

# Hash-Based Multi-Signatures for Post-Quantum Ethereum



**Benedikt Wagner, Ethereum Foundation**

Joint work with  
Justin Drake, Dmitry Khovratovich, Mikhail Kudinov

Context: Beam Chain

# Context: Beam Chain



Paper

### Hash-Based Multi-Signatures for Post-Quantum Ethereum

Justin Drake<sup>1</sup>

Dmitry Khovratovich<sup>1</sup>  
Benedikt Wagner<sup>1</sup>

Mikhail Kudinov<sup>2</sup>

January 14, 2025

<sup>1</sup> Ethereum Foundation  
`{justin.drake,dmitry.khovratovich,benedikt.wagner}@ethereum.org`

<sup>2</sup> Eindhoven University of Technology  
`mishel.kudinov@gmail.com`

#### Abstract

With the threat posed by quantum computers on the horizon, systems like Ethereum must transition to cryptographic primitives resistant to quantum attacks. One of the most critical of these primitives is the non-interactive multi-signature scheme used in Ethereum’s proof-of-stake consensus, currently implemented with BLS signatures. This primitive enables validators to independently sign blocks, with their signatures then publicly aggregated into a compact aggregate signature.

In this work, we introduce a family of hash-based signature schemes as post-quantum alternatives to BLS. We consider the folklore method of aggregating signatures via (hash-based) succinct arguments, and our work is focused on instantiating the underlying signature scheme. The proposed schemes are variants of the XMSS signature scheme, analyzed within a novel and unified framework. While being generic, this framework is designed to minimize security loss, facilitating efficient parameter selection. A key feature of our work is the avoidance of random oracles in the security proof. Instead, we define explicit standard model requirements for the underlying hash functions. This eliminates the paradox of simultaneously treating hash functions as random oracles and as explicit circuits for aggregation. Furthermore, this provides cryptanalysts with clearly defined targets for evaluating the security of hash functions. Finally, we provide recommendations for practical instantiations of hash functions and concrete parameter settings, supported by known and novel heuristic bounds on the standard model properties.

Code

# Outline

The Problem

Overall Paradigm

Signature Design

Next Steps

# Outline

The Problem

Overall Paradigm

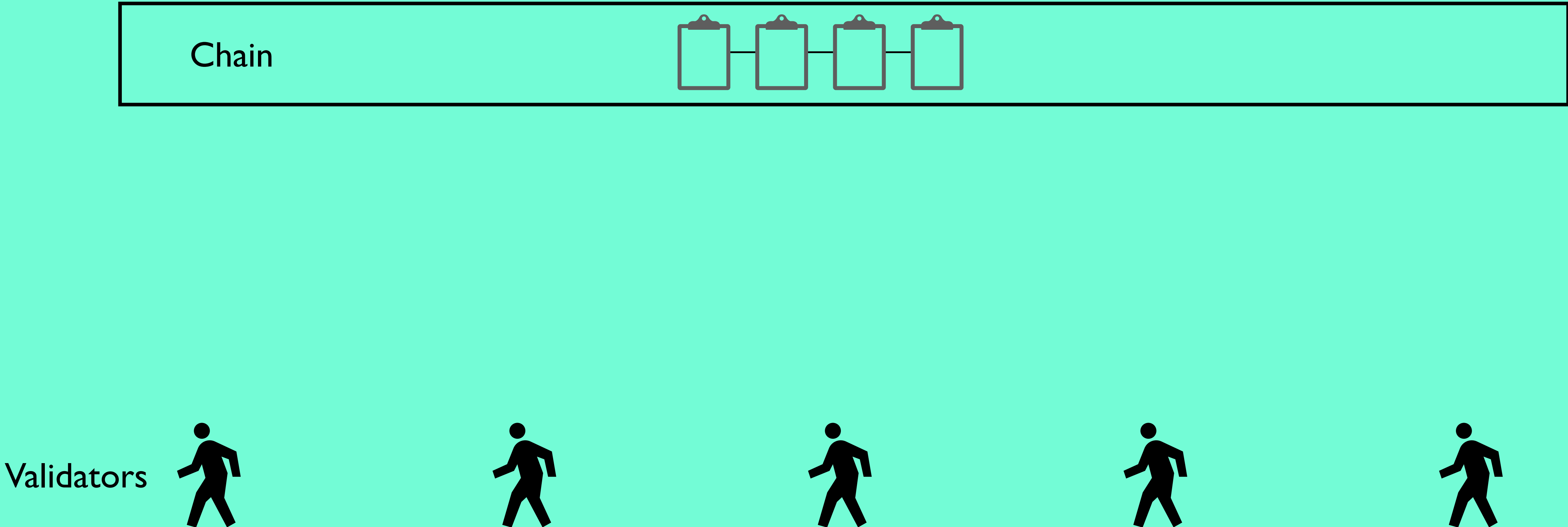
Signature Design

Next Steps

# Multi-Signatures in Ethereum Consensus

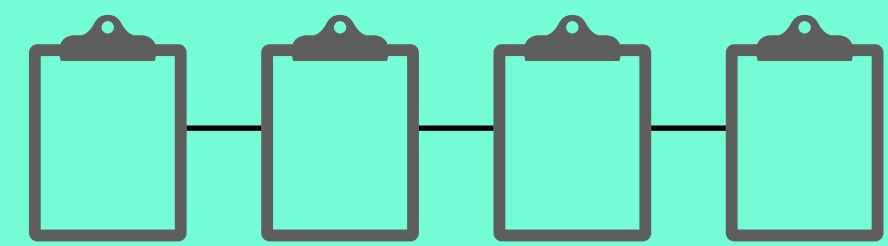


# Multi-Signatures in Ethereum Consensus



# Multi-Signatures in Ethereum Consensus

Chain  $pk_1, pk_2, pk_3, pk_4, pk_5$



Validators



$pk_1, sk_1$



$pk_2, sk_2$



$pk_3, sk_3$



$pk_4, sk_4$

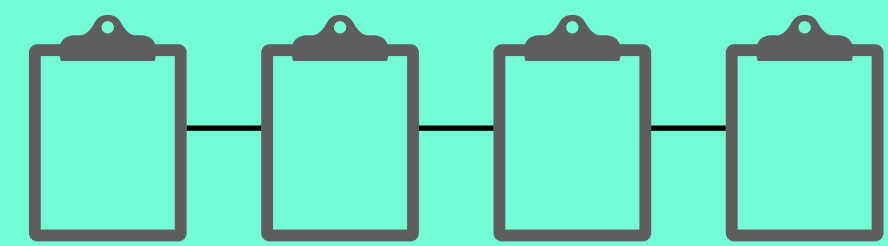


$pk_5, sk_5$



# Multi-Signatures in Ethereum Consensus

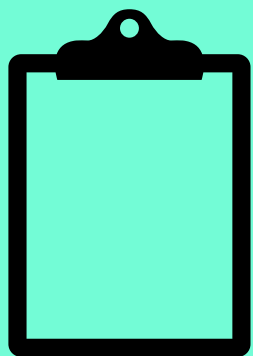
Chain     $pk_1, pk_2, pk_3, pk_4, pk_5$



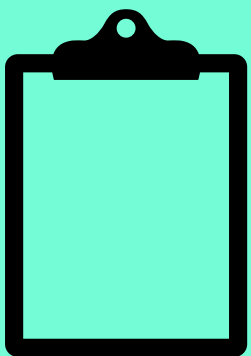
Validators



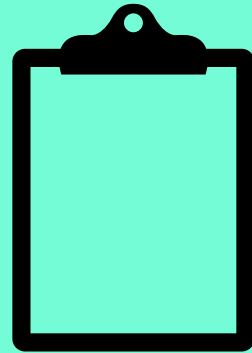
$pk_1, sk_1$



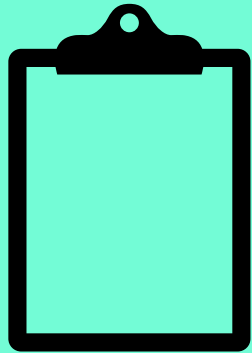
$pk_2, sk_2$



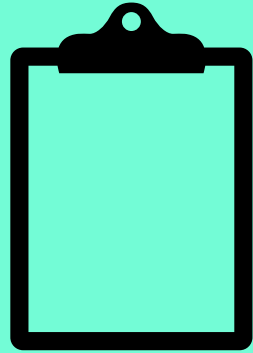
$pk_3, sk_3$



$pk_4, sk_4$

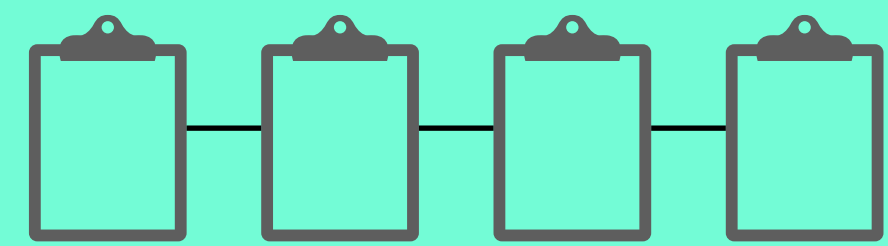


$pk_5, sk_5$



# Multi-Signatures in Ethereum Consensus

Chain     $pk_1, pk_2, pk_3, pk_4, pk_5$

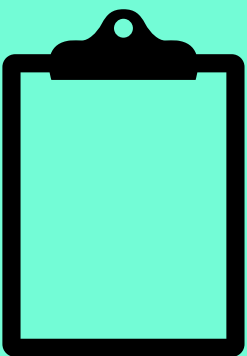


Validators

$Sig_1$



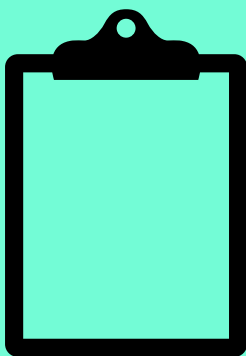
$pk_1, sk_1$



$Sig_2$



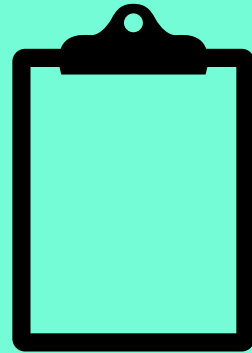
$pk_2, sk_2$



$Sig_3$



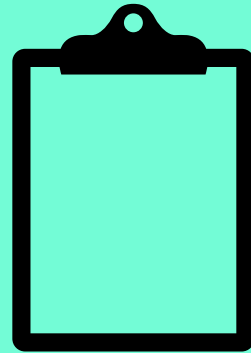
$pk_3, sk_3$



$Sig_4$



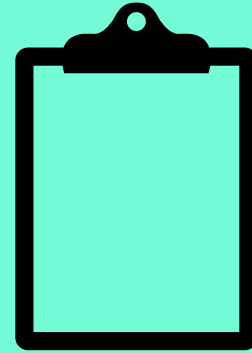
$pk_4, sk_4$



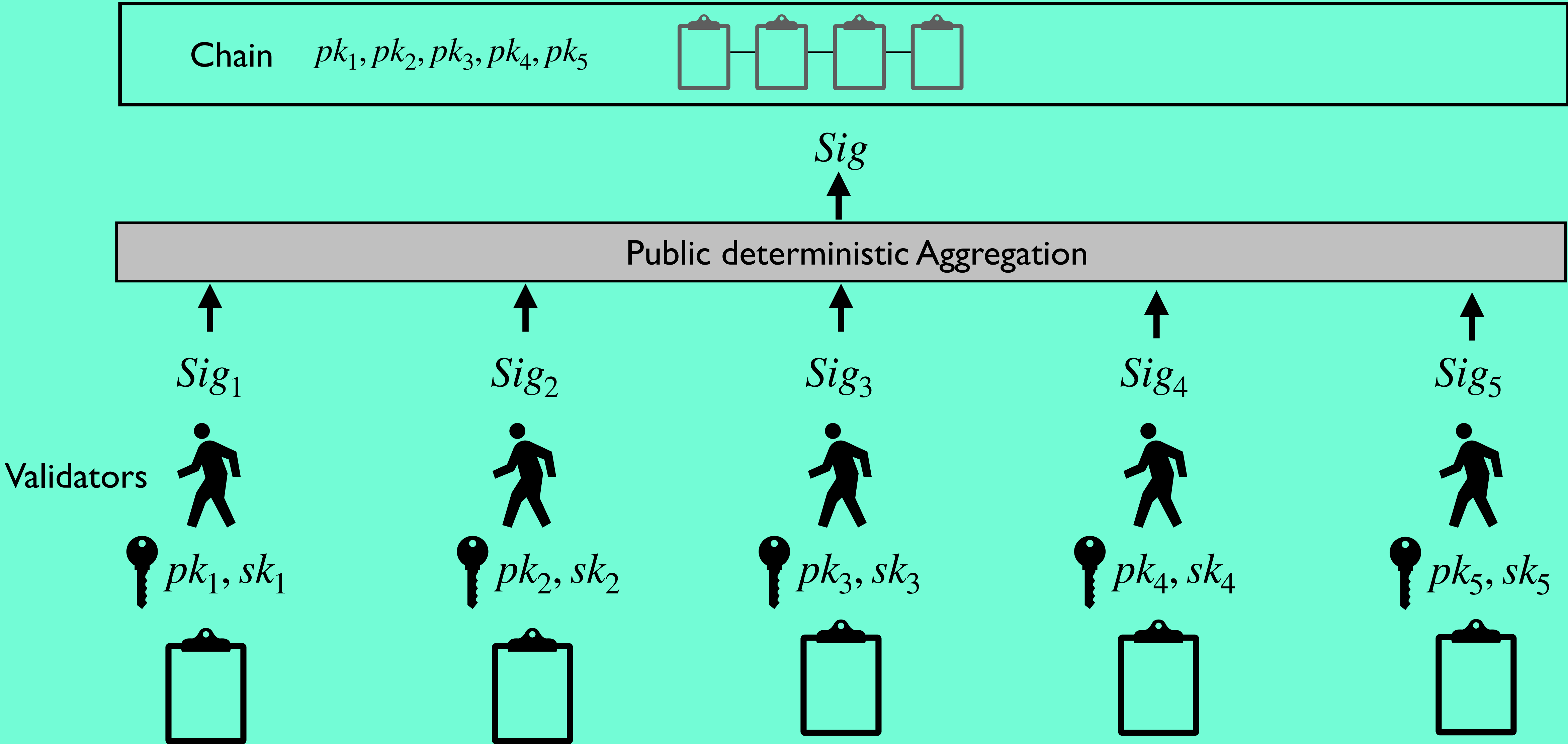
$Sig_5$



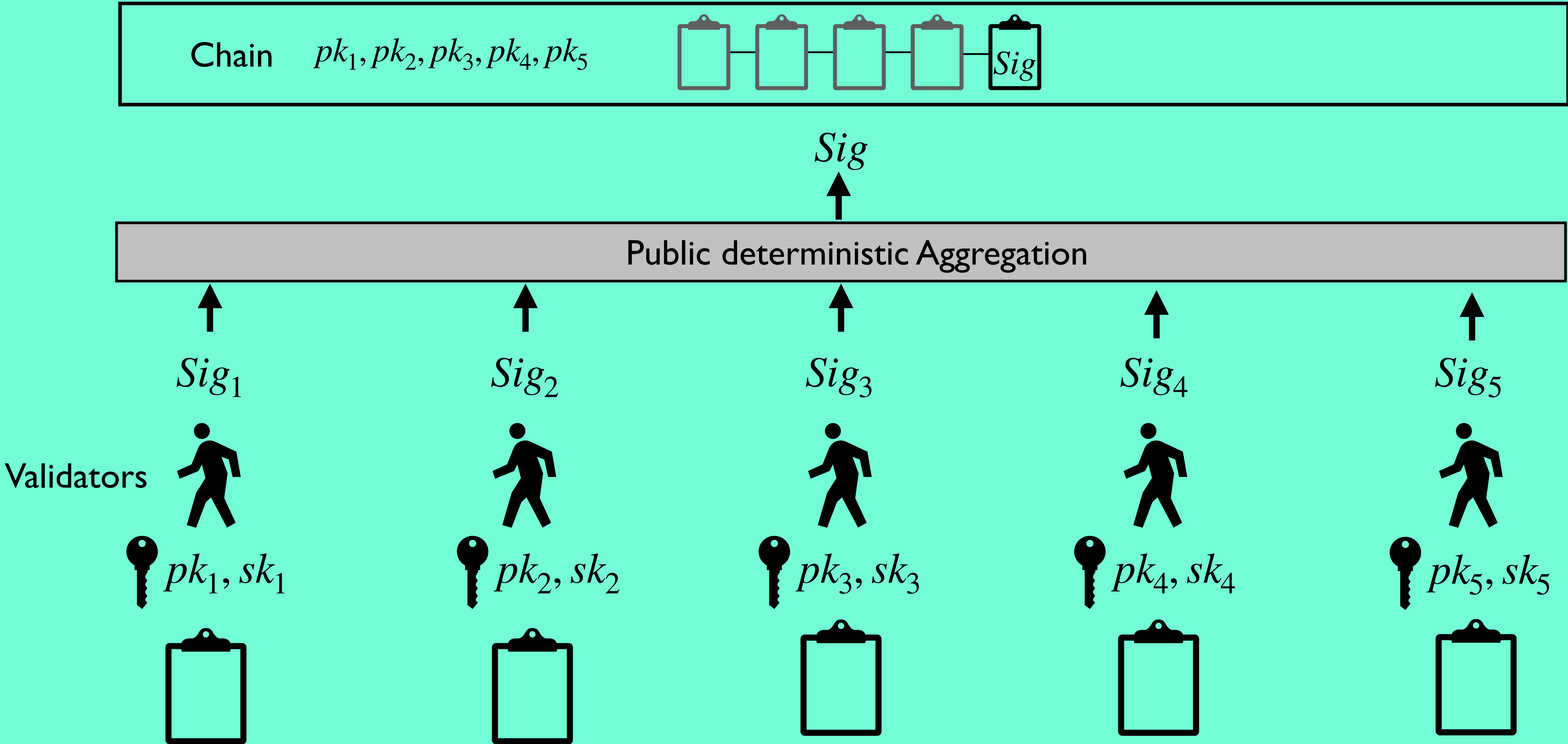
$pk_5, sk_5$



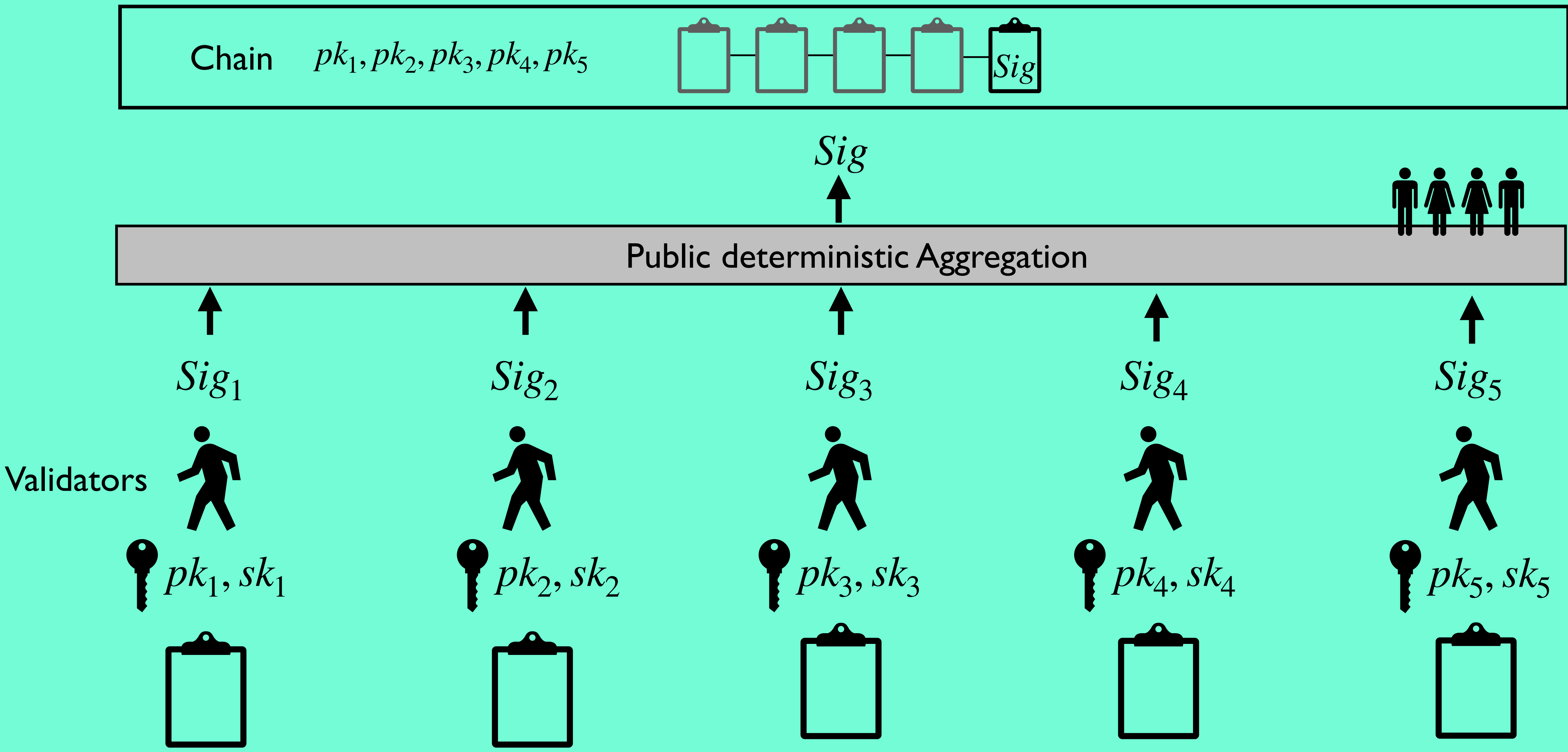
# Multi-Signatures in Ethereum Consensus



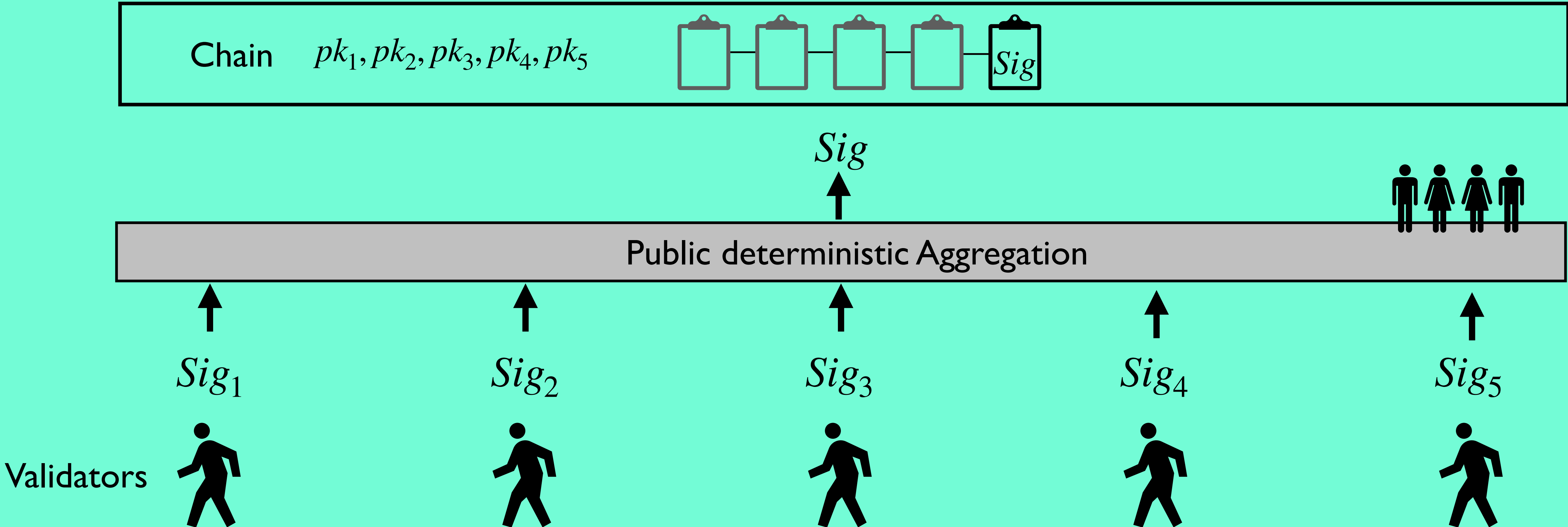
# Multi-Signatures in Ethereum Consensus



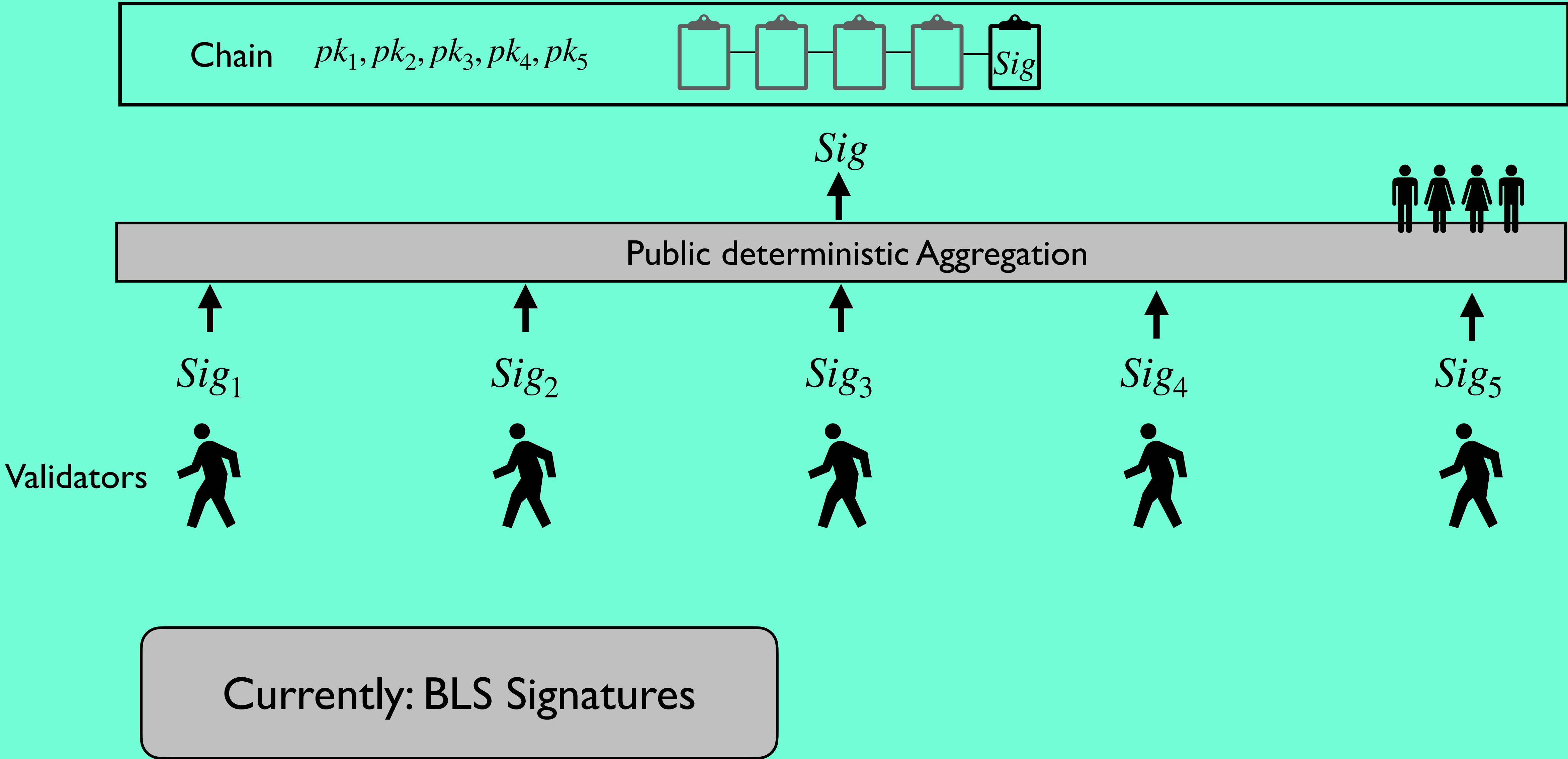
# Multi-Signatures in Ethereum Consensus



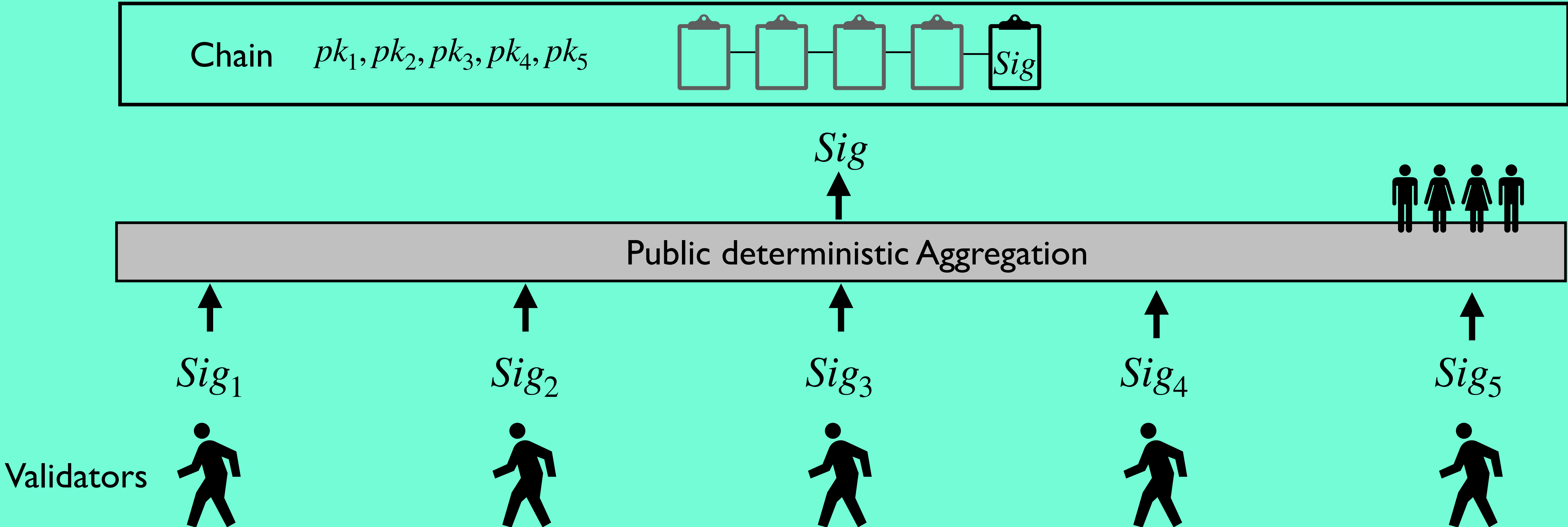
# Multi-Signatures in Ethereum Consensus



# Multi-Signatures in Ethereum Consensus



# Multi-Signatures in Ethereum Consensus



Currently: BLS Signatures

Quantum Insecurity!



