend using Merkle-style constructions
- Only one cryptographic object: hash function (SHA)
- Crucially:** Security of signature (unforgeability) reduces only to *security of underlying hash function!*



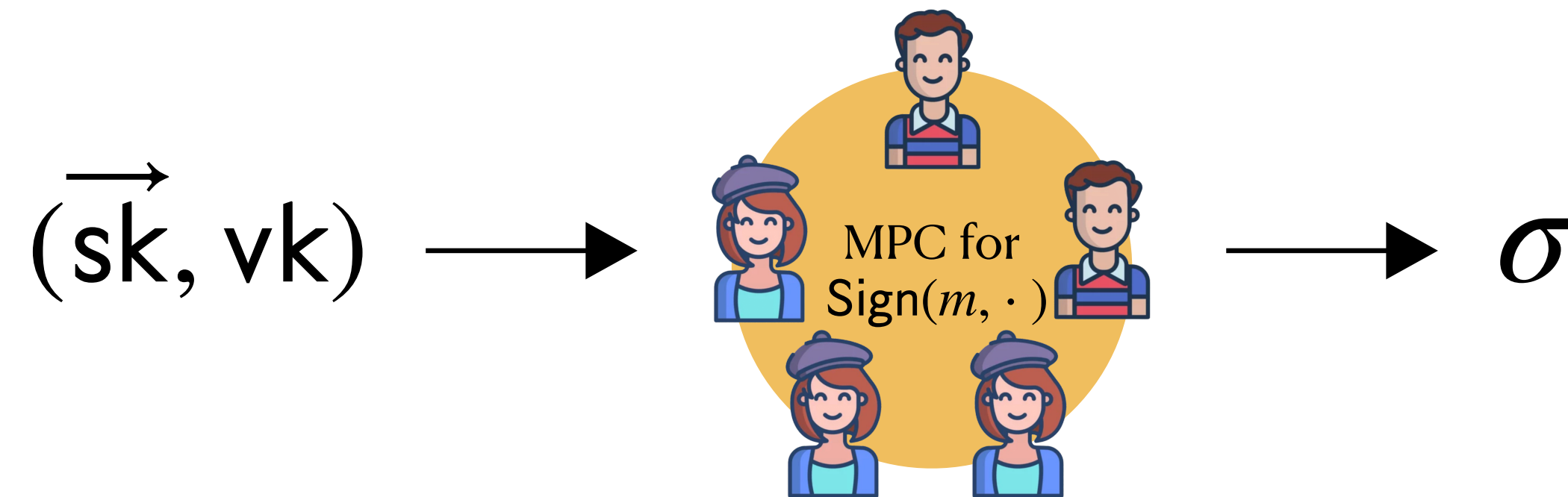
Natural Idea: Random Oracle Model

Definition: Security reduction requires no hardness outside the Random Oracle!

Threshold Signing

What kind of algorithms do we consider?

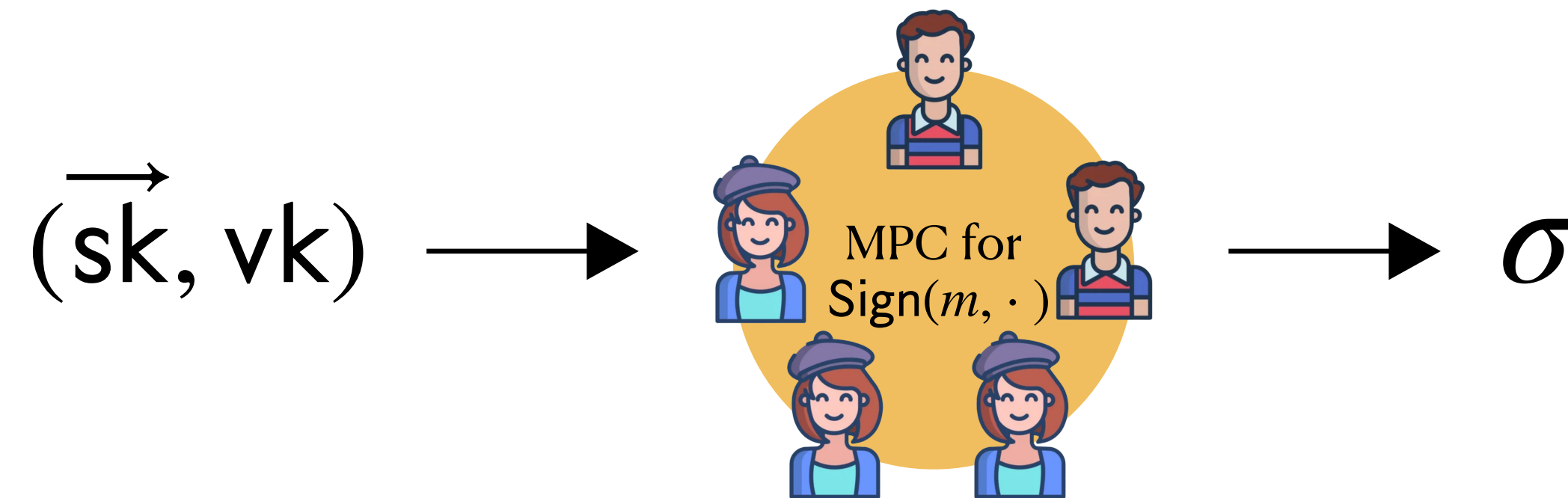
- In general: thresholdize signing using generic MPC



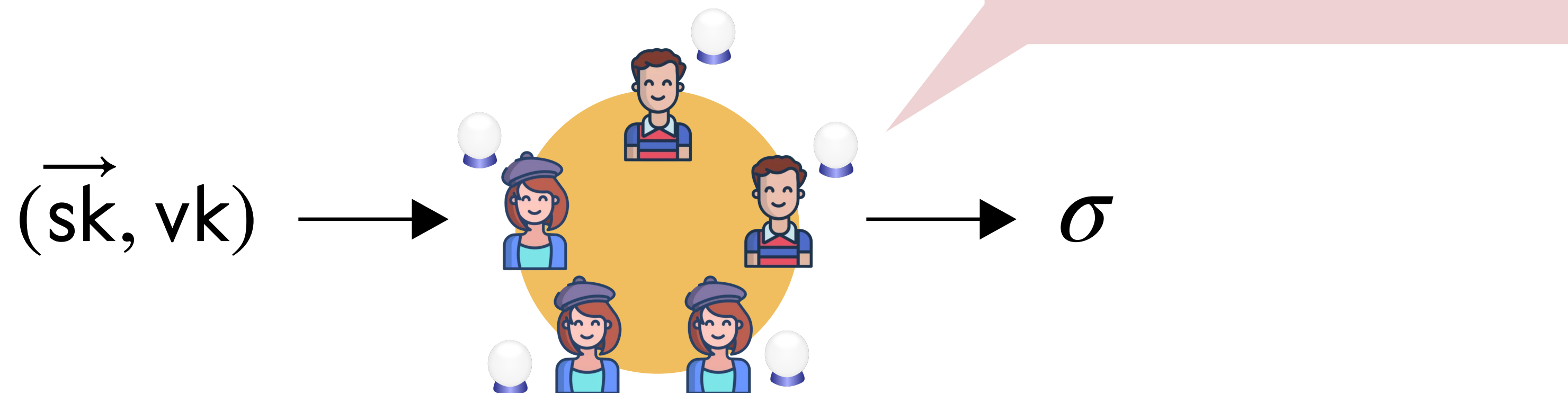
Threshold Signing

What kind of algorithms do we consider?

- In general: thresholdize signing using generic MPC



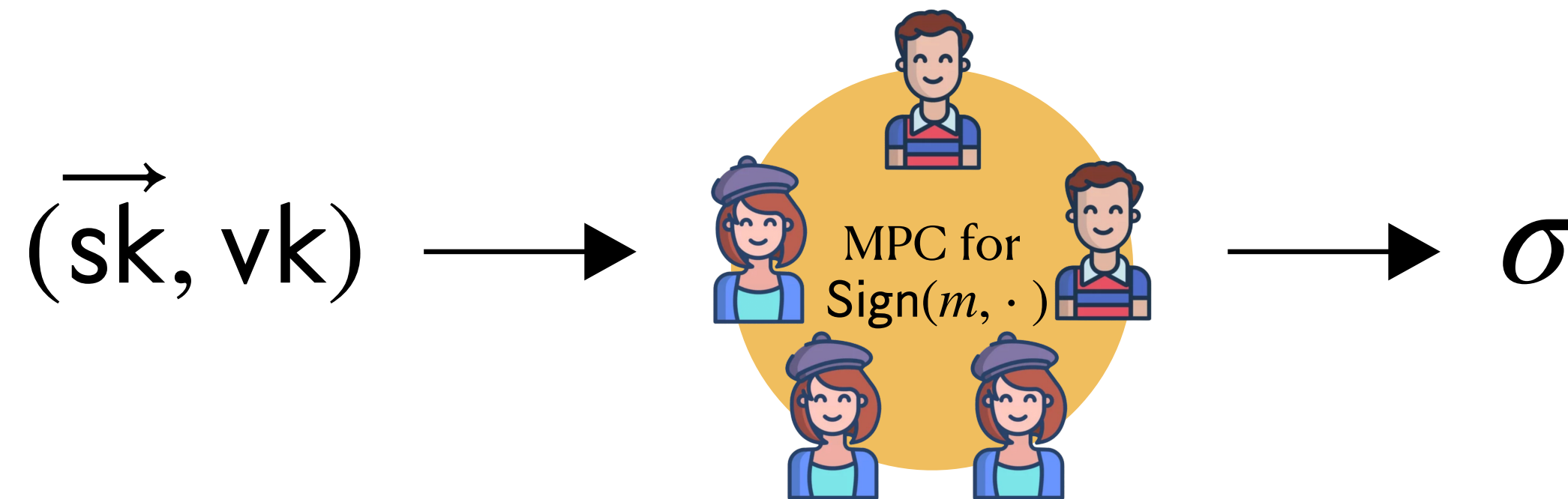
- Downside: evaluate SHA *inside* MPC (expensive!)
- Prefer if all Oracle calls are *black-box*



