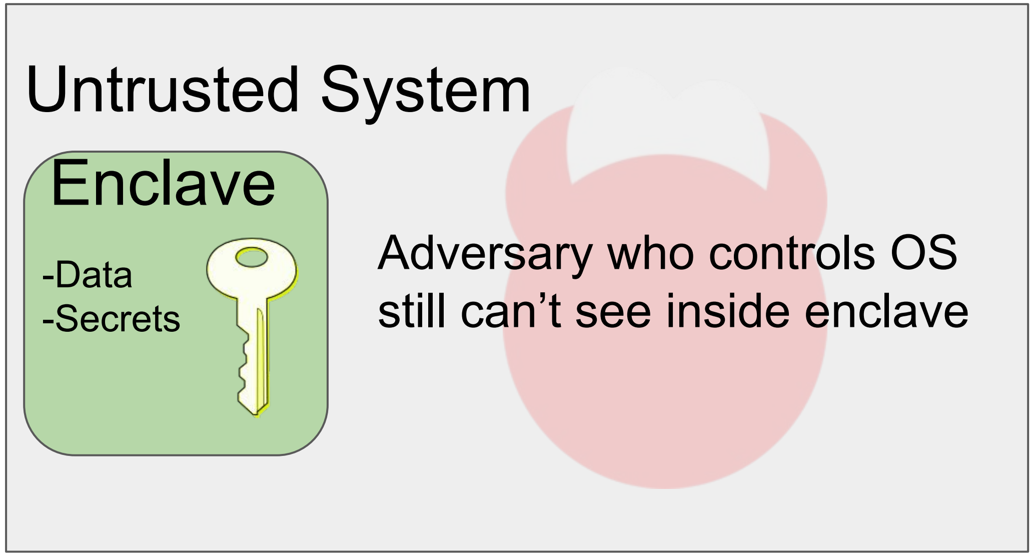
Post-Quantum EPID Signatures from Symmetric Primitives