# Our Goal

Goal: Post-Quantum Multi-Signatures for Ethereum

# Our Goal

Goal: Post-Quantum Multi-Signatures for Ethereum

Post-Quantum Cryptography

# Our Goal

Goal: Post-Quantum Multi-Signatures for Ethereum

Post-Quantum Cryptography

Lattices and Codes

# Our Goal

Goal: Post-Quantum Multi-Signatures for Ethereum

Post-Quantum Cryptography

Lattices and Codes

Isogenies

Multivariate

# Our Goal

Goal: Post-Quantum Multi-Signatures for Ethereum

Post-Quantum Cryptography

Lattices and Codes

Isogenies

Multivariate

Hash Functions

# Our Goal

Goal: Post-Quantum Multi-Signatures for Ethereum

Post-Quantum Cryptography

Lattices and Codes

Isogenies

Multivariate

Hash Functions

# Outline

The Problem

Overall Paradigm

Signature Design

Next Steps

# Outline

The Problem

Overall Paradigm

Signature Design

Next Steps

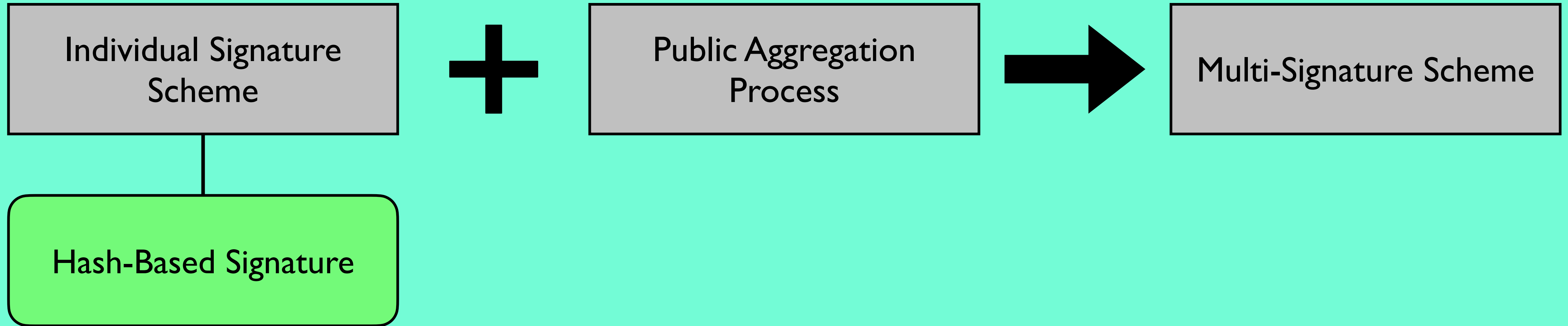


# Paradigm (Folklore)

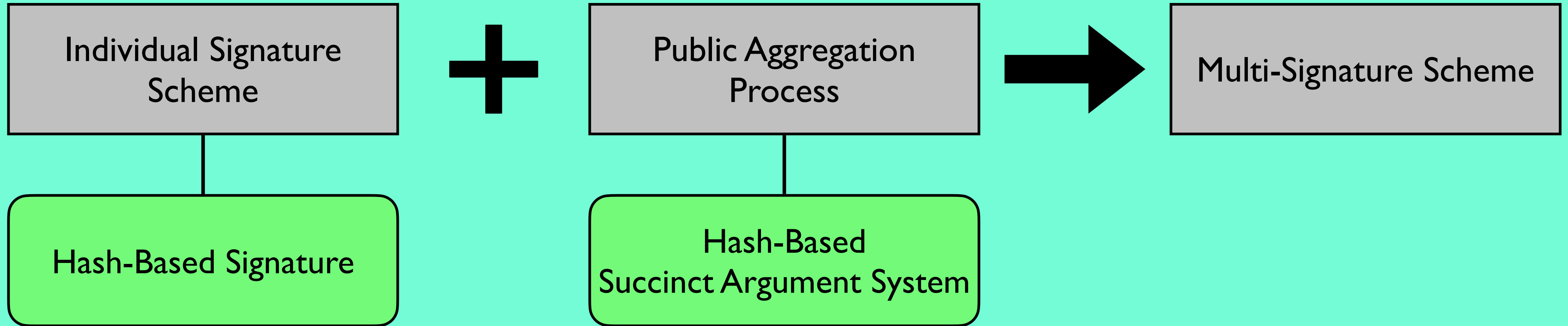
# Paradigm (Folklore)



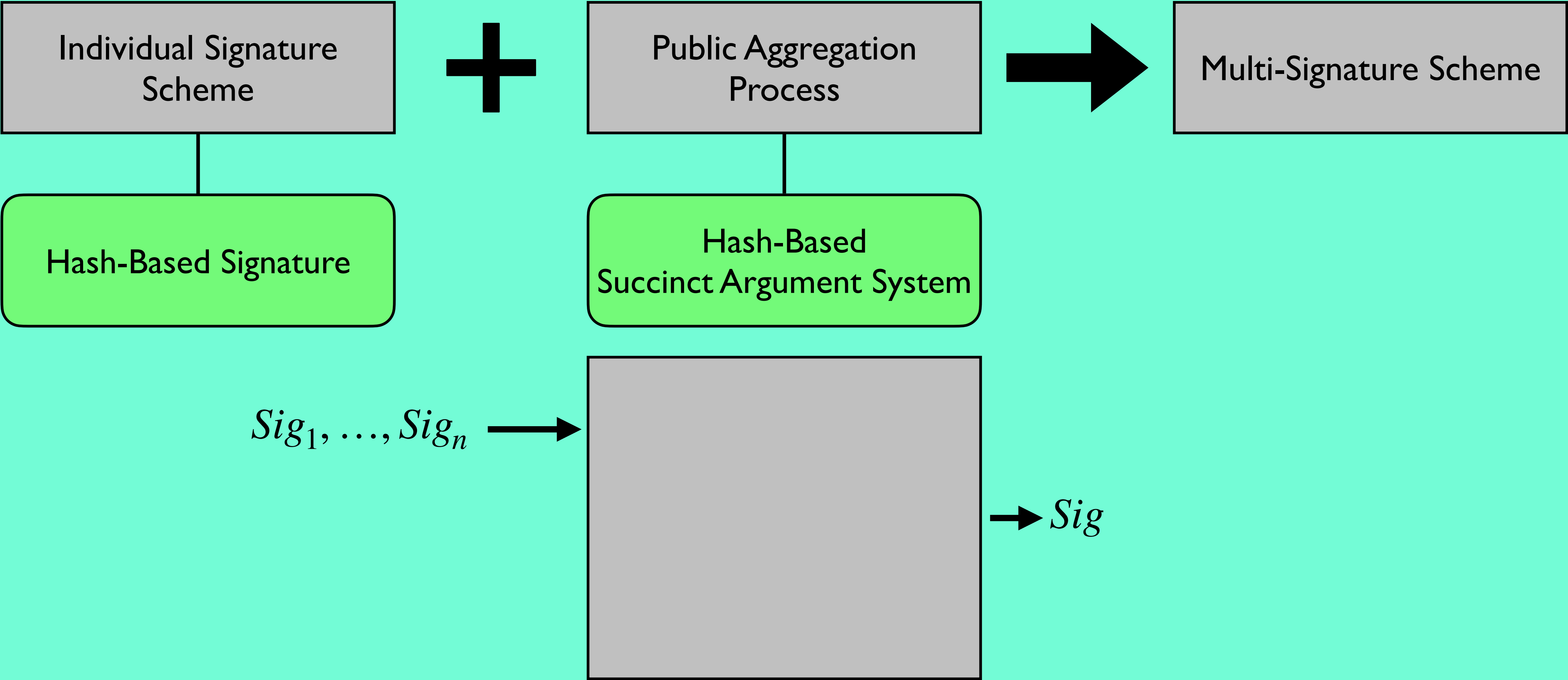
# Paradigm (Folklore)



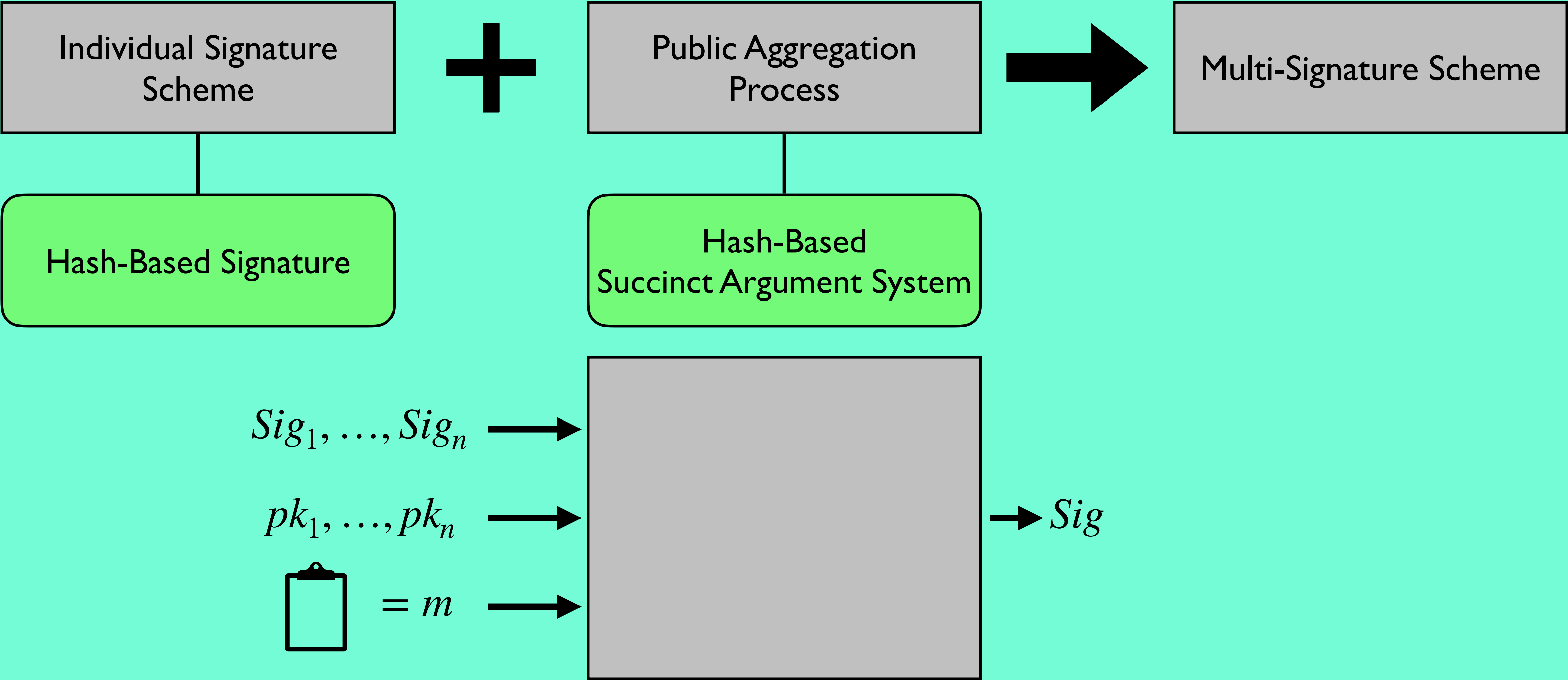
# Paradigm (Folklore)



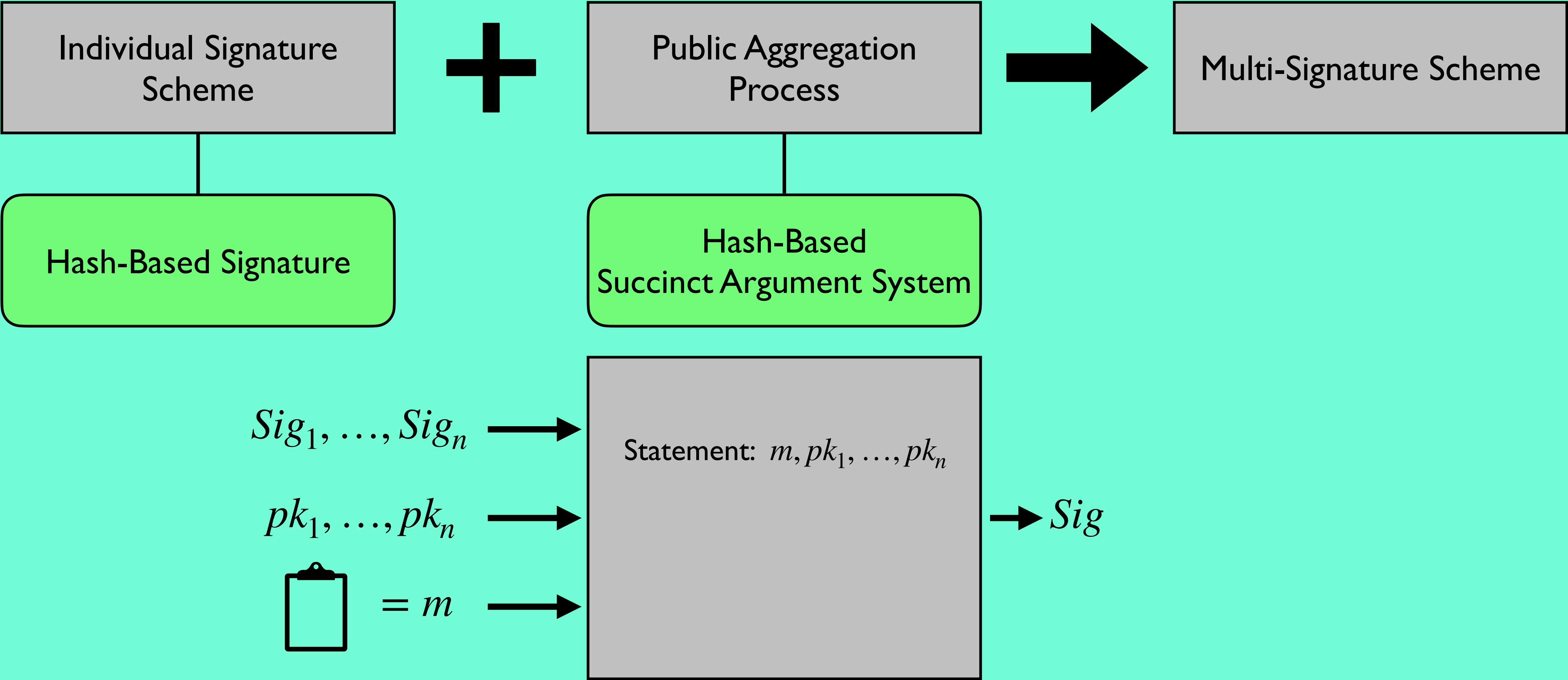
# Paradigm (Folklore)



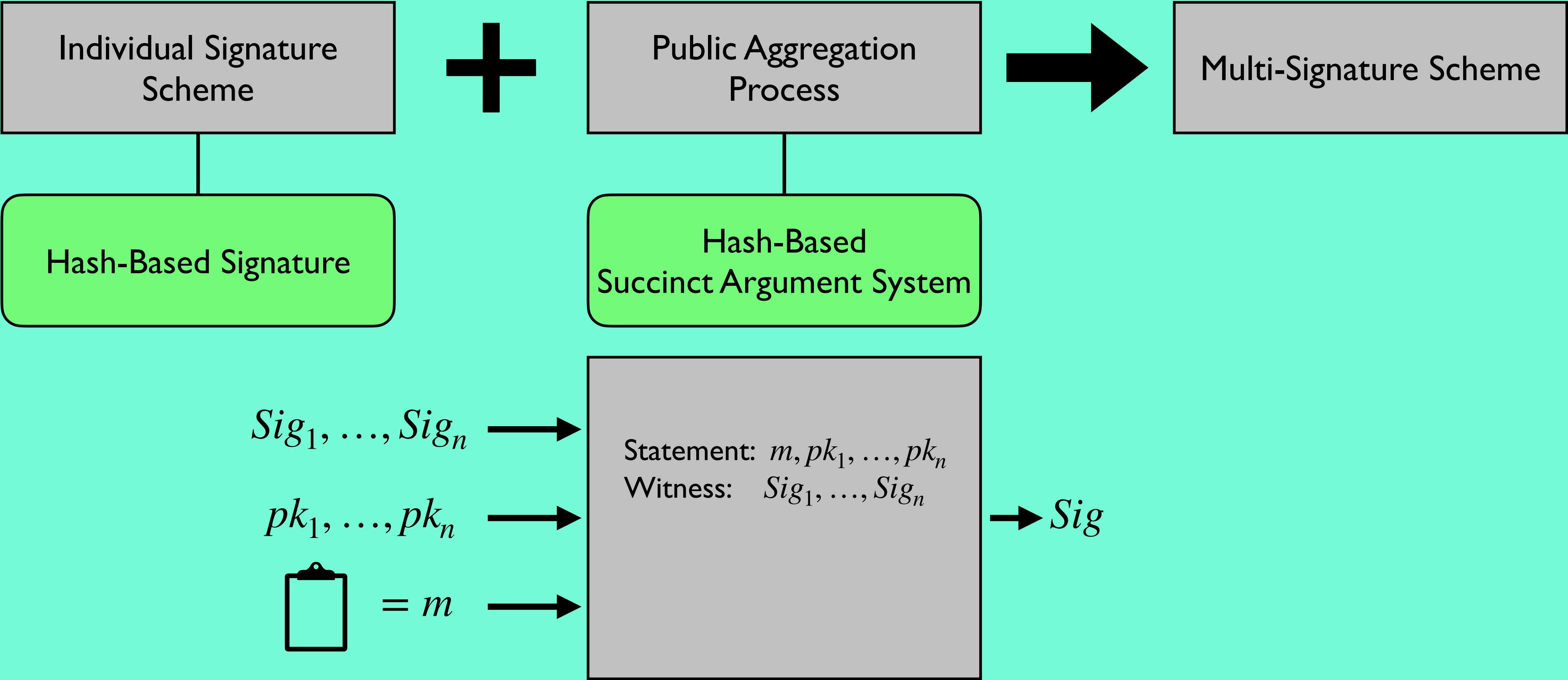
# Paradigm (Folklore)



# Paradigm (Folklore)

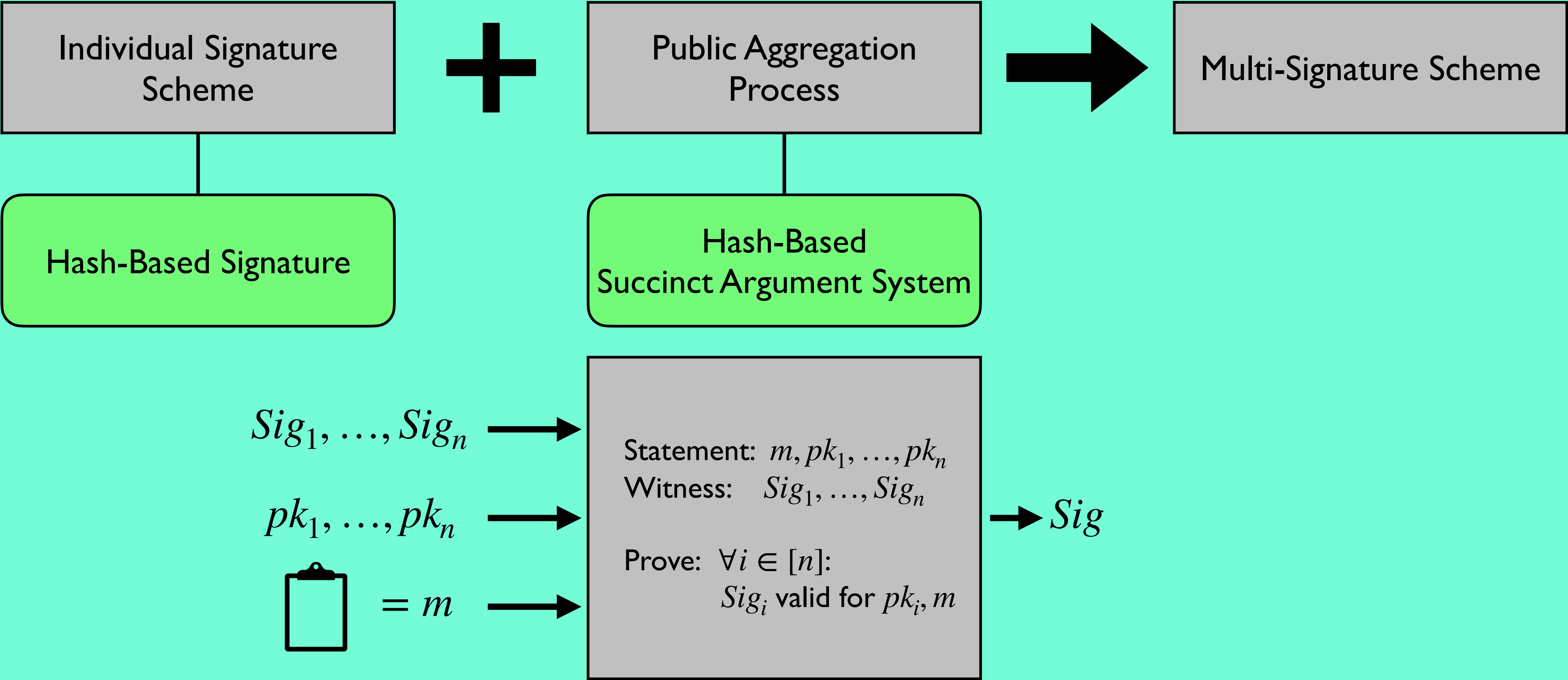


# Paradigm (Folklore)

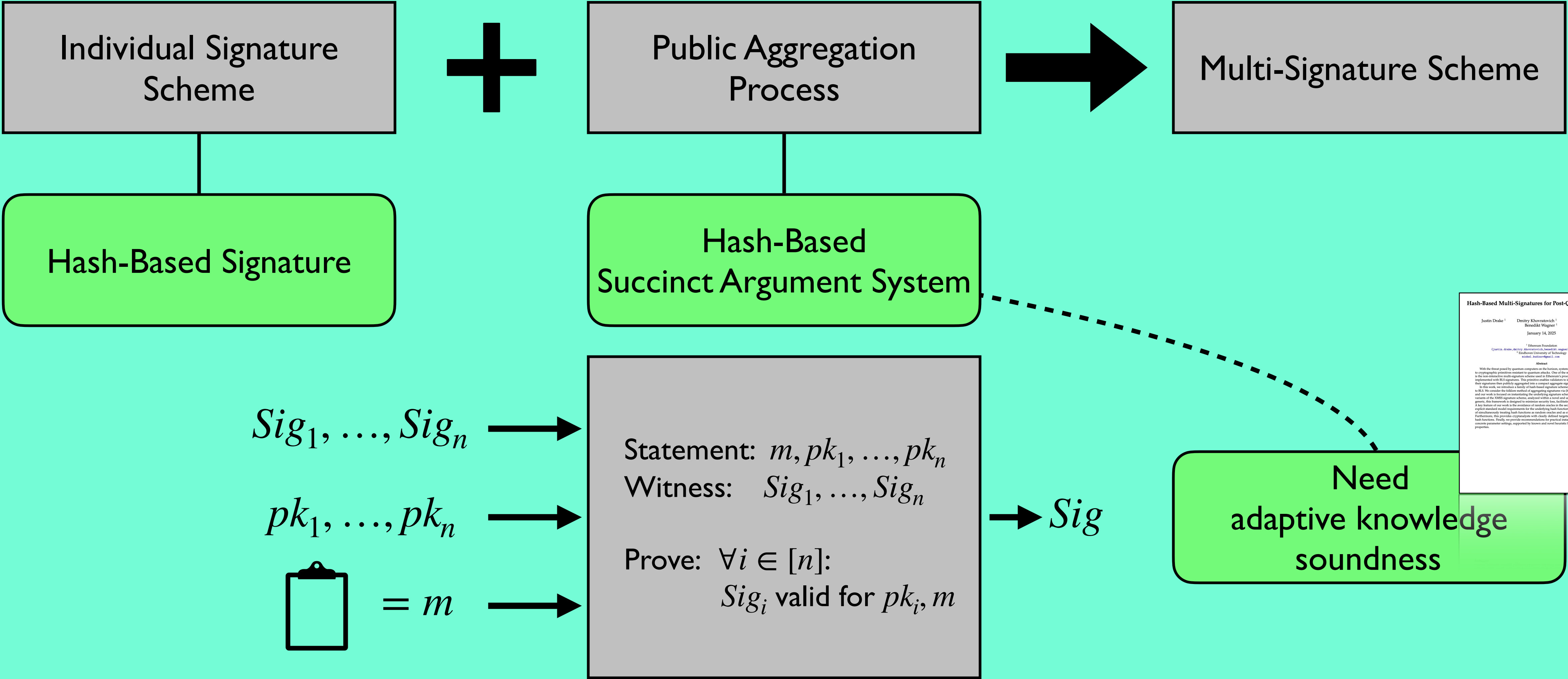




# Paradigm (Folklore)



# Paradigm (Folklore)



Hash-Based Multi-Signatures for Post-Quantum Ethereum

Justin Drake<sup>1</sup> Dmitry Khovratovich<sup>1</sup> Mikhail Kudinov<sup>2</sup>  
Benedikt Wagner<sup>1</sup>  
January 14, 2025

<sup>1</sup> Ethereum Foundation  
Justin.drake, dmitry.khovratovich, benedikt.wagner@ethereum.org  
<sup>2</sup> Eindhoven University of Technology  
mikhail.kudinov@tue.nl

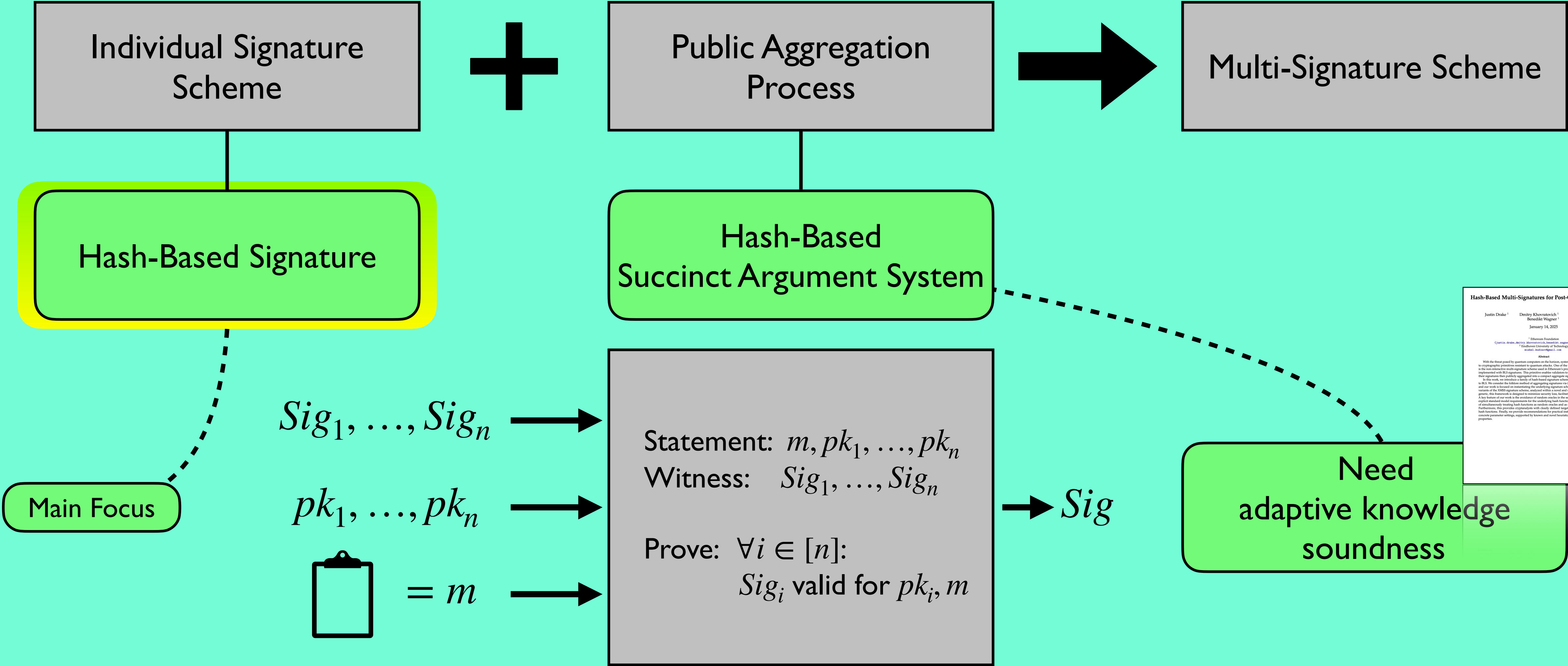
**Abstract**

With the threat posed by quantum computers on the horizon, systems like Ethereum must transition to cryptographically primitive resistant to quantum attack. One of the most critical of these primitives is the hash-based multi-signature scheme used in Ethereum's proof-of-stake consensus, currently implemented with BLS signatures. This primitive enables validators to independently sign blocks, with their signatures then publicly aggregated into a compact aggregate signature.

In this work, we introduce a family of hash-based signature schemes as post-quantum alternatives to BLS. We consider the folklore method of aggregating signatures via hash-based succinct arguments, and our work is based on instantiating the underlying signature scheme. The proposed schemes are variants of the BLS signature scheme, analyzed within a novel and refined framework. While being generic, this framework is designed to maximize security by facilitating efficient parameter selection.

A key feature of our work is the avoidance of random oracles in the security proof. Instead, we define explicit statistical model requirements for the underlying hash functions. This framework for proving of simultaneously meeting both functions as random oracles and as explicit circuits for aggregation. Furthermore, this provides cryptanalysts with clearly defined targets for evaluating the security of hash functions. Finally, we provide recommendations for practical instantiations of hash functions and concrete parameter settings, supported by known and novel heuristic bounds on the standard model properties.

# Paradigm (Folklore)



Hash-Based Multi-Signatures for Post-Quantum Ethereum

Justin Drake<sup>1</sup>   Dmitry Khovratovich<sup>1</sup>   Mikhail Kudinov<sup>2</sup>  
Benedikt Wagner<sup>1</sup>  
January 14, 2025

<sup>1</sup> Ethereum Foundation  
Justin.drake, dmitry.khovratovich, benedikt.wagner@ethereum.org  
<sup>2</sup> Eindhoven University of Technology  
mikhail.kudinov@tue.nl

**Abstract**

With the threat posed by quantum computers on the horizon, systems like Ethereum must transition to cryptographically primitive resistant to quantum attack. One of the most critical of these primitives is the hash-based multi-signature scheme used in Ethereum's proof-of-stake consensus, currently implemented with BLS signatures. This primitive enables validators to independently sign blocks, with their signatures then publicly aggregated into a compact aggregate signature.