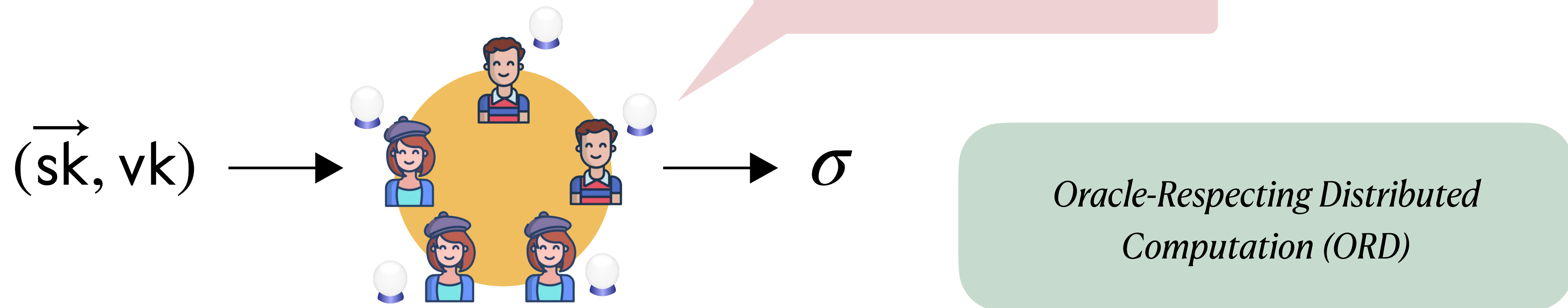
Threshold Signing

What kind of algorithms do we consider?

- In general: thresholdize signing using generic MPC



- Downside: evaluate SHA *inside* MPC (expensive!)
- Prefer if all Oracle calls are *black-box*



Our Results

- Define natural ‘**extractability**’ property of hash-based signatures, borrowed from NIZKs
- Rule out $(n-1, n)$ -protocols for ORD-computation of extractable signature schemes
- Our model: no CRS, no dealer, no preprocessing (more on that later)



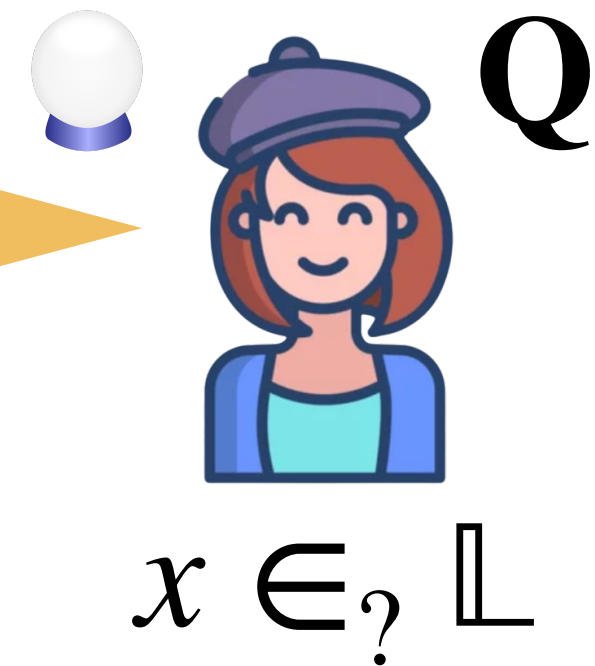
(Straight-line) Extractability for Signatures

- We recall the notion of NIZKs in the ROM.

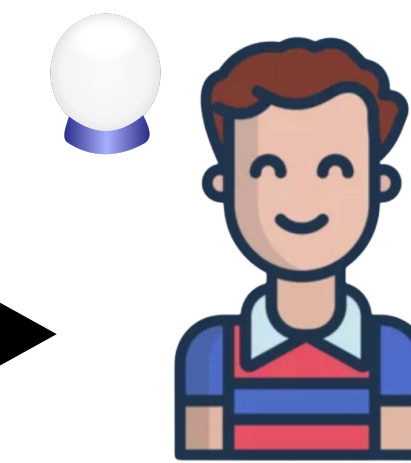
There exists \mathcal{E} such that given a correct proof,

$$\mathcal{E}(x, \pi, \mathbf{Q}) = w$$

where $(x, w) \in \mathbb{R}$



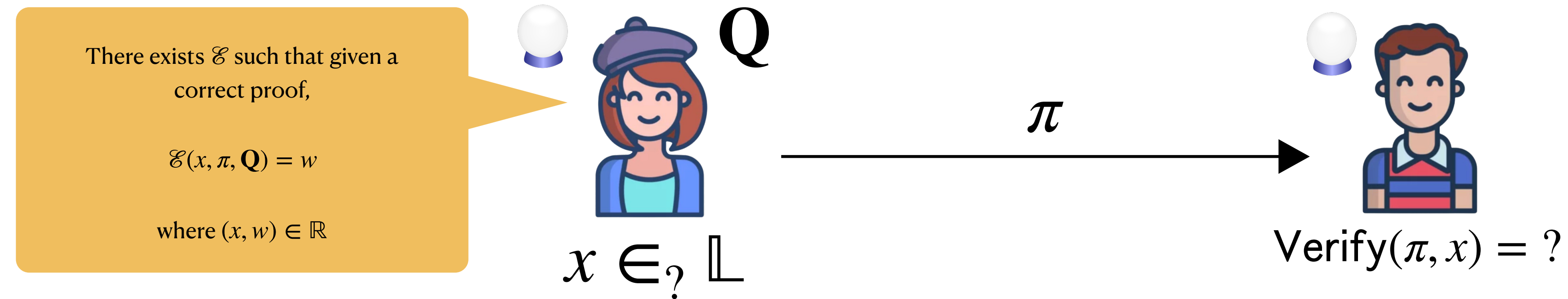
π



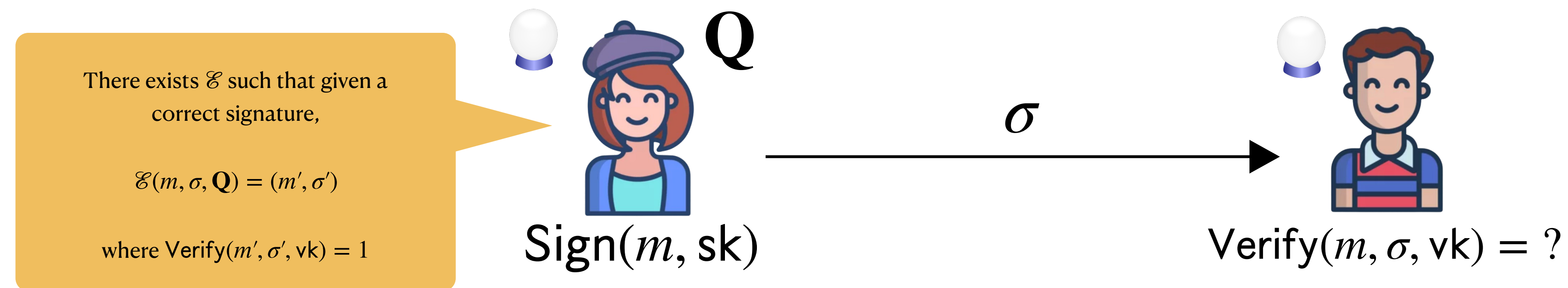
Verify(π, x) = ?

(Straight-line) Extractability for Signatures

- We recall the notion of NIZKs in the ROM.



- Similarly, we can define extractability for signatures:



(Straight-line) Extractability for Signatures

We call a signature scheme (k, t) -extractable if given k signatures and all but the first t oracle queries used to generate them, it is possible to forge a new signature on any message of choice.