Dan Boneh Saba Eskandarian Ben Fisch

Hardware Enclaves

A trusted component in an untrusted system

- Protected memory isolates enclave from compromised OS



Untrusted System

Enclave

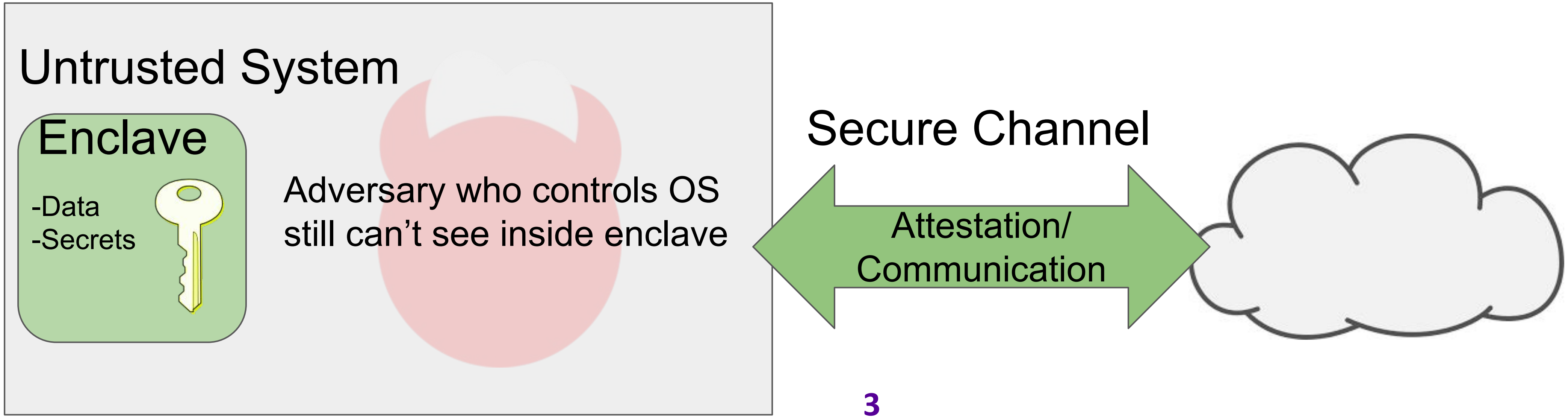
-Data
-Secrets

Adversary who controls OS still can’t see inside enclave

Hardware Enclaves

A trusted component in an untrusted system

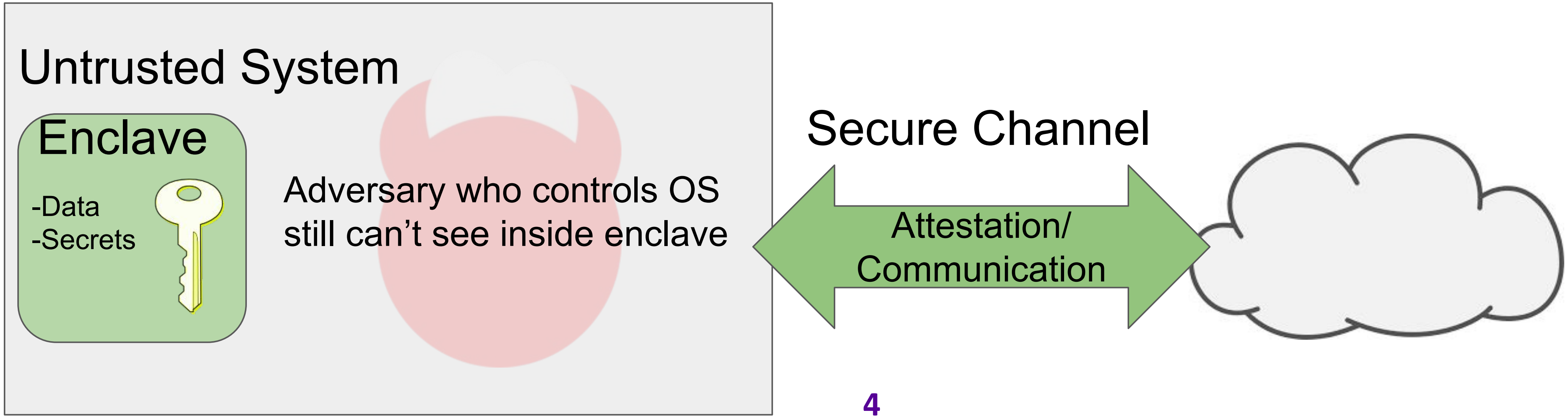
- Protected memory isolates enclave from compromised OS
- Proves authenticity via a process called *attestation*



Hardware Enclaves

A trusted component in an untrusted system

- Protected memory isolates enclave from compromised OS
- Proves authenticity via a process called *attestation*
  - Is it “post-quantum” secure?



Untrusted System

Enclave

-Data
-Secrets

Adversary who controls OS
still can’t see inside enclave

Secure Channel

Attestation/
Communication

EPID Signatures [BL09]

Group signature-like primitive that provides two properties:

1. Signatures from any member of a group are indistinguishable from each other
2. Users can have their credentials revoked either by a blacklisted key or a blacklisted signature

Intel's EPID signature scheme relies on pairings and is not post-quantum secure

EPID Signatures [BL09]

$\mathtt{sk}_{\mathtt{i}}$, $\mathtt{cert}_{\mathtt{i}}$←`Join(...)` - interactive protocol between group member and manager to join group

σ←`Sign(gpk,`$\mathtt{sk}_{\mathtt{i}}$`,`$\mathtt{cert}_{\mathtt{i}}$`,m,SIG-RL)` - any user who has joined can sign a message anonymously as a group member

`1/0`←`Verify(gpk,m,KEY-RL,SIG-RL,`σ`)` - signatures only verify if signed by a valid, unrevoked group member