In this work, we introduce a family of hash-based signature schemes as post-quantum alternatives to BLS. We consider the folklore method of aggregating signatures via hash-based vector equations, and our work is based on instantiating the underlying signature scheme. The proposed schemes are variants of the Schnorr signature scheme, analyzed within a novel and refined framework. While being generic, this framework is designed to maximize security by facilitating efficient parameter selection. A key feature of our work is the avoidance of random oracles in the security proof. Instead, we define explicit random oracle responses for the underlying hash functions. This framework is proven to simultaneously meeting both functions as random oracles and as explicit circuits for aggregation. Furthermore, this provides cryptanalysts with clearly defined targets for evaluating the security of hash functions. Finally, we provide recommendations for practical instantiations of hash functions and concrete parameter settings, supported by known and novel heuristic bounds on the standard model properties.

# Signature Requirements

Goal: Post-Quantum Multi-Signatures for Ethereum

# Signature Requirements

Goal: Post-Quantum Multi-Signatures for Ethereum

Aggregation-Friendly Hash-Based Signatures

# Signature Requirements

Goal: Post-Quantum Multi-Signatures for Ethereum

Aggregation-Friendly Hash-Based Signatures

Small Signatures

```
graph TD; A[Goal: Post-Quantum Multi-Signatures for Ethereum] --> B[Aggregation-Friendly Hash-Based Signatures]; B --> C[Small Signatures];
```

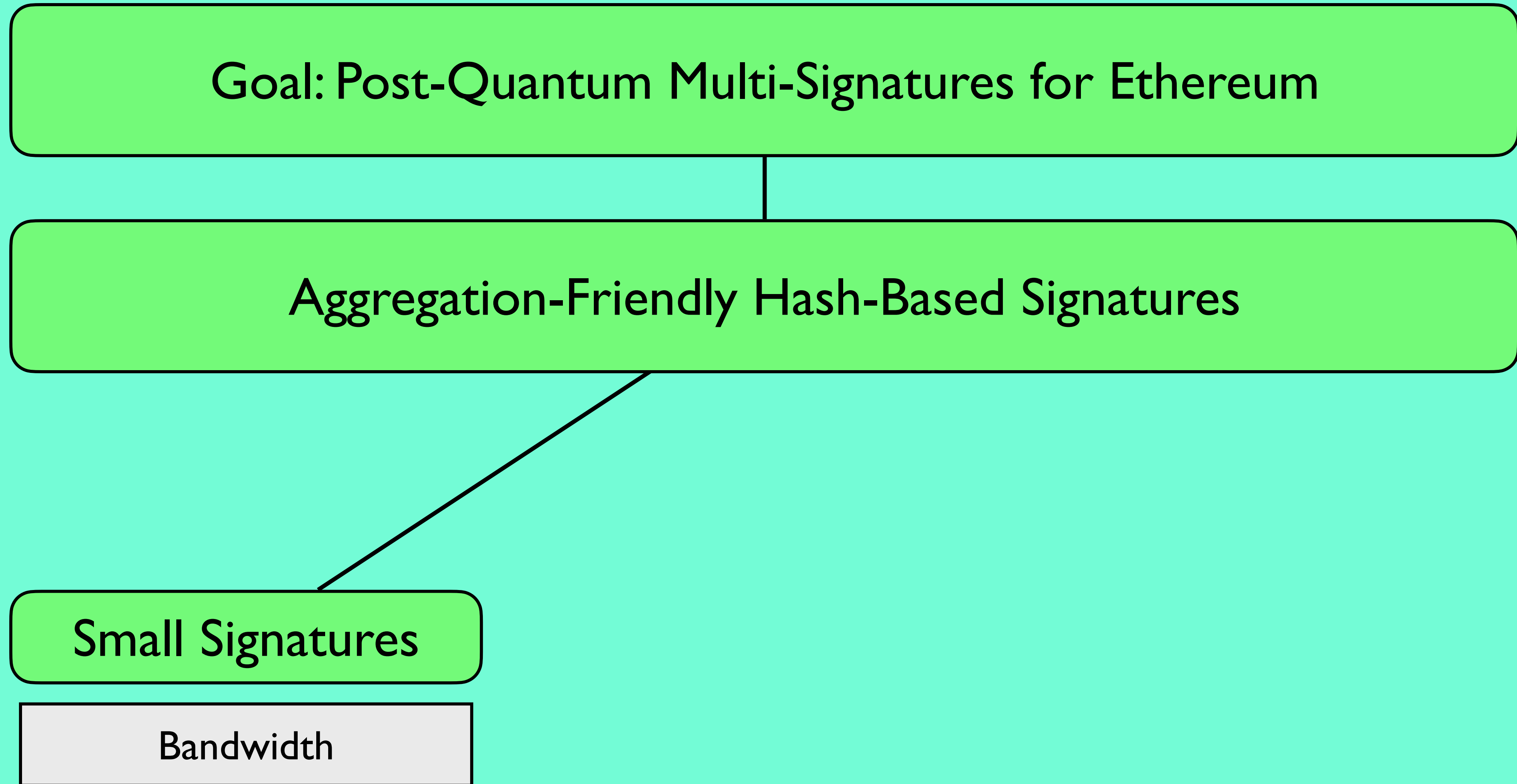
# Signature Requirements

Goal: Post-Quantum Multi-Signatures for Ethereum

Aggregation-Friendly Hash-Based Signatures

Small Signatures

Bandwidth



# Signature Requirements

Goal: Post-Quantum Multi-Signatures for Ethereum

Aggregation-Friendly Hash-Based Signatures

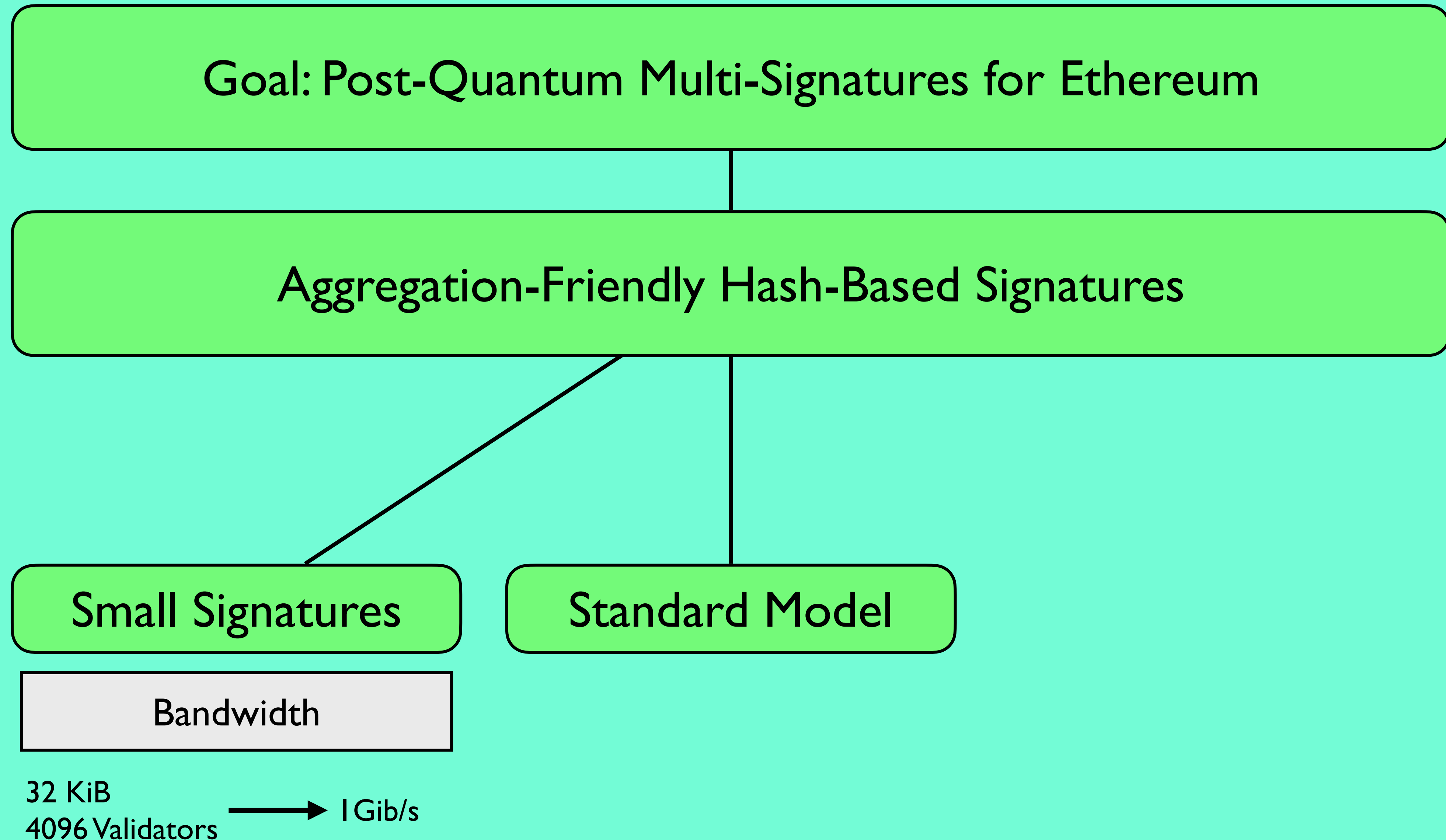
Small Signatures

Bandwidth

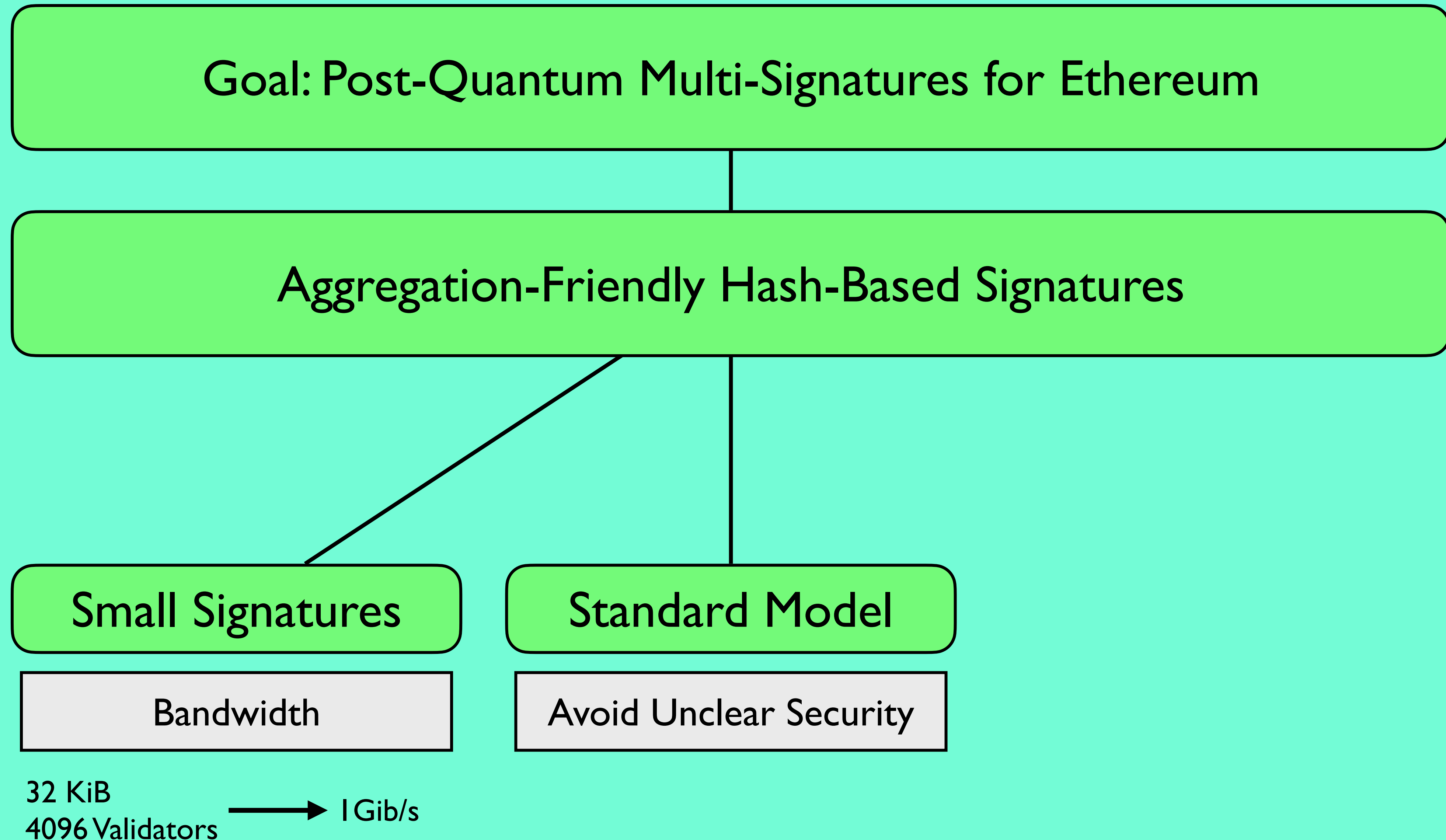
32 KiB  
4096 Validators → 1 Gib/s



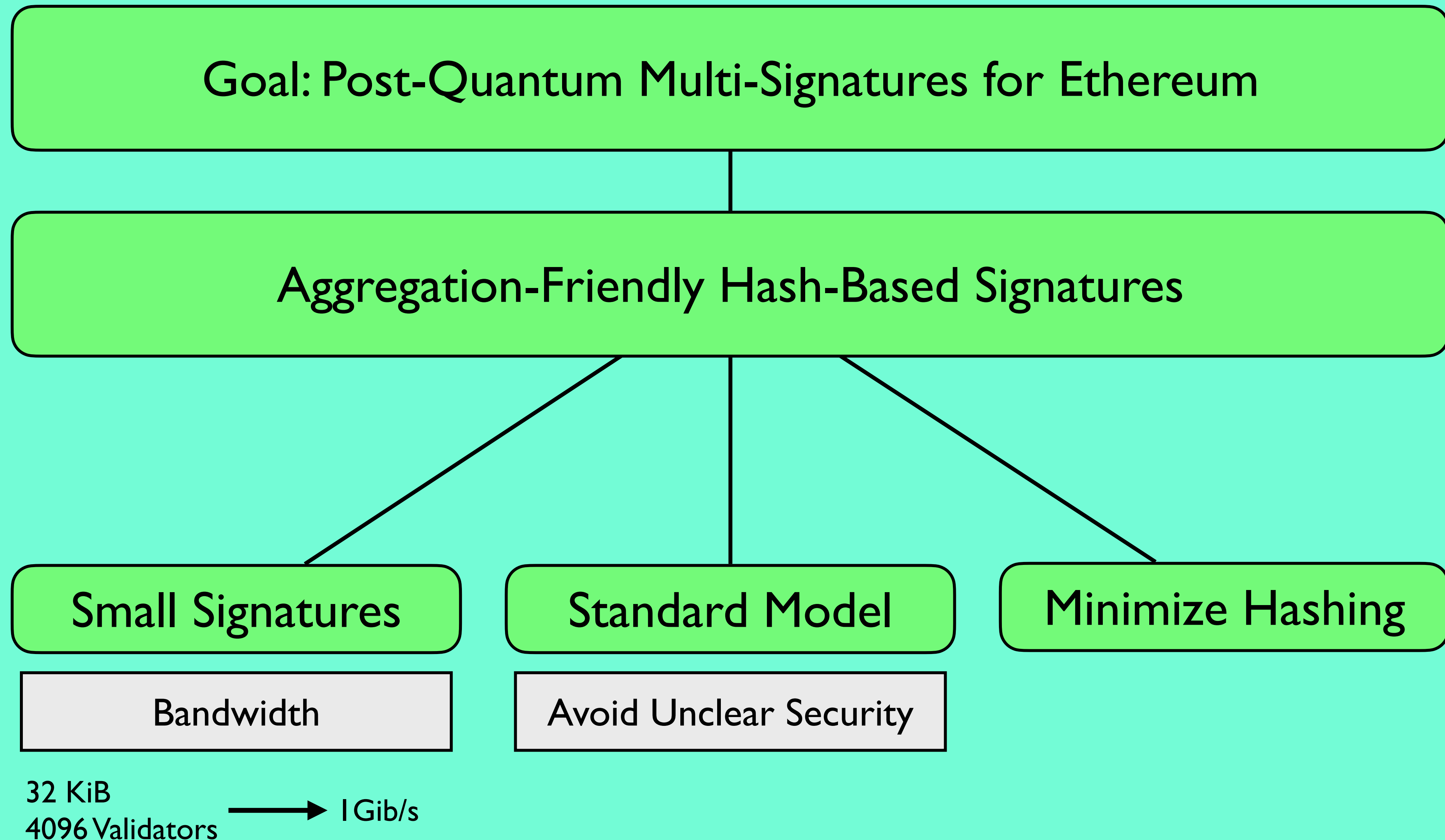
# Signature Requirements



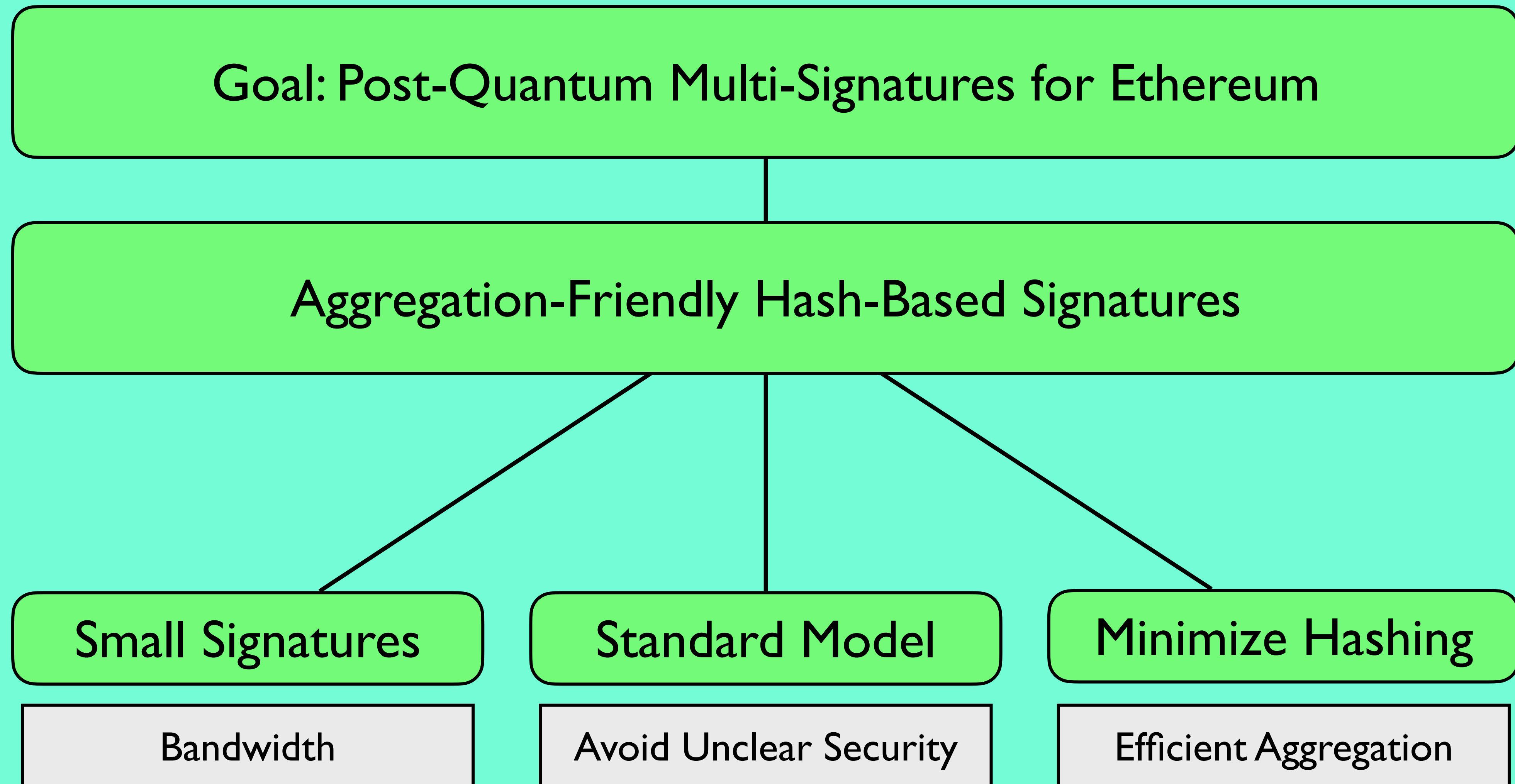
# Signature Requirements



# Signature Requirements



# Signature Requirements



32 KiB  
4096 Validators → 1 Gib/s

# Outline

The Problem

Overall Paradigm

Signature Design

Next Steps

# Outline

The Problem

Overall Paradigm

Signature Design

Next Steps

# Existing Hash-Based Schemes

Scheme	Input	Output	Properties
MD5	Variable length	128 bits	Fast, widely used, but vulnerable to collisions
SHA-1	Variable length	160 bits	More secure than MD5, but still vulnerable to collisions
SHA-256	Variable length	256 bits	Secure, widely used in Bitcoin and other cryptocurrencies
SHA-384	Variable length	384 bits	Secure, used in some government and military applications
SHA-512	Variable length	512 bits	Secure, used in some government and military applications
RIPEMD-160	Variable length	160 bits	Secure, used in some cryptographic protocols
Whirlpool	Variable length	512 bits	Secure, used in some cryptographic protocols
Grindr	Variable length	128 bits	Secure, used in some cryptographic protocols
SHA-224	Variable length	224 bits	Secure, used in some cryptographic protocols
SHA-256/224	Variable length	224 bits	Secure, used in some cryptographic protocols