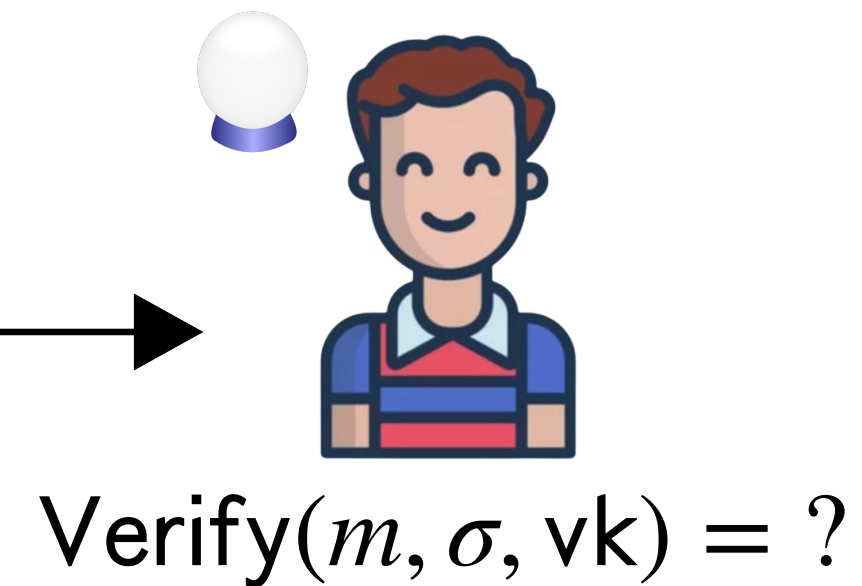
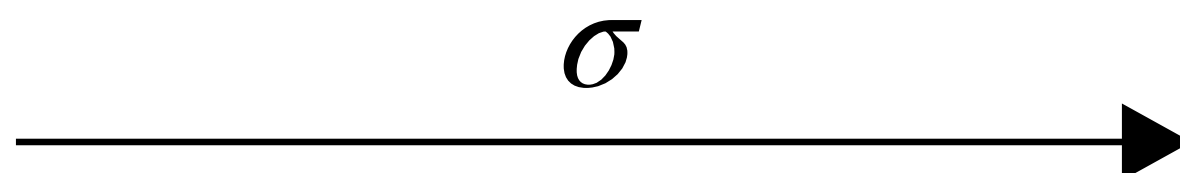
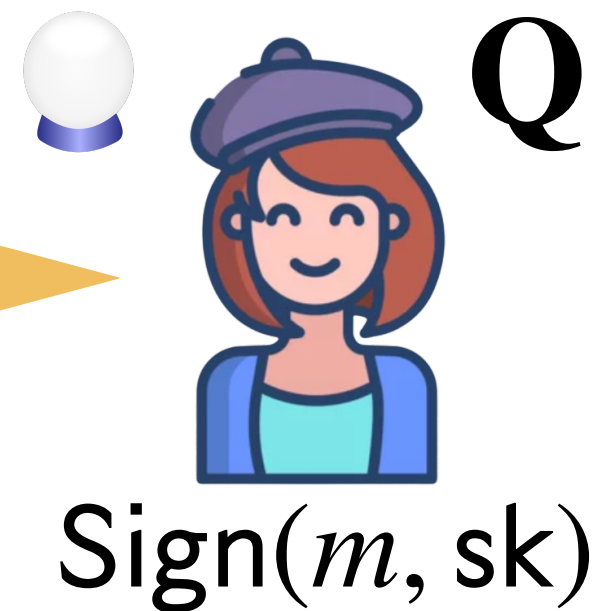
(k, t) -extractability

- Similarly, we can define extractability for signatures:

There exists \mathcal{E} such that given a correct signature,

$$\mathcal{E}(m, \sigma, \mathbf{Q}) = (m', \sigma')$$

where $\text{Verify}(m', \sigma', \text{vk}) = 1$



(Straight-line) Extractability for Signatures

We call a signature scheme (k, t) -extractable if given k signatures and all but the first t oracle queries used to generate them, it is possible to forge a new signature on any message of choice.

(k, t) -extractability

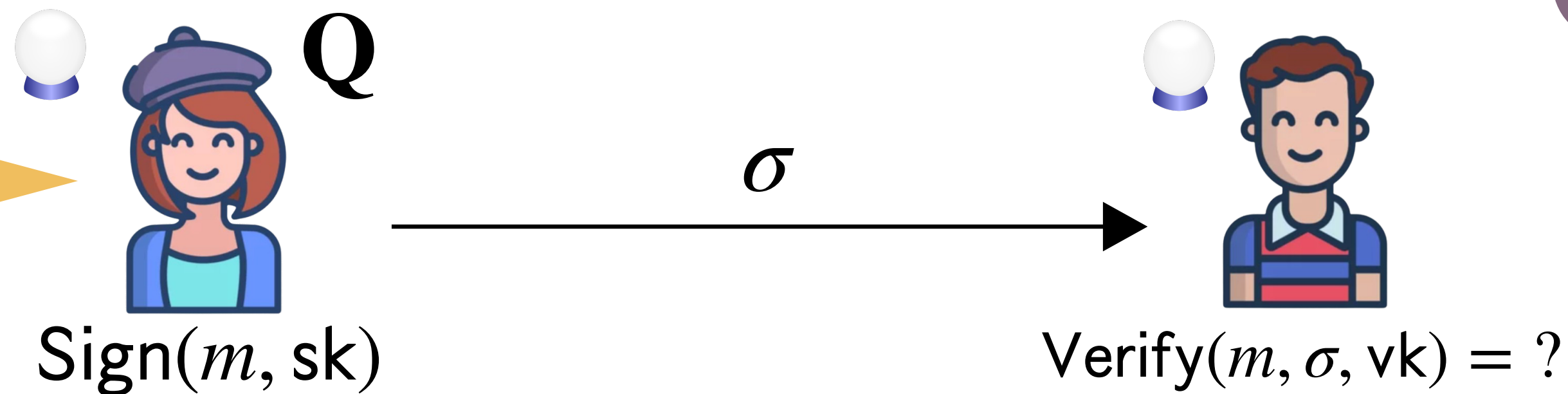
Side Note: (k, ∞) -extractability is just unforgeability

- Similarly, we can define extractability for signatures:

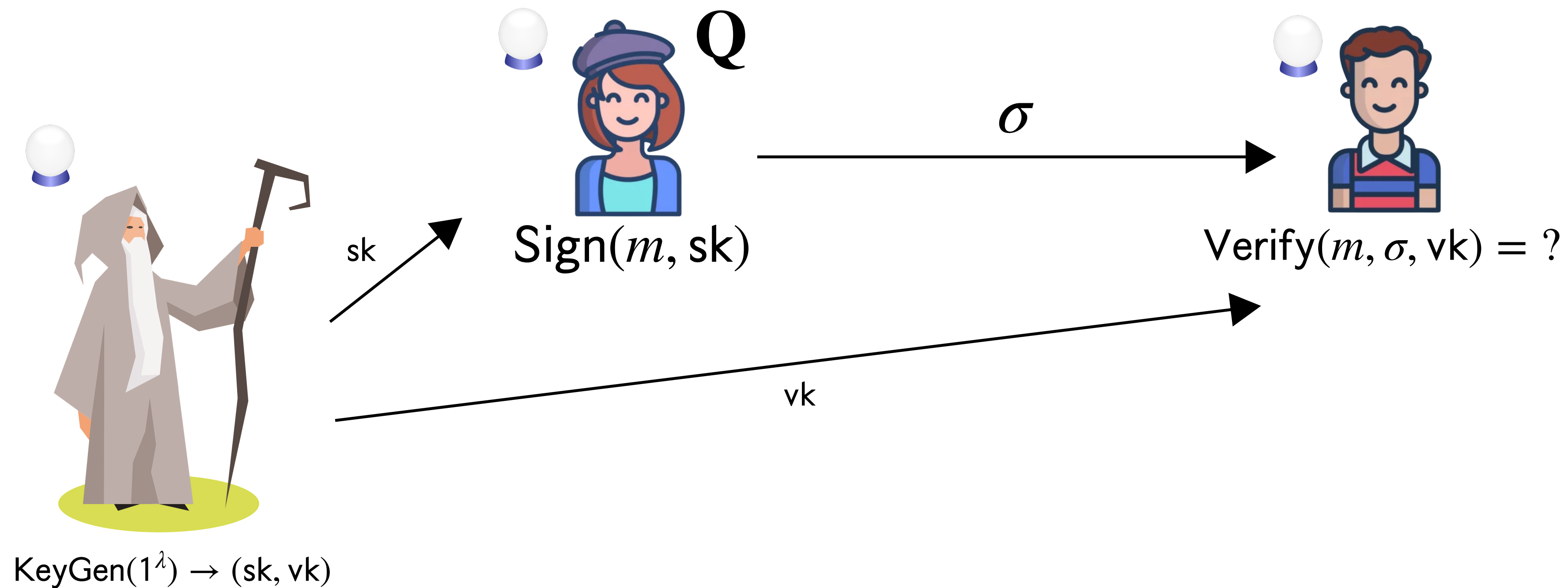
There exists \mathcal{E} such that given a correct signature,