`KEY-RL’`←`RevokeKey(KEY-RL,`$\mathtt{sk}_{\mathtt{i}}$`)` - revoke a group member by key

`SIG-RL’`←`RevokeSig(SIG-RL,`σ`)` - revoke a group member by signature

Security properties: **Anonymity** and **Unforgeability**

EPID Signatures [BL09]

$\texttt{sk}_i, \texttt{cert}_i \leftarrow \texttt{Join}(\ldots)$ - interactive protocol between group member and manager to join group

$\sigma \leftarrow \texttt{Sign}(\texttt{gpk}, \texttt{sk}_i, \texttt{cert}_i, \texttt{m}, \texttt{SIG-RL})$ - any user who has joined can sign a message anonymously as a group member

$\texttt{1/0} \leftarrow \texttt{Verify}(\texttt{gpk}, \texttt{m}, \texttt{KEY-RL}, \texttt{SIG-RL}, \sigma)$ - signatures only verify if signed by a valid, unrevoked group member

$\texttt{KEY-RL}' \leftarrow \texttt{RevokeKey}(\texttt{KEY-RL}, \texttt{sk}_i)$ - revoke a group member by key

$\texttt{SIG-RL}' \leftarrow \texttt{RevokeSig}(\texttt{SIG-RL}, \sigma)$ - revoke a group member by signature

Security properties: **Anonymity** and **Unforgeability**

Our design goal: post-quantum security from **symmetric primitives only**

Picnic Signatures [CDGORRSZ17]

Uses ZKB++ MPC-in-the-head type proof system [IKOS07, GMO16]

i.e. proof of knowledge from symmetric primitives

High-level idea: Signature is proof of knowledge of preimage of a one-way function

e.g. I know sk such that f(sk)=y

Our Basic Approach [BMW03,CG04]

Join

User generates pk, sk

Group manager signs pk to form cert

Sign

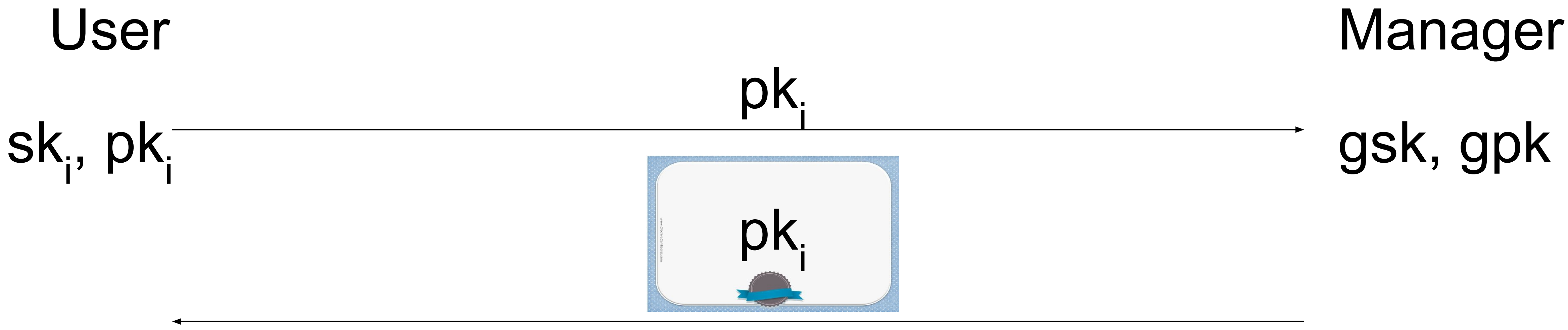
User signs message with sk

User publishes proof of knowledge of signature as σ

Additionally need to support revocation

Our Basic Approach [BMW03,CG04]

Join



User

Manager

sk_i, pk_i

pk_i

gsk, gpk

pk_i

Sign

s = Sign(sk_i, m)

Proof of Knowledge: I have a certificate on a key *sk** and a signature *s* on message *m* signed with *sk**