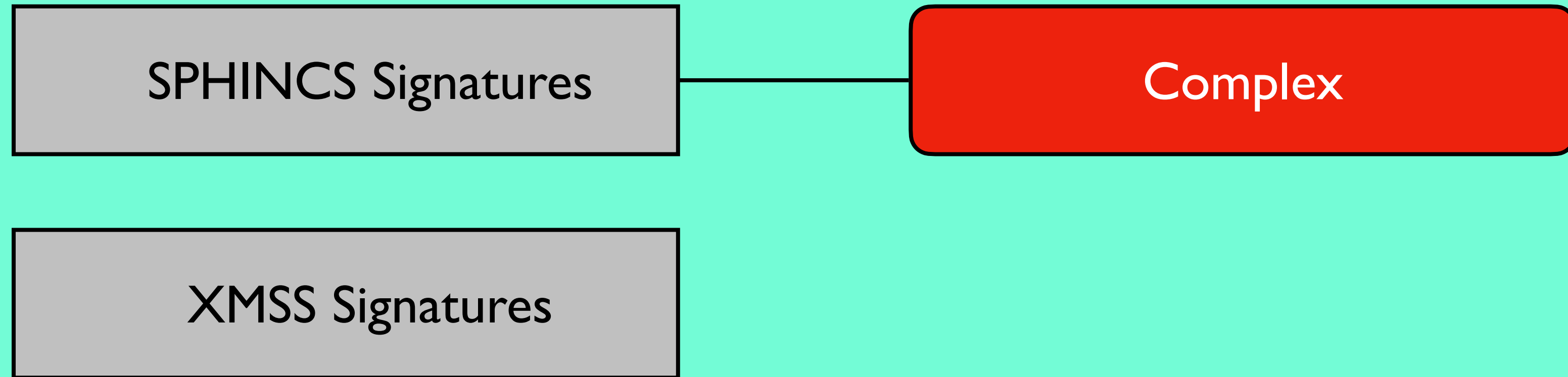
# Existing Hash-Based Schemes

SPHINCS Signatures

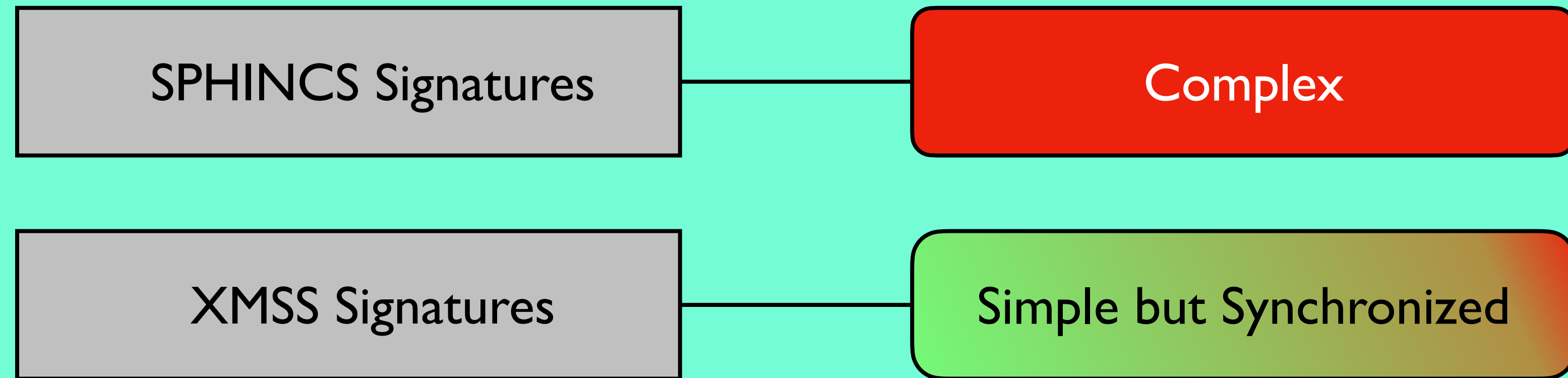
XMSS Signatures



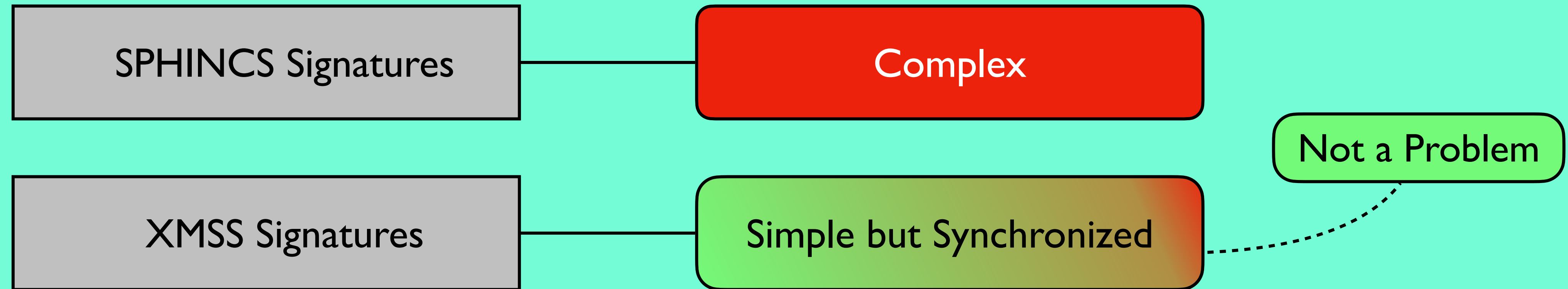
# Existing Hash-Based Schemes



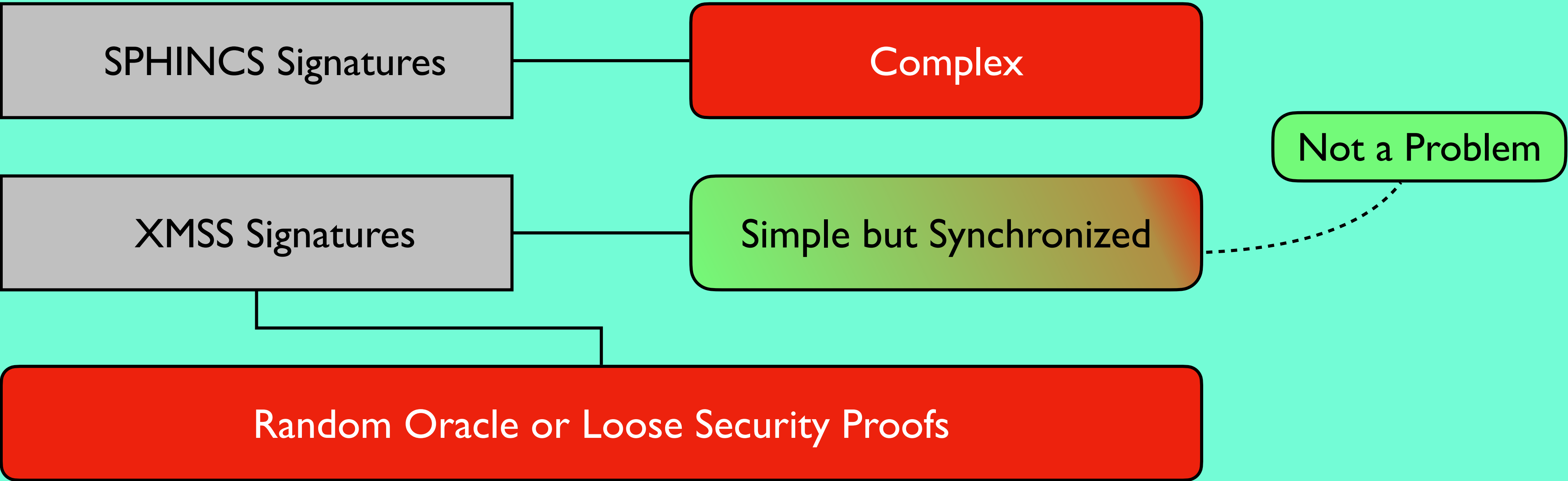
# Existing Hash-Based Schemes



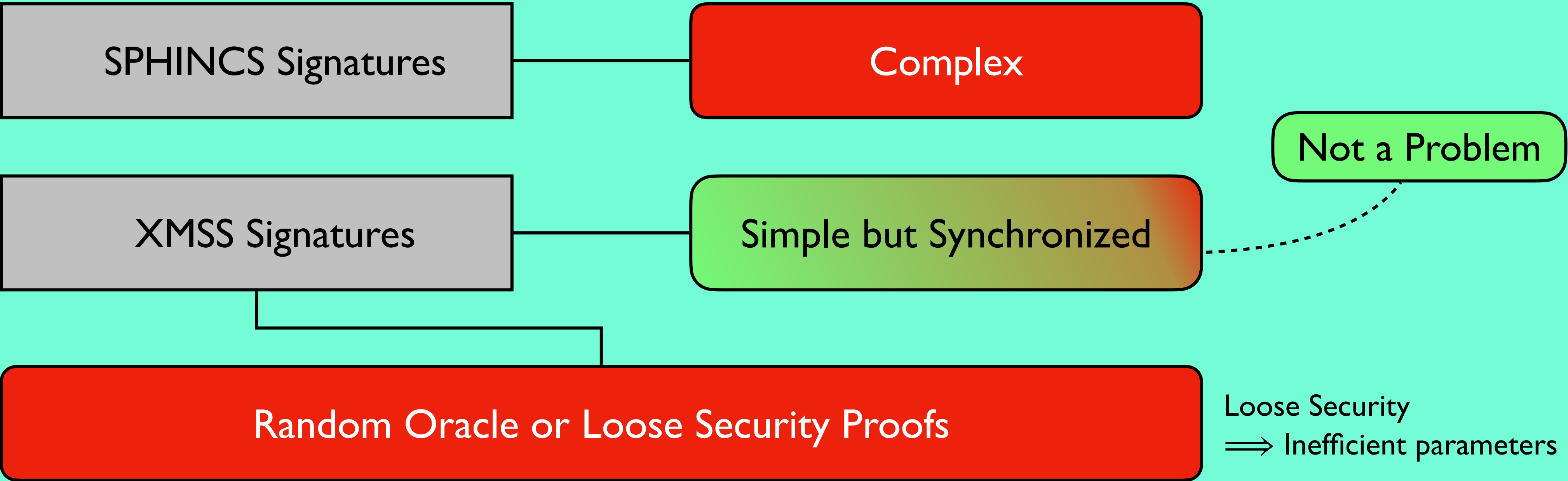
# Existing Hash-Based Schemes



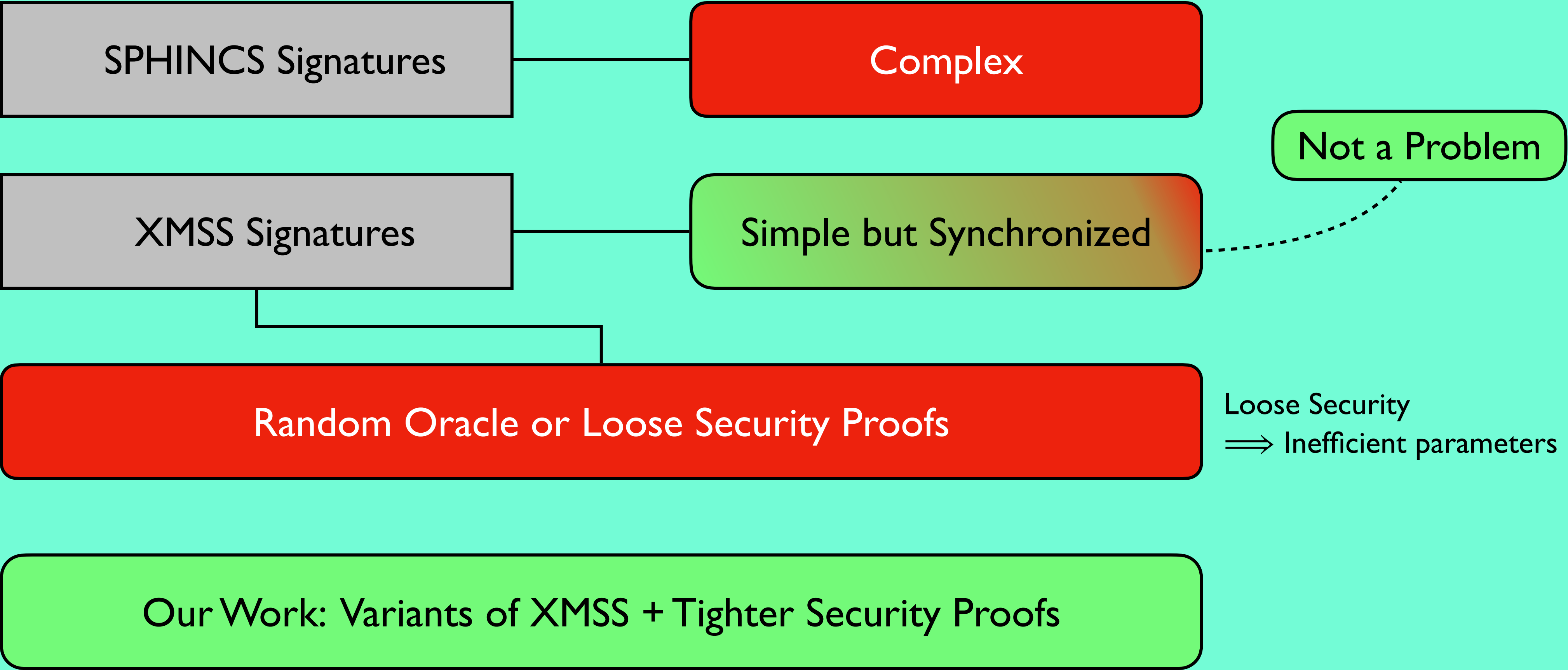
# Existing Hash-Based Schemes



# Existing Hash-Based Schemes



# Existing Hash-Based Schemes



# The XMSS Approach

• **Efficient** (small signatures)

• **Secure** (proven security)

• **Simple** (easy to implement)

• **Scalable** (works for large datasets)

• **Flexible** (works for many applications)

• **Fast** (quick to generate and verify)

• **Robust** (works in many environments)

• **Secure** (proven security)

• **Simple** (easy to implement)

# The XMSS Approach

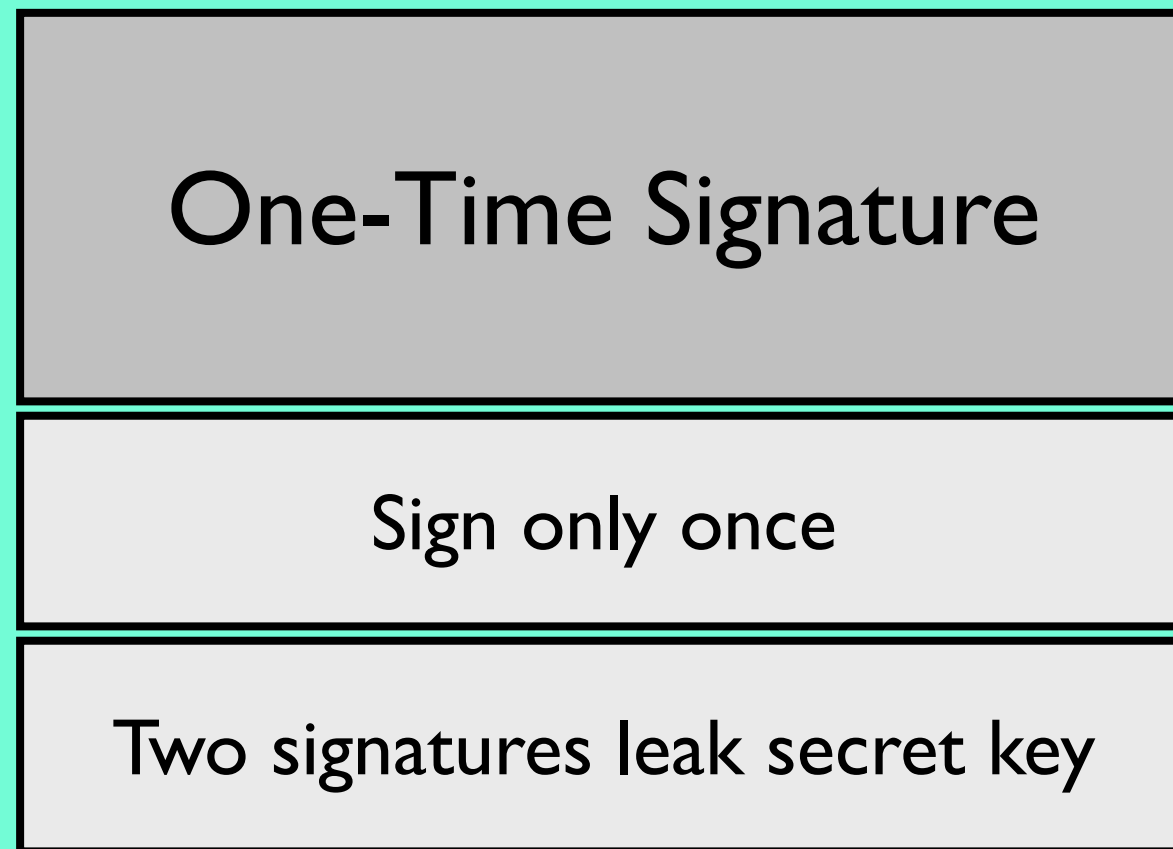


One-Time Signature

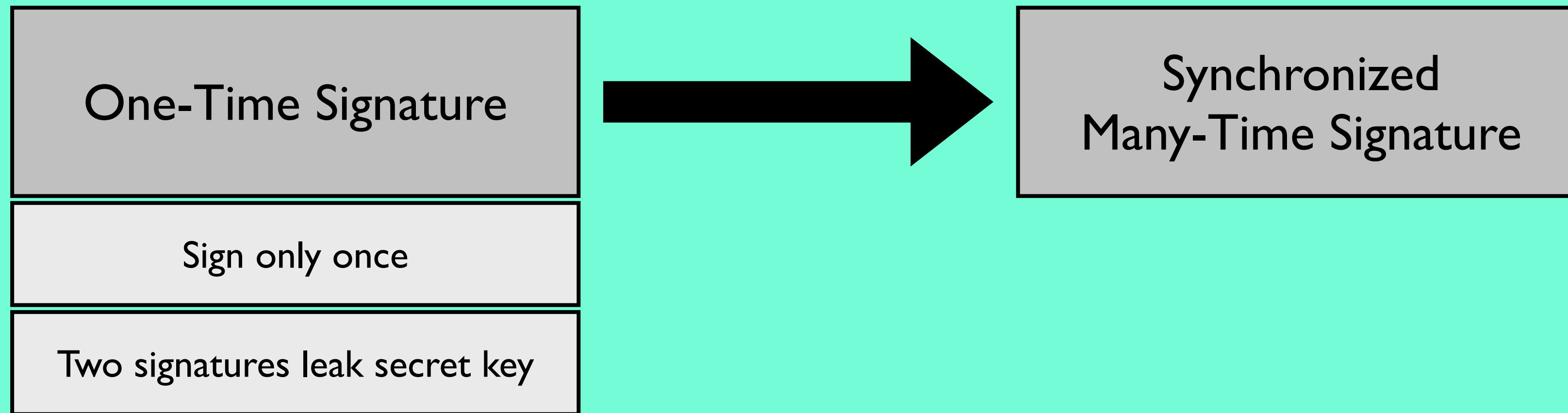
Commitment



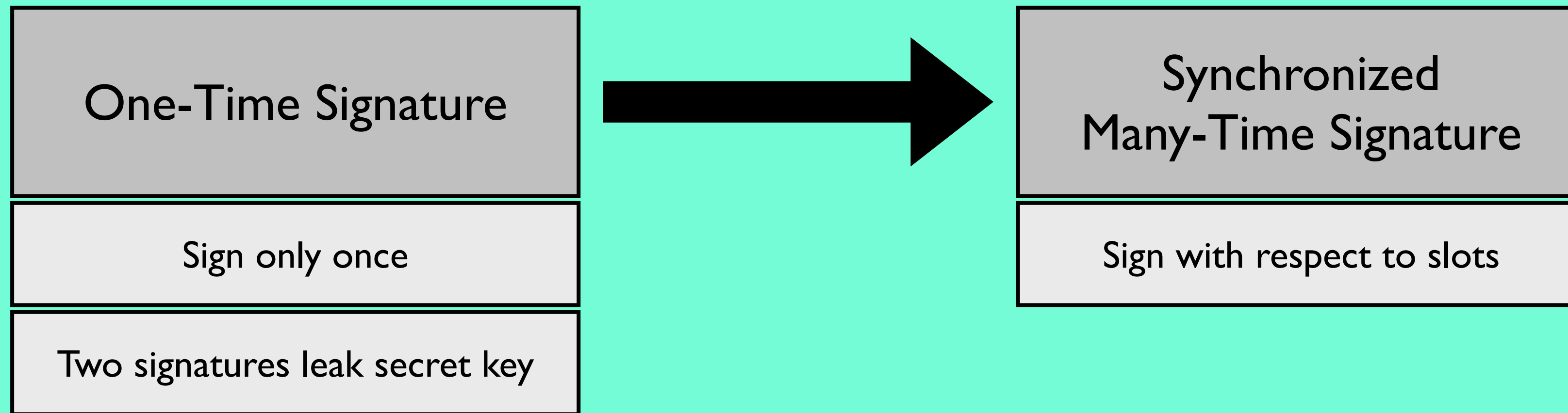
# The XMSS Approach



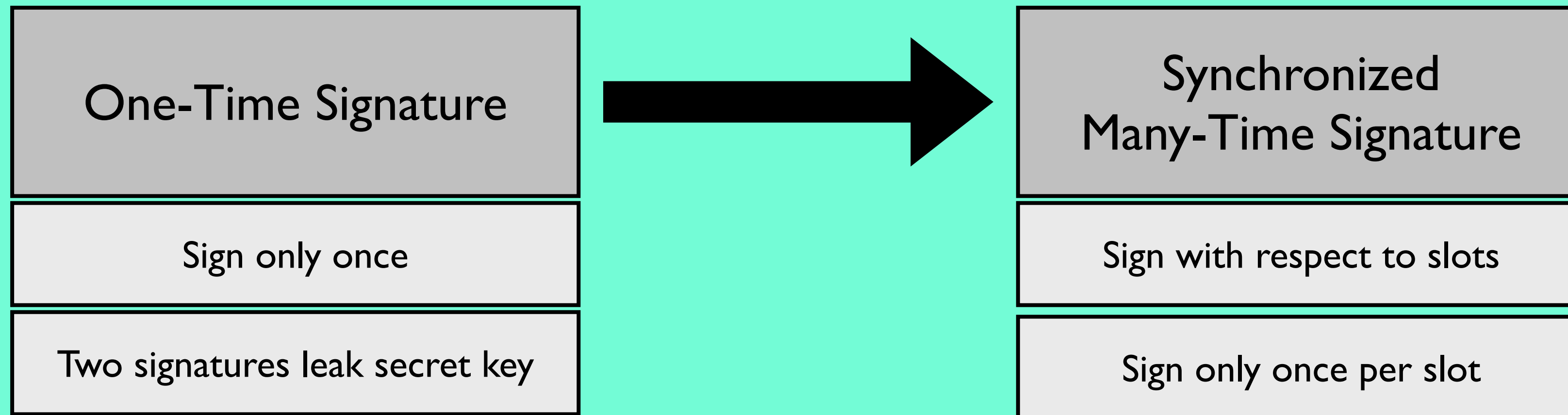
# The XMSS Approach



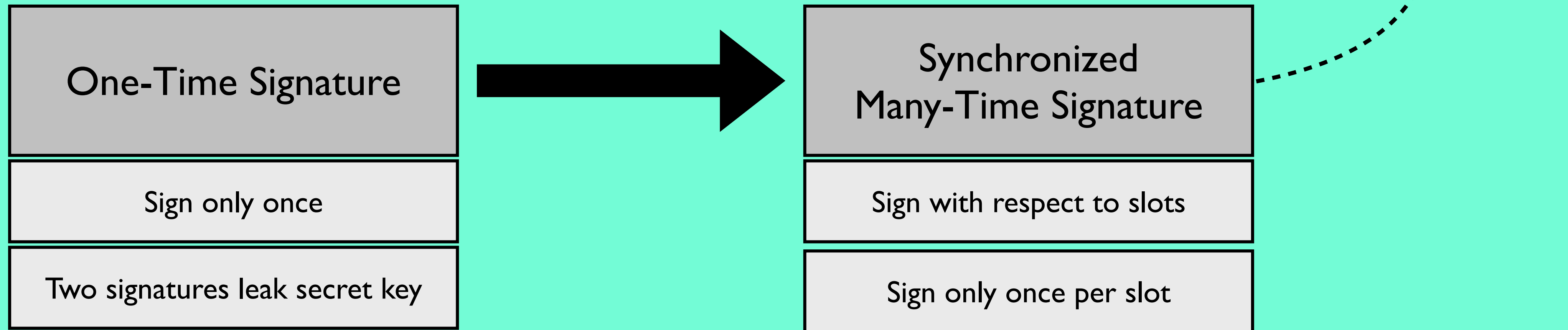
# The XMSS Approach



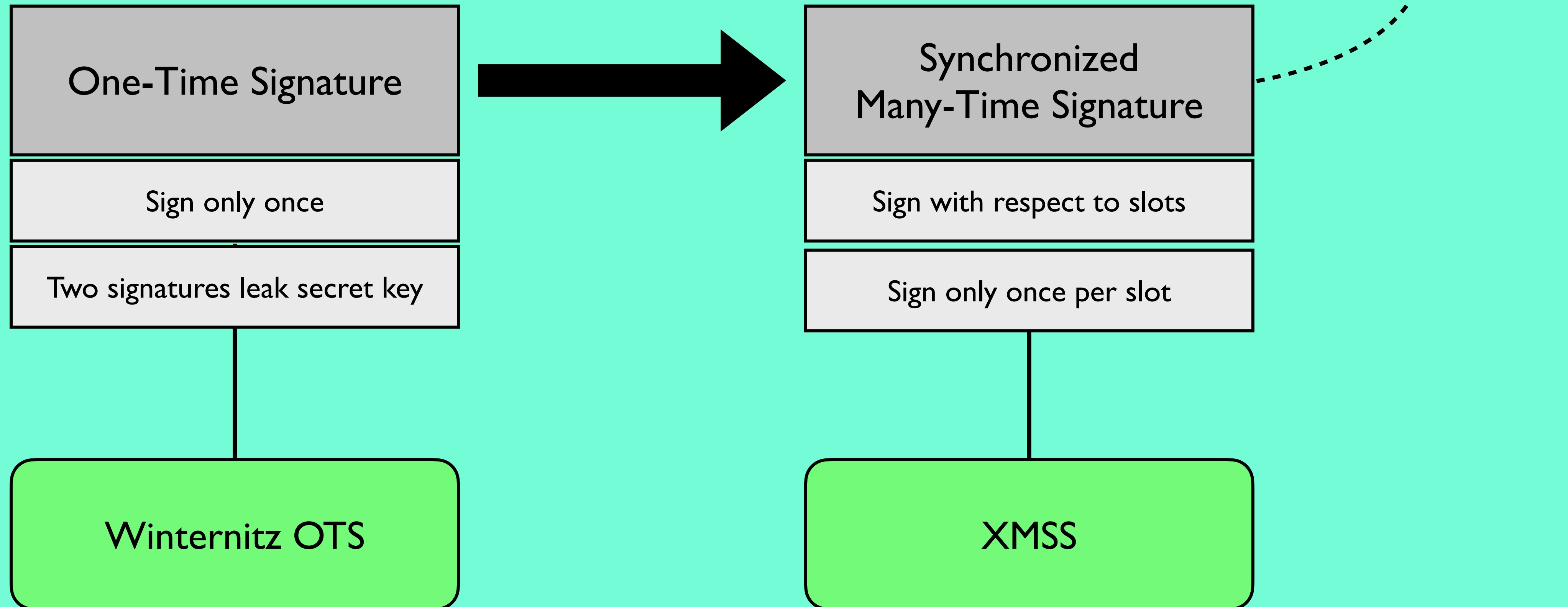
# The XMSS Approach



# The XMSS Approach



# The XMSS Approach



# From One-Time Signing to Many-Time Signing

# From One-Time Signing to Many-Time Signing

$L$  = Key lifetime



# From One-Time Signing to Many-Time Signing

$L$  = Key lifetime

$pk_1^{\text{OTS}}$

...

$pk_L^{\text{OTS}}$

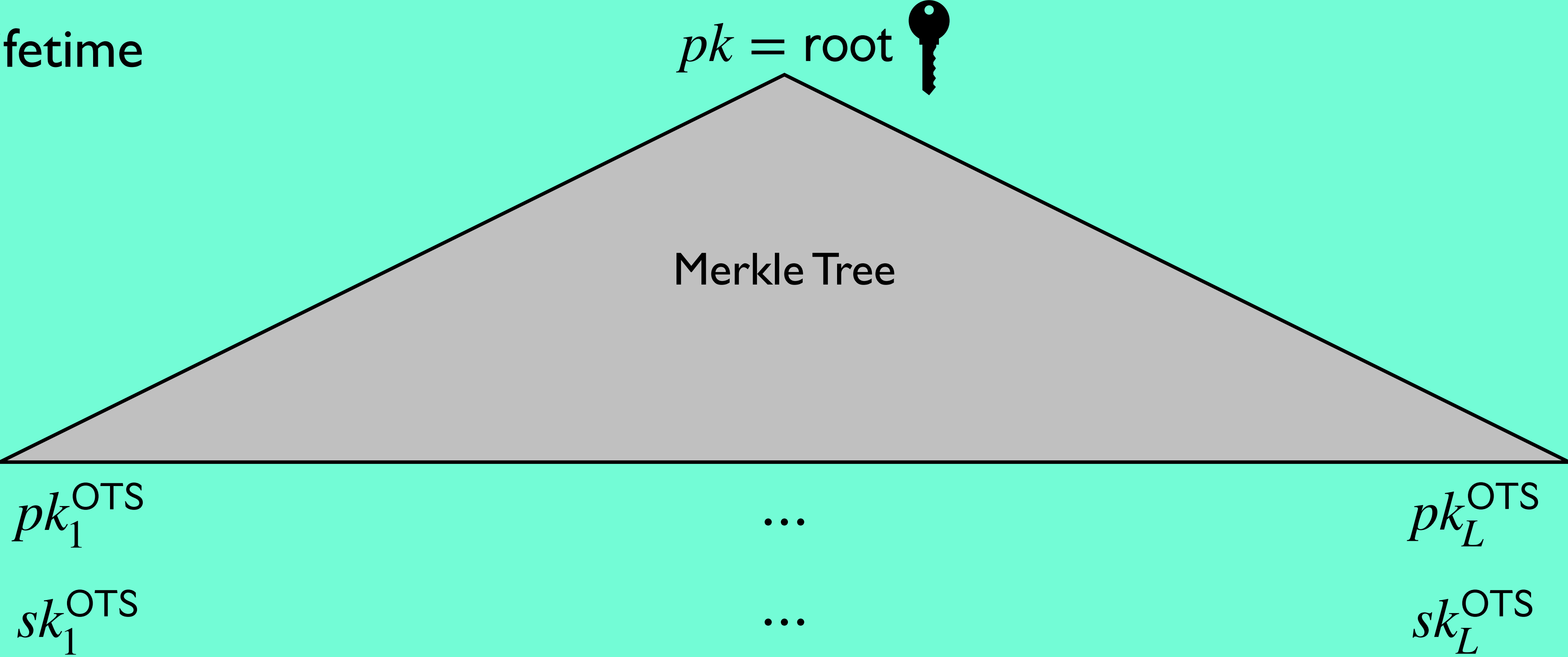
$sk_1^{\text{OTS}}$

...

$sk_L^{\text{OTS}}$

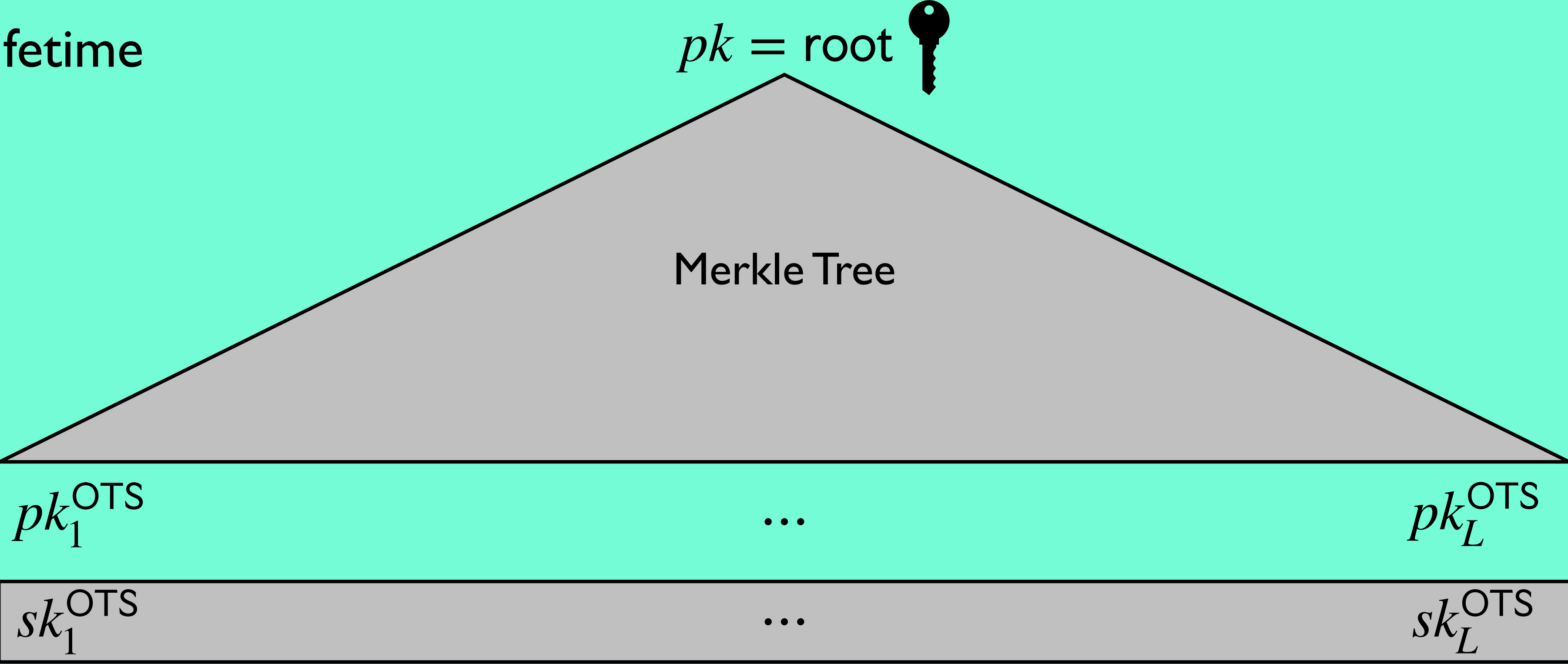
# From One-Time Signing to Many-Time Signing

$L$  = Key lifetime



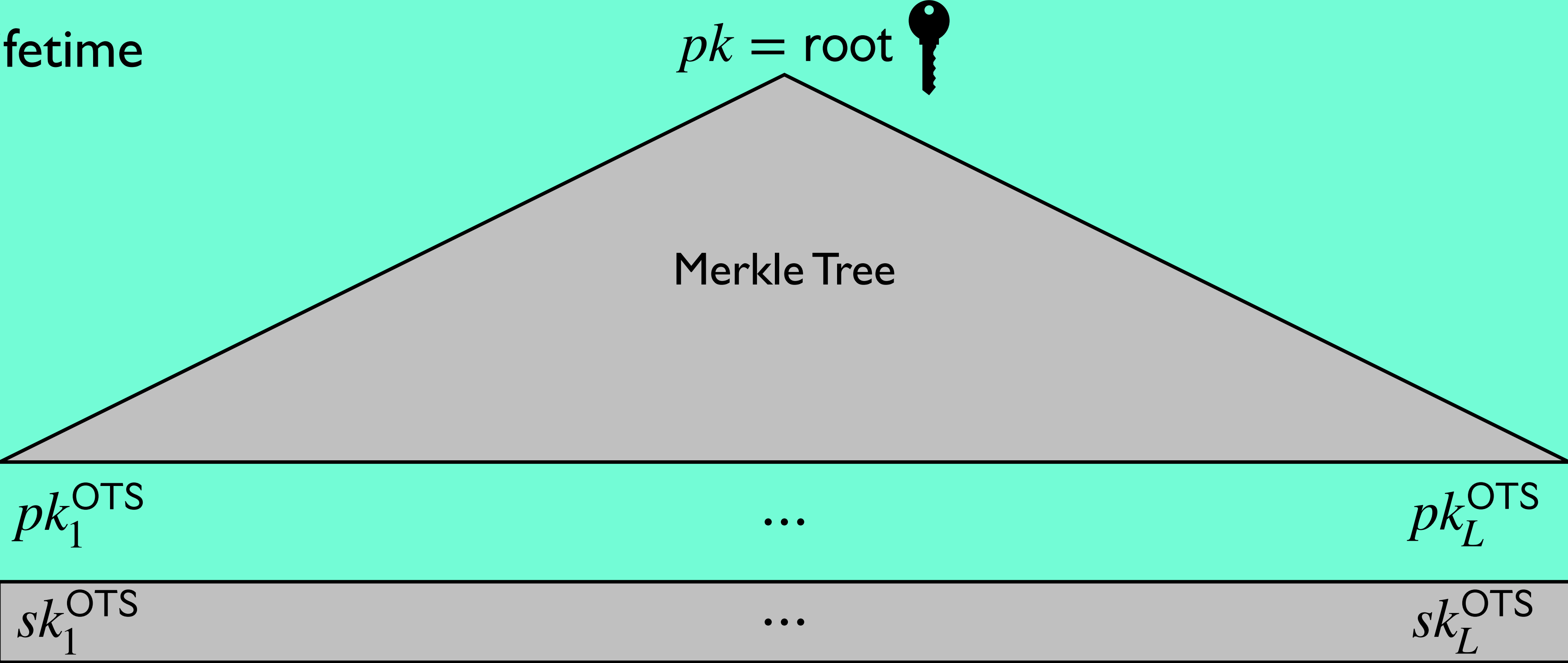
# From One-Time Signing to Many-Time Signing