$$\mathcal{E}(m, \sigma, \mathbf{Q}) = (m', \sigma')$$

where $\text{Verify}(m', \sigma', \text{vk}) = 1$



Extractable vs Fully Extractable

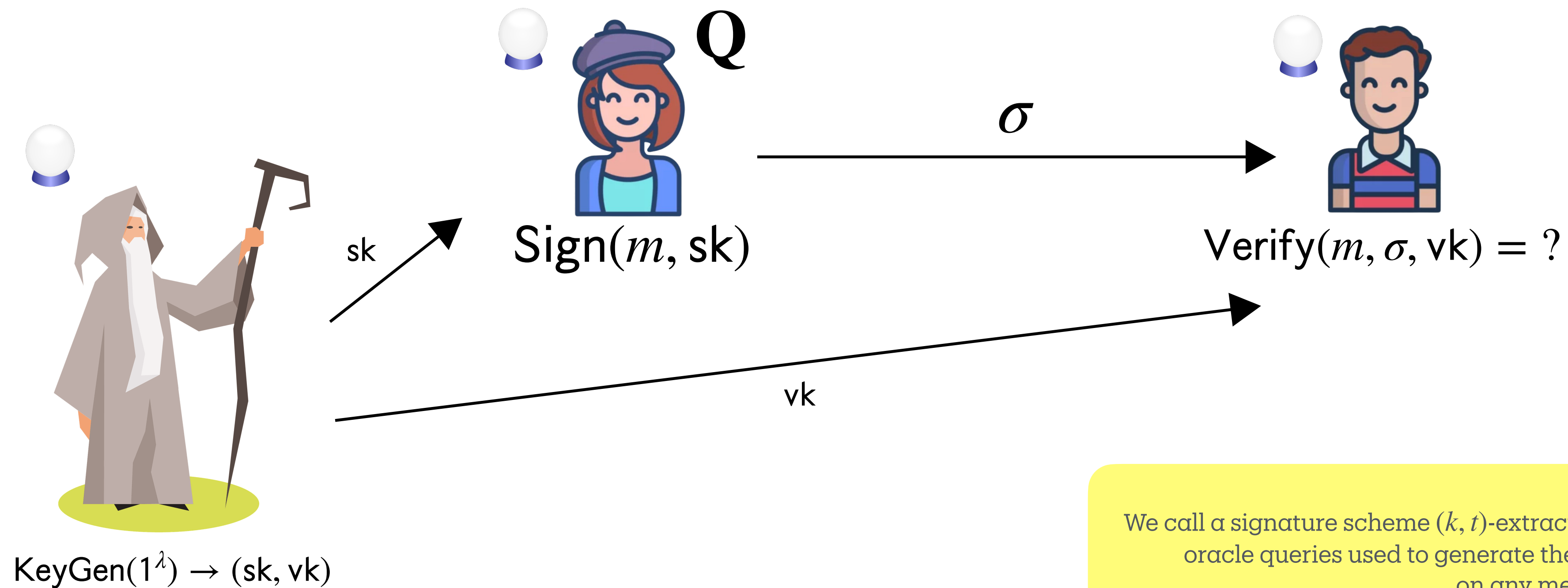


- Often, Key Generation is also distributed
- For something like Lamport, it is possible to sign with zero hash queries!
- Define fully extractable, including the *KeyGen* queries

We call a signature scheme (k, t) -extractable if given k signatures and all but the first t oracle queries used to generate them, it is possible to forge a new signature on any message of choice.

(k, t)-extractability

Extractable vs Fully Extractable



- Often, Key Generation is also distributed
- For something like Lamport, it is possible to sign with zero hash queries!
- Define fully extractable, including the *KeyGen* queries

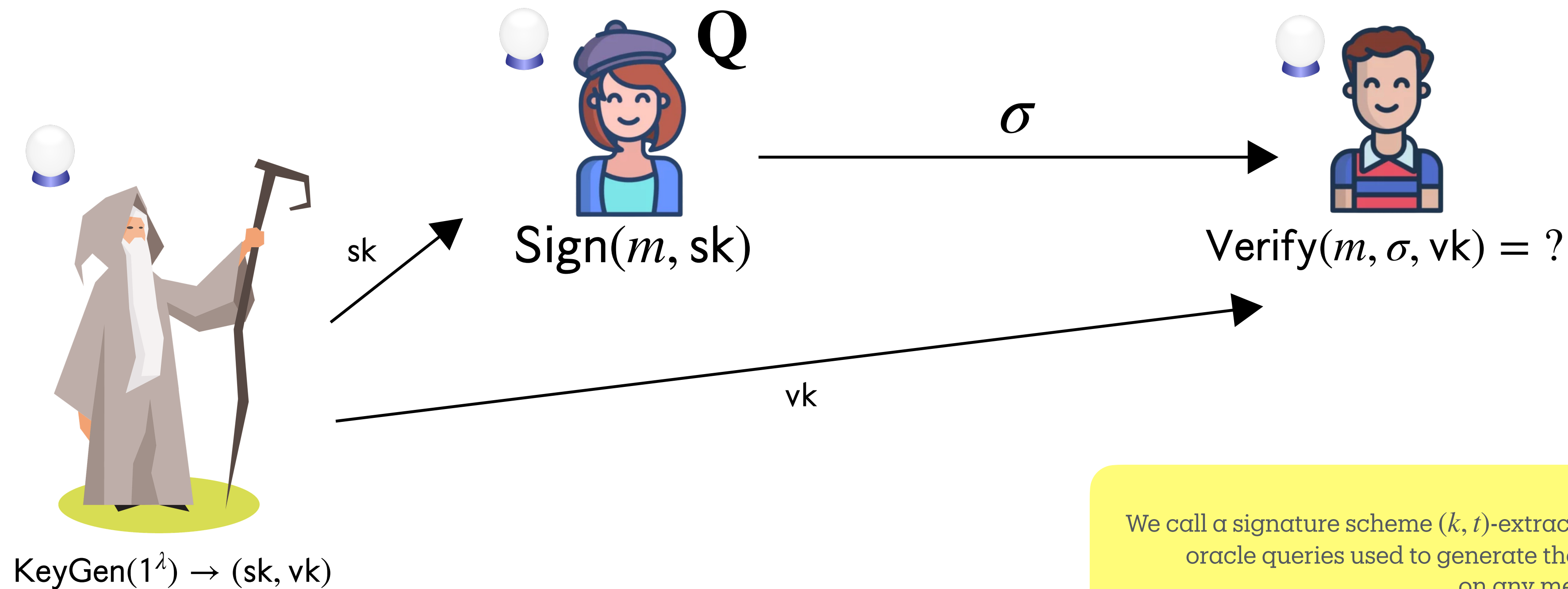
We call a signature scheme (k, t) -extractable if given k signatures and all but the first t oracle queries used to generate them, it is possible to forge a new signature on any message of choice.

(k, t)-extractability

We call a signature scheme (k, t) -extractable if given k signatures and all but the first t oracle queries used to generate them (including queries made during key generation), it is possible to forge a new signature on any message of choice.

(k, t)-full extractability

Extractable vs Fully Extractable



- Often, Key Generation is also distributed
- For something like Lamport, it is possible to sign with zero hash queries!
- Define fully extractable, including the *KeyGen* queries

We call a signature scheme (k, t) -extractable if given k signatures and all but the first t oracle queries used to generate them, it is possible to forge a new signature on any message of choice.

(k, t)-extractability

We call a signature scheme (k, t) -fully extractable if given k signatures and all but the first t oracle queries used to generate them (including queries made during key generation), it is possible to forge a new signature on any message of choice.

(k, t)-full extractability

Extractability for Known Signatures

We call a signature scheme (k, t) -extractable if given k signatures and all but the first t oracle queries used to generate them, it is possible to forge a new signature on any message of choice.

(k, t) -extractability

- Are popular hash based signatures extractable? *Yes!*
- Can be proven easily for XMSS (fully extractable), SPHINCS, SPHINCS+

Extractability for Known Signatures

We call a signature scheme (k, t) -extractable if given k signatures and all but the first t oracle queries used to generate them, it is possible to forge a new signature on any message of choice.

(k, t) -extractability

- Are popular hash based signatures extractable? *Yes!*
- Can be proven easily for XMSS (fully extractable), SPHINCS, SPHINCS+

Intuition:

1. Lamport is extractable, and all schemes use Lamport-style subschemes
2. If no source of hardness outside the RO can be used, then hiding the queries is the only option available!

Impossibility of ORD-Computing Hash-Based Schemes

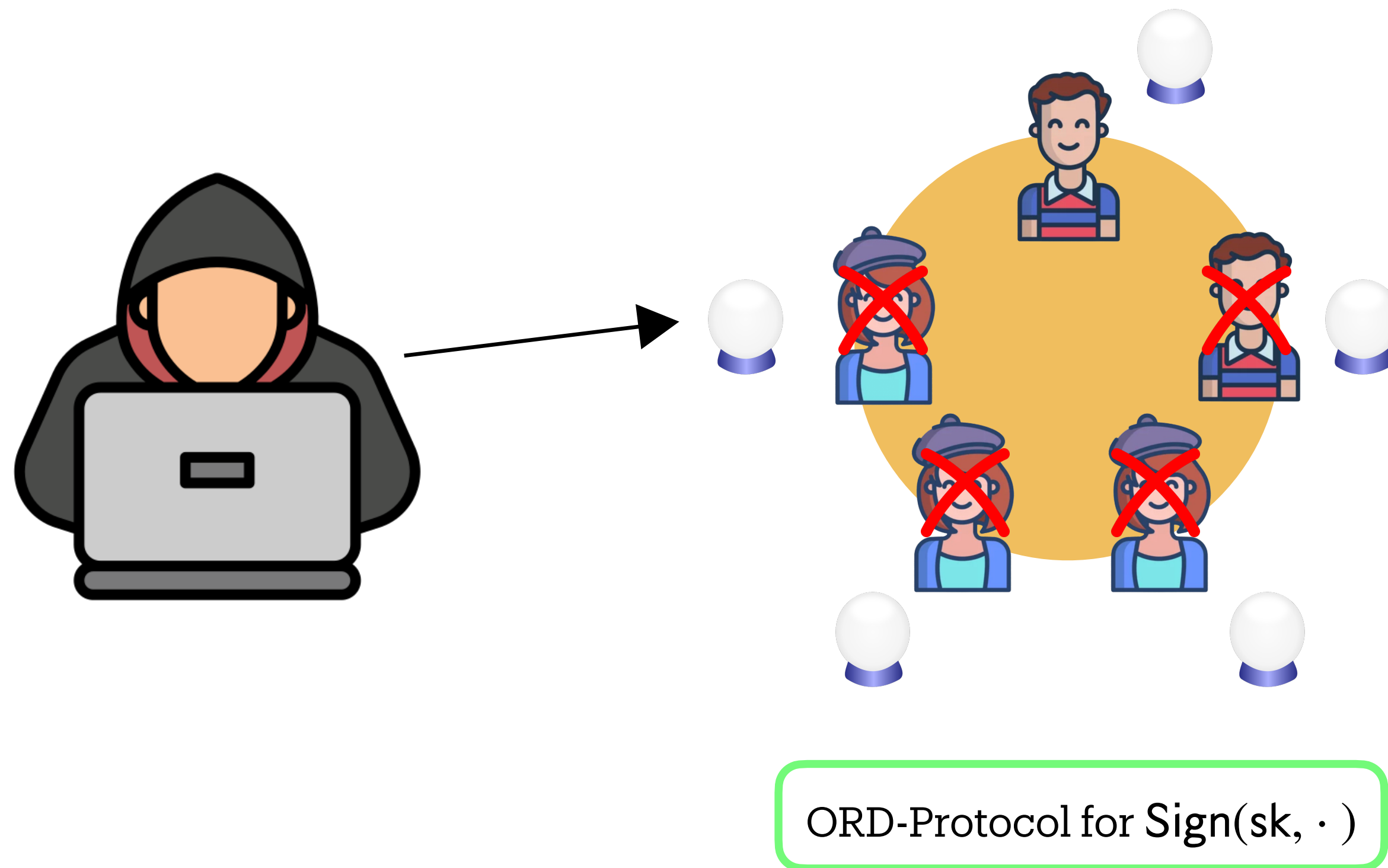
(Extractable Case; Fully Extractable is Similar)



