Subset-optimized BLS Multi-signature with Key Aggregation

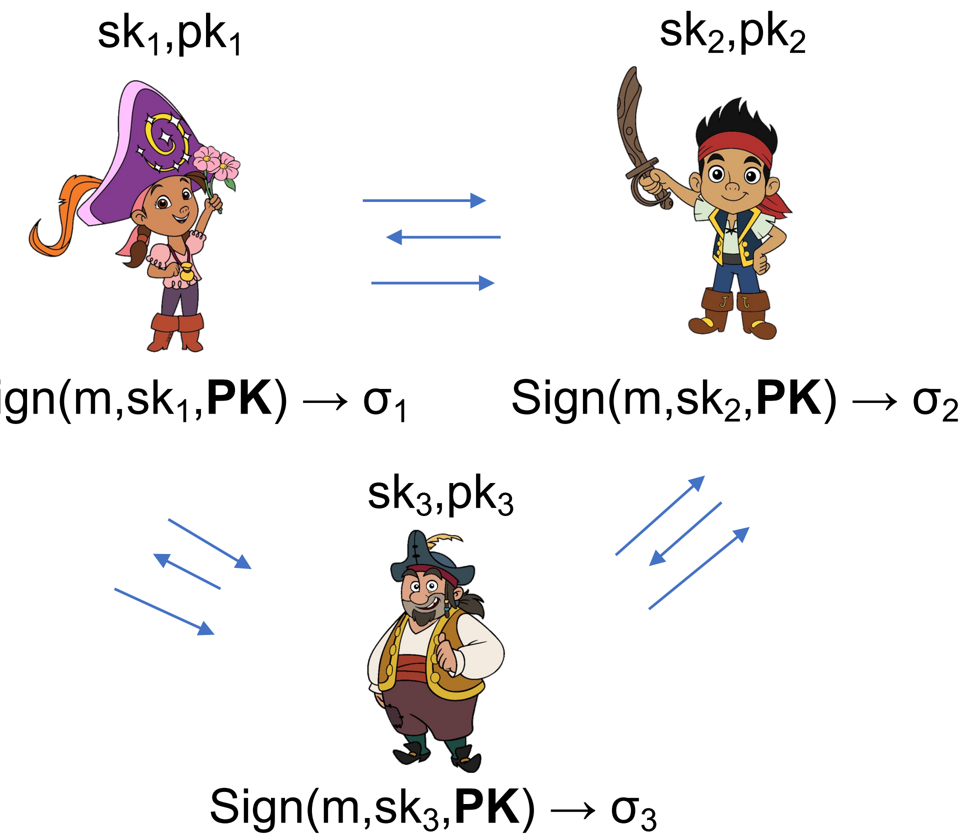
COSIC

F. Baldimtsi, K. Chalkias, F. Garrilot, J. Lindstrøm, B. Riva, A. Roy, M. Sedaghat, A. Sonnino, P. Waiwitlikhit, J. Wang

FC'24

MystenLabs

Multi-signatures:



Where $\mathbf{PK}=\{pk_1,pk_2,pk_3\}$

Output $\sigma = <\sigma_1,\sigma_2,\sigma_3>$ O(n) size 🚫

n signers produce a single signature on the **<u>same message</u> m**.

Multi-signatures:

sk_1,pk_1

$Sign(m,sk_1,\mathbf{PK}) \rightarrow \sigma_1$

sk_2,pk_2

$Sign(m,sk_2,\mathbf{PK}) \rightarrow \sigma_2$

sk_3,pk_3

$Sign(m,sk_3,\mathbf{PK}) \rightarrow \sigma_3$

n signers produce a single signature on the **same message m**.

Where $\mathbf{PK}=\{pk_1,pk_2,pk_3\}$

~~Output $\sigma = <\sigma_1,\sigma_2,\sigma_3>$~~ O(n) size 🚫

Create a **short** σ via:

- interactive protocol
- signature aggregation

efficient verification

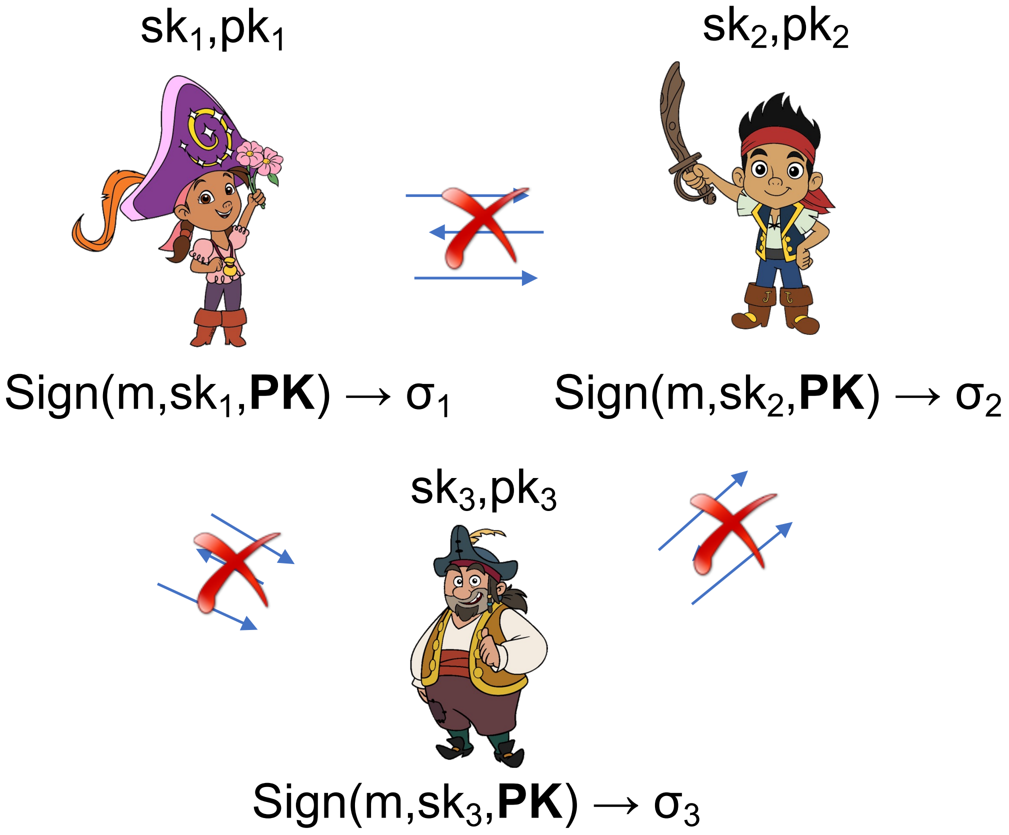
$Ver(\mathbf{PK},m,\sigma)=1$

Additional goal: Key Aggregation

$KAgg(pk_1,pk_2,pk_3) \rightarrow apk$

$Ver(apk,m,\sigma)=1$

Multi-signatures:



Security:

- Correctness
- Unforgeability (special attention to rogue key attacks)

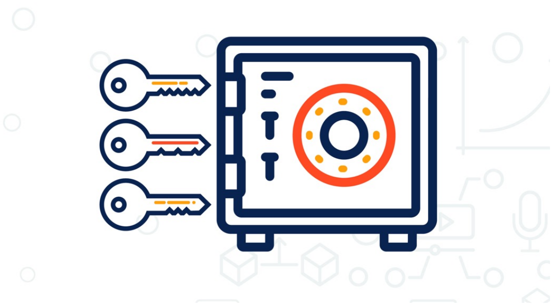
Constructions:

- A variety of constructions with various trade-offs, secure under different assumptions
- Our focus is BLS

n signers produce a single signature on the **<u>same message</u> m**.

Multi-signatures Applications:

Multi-user wallets



Collective Signing of Digital Certificates

Layer-2 protocols

BITCOIN LIGHTNING NETWORK

Block Validation in PoS/ permissioned ledgers

In this talk:

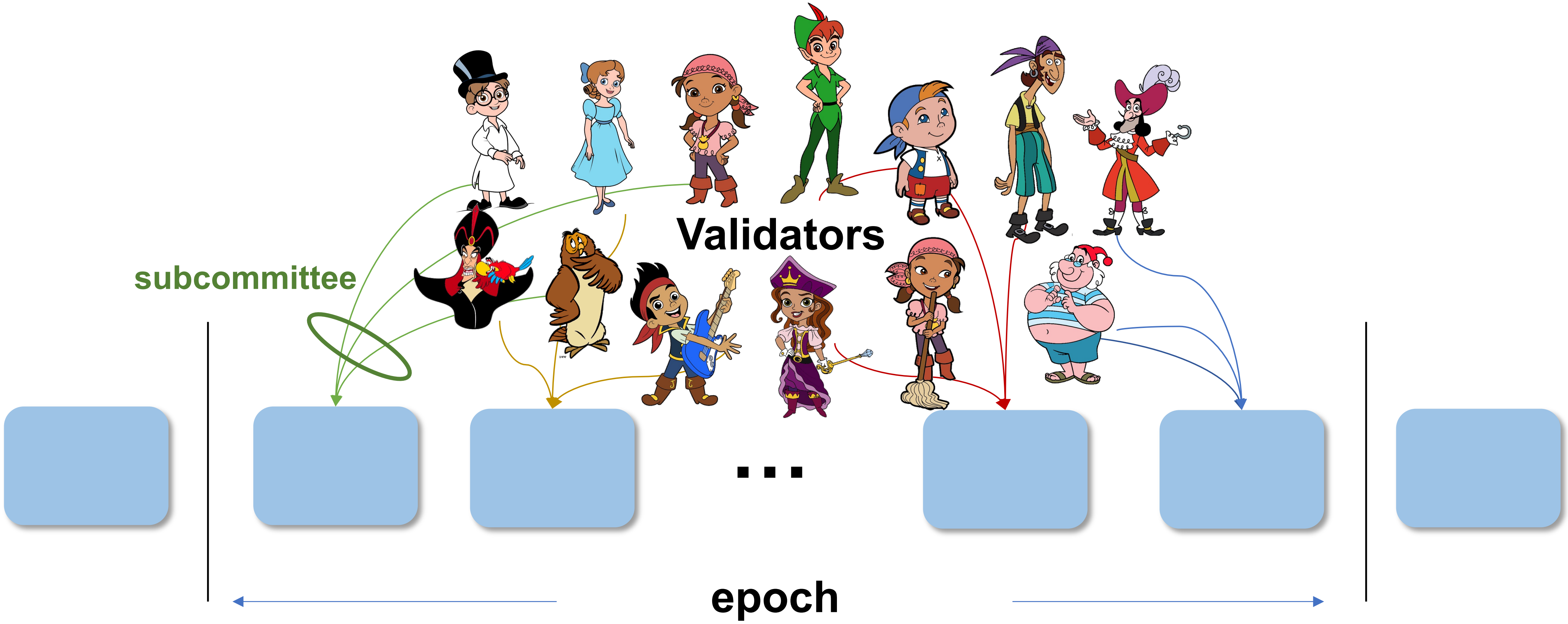
Multi-user wallets

Collective Signing of Digital Certificates

Layer-2 protocols

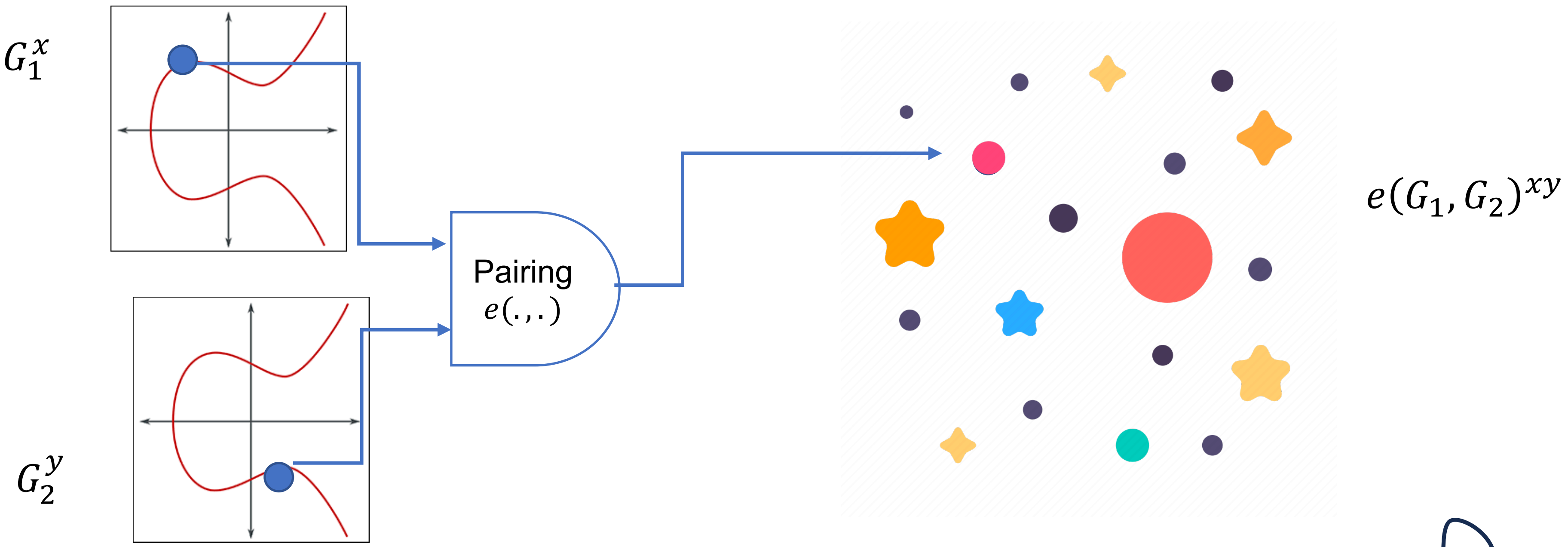
Block Validation in PoS/ permissioned ledgers