$L$  = Key lifetime



# From One-Time Signing to Many-Time Signing

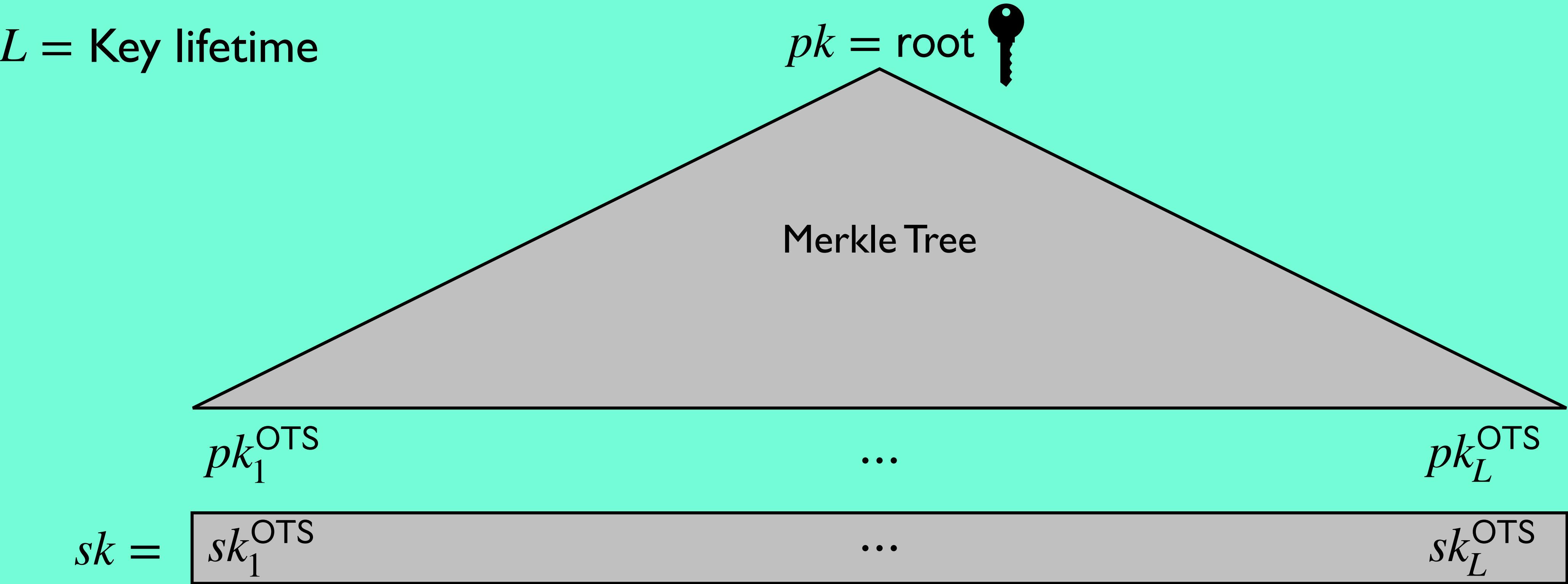
$L$  = Key lifetime



**Signing message  $m$  wrt slot  $i$**

1. OT Public Key  $pk_i^{\text{OTS}}$
2. Merkle Path
3. OT Sig  $\text{Sig}(sk_i^{\text{OTS}}, m)$

# From One-Time Signing to Many-Time Signing



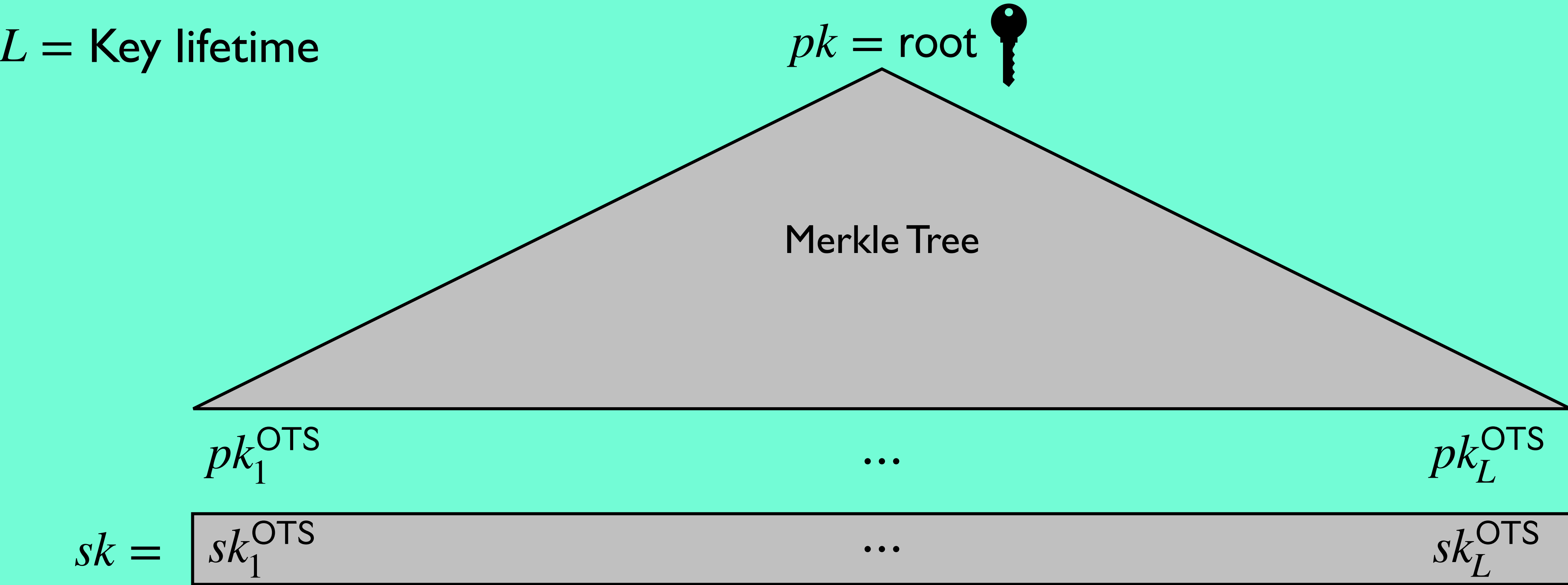
## Signing message $m$ wrt slot $i$

1. OT Public Key  $pk_i^{\text{OTS}}$
2. Merkle Path
3. OT Sig  $\text{Sig}(sk_i^{\text{OTS}}, m)$

## Verification

1. Check Merkle Path
2. Check OT Sig wrt to OT Public Key

# From One-Time Signing to Many-Time Signing



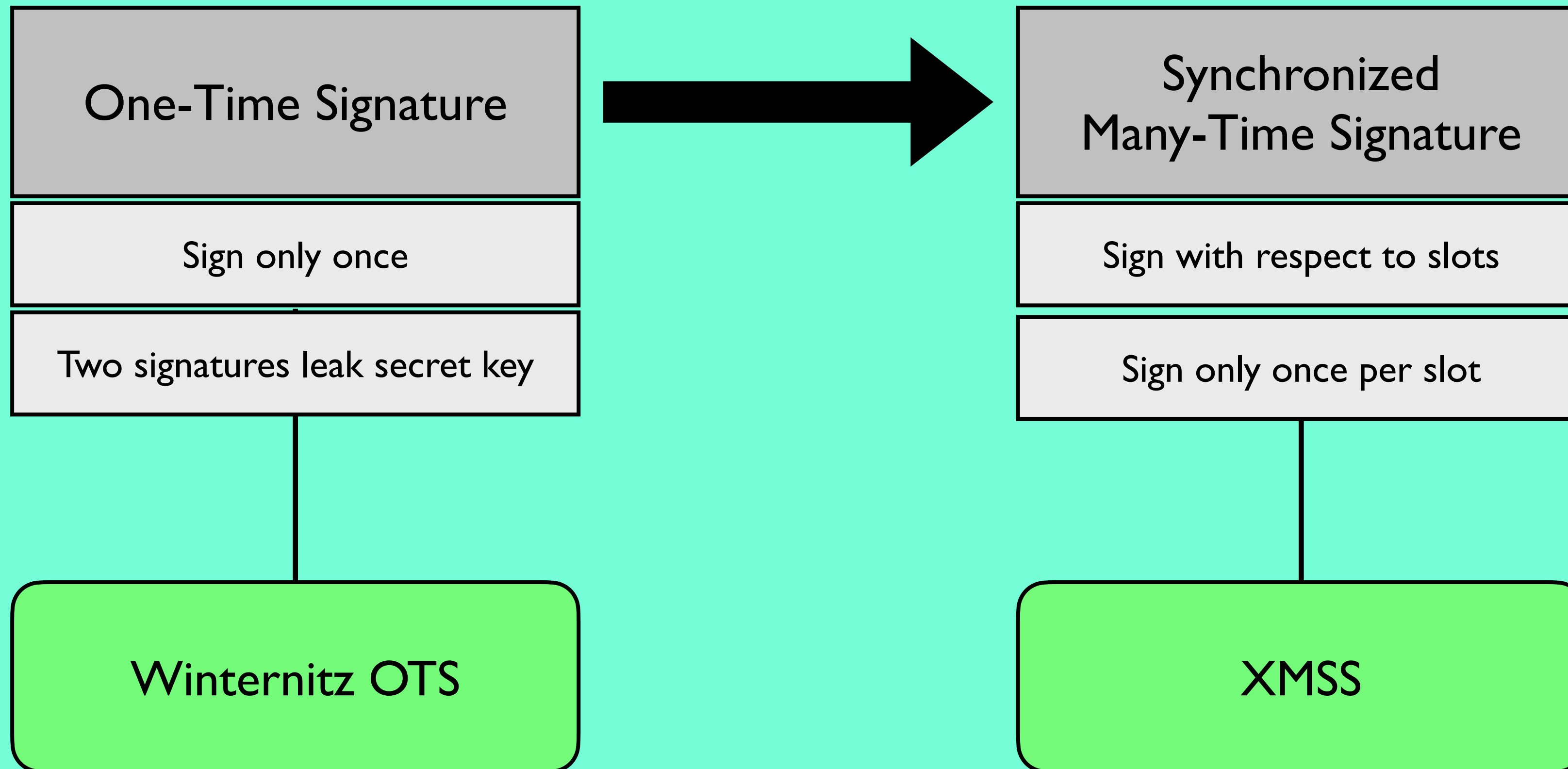
## Signing message $m$ wrt slot $i$

- ~~1. OT Public Key  $pk_i^{\text{OTS}}$~~
2. Merkle Path
3. OT Sig  $\text{Sig}(sk_i^{\text{OTS}}, m)$

## Verification

1. Check Merkle Path
2. Check OT Sig wrt to OT Public Key

# The XMSS Approach



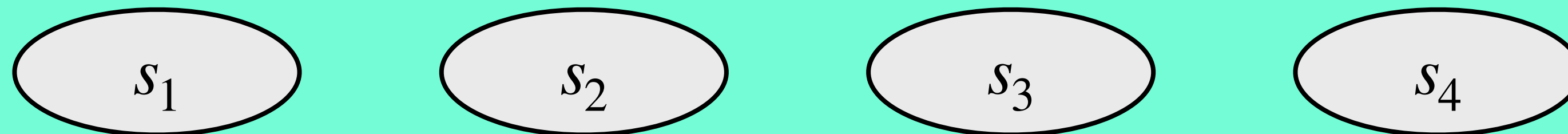
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



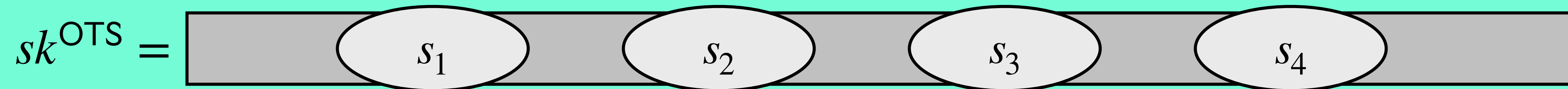
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



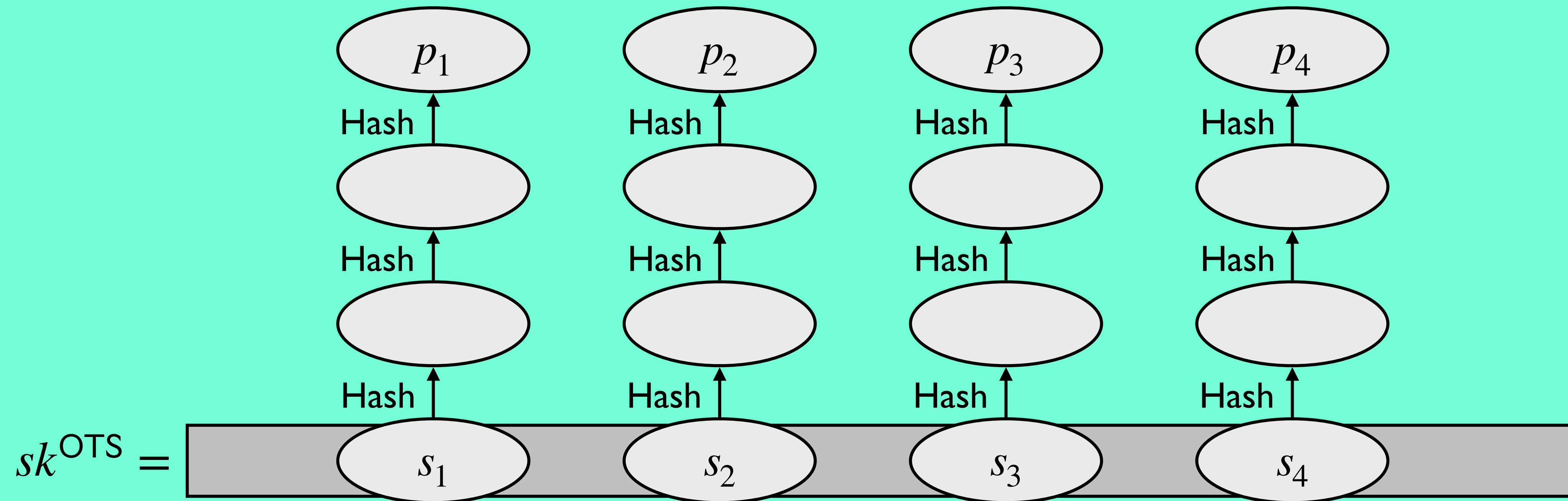
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



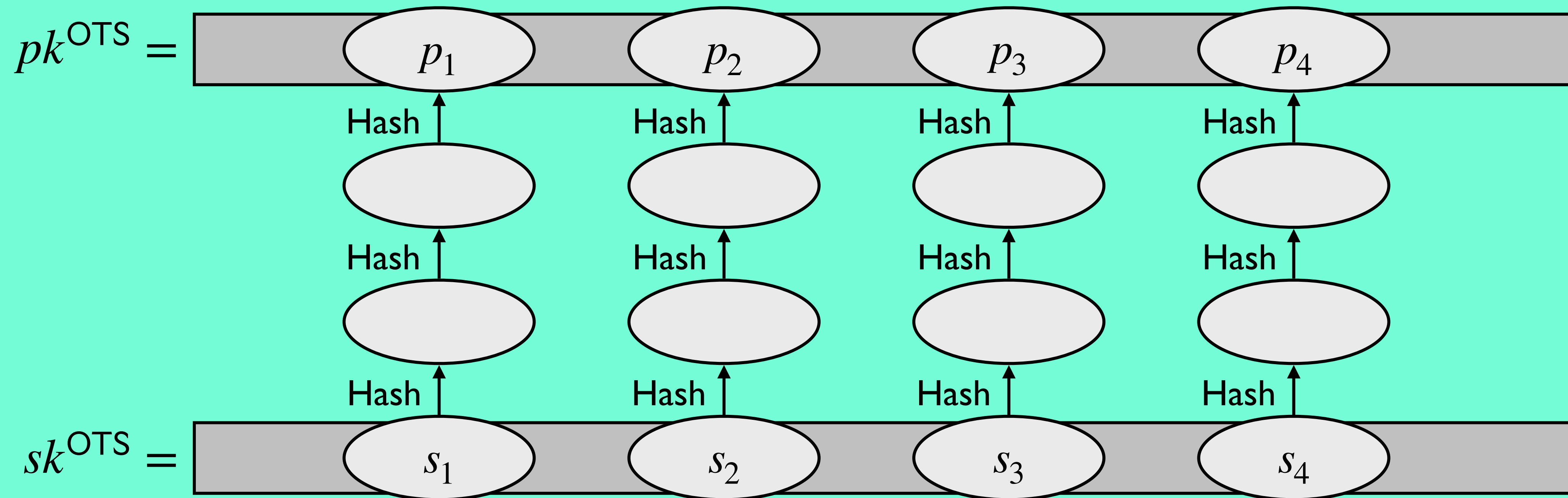
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



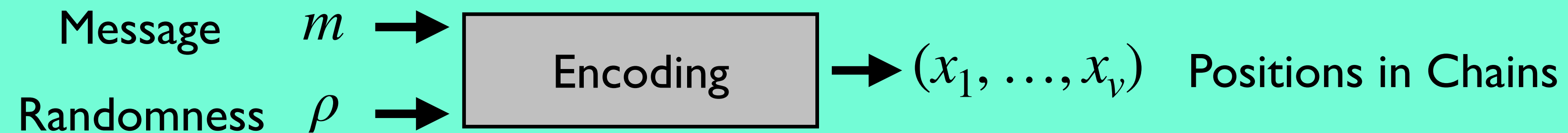
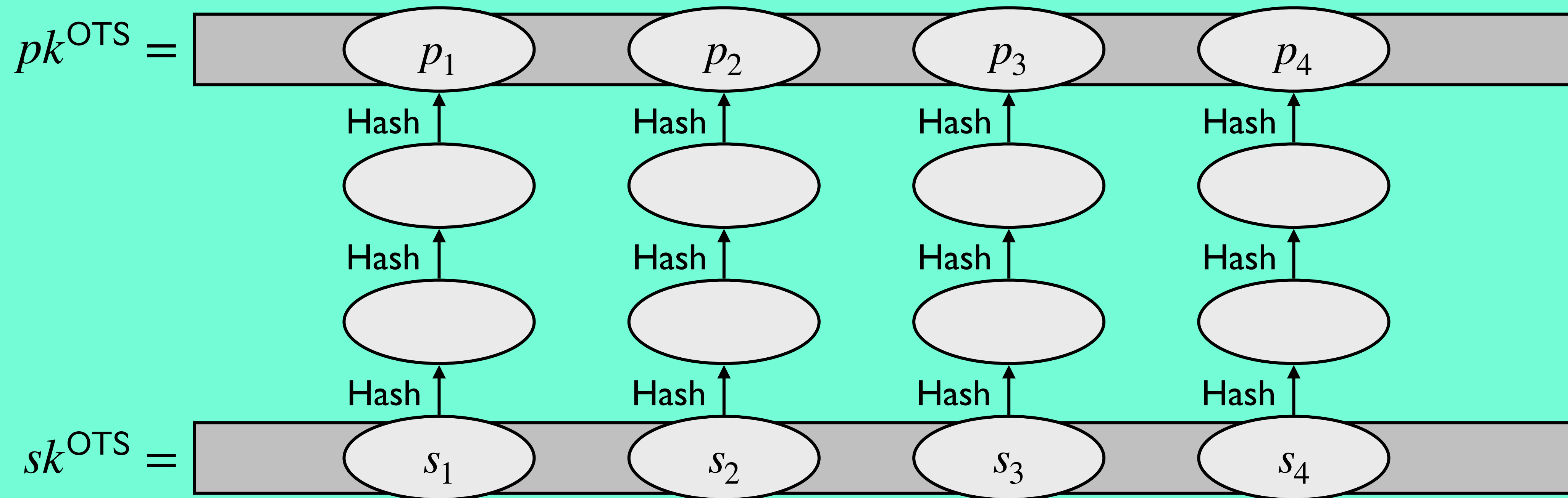
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



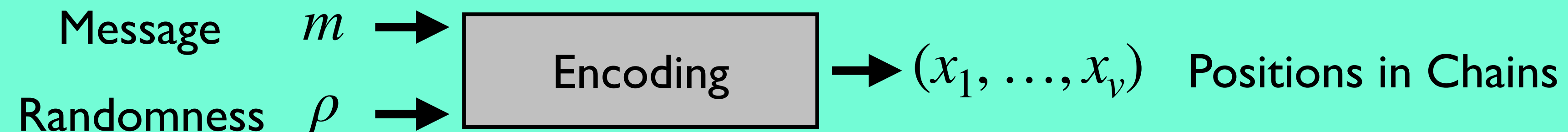
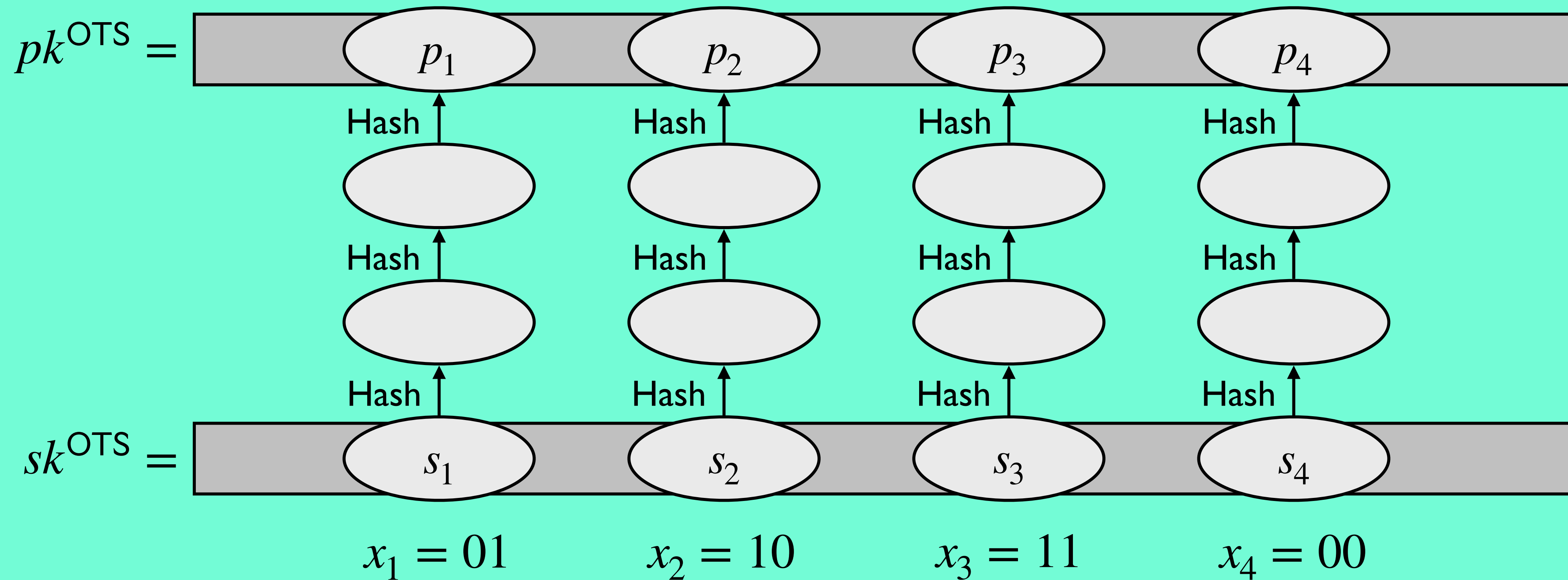
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



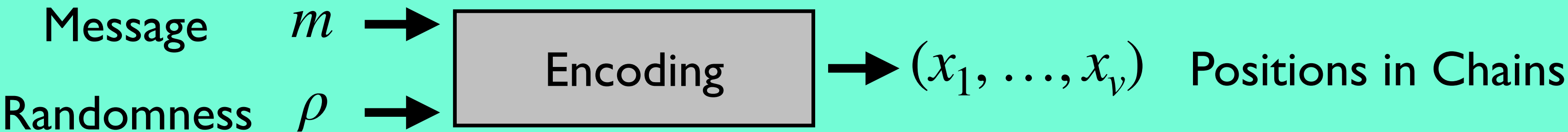
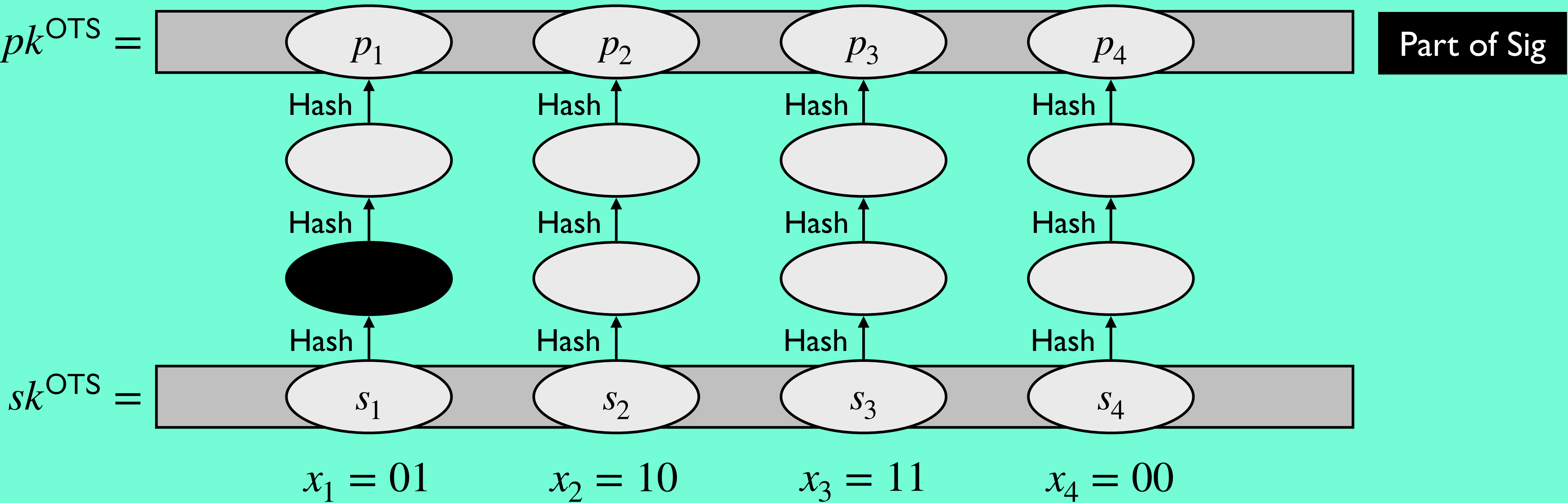
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



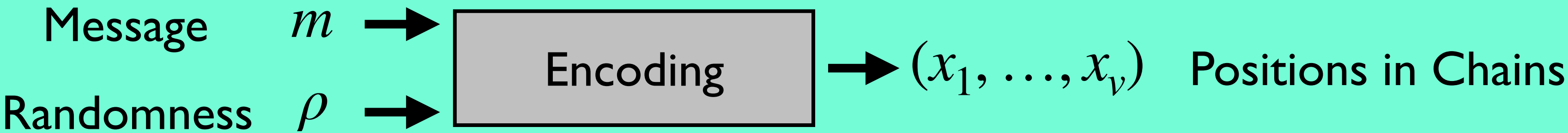
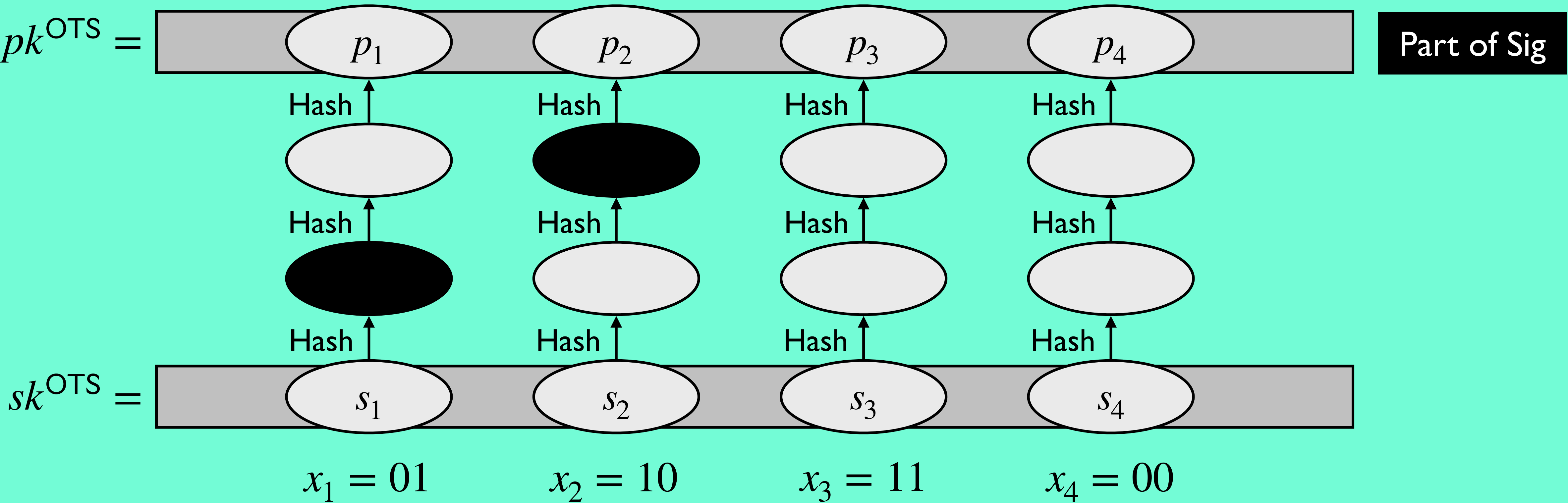
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



# One-Time Signatures via Hash Chains

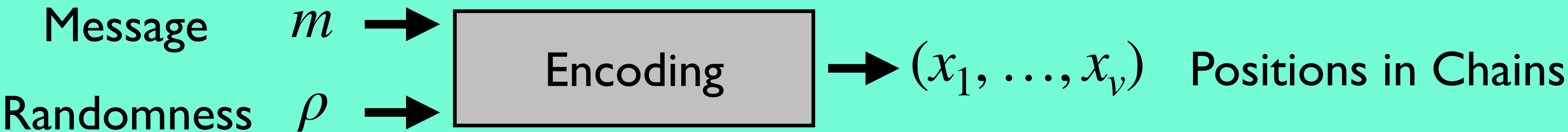
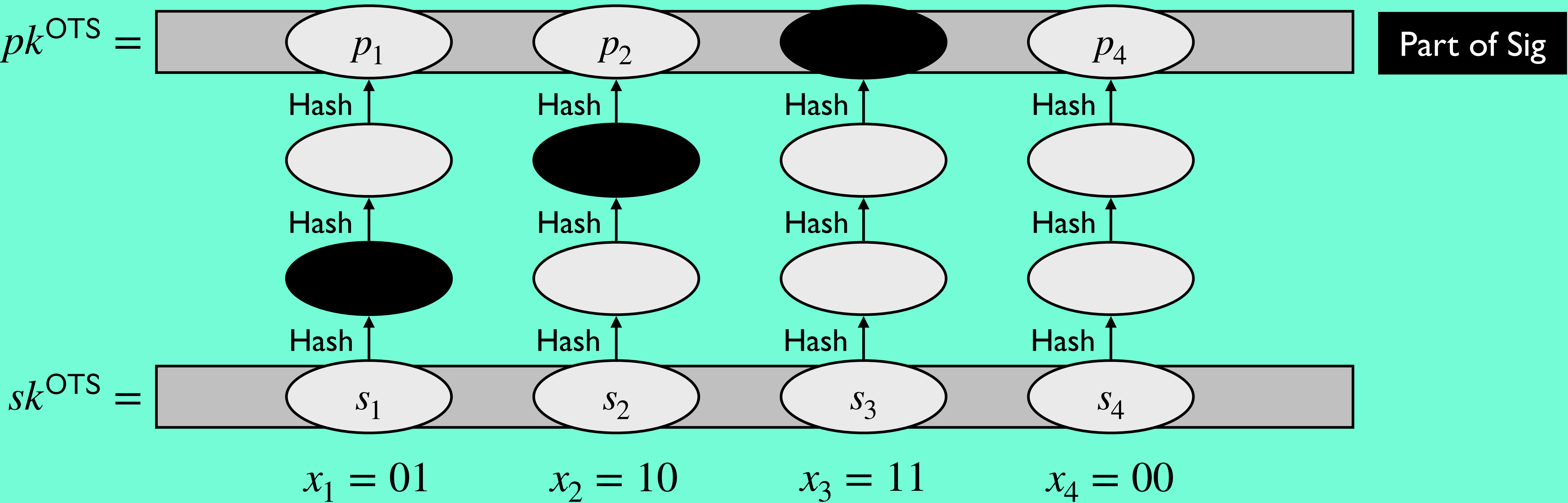
Example:  $v = 4$  chains of length  $2^w = 4$