ORD-Protocol for $\text{Sign}(\text{sk}, \cdot)$

Impossibility of ORD-Computing Hash-Based Schemes

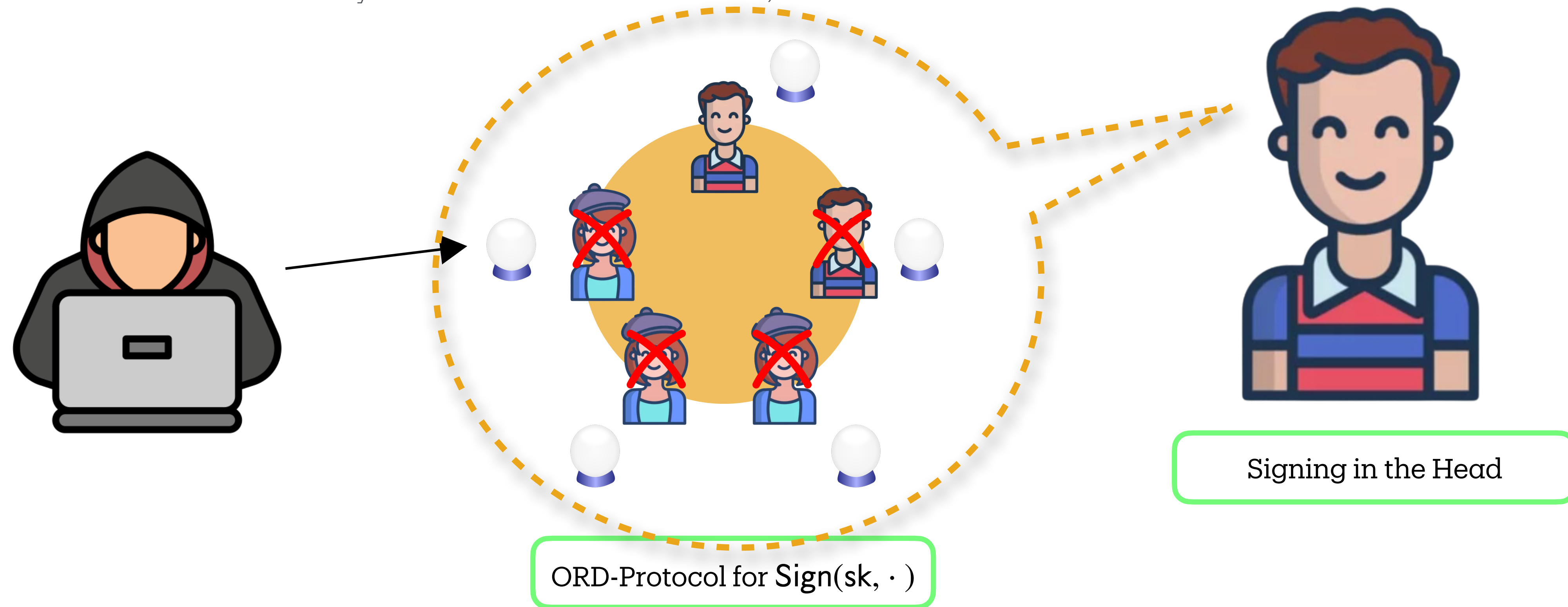
(Extractable Case; Fully Extractable is Similar)



- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party's queries

Impossibility of ORD-Computing Hash-Based Schemes

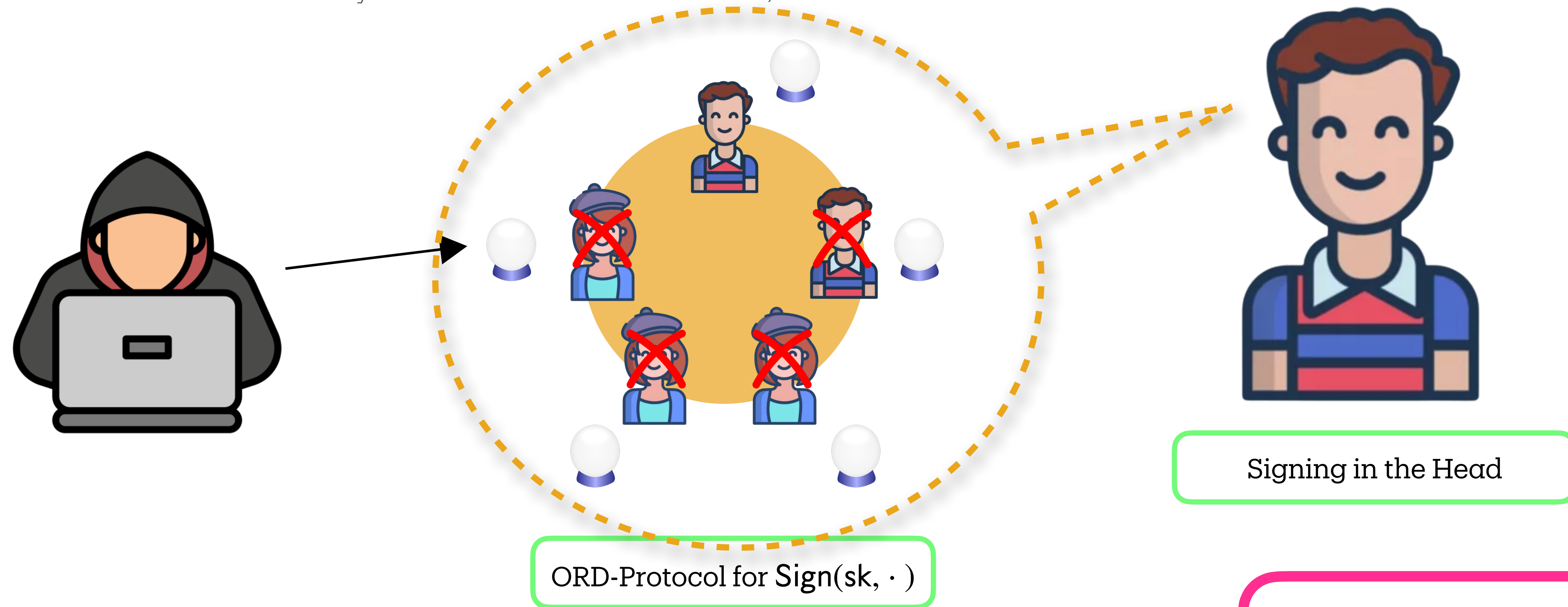
(Extractable Case; Fully Extractable is Similar)



- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party's queries

Impossibility of ORD-Computing Hash-Based Schemes

(Extractable Case; Fully Extractable is Similar)