Post-Quantum EPID Signature

Join

User

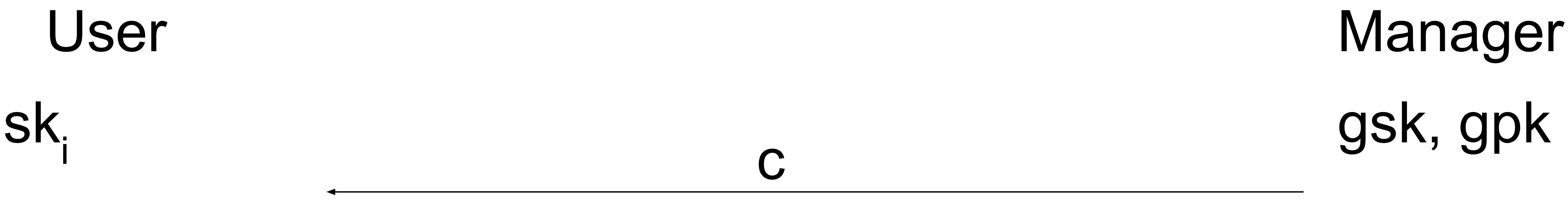
Manager

sk_i

gsk, gpk

Post-Quantum EPID Signature

Join



User

sk_i

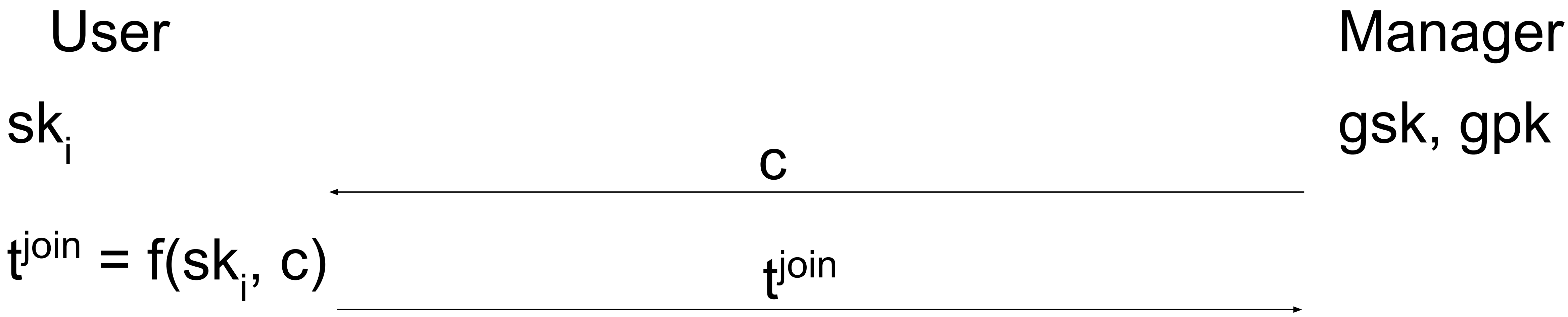
Manager

gsk, gpk

c

Post-Quantum EPID Signature

Join



User

Manager

sk_i

gsk, gpk

c

$t^{join} = f(sk_i, c)$

t^{join}

Post-Quantum EPID Signature

Join

User

Manager

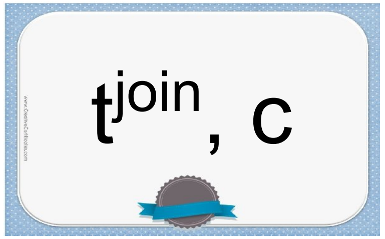
sk_i

gsk, gpk

$\xleftarrow{\quad c \quad}$

$t^{join} = f(sk_i, c)$

$\xrightarrow{\quad t^{join} \quad}$



t^{join}, c

$\longleftarrow$

Post-Quantum EPID Signature

Sign

$r \leftarrow \{0,1\}^\lambda$

$t = f(sk_i, r), r$

Post-Quantum EPID Signature

Sign

$r \leftarrow \{0,1\}^{\lambda}$

$t = f(sk_i, r), r$

Proof of Knowledge:

1. I know a valid certificate for t^{join}, c

Post-Quantum EPID Signature

Sign

$r \leftarrow \{0,1\}^\lambda$

$t = f(sk_i, r), r$

Proof of Knowledge:

1. I know a valid certificate for t^{join}, c
2. I know sk_i such that $t = f(sk_i, r)$ and $t^{join} = f(sk_i, c)$

Post-Quantum EPID Signature

Sign

$r \leftarrow \{0,1\}^\lambda$

$t = f(sk_i, r), r$

Proof of Knowledge:

1. I know a valid certificate for t^{join}, c
2. I know sk_i such that $t = f(sk_i, r)$ and $t^{join} = f(sk_i, c)$
3. There is no signature in SIG-RL such that $f(sk_i, r')=t'$

publish proof and t as signature

Instantiation

Need	Choices
Zero Knowledge PoK	ZKB++, Ligero, zk-STARK
PRF/CRHF	
Post-Quantum Signature from symmetric primitives	

Instantiation

Need	**Choices**
Zero Knowledge PoK	**ZKB++**, Ligero, zk-STARK
PRF/CRHF	
Post-Quantum Signature from symmetric primitives	

Instantiation

Need	Choices
Zero Knowledge PoK	**ZKB++**, Ligero, zk-STARK
PRF/CRHF	AES, MiMC, LowMC
Post-Quantum Signature from symmetric primitives	

Instantiation

Need	**Choices**
Zero Knowledge PoK	**ZKB++**, Ligero, zk-STARK
PRF/CRHF	AES, MiMC, **LowMC**
Post-Quantum Signature from symmetric primitives	

Instantiation

Need	**Choices**
Zero Knowledge PoK	**ZKB++**, Ligero, zk-STARK
PRF/CRHF	AES, MiMC, **LowMC**
Post-Quantum Signature from symmetric primitives	Tree-based, SPHINCS, Fish

Instantiation

Need	**Choices**
Zero Knowledge PoK	**ZKB++**, Ligero, zk-STARK
PRF/CRHF	AES, MiMC, **LowMC**
Post-Quantum Signature from symmetric primitives	**Tree-based**, **SPHINCS**, Fish

Instantiation

Need	Choices
Zero Knowledge PoK	**ZKB++**, Ligero, zk-STARK
PRF/CRHF	AES, MiMC, **LowMC**
Post-Quantum Signature from symmetric primitives	**Tree-based**, **SPHINCS**, Fish

Post-quantum EPID signature size (group size 2^{30}):

Instantiation

Need	**Choices**
Zero Knowledge PoK	**ZKB++**, Ligero, zk-STARK
PRF/CRHF	AES, MiMC, **LowMC**
Post-Quantum Signature from symmetric primitives	**Tree-based**, **SPHINCS**, Fish

Post-quantum EPID signature size (group size 2^{30}): **217MB**

Way too big!! Culprit: signature verification inside PoK

Post-Quantum EPID Signature

Requires signature verification!
How can we remove this?