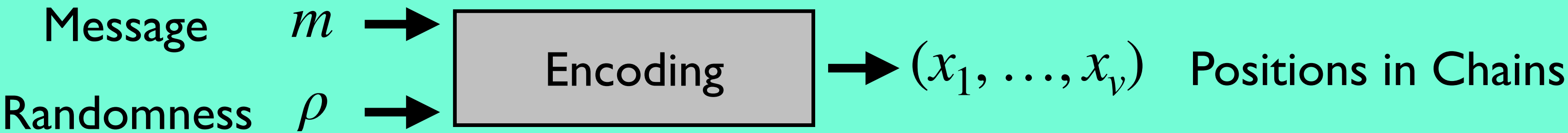
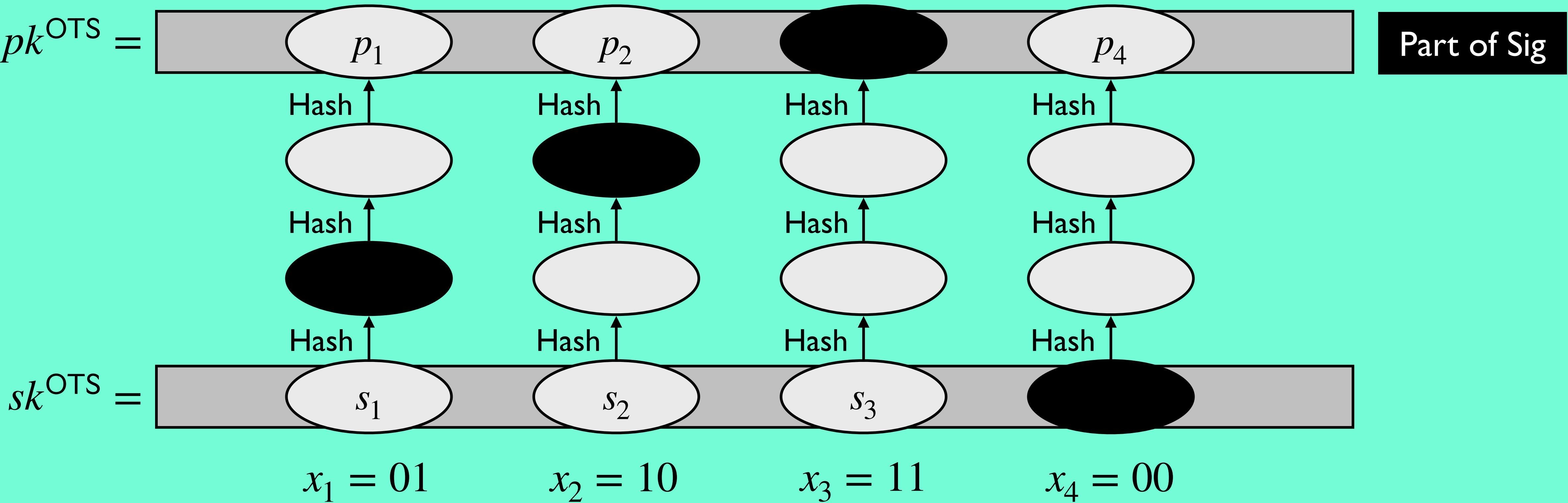
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



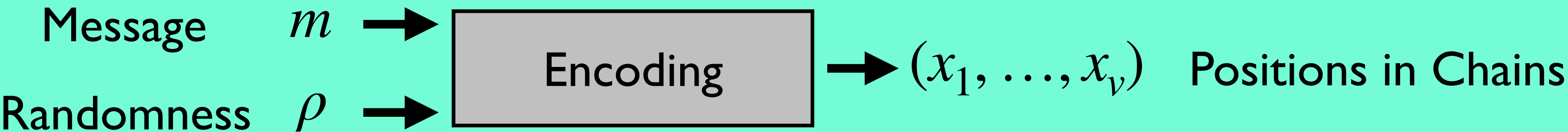
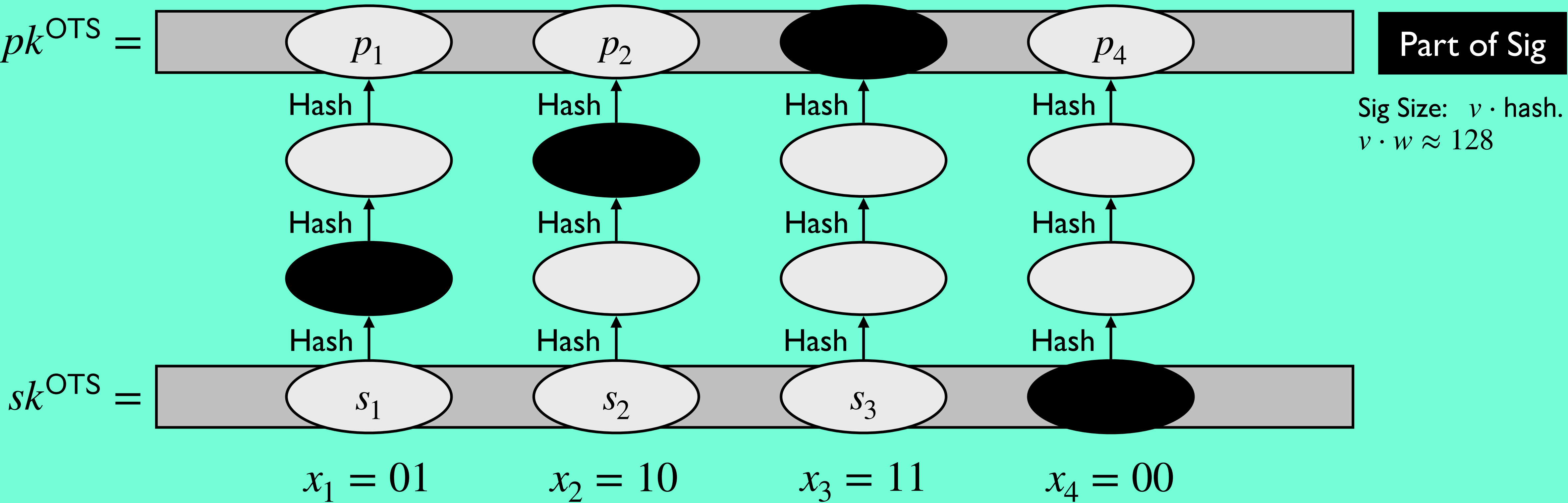
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



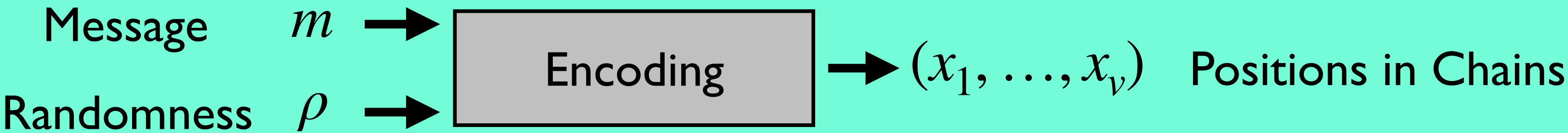
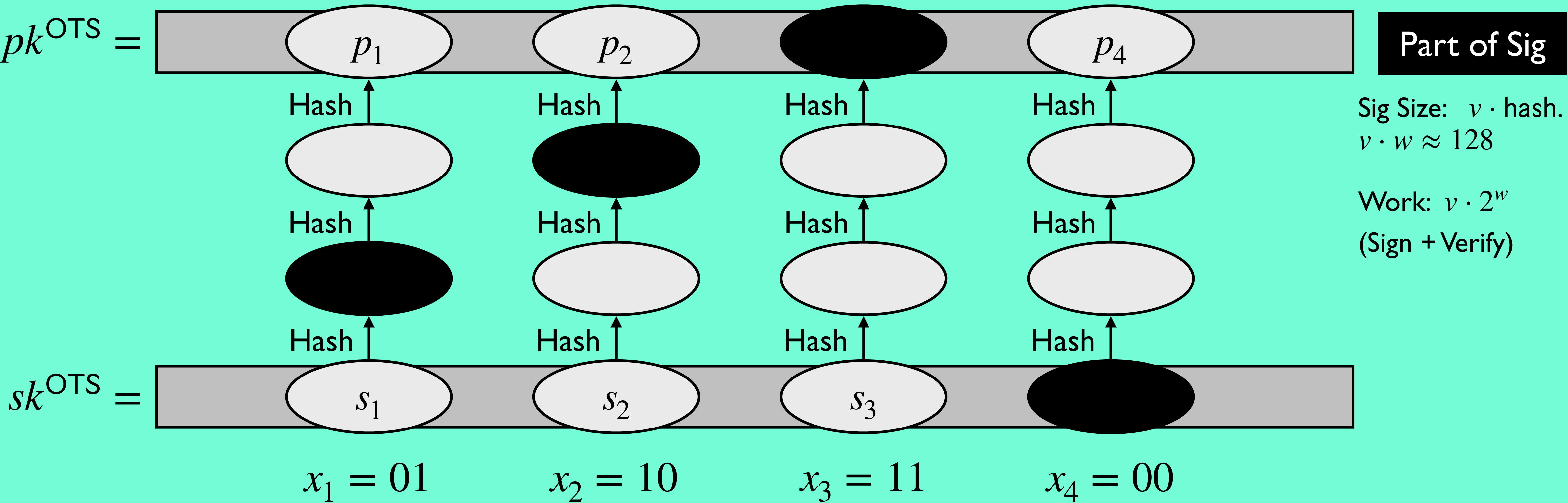
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



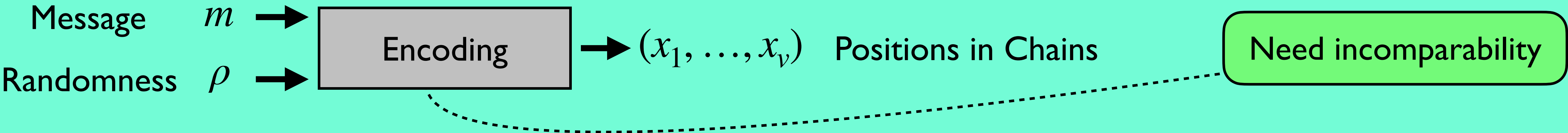
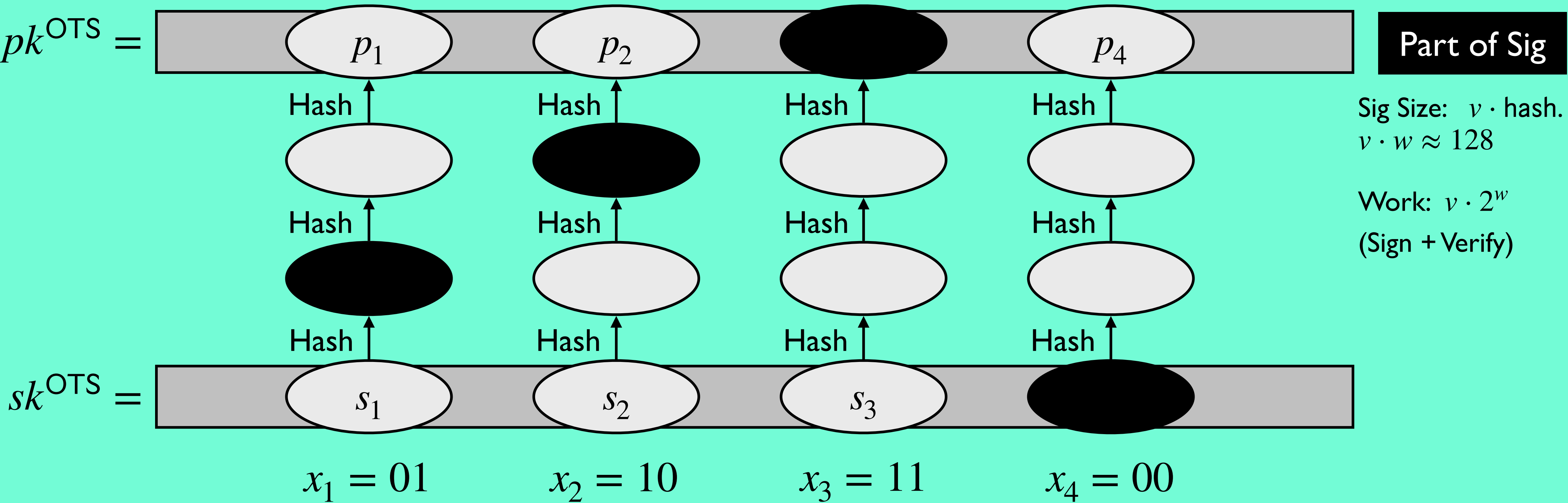
# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