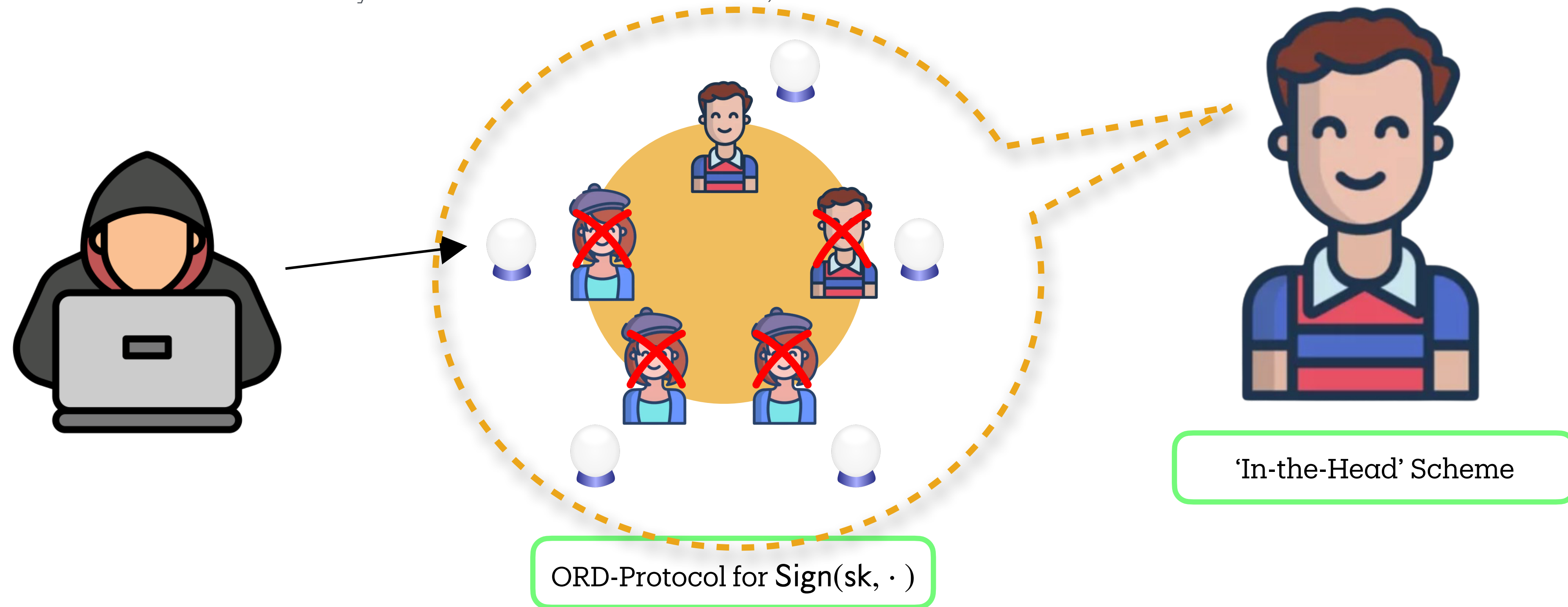


- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party's queries

Theorem (from the definition of extractability): *If a scheme is extractable, every other scheme which has the same verifier is also extractable!*

Impossibility of ORD-Computing Hash-Based Schemes

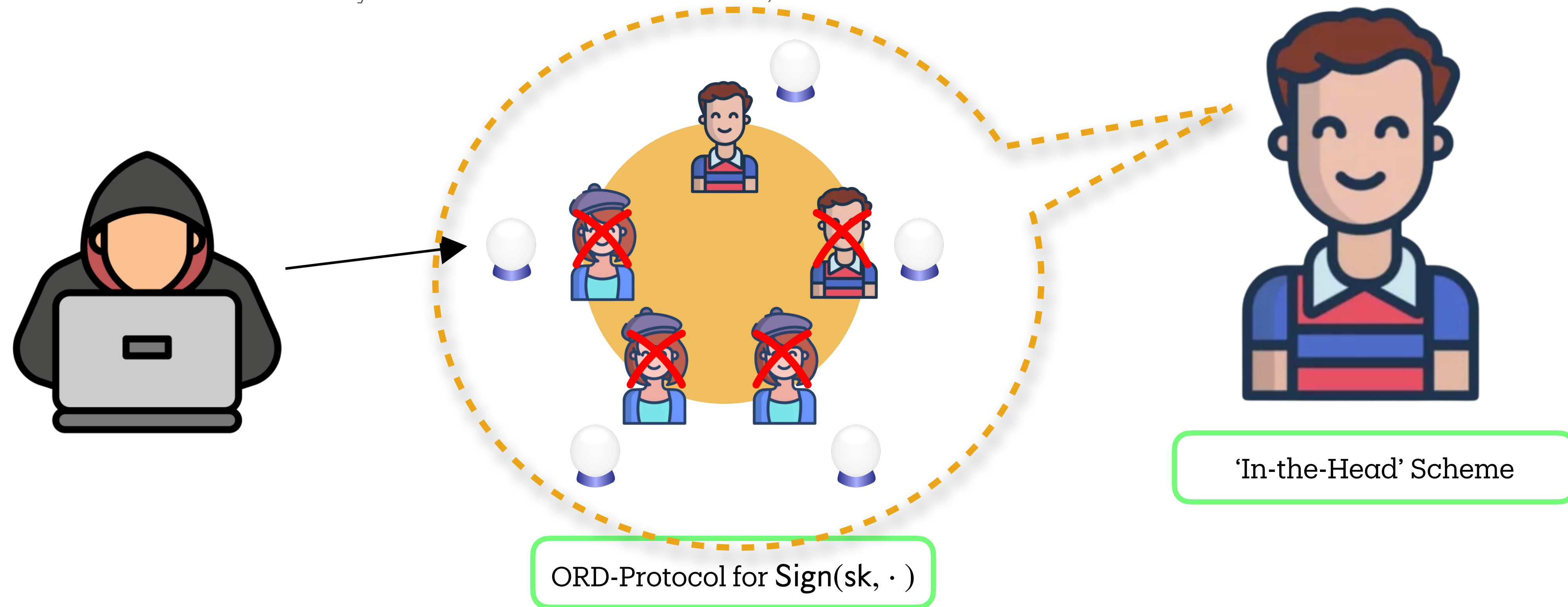
(Extractable Case; Fully Extractable is Similar)



- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party's queries
- **Step 3:** Run the extractor for the 'in-the-head' scheme to forge

Impossibility of ORD-Computing Hash-Based Schemes

(Extractable Case; Fully Extractable is Similar)

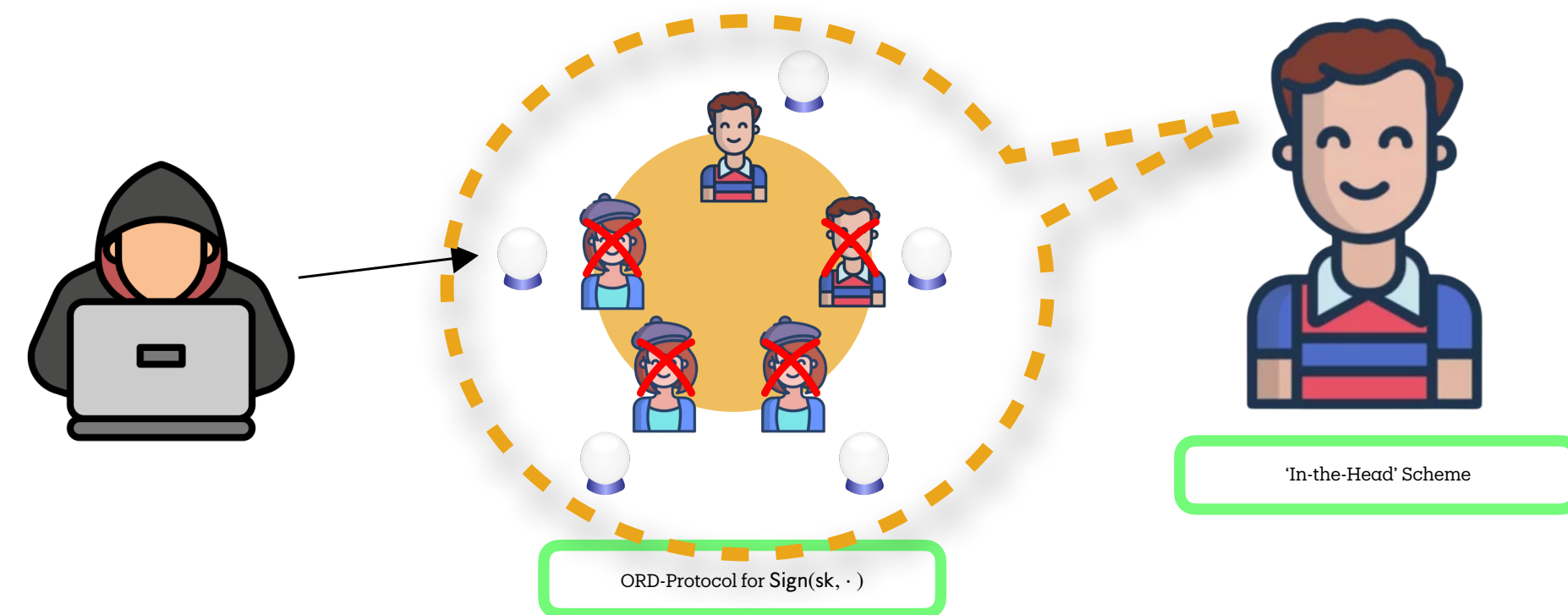


- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party's queries
- **Step 3:** Run the extractor for the 'in-the-head' scheme to forge

But you don't have all the oracle queries!

Impossibility of ORD-Computing Hash-Based Schemes

(Extractable Case; Fully Extractable is Similar)



- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party's queries
- **Step 3:** Run the extractor for the 'in-the-head' scheme to forge

But you don't have all the oracle queries!