Sign

$r \leftarrow \{0,1\}^\lambda$

$t = f(sk_i, r), r$

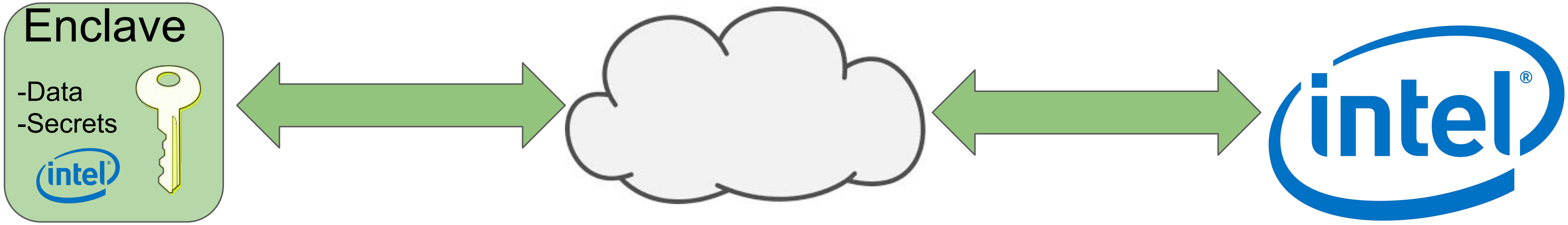
Proof of Knowledge:

1. **I know a valid certificate for t^{join}, c**
2. I know sk_i such that $t = f(sk_i, r)$ and $t^{join} = f(sk_i, c)$
3. There is no signature in SIG-RL such that $f(sk_i, r')=t'$

publish proof and t as signature

The Attestation Setting

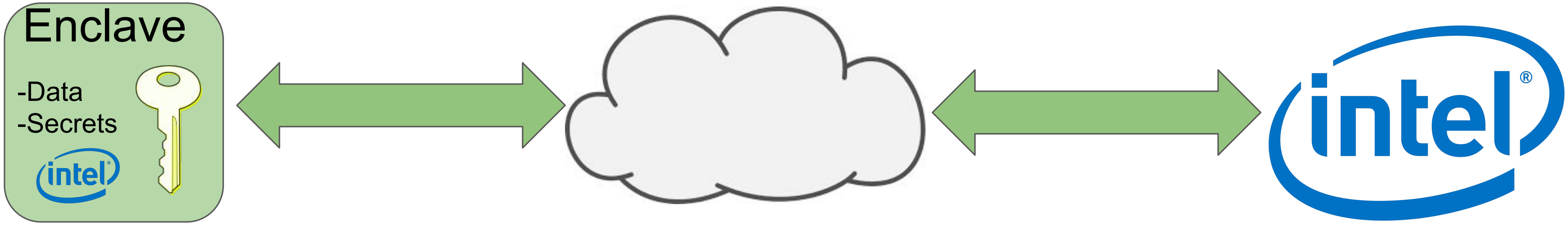
Each Intel SGX attestation involves contacting Intel, who verifies the attestation for you.



How can we leverage this to reduce signature sizes?

The Attestation Setting

Each Intel SGX attestation involves contacting Intel, who verifies the attestation for you.



How can we leverage this to reduce signature sizes?

Idea: If group manager has to be online, maybe it can update users’ certificates

User anonymity sets relative to last certificate update

Signatures for Attestation

Manager puts user credentials in a Merkle tree and signs root

Users get newest Merkle root/inclusion proof when they connect to the manager

Signatures for Attestation

Manager puts user credentials in a Merkle tree and signs root

Users get newest Merkle root/inclusion proof when they connect to the manager

Signature on Merkle tree root can be verified outside PoK

Only need much smaller Merkle inclusion proof inside PoK

Signatures for Attestation

$r \leftarrow \{0,1\}^\lambda$

$t = f(sk_i, r), r$

Proof of Knowledge:

1. **I know an inclusion proof for t^{join}, c**
2. I know sk_i such that $t = f(sk_i, r)$ and $t^{join} = f(sk_i, c)$
3. There is no signature in SIG-RL such that $f(sk_i, r')=t'$

publish proof, t, **and signed Merkle root** as signature

Similar to post-quantum Ring signatures of Derler et al [DRS17]

Signature Sizes

Group Size	RO Model*	QRO Model*
2^{7}	1.37MB	2.64MB
2^{10}	1.85MB	3.59MB
2^{20}	3.45MB	6.74MB
2^{30}	5.05MB	9.89MB
2^{40}	6.65MB	13.0MB

Potential application: large data transfer, e.g. streaming movies

*under ideal cipher assumption on LowMC