# One-Time Signatures via Hash Chains

Example:  $v = 4$  chains of length  $2^w = 4$



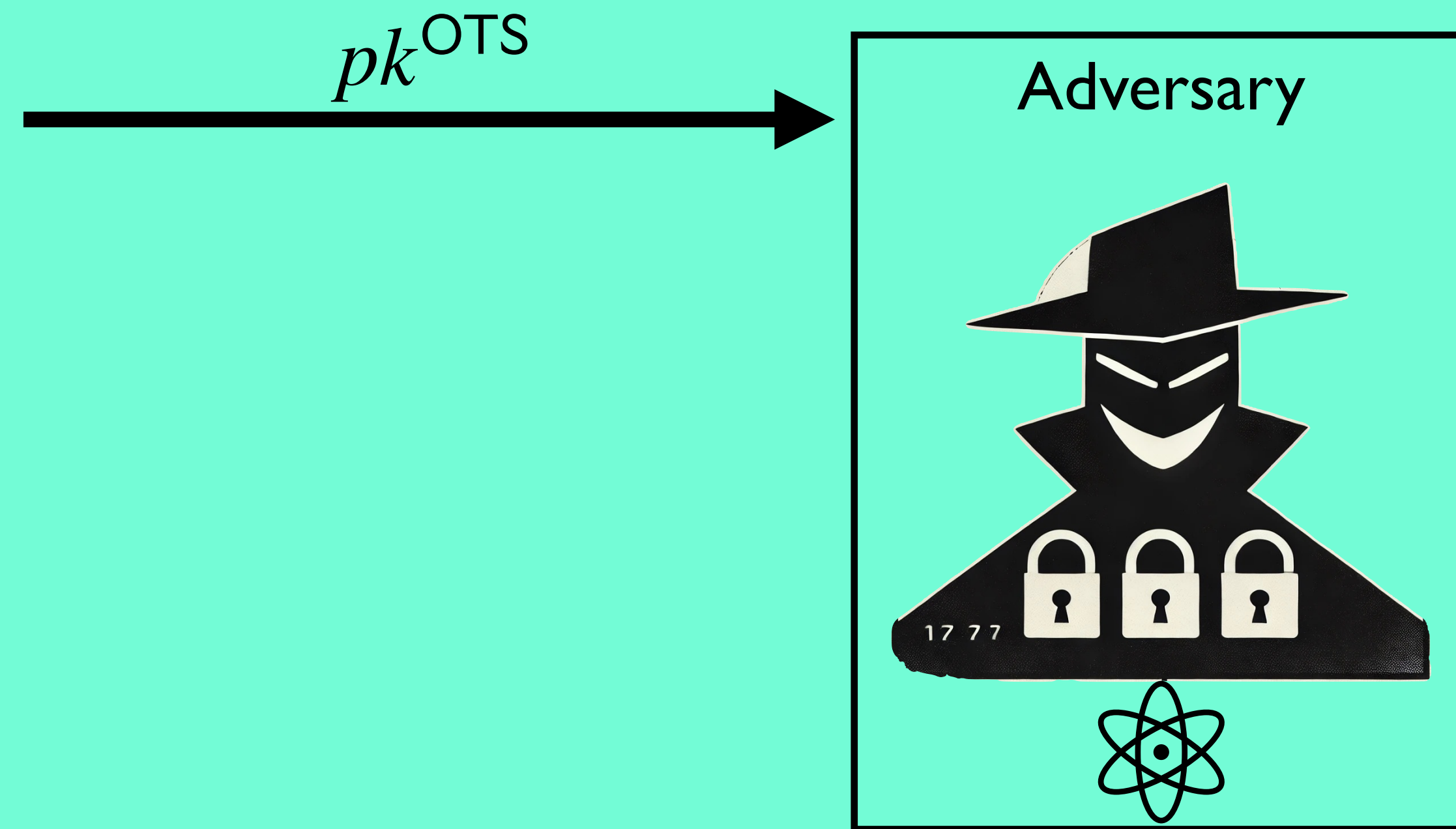
# One-Time Signatures via Hash Chains



# One-Time Signatures via Hash Chains

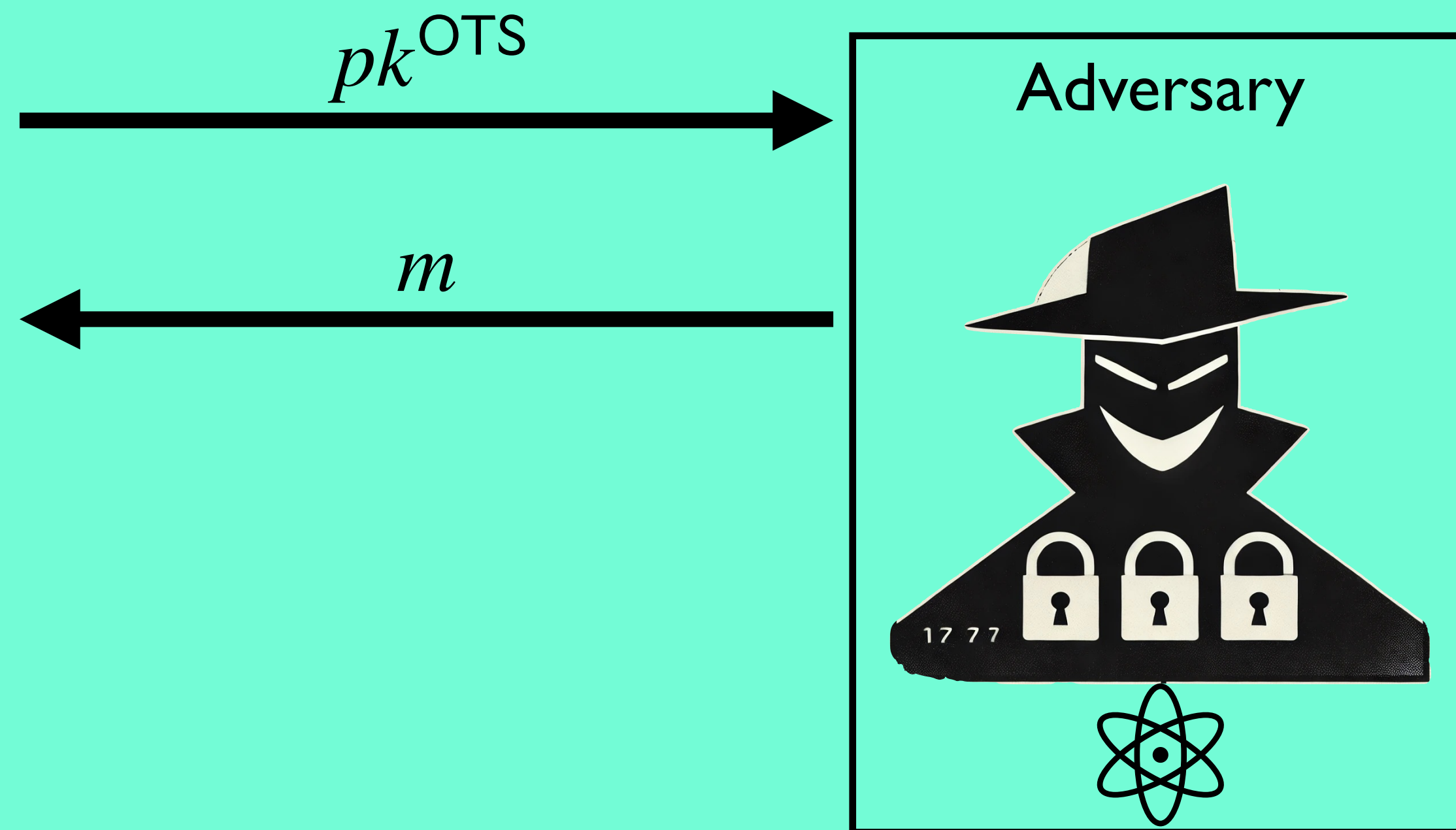


# One-Time Signatures via Hash Chains

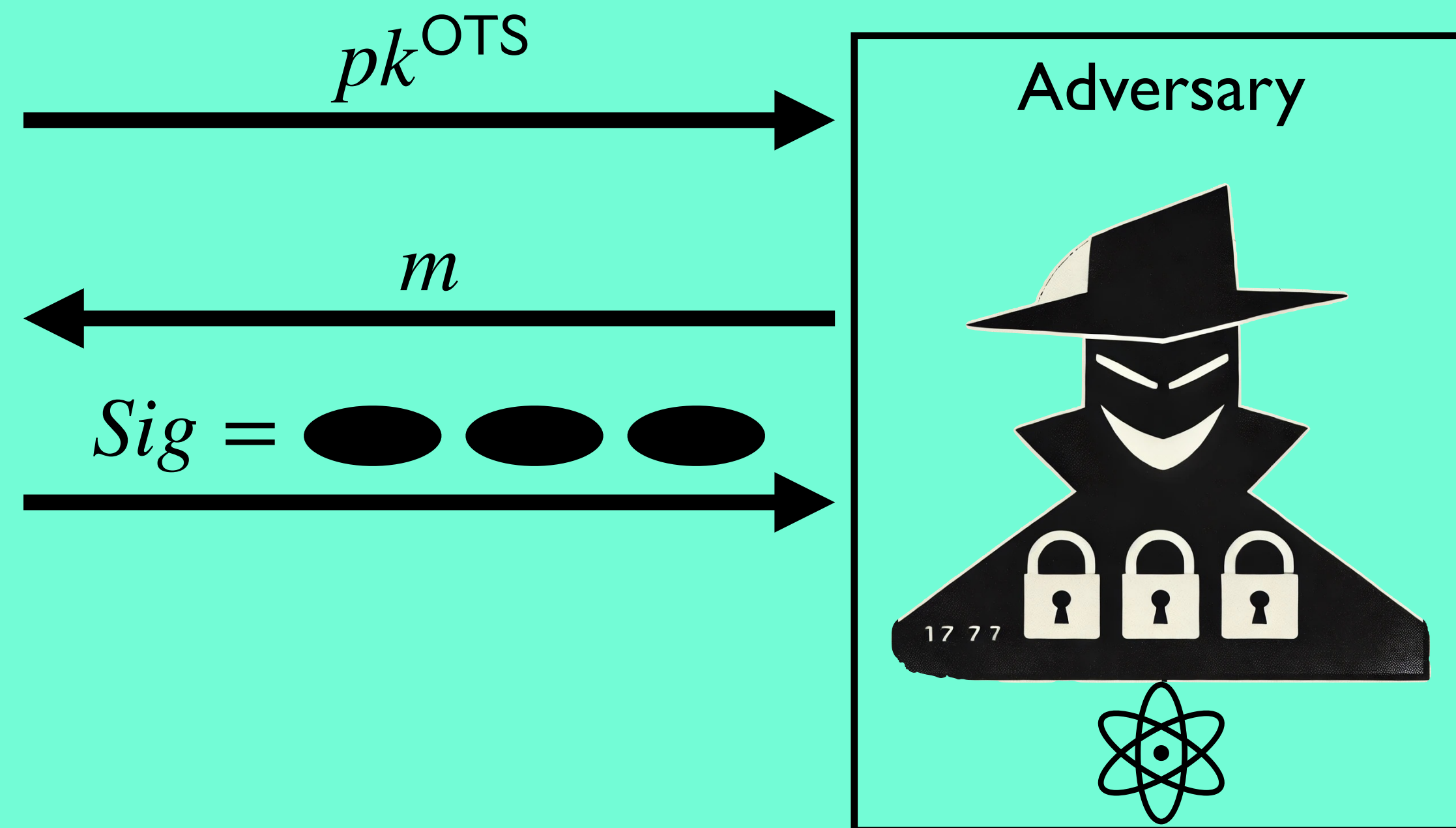




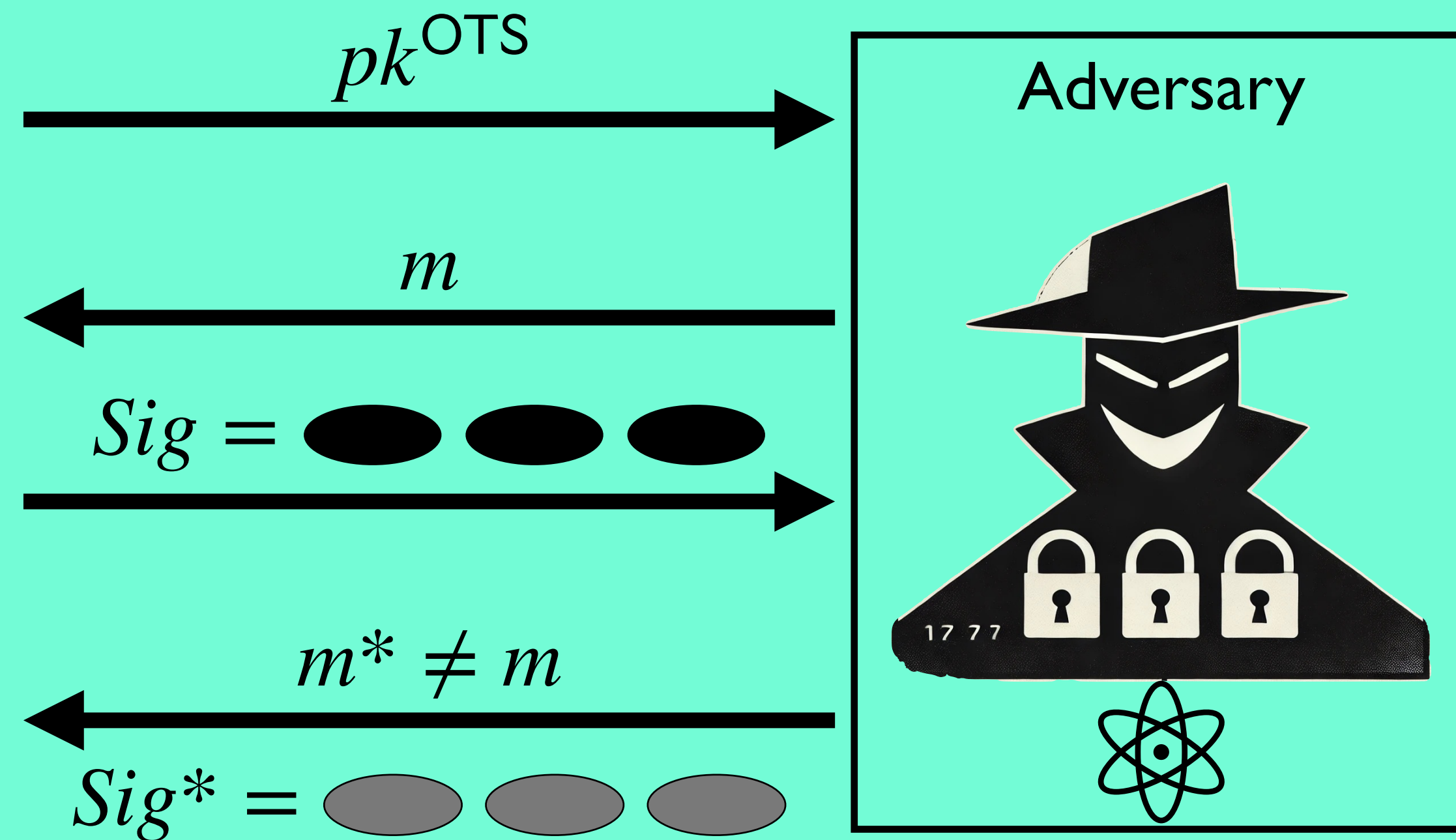
# One-Time Signatures via Hash Chains



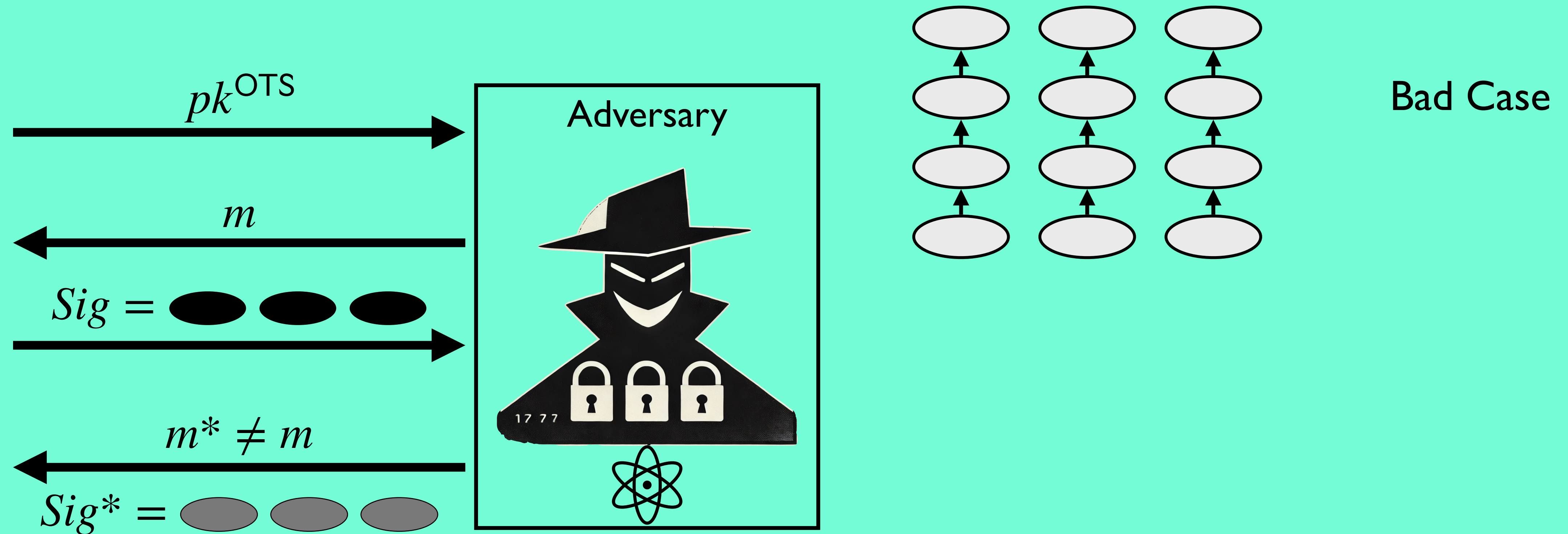
# One-Time Signatures via Hash Chains



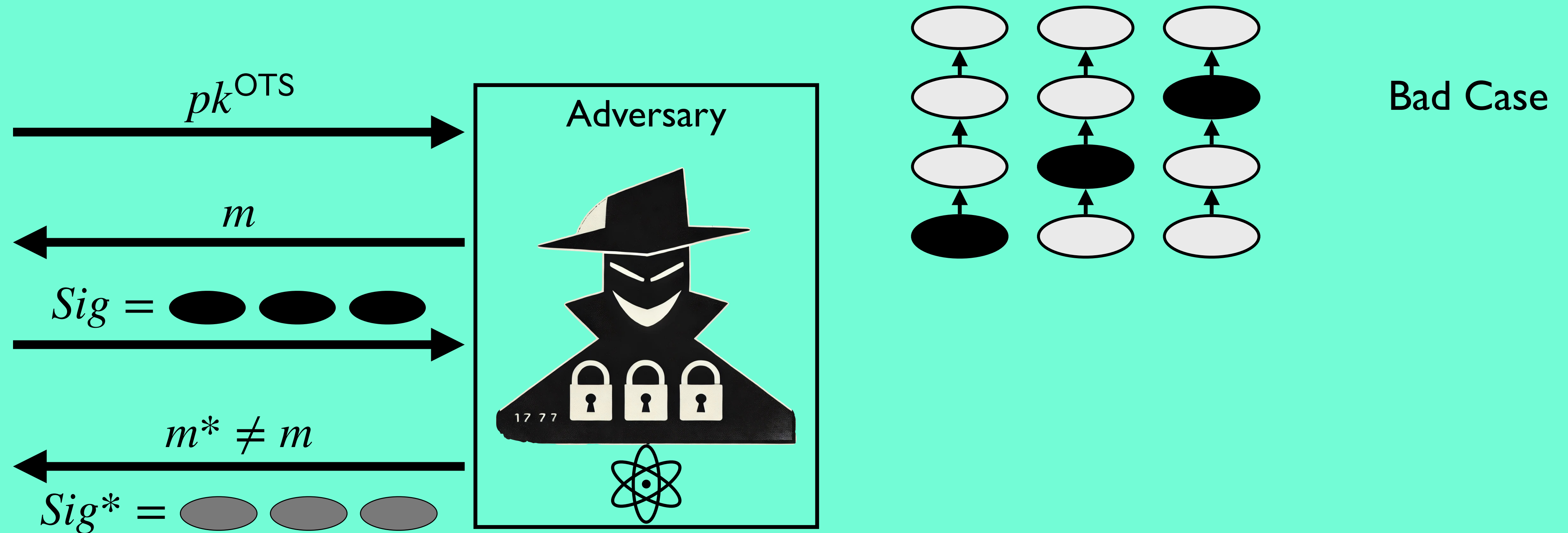
# One-Time Signatures via Hash Chains



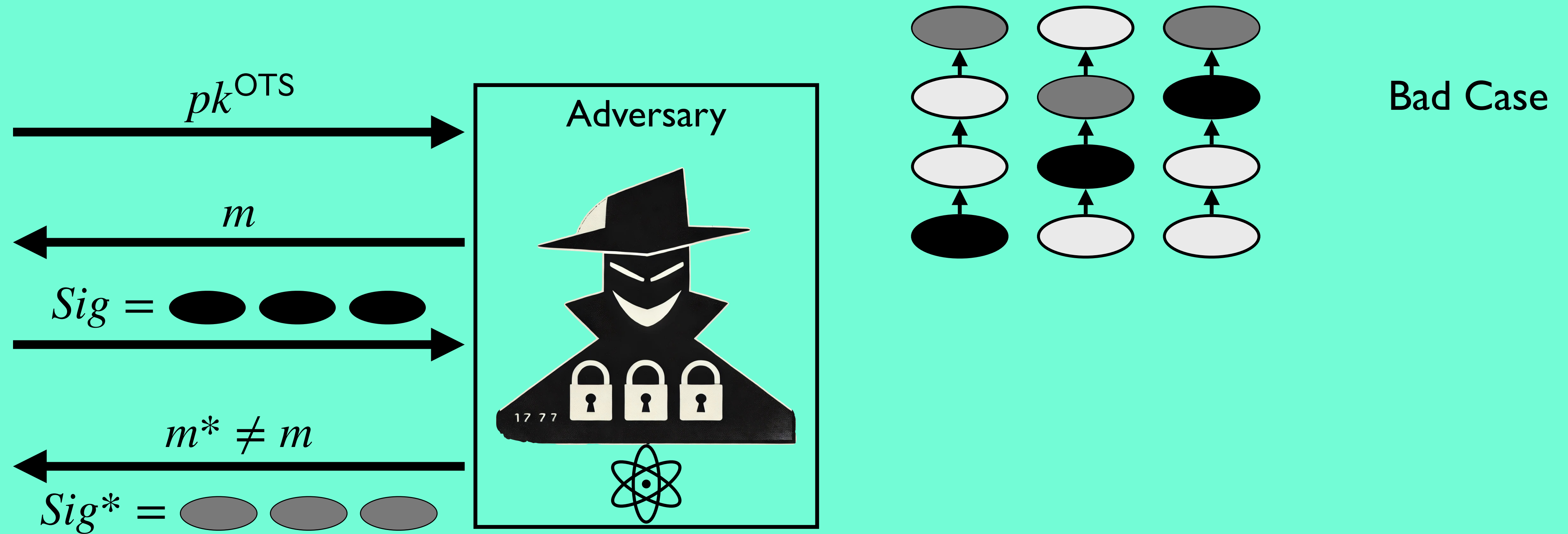
# One-Time Signatures via Hash Chains



# One-Time Signatures via Hash Chains

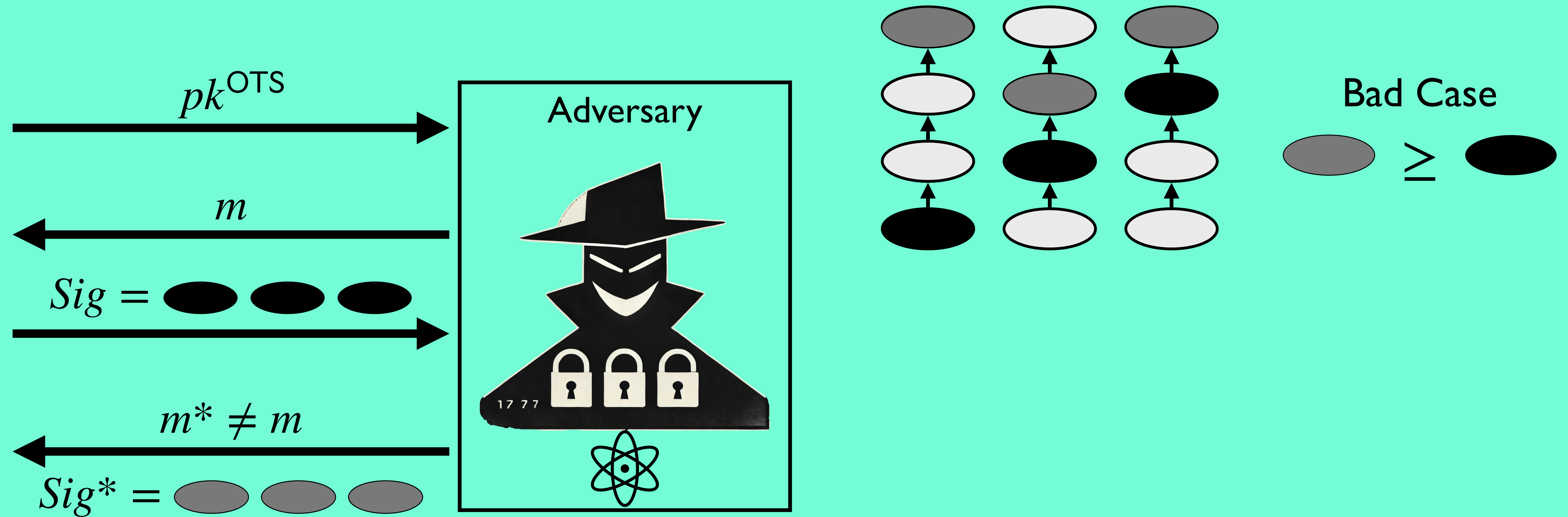


# One-Time Signatures via Hash Chains

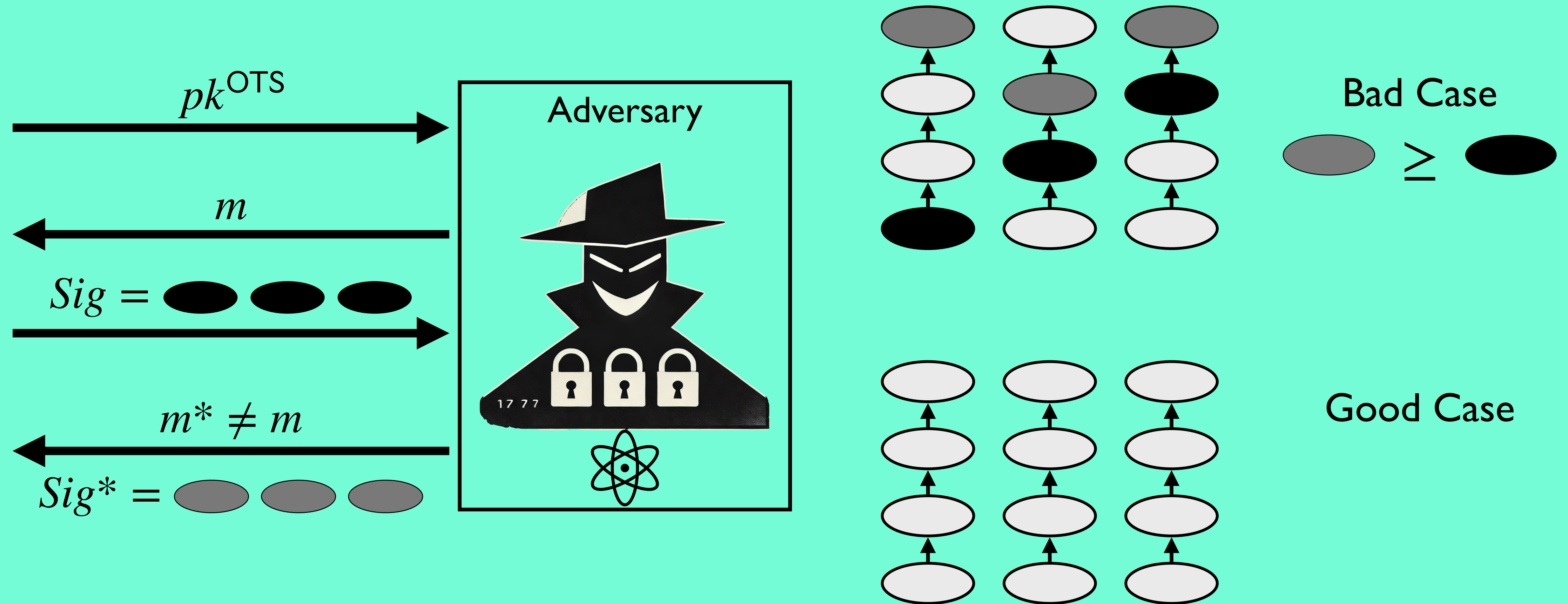




# One-Time Signatures via Hash Chains

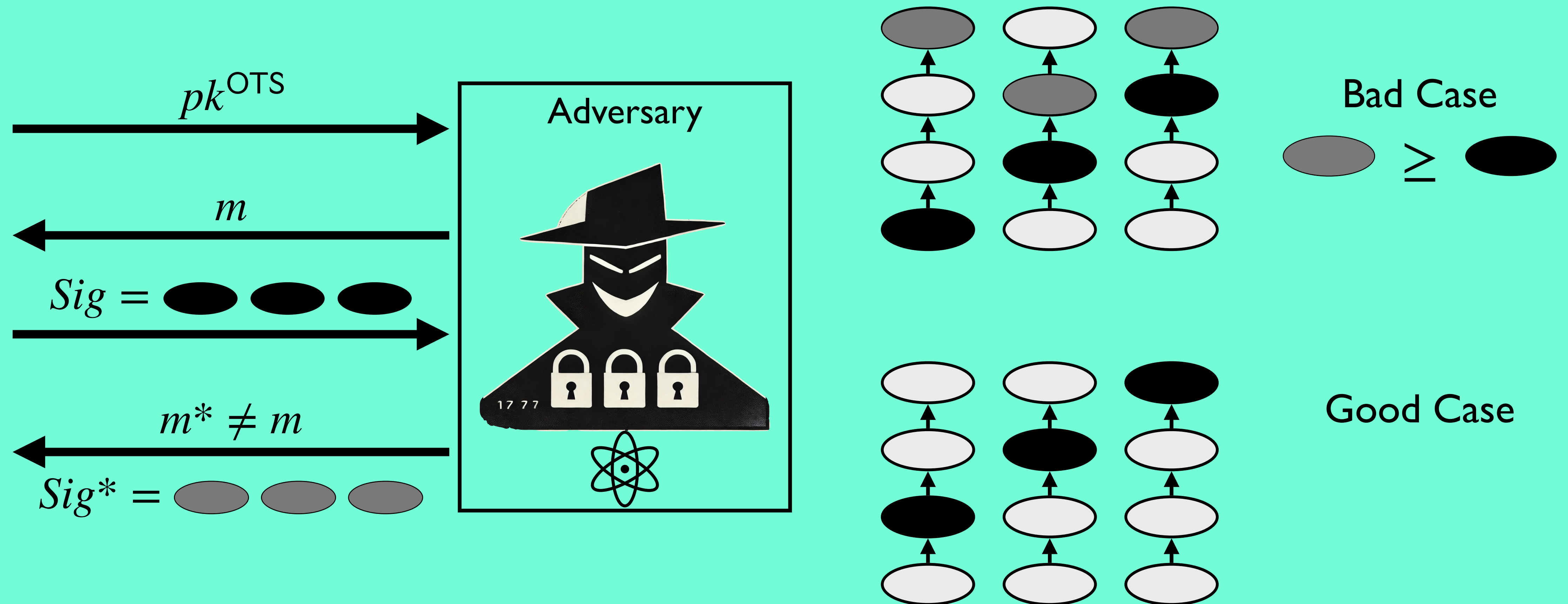


# One-Time Signatures via Hash Chains

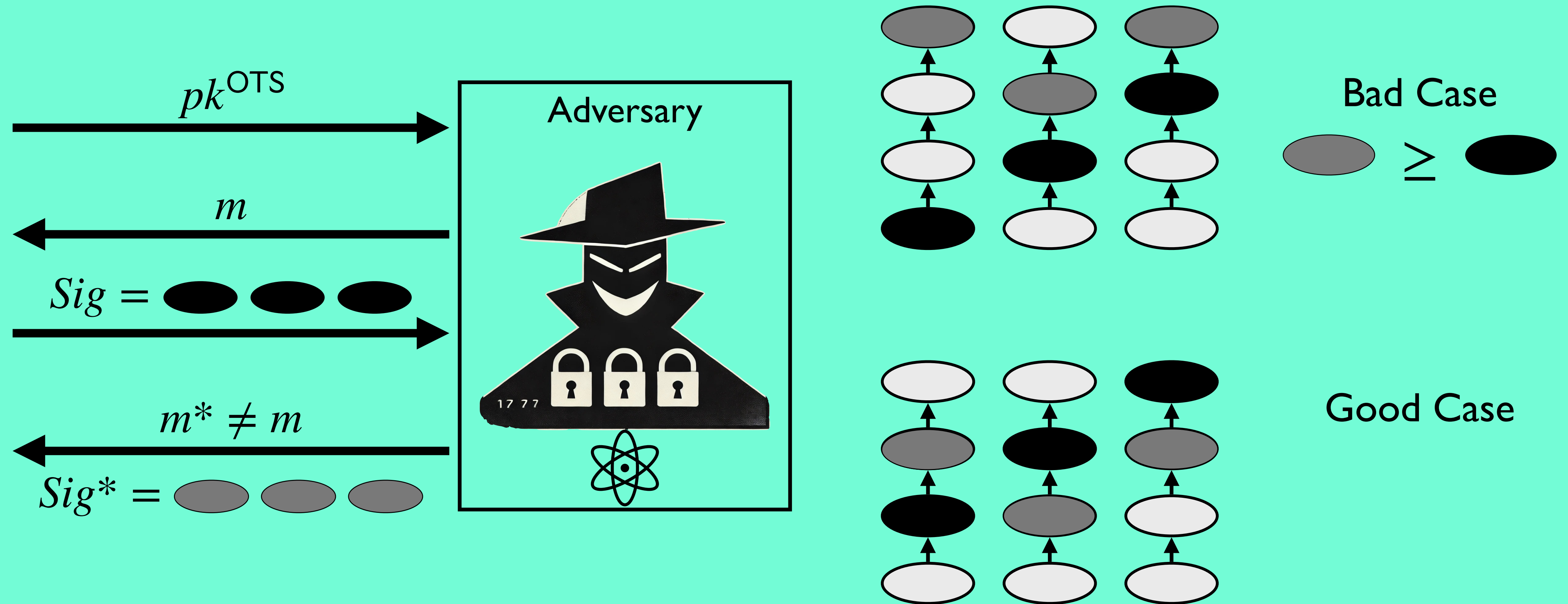




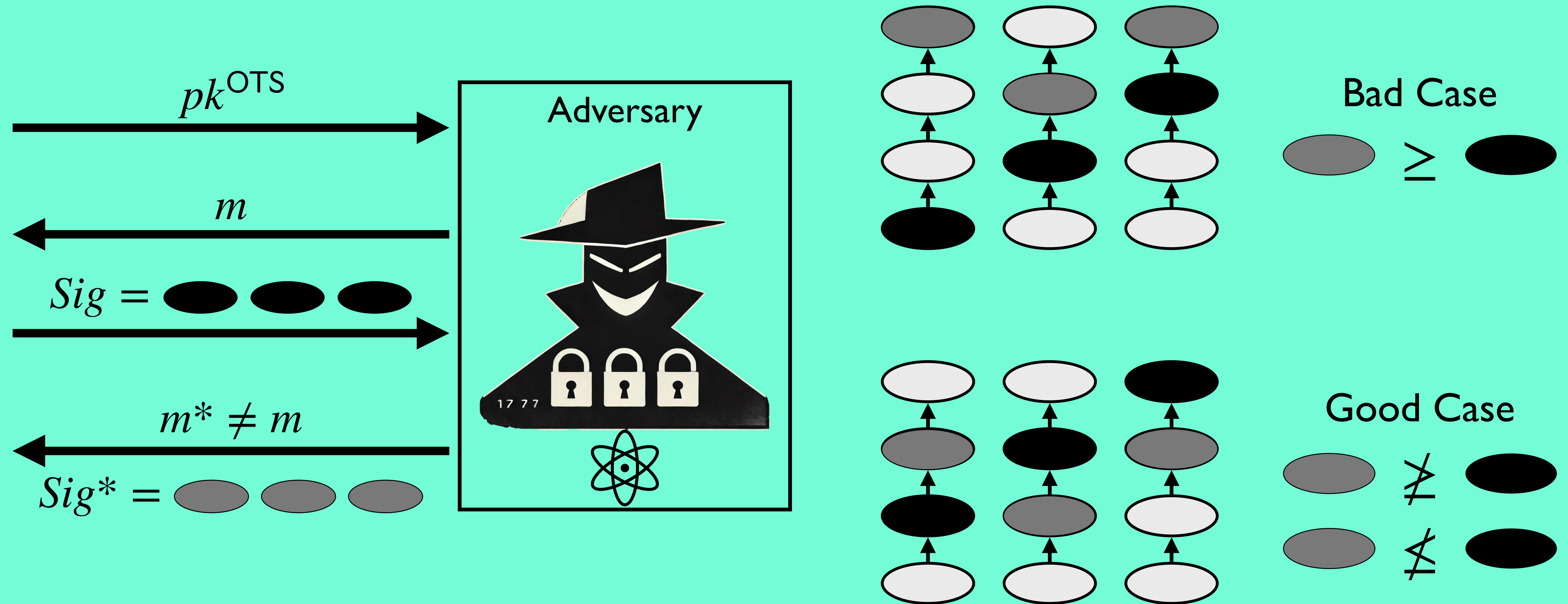
# One-Time Signatures via Hash Chains



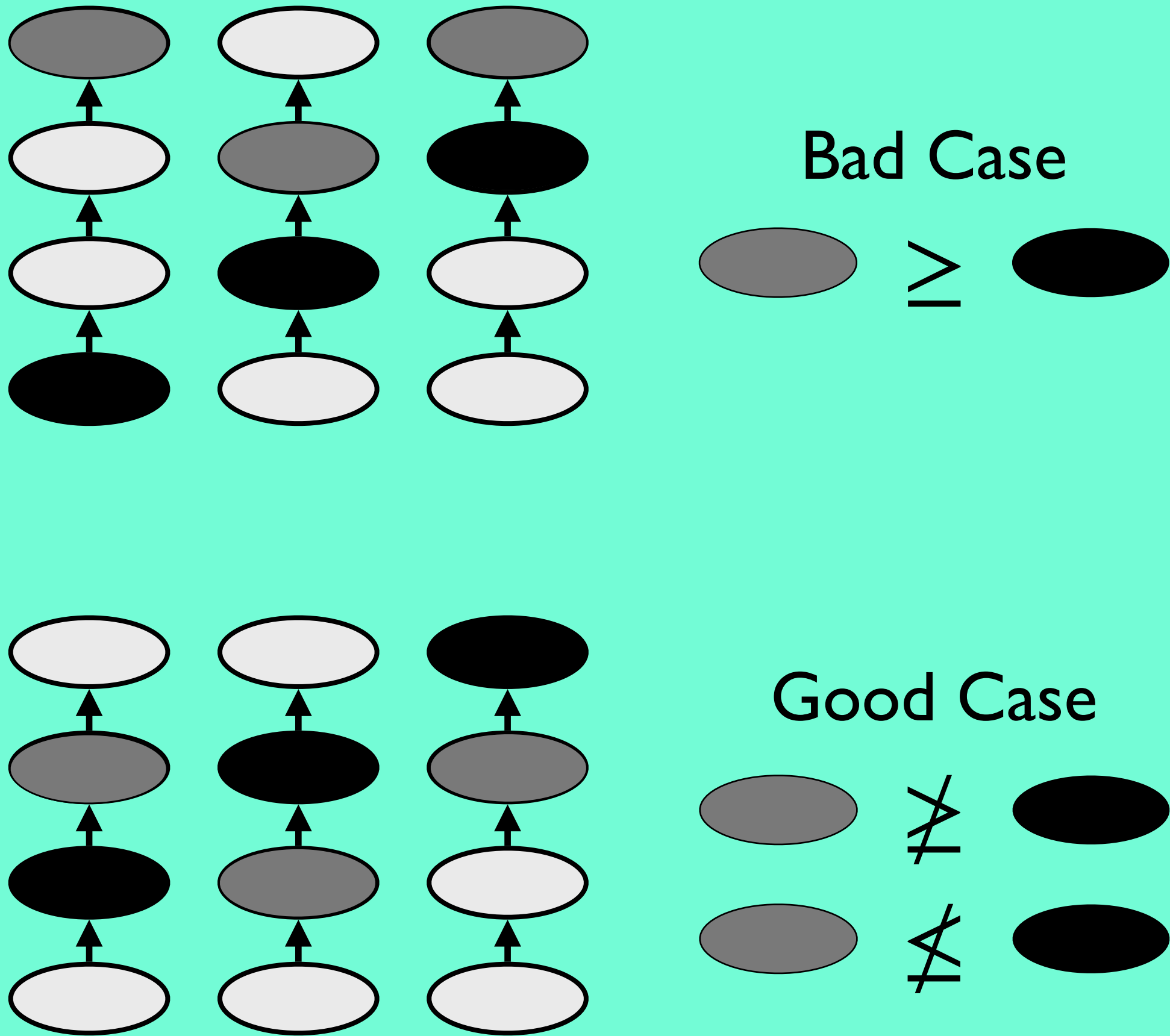
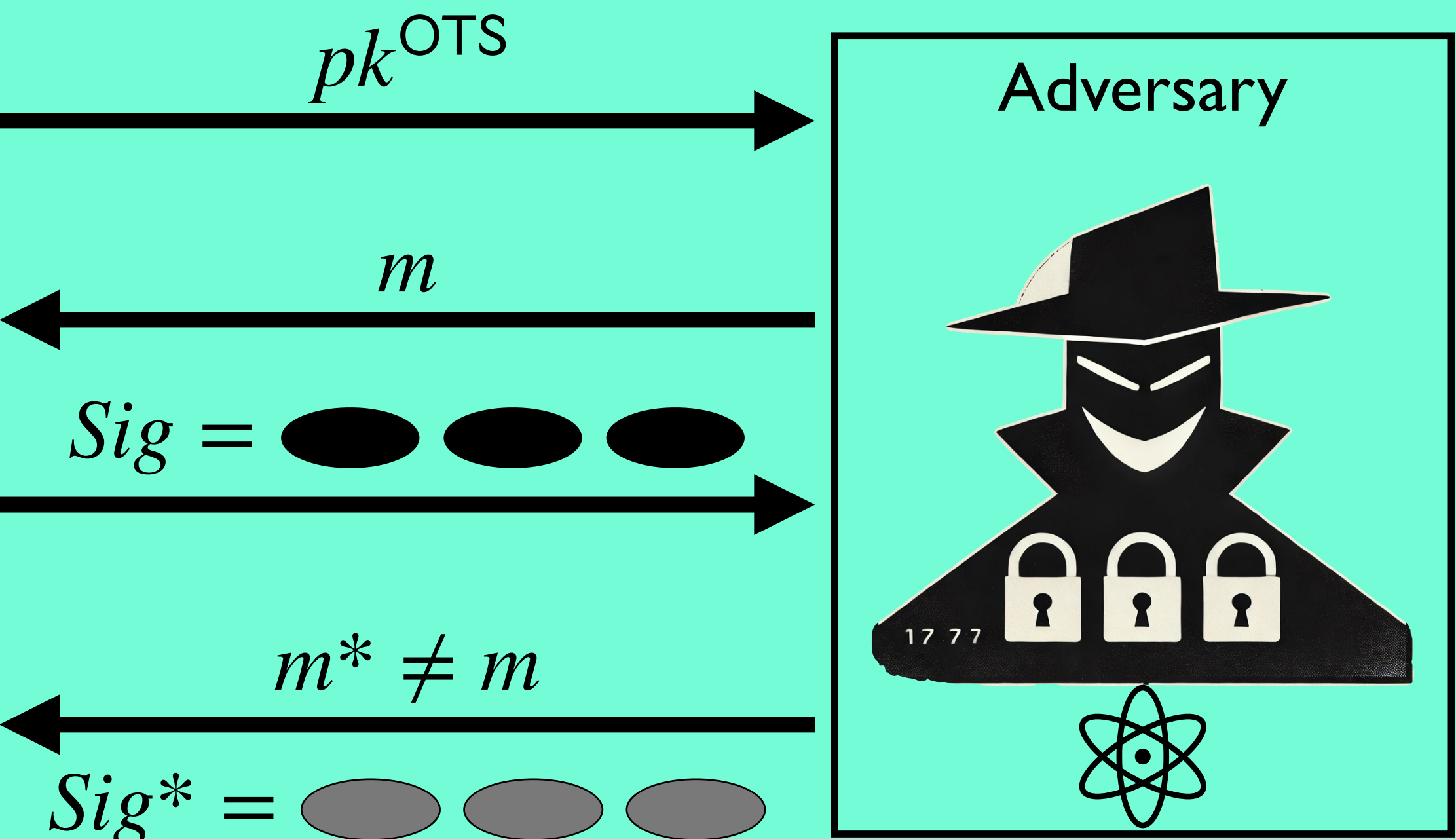
# One-Time Signatures via Hash Chains



# One-Time Signatures via Hash Chains



# One-Time Signatures via Hash Chains



Incomparable Encoding: Only Good Case

# The XMSS Approach

• **Efficient** (small signatures)

• **Secure** (proven security)

• **Simple** (easy to implement)

• **Scalable** (large number of keys)

• **Flexible** (many applications)

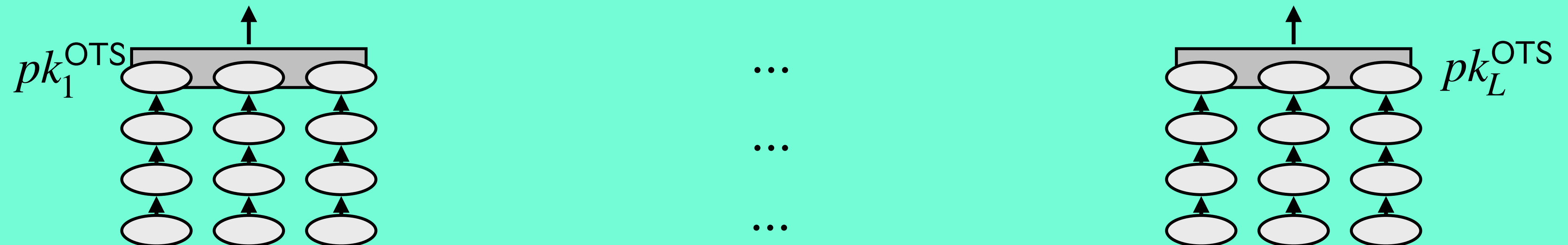
• **Fast** (fast verification)

• **Robust** (resistant to attacks)