Sometimes You Can't Distribute
Random-Oracle-Based Proofs

Jack Doerner
j@ckdoerner.net
Brown University, Technion, Reichman University

Yashvanth Kondi Leah Namisa Rosenbloom
yash@ykondi.net leah_rosenbloom@brown.edu
Silence Labs (Deel) Brown University

May 21, 2024

Abstract

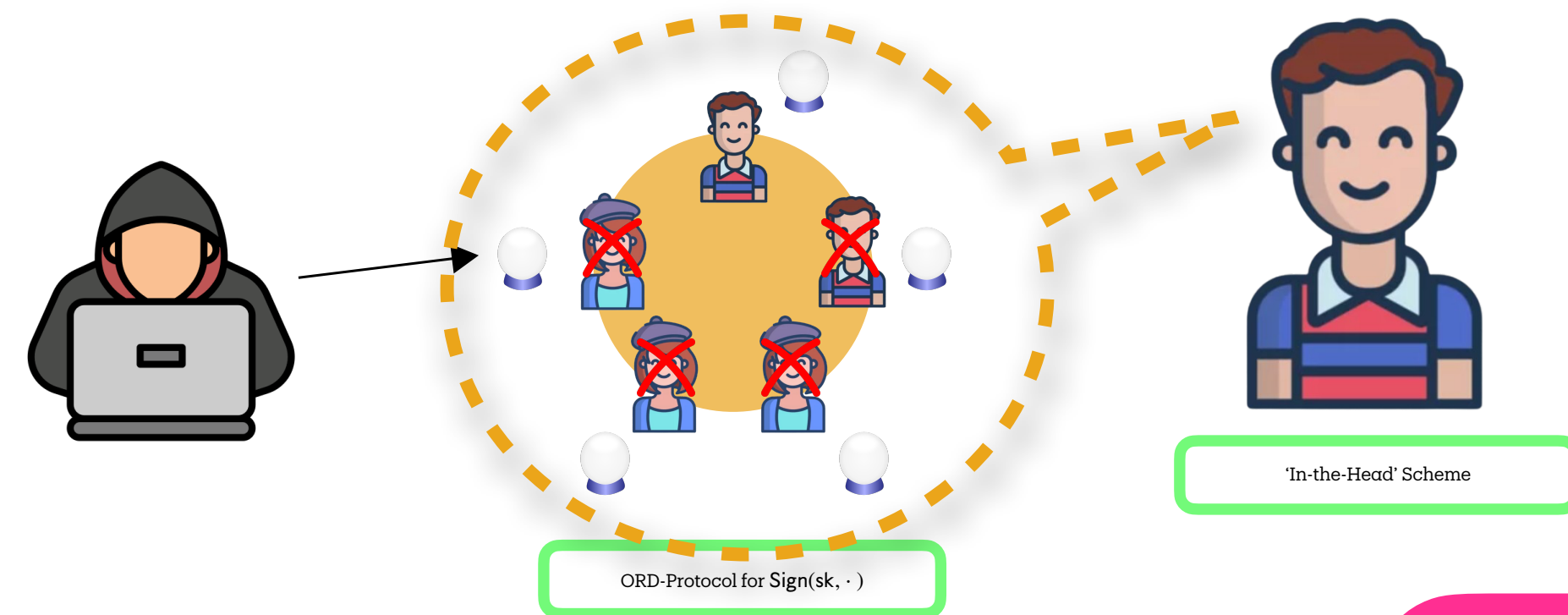
We investigate the conditions under which straight-line extractable NIZKs in the random oracle model (i.e. without a CRS) permit multiparty realizations that are black-box in the same random oracle. We show that even in the semi-honest setting, any MPC protocol to compute such a NIZK cannot make black-box use of the random oracle or a hash function instantiating it if security against all-but-one corruptions is desired, unless the number of queries made by the verifier to the oracle grows linearly with the number of parties. This presents a fundamental barrier to constructing efficient protocols to securely distribute the computation of NIZKs (and signatures) based on MPC-in-the-head, PCPs/IOPs, and sigma protocols compiled with transformations due to Fischlin, Pass, or Unruh.

When the adversary is restricted to corrupt only a constant fraction of parties, we give a positive result by means of a tailored construction, which demonstrates that our impossibility does not extend to weaker corruption models in general.

[DKR24]: Crypto '24

Impossibility of ORD-Computing Hash-Based Schemes

(Extractable Case; Fully Extractable is Similar)



- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party's queries
- **Step 3:** Run the extractor for the 'in-the-head' scheme to forge

But you don't have all the oracle queries!

Lemma:
It's possible to simulate the queries you don't have and still succeed given an n that is 'large enough.'

Just picking the queries to be random still allows you to successfully forge with $\frac{1}{\text{poly}(\lambda)}$ probability.

Sometimes You Can't Distribute
Random-Oracle-Based Proofs

Jack Doerner
j@ckdoerner.net

Brown University, Technion, Reichman University
Yashvanth Kondi Leah Namisa Rosenbloom
yash@ykondi.net leah_rosenbloom@brown.edu
Silence Labs (Deel) Brown University

May 21, 2024

Abstract

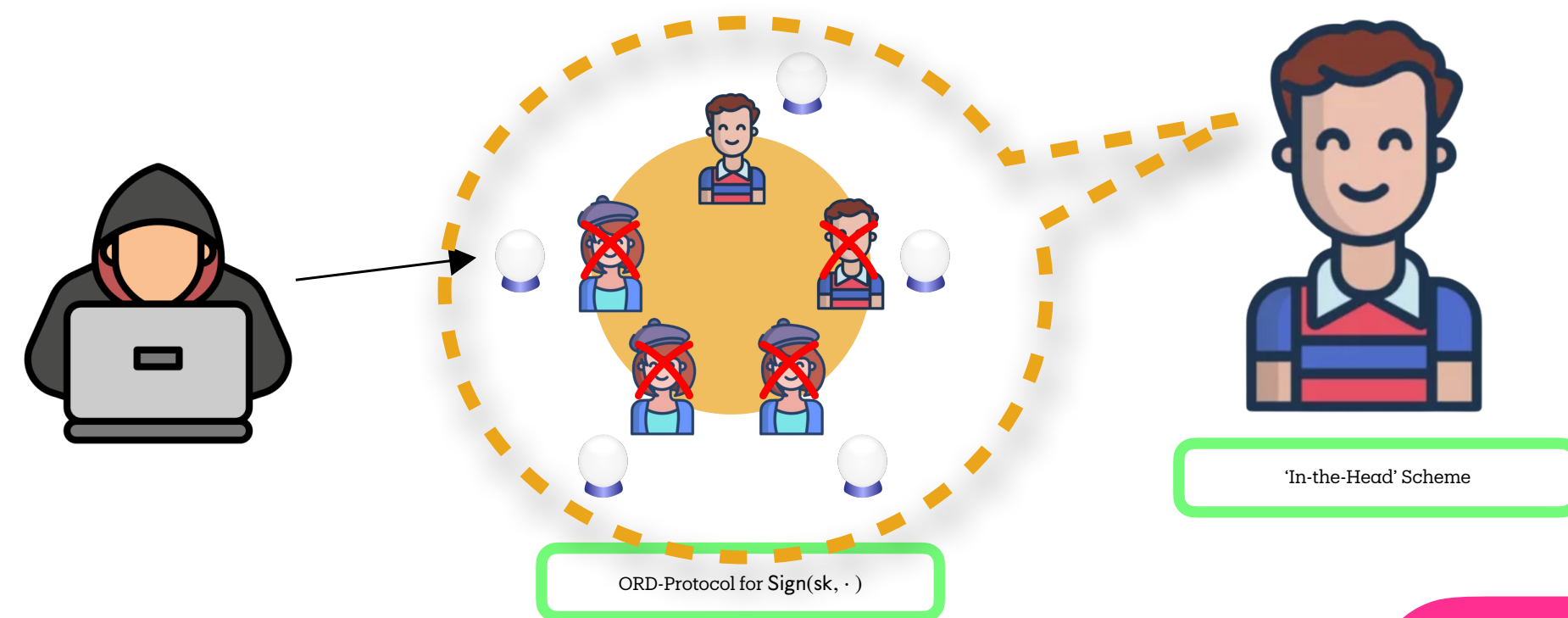
We investigate the conditions under which straight-line extractable NIZKs in the random oracle model (i.e. without a CRS) permit multiparty realizations that are black-box in the same random oracle. We show that even in the semi-honest setting, any MPC protocol to compute such a NIZK cannot make black-box use of the random oracle or a hash function instantiating it if security against all-but-one corruptions is desired, unless the number of queries made by the verifier to the oracle grows linearly with the number of parties. This presents a fundamental barrier to constructing efficient protocols to securely distribute the computation of NIZKs (and signatures) based on MPC-in-the-head, PCPs/IOPs, and sigma protocols compiled with transformations due to Fischlin, Pass, or Unruh.

When the adversary is restricted to corrupt only a constant fraction of parties, we give a positive result by means of a tailored construction, which demonstrates that our impossibility does not extend to weaker corruption models in general.

[DKR24]: Crypto '24

Impossibility of ORD-Computing Hash-Based Schemes

(Extractable Case; Fully Extractable is Similar)



How big? $n > \#$ of queries made by the verifier, so thousands/tens of thousands!

- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party's queries
- **Step 3:** Run the extractor for the 'in-the-head' scheme to forge

But you don't have all the oracle queries!

Lemma:
It's possible to simulate the queries you don't have and still succeed given an n that is 'large enough.'

Just picking the queries to be random still allows you to successfully forge with $\frac{1}{\text{poly}(\lambda)}$ probability.

Sometimes You Can't Distribute Random-Oracle-Based Proofs

Jack Doerner
j@ckdoerner.net
Brown University, Technion, Reichman University

Yashvanth Kondi Leah Namisa Rosenbloom
yash@ykondi.net leah_rosenbloom@brown.edu
Silence Labs (Deel) Brown University

May 21, 2024

Abstract

We investigate the conditions under which straight-line extractable NIZKs in the random oracle model (i.e. without a CRS) permit multiparty realizations that are black-box in the same random oracle. We show that even in the semi-honest setting, any MPC protocol to compute such a NIZK cannot make black-box use of the random oracle or a hash function instantiating it if security against all-but-one corruptions is desired, unless the number of queries made by the verifier to the oracle grows linearly with the number of parties. This presents a fundamental barrier to constructing efficient protocols to securely distribute the computation of NIZKs (and signatures) based on MPC-in-the-head, PCPs/IOPs, and sigma protocols compiled with transformations due to Fischlin, Pass, or Unruh.