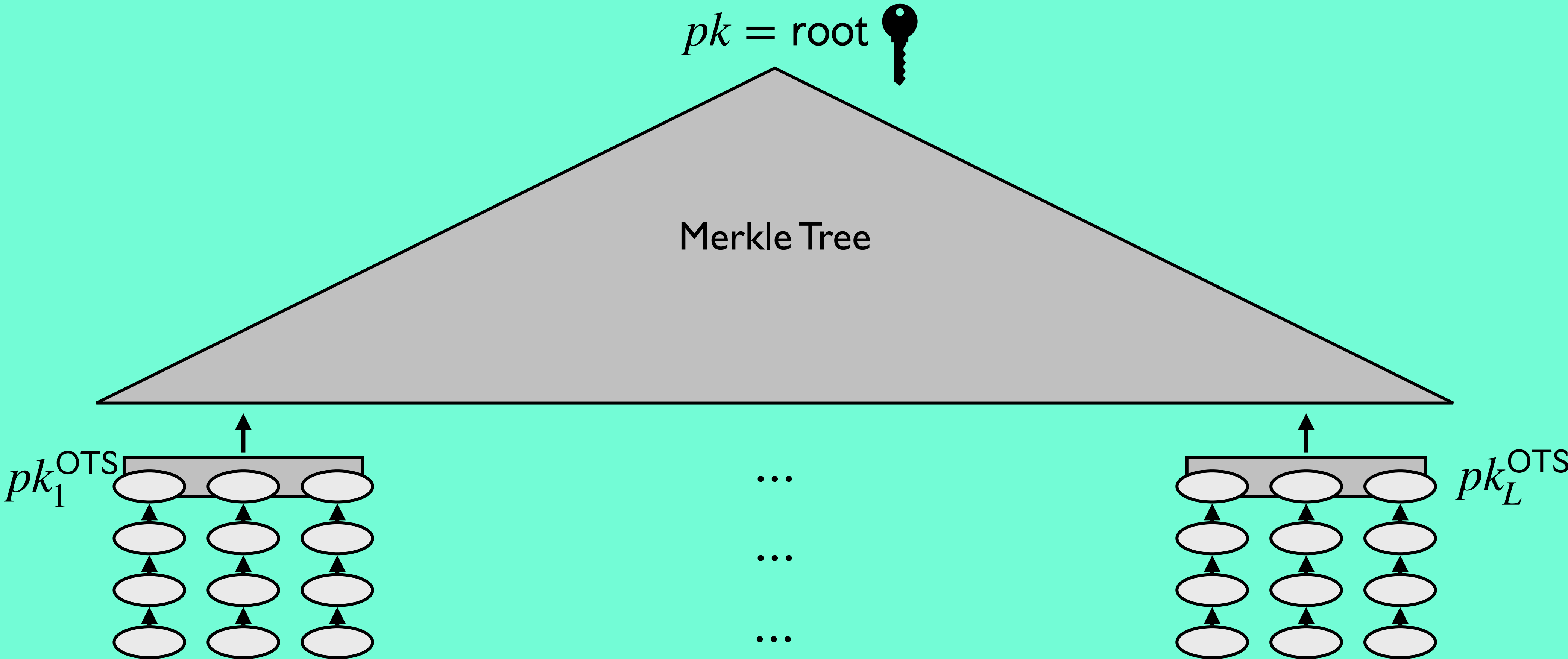
• **Adapted** (tailored for IoT)

• **Open** (publicly available)

# The XMSS Approach

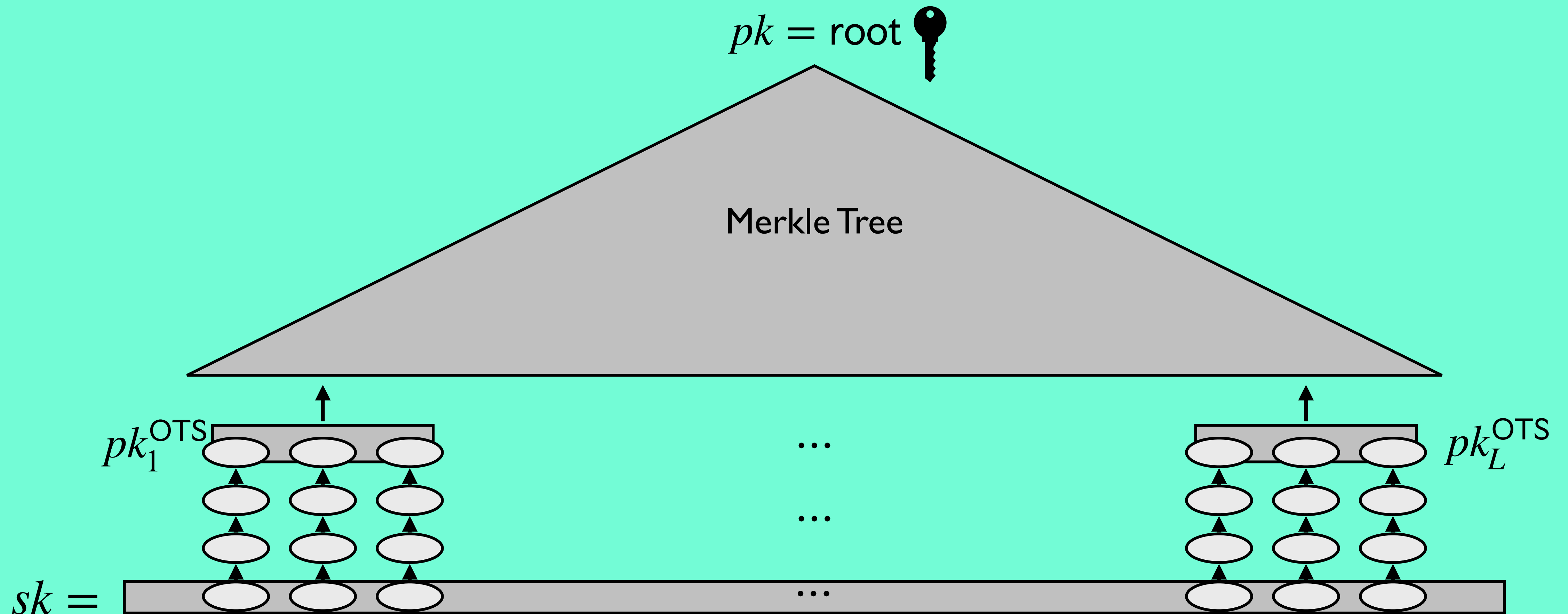


# The XMSS Approach



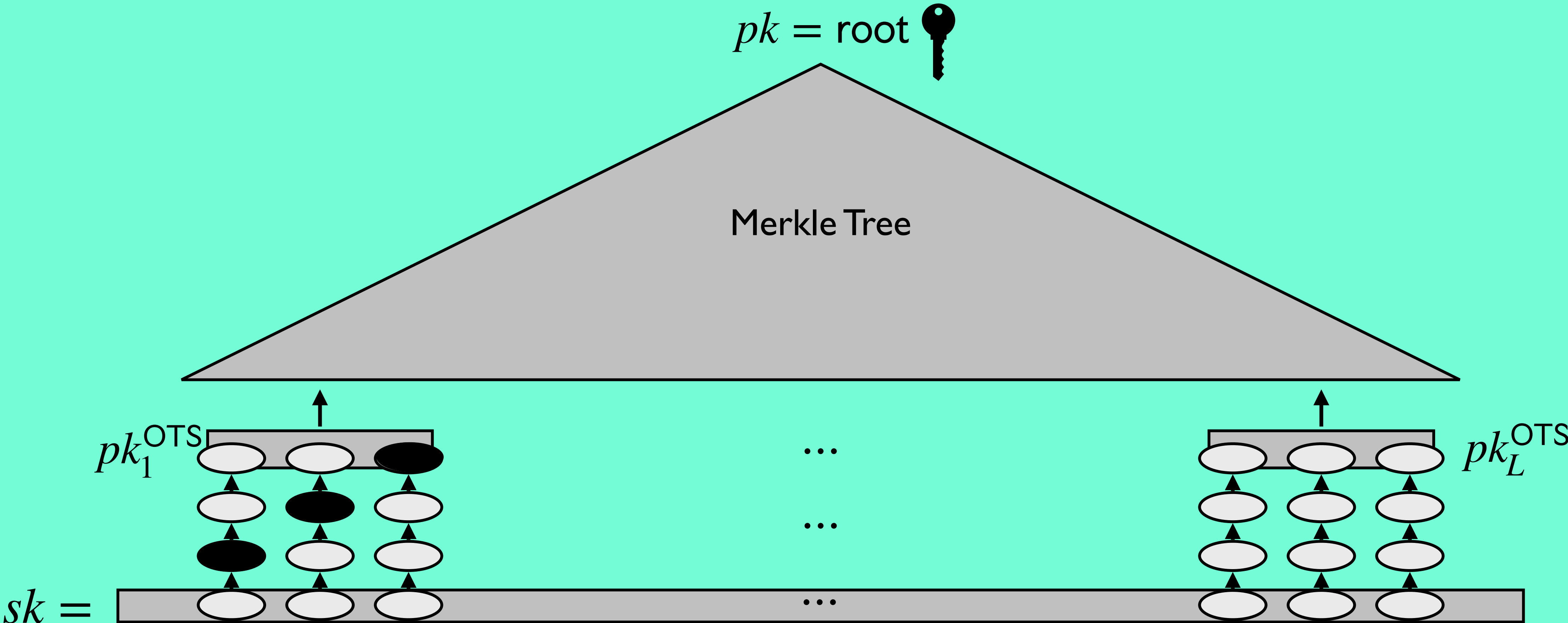


# The XMSS Approach

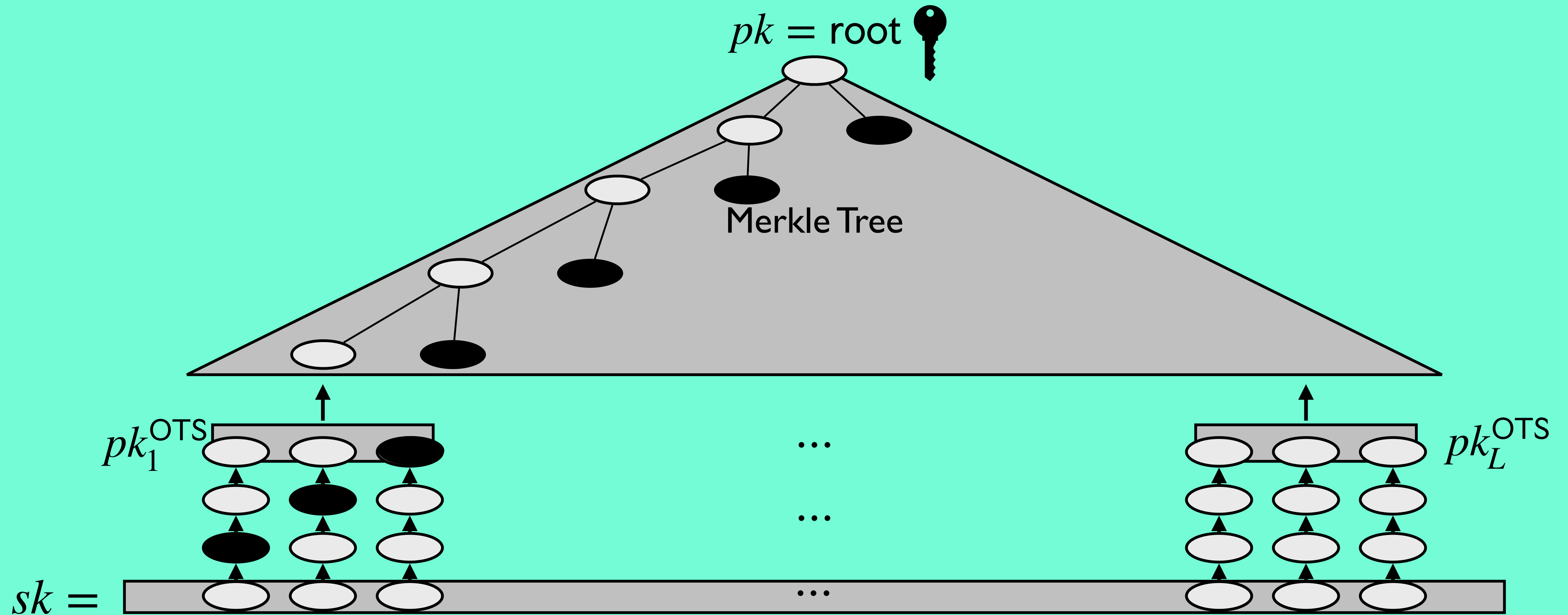




# The XMSS Approach



# The XMSS Approach



# Security Properties for Hash-Functions

Property	Definition	Importance
Preimage Resistance	Given a hash value $h$ , it is computationally infeasible to find any input $x$ such that $H(x) = h$ .	Ensures that the hash function is one-way, preventing attackers from reversing the hash to find the original data.
Second-Preimage Resistance	Given an input $x$ , it is computationally infeasible to find a different input $x'$ such that $H(x) = H(x')$ .	Prevents attackers from finding a different input that produces the same hash value as a given input.
Collision Resistance	It is computationally infeasible to find any two distinct inputs $x$ and $x'$ such that $H(x) = H(x')$ .	Ensures that no two different inputs can produce the same hash value, which is crucial for data integrity and security.

# Security Properties for Hash-Functions

Security of XMSS Variants

# Security Properties for Hash-Functions

Security of XMSS Variants



Hash Function Properties

Multi-Target Preimage Resistance

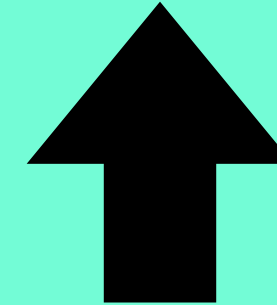
Multi-Target Undetectability

Multi-Target Collision Resistance

Multi-Target Collision Resistance with random Sampling

# Security Properties for Hash-Functions

Security of XMSS Variants



Hash Function Properties

Multi-Target Preimage Resistance

Multi-Target Undetectability

Multi-Target Collision Resistance

Multi-Target Collision Resistance with random Sampling

Concrete Targets for Cryptanalysis

# Efficiency

	Encoding	Parameters	Gen [s]	Sign [ $\mu$ s]	Ver [ $\mu$ s]	Sig [KiB]	$\pi_{16}$ AC	$\pi_{24}$ AC	$\pi_{16}$ WC	$\pi_{24}$ WC
Lifetime $L = 2^{18}$	W	$w = 1$	179.01	362.59	416.54	4.97	81	97	158	97
	W	$w = 2$	168.19	350.04	408.67	2.75	122	59	237	59
	W	$w = 4$	330.52	638.08	769.41	1.66	325	41	615	41
	W	$w = 8$	2717.28	4820	5820	1.11	2917	31	5355	31
	TSW	$w = 1, \delta = 1$	172.67	541.45	396.56	4.75	77	93	77	93
	TSW	$w = 1, \delta = 1.1$	172.29	898.22	376.62	4.75	69	93	69	93
	TSW	$w = 2, \delta = 1$	166.51	530.83	372.93	2.65	117	57	117	57
	TSW	$w = 2, \delta = 1.1$	166.22	888.55	351.37	2.65	105	57	105	57
	TSW	$w = 4, \delta = 1$	312.49	1090.00	650.82	1.58	292	39	292	39
	TSW	$w = 4, \delta = 1.1$	312.64	1670.00	602.75	1.58	263	39	263	39
	TSW	$w = 8, \delta = 1$	2501.01	9760.00	4900.00	1.06	2550	30	2550	30
	TSW	$w = 8, \delta = 1.1$	2499.97	14570.00	4320.00	1.06	2295	30	2295	30
Lifetime $L = 2^{20}$	W	$w = 1$	780.89	362.44	418.31	5.03	82	99	158	99
	W	$w = 2$	705.42	336.30	400.60	2.81	122	61	237	61
	W	$w = 4$	1353.18	617.48	746.28	1.72	326	43	615	43
	W	$w = 8$	11122.95	4981.20	6039.40	1.34	2917	35	5355	35
	TSW	$w = 1, \delta = 1$	752.57	520.42	401.32	4.81	77	95	77	95
	TSW	$w = 1, \delta = 1.1$	731.79	844.01	381.23	4.81	69	95	69	95
	TSW	$w = 2, \delta = 1$	667.76	527.17	379.56	2.7	117	59	117	59
	TSW	$w = 2, \delta = 1.1$	668.14	853.66	354.09	2.7	105	59	105	59
	TSW	$w = 4, \delta = 1$	1249.52	1057.40	661.61	1.64	292	41	292	41
	TSW	$w = 4, \delta = 1.1$	1248.35	1600.00	603.65	1.64	263	41	263	41
	TSW	$w = 8, \delta = 1$	9972.32	9509.50	4870.60	1.27	2550	34	2550	34
	TSW	$w = 8, \delta = 1.1$	9927.97	14271.00	4358.60	1.27	2295	34	2295	34

\* Assuming Poseidon2 is used, 128-bit classical security



# Efficiency

	Encoding	Parameters	Gen [s]	Sign [ $\mu$ s]	Ver [ $\mu$ s]	Sig [KiB]	$\pi_{16}$ AC	$\pi_{24}$ AC	$\pi_{16}$ WC	$\pi_{24}$ WC
Lifetime $L = 2^{18}$	W	$w = 1$	179.01	362.59	416.54	4.97	81	97	158	97
	W	$w = 2$	168.19	350.04	408.67	2.75	122	59	237	59
	W	$w = 4$	330.52	638.08	769.41	1.66	325	41	615	41
	W	$w = 8$	2717.28	4820	5820	1.11	2917	31	5355	31
	TSW	$w = 1, \delta = 1$	172.67	541.45	396.56	4.75	77	93	77	93
	TSW	$w = 1, \delta = 1.1$	172.29	898.22	376.62	4.75	69	93	69	93
	TSW	$w = 2, \delta = 1$	166.51	530.83	372.93	2.65	117	57	117	57
	TSW	$w = 2, \delta = 1.1$	166.22	888.55	351.37	2.65	105	57	105	57
	TSW	$w = 4, \delta = 1$	312.49	1090.00	650.82	1.58	292	39	292	39
	TSW	$w = 4, \delta = 1.1$	312.64	1670.00	602.75	1.58	263	39	263	39
	TSW	$w = 8, \delta = 1$	2501.01	9760.00	4900.00	1.06	2550	30	2550	30
	TSW	$w = 8, \delta = 1.1$	2499.97	14570.00	4320.00	1.06	2295	30	2295	30
Lifetime $L = 2^{20}$	W	$w = 1$	780.89	362.44	418.31	5.03	82	99	158	99
	W	$w = 2$	705.42	336.30	400.60	2.81	122	61	237	61
	W	$w = 4$	1353.18	617.48	746.28	1.72	326	43	615	43
	W	$w = 8$	11122.95	4981.20	6039.40	1.34	2917	35	5355	35
	TSW	$w = 1, \delta = 1$	752.57	520.42	401.32	4.81	77	95	77	95
	TSW	$w = 1, \delta = 1.1$	731.79	844.01	381.23	4.81	69	95	69	95
	TSW	$w = 2, \delta = 1$	667.76	527.17	379.56	2.7	117	59	117	59
	TSW	$w = 2, \delta = 1.1$	668.14	853.66	354.09	2.7	105	59	105	59
	TSW	$w = 4, \delta = 1$	1249.52	1057.40	661.61	1.64	292	41	292	41
	TSW	$w = 4, \delta = 1.1$	1248.35	1600.00	603.65	1.64	263	41	263	41
	TSW	$w = 8, \delta = 1$	9972.32	9509.50	4870.60	1.27	2550	34	2550	34
	TSW	$w = 8, \delta = 1.1$	9927.97	14271.00	4358.60	1.27	2295	34	2295	34

\* Assuming Poseidon2 is used, 128-bit classical security



# Outline

The Problem

Overall Paradigm

Signature Design

Next Steps

# Outline

The Problem

Overall Paradigm

Signature Design

Next Steps

# Next Steps

# Next Steps

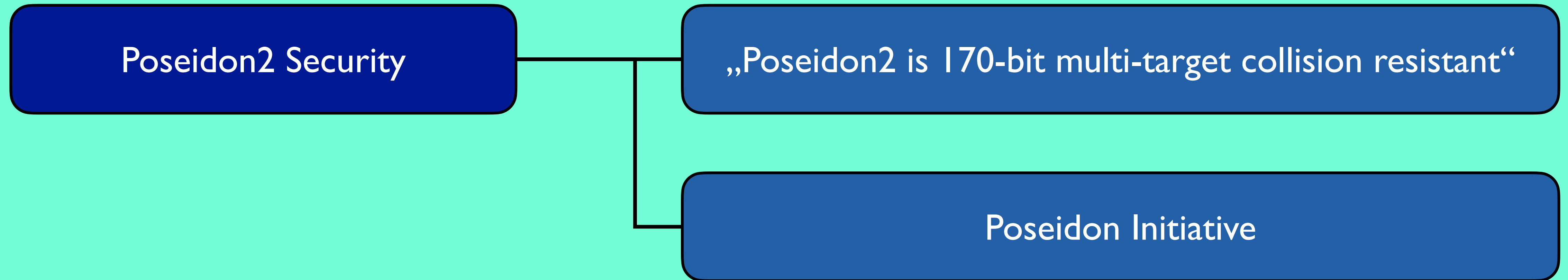
Poseidon2 Security

# Next Steps

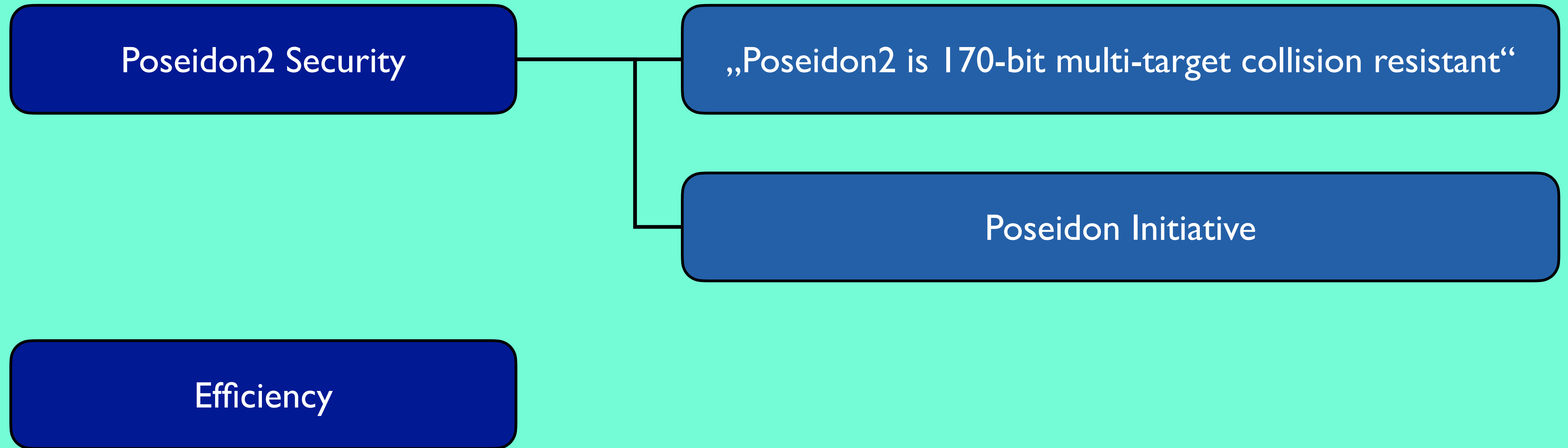
Poseidon2 Security

„Poseidon2 is 170-bit multi-target collision resistant“

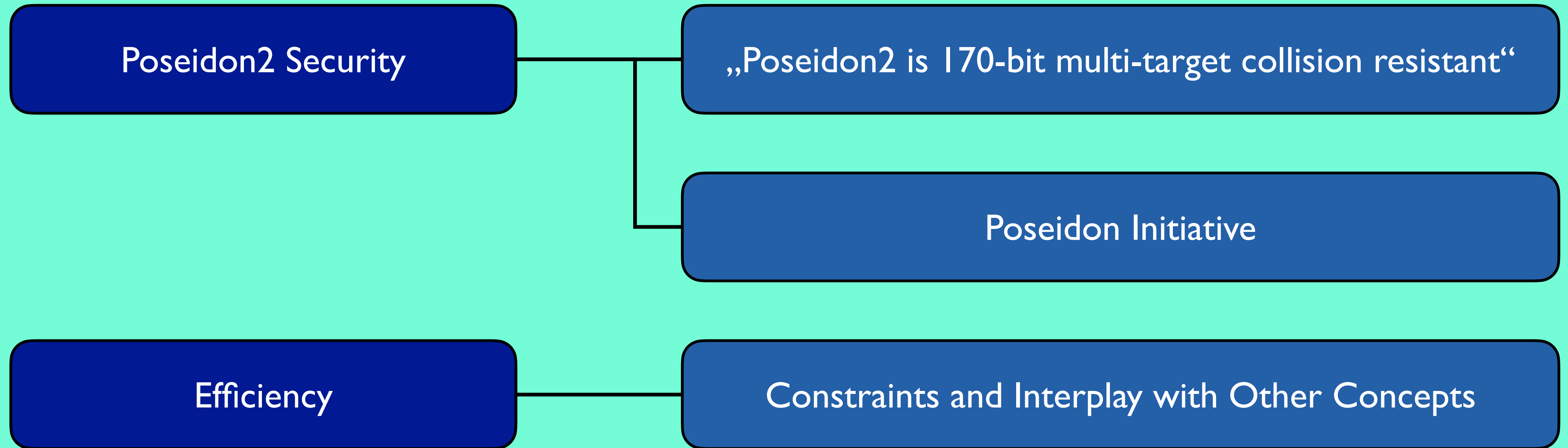
# Next Steps



# Next Steps

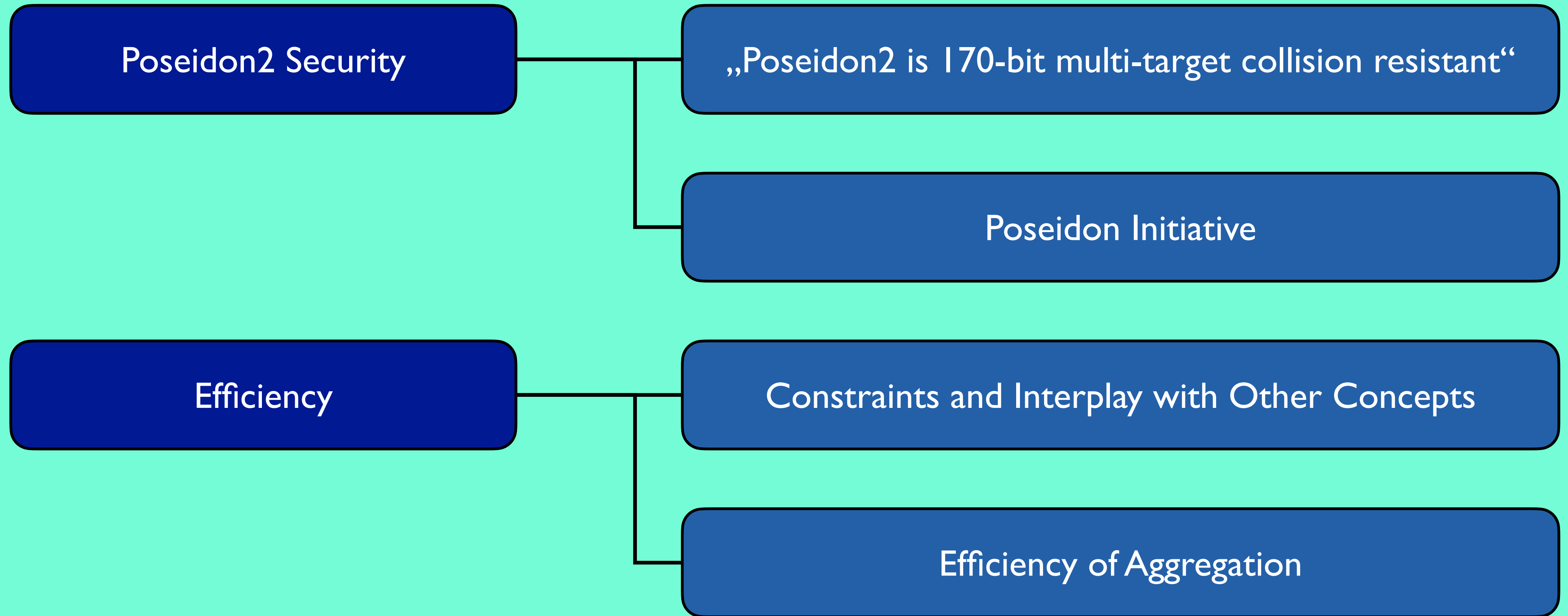


# Next Steps





# Next Steps



# Summary

# Summary

Post-Quantum Multi-Signatures for Ethereum

# Summary

Post-Quantum Multi-Signatures for Ethereum

Our Work

XMSS Variants + Succinct Argument

# Summary

## Post-Quantum Multi-Signatures for Ethereum

### Our Work

XMSS Variants + Succinct Argument

Tight Proofs without RO

# Summary

## Post-Quantum Multi-Signatures for Ethereum

### Our Work

XMSS Variants + Succinct Argument

Tight Proofs without RO

Hash Function  
Security Requirements

# Summary

## Post-Quantum Multi-Signatures for Ethereum

### Our Work

XMSS Variants + Succinct Argument

Tight Proofs without RO

Hash Function  
Security Requirements

### Next Steps

# Summary

## Post-Quantum Multi-Signatures for Ethereum

### Our Work

XMSS Variants + Succinct Argument

Tight Proofs without RO

Hash Function  
Security Requirements

### Next Steps

Is Poseidon2 secure enough?



# Summary

## Post-Quantum Multi-Signatures for Ethereum

### Our Work

XMSS Variants + Succinct Argument

Tight Proofs without RO

Hash Function  
Security Requirements

### Next Steps

Is Poseidon2 secure enough?

Is this efficient enough?

# Thank you!



Paper

### Hash-Based Multi-Signatures for Post-Quantum Ethereum

Justin Drake<sup>1</sup>

Dmitry Khovratovich<sup>1</sup>  
Benedikt Wagner<sup>1</sup>

Mikhail Kudinov<sup>2</sup>

January 14, 2025

<sup>1</sup> Ethereum Foundation  
`{justin.drake,dmitry.khovratovich,benedikt.wagner}@ethereum.org`

<sup>2</sup> Eindhoven University of Technology  
`mishel.kudinov@gmail.com`

#### Abstract

With the threat posed by quantum computers on the horizon, systems like Ethereum must transition to cryptographic primitives resistant to quantum attacks. One of the most critical of these primitives is the non-interactive multi-signature scheme used in Ethereum’s proof-of-stake consensus, currently implemented with BLS signatures. This primitive enables validators to independently sign blocks, with their signatures then publicly aggregated into a compact aggregate signature.

In this work, we introduce a family of hash-based signature schemes as post-quantum alternatives to BLS. We consider the folklore method of aggregating signatures via (hash-based) succinct arguments, and our work is focused on instantiating the underlying signature scheme. The proposed schemes are variants of the XMSS signature scheme, analyzed within a novel and unified framework. While being generic, this framework is designed to minimize security loss, facilitating efficient parameter selection. A key feature of our work is the avoidance of random oracles in the security proof. Instead, we define explicit standard model requirements for the underlying hash functions. This eliminates the paradox of simultaneously treating hash functions as random oracles and as explicit circuits for aggregation. Furthermore, this provides cryptanalysts with clearly defined targets for evaluating the security of hash functions. Finally, we provide recommendations for practical instantiations of hash functions and concrete parameter settings, supported by known and novel heuristic bounds on the standard model properties.

Code