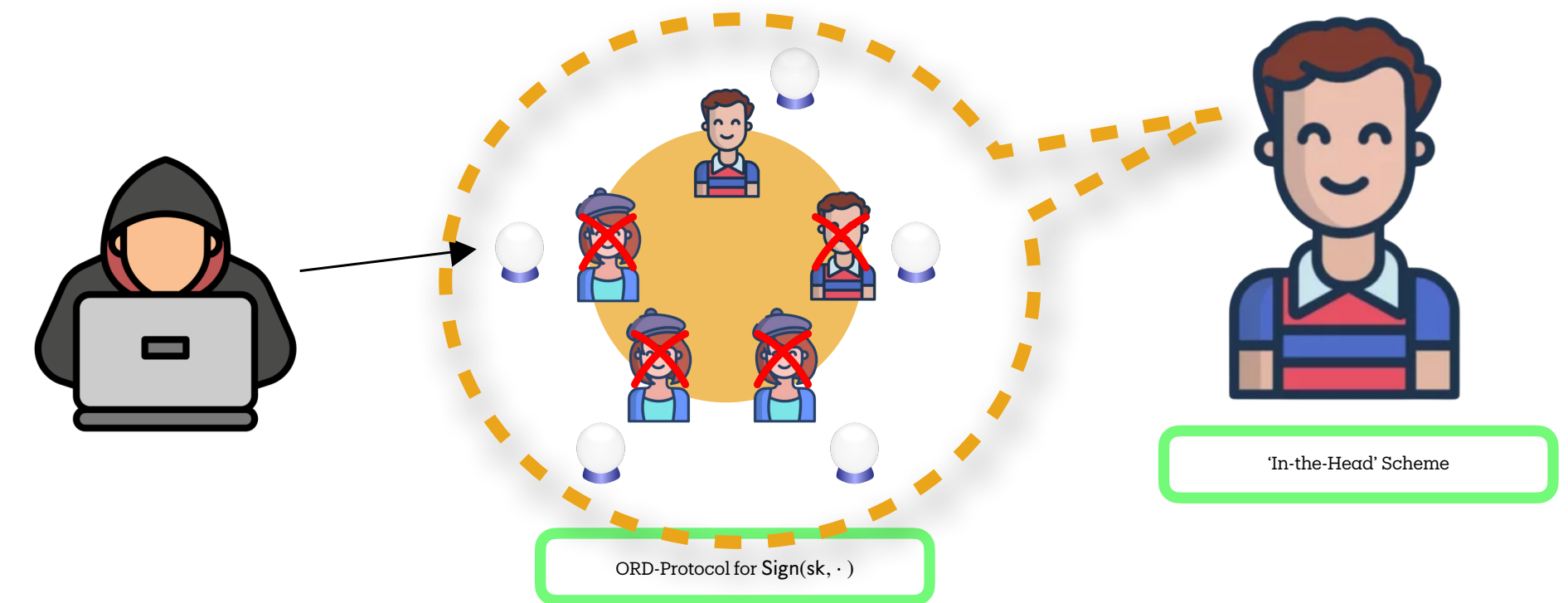
When the adversary is restricted to corrupt only a constant fraction of parties, we give a positive result by means of a tailored construction, which demonstrates that our impossibility does not extend to weaker corruption models in general.

[DKR24]: Crypto '24

It can get a lot worse

What about 2 parties?

- Most schemes like SPHINCS/SPHINCS+ have an ‘important’ hash query that cannot be leaked: usually a PRF key
- Even for 2 parties, this means that you can’t thresholdize without a trusted dealer or preprocessing.



- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party’s queries
- **Step 3:** Run the extractor for the ‘in-the-head’ scheme to forge

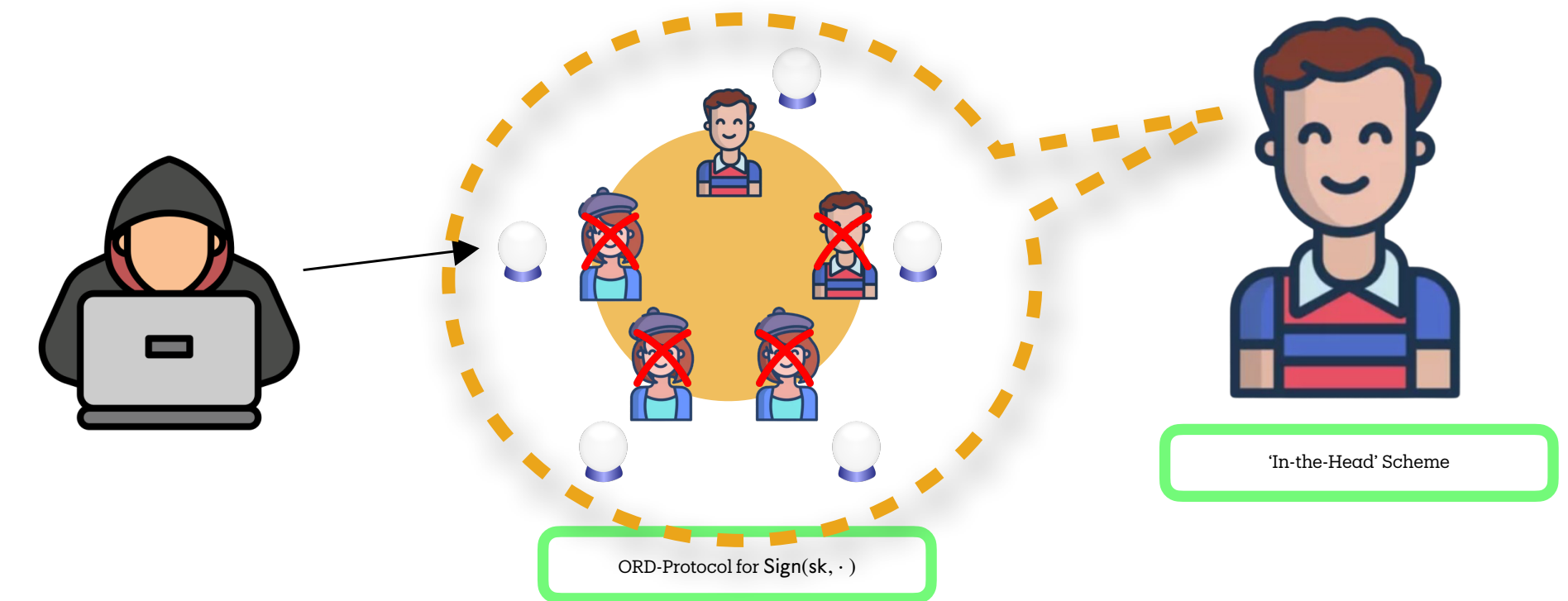
It can get a lot worse

What about 2 parties?

- Most schemes like SPHINCS/SPHINCS+ have an ‘important’ hash query that cannot be leaked: usually a PRF key
- Even for 2 parties, this means that you can’t thresholdize without a trusted dealer or preprocessing.

Any scheme which ORD-computes signatures for k messages requires at least k steps of preprocessing.

Impossibility with Preprocessing



- **Step 1:** Let the parties produce a signature σ , adversary acts honestly
- **Step 2:** Adversary now has all but one party’s queries
- **Step 3:** Run the extractor for the ‘in-the-head’ scheme to forge

Conclusions and Alternatives

- Threshold Signing Hash-Based Schemes has the following caveats:

Either:

1. The protocol must make non-black box use of the hash function (ie. evaluate SHA inside MPC or use structured, algebraic hashes which support efficient MPC protocols)
2. Schemes require a (not-small) amount of preprocessing by a trusted dealer, making them unsuitable for applications where such a dealer is absent unless efficiency can be compromised on.

Conclusions and Alternatives

- Threshold Signing Hash-Based Schemes has the following caveats:

Either:

1. The protocol must make non-black box use of the hash function (ie. evaluate SHA inside MPC or use structured, algebraic hashes which support efficient MPC protocols)
2. Schemes require a (not-small) amount of preprocessing by a trusted dealer, making them unsuitable for applications where such a dealer is absent unless efficiency can be compromised on.

What if you simply use
generic MPC?

Conclusions and Alternatives

- Threshold Signing Hash-Based Schemes has the following caveats:

Either:

1. The protocol must make non-black box use of the hash function (ie. evaluate SHA inside MPC or use structured, algebraic hashes which support efficient MPC protocols)
2. Schemes require a (not-small) amount of preprocessing by a trusted dealer, making them unsuitable for applications where such a dealer is absent unless efficiency can be compromised on.

What if you simply use generic MPC?

Some Basic Estimations

We signed XMSS for two parties using a garbled circuit implementation of the EMP-tool library for a 256-bit string with

$$w = 16, \ell = 67$$

and a naive, unoptimized implementation takes time

$$\approx 316.52 \pm 1.45 \text{ ms}$$

for computing the W-OTS function $f_{sk_i}^{w-1}(\cdot)$.

For SPHINCS+, the total number of hashes for standard parameters is >1,000,000, which comes up to an estimated runtime of >20,000s with naive garbling techniques!

(Note that the above can almost certainly be optimized to yield much better efficiency with preprocessing and more sophisticated protocol design).

Thank You!