Multi-signatures in Proof-of-Stake:



- Fixed committee of **n** validators/epoch
- Subset/subcommittee of **k** validators multi-signs each block

BLS signature [BLS04]: A digital signature over bilinear groups*



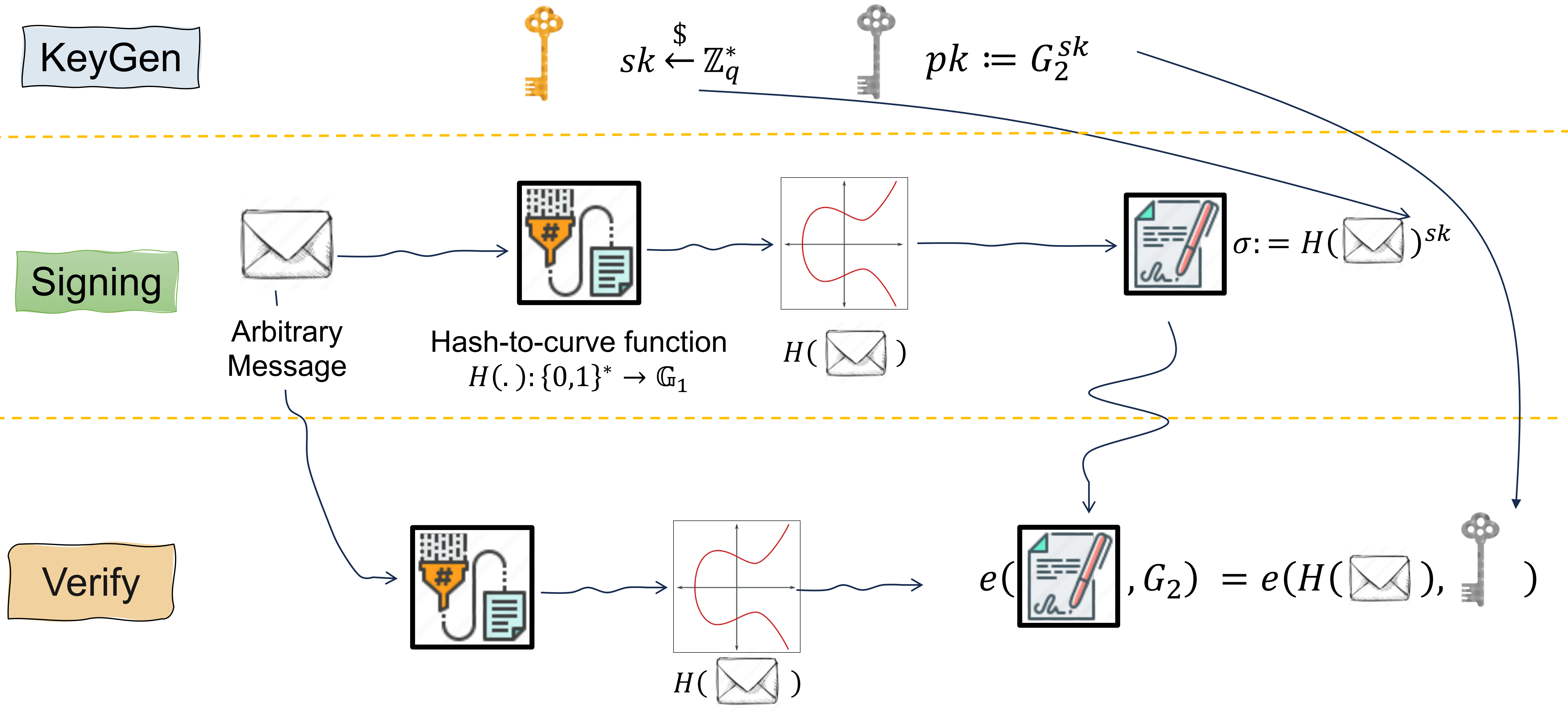
Source groups

Target group

* (Type-III) Bilinear Groups:

- There exists an efficient map e: $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$:
  - **Bilinearity**: $e\left(G_1^x, G_2^y\right) = e(G_1, G_2)^{xy}, \forall x, y \in \mathbb{Z}_q$
  - **Non-degenerate**: $e(G_1, G_2) \neq 1_{\mathbb{G}_T}$
  - $\mathbb{G}_1 =< G_1 >, \mathbb{G}_2 =< G_2 >, \mathbb{G}_T =< e(G_1, G_2) >$

BLS signature [BLS04]:



First Attempt: Rogue-key attack

$\sigma_{agg} \coloneqq \prod \sigma_i$

$apk \coloneqq \prod pk_i$

$e(\sigma_{agg}, G_2) = e(H(m), apk)$

$e(H(m)^{sk_3}, G_2) = e(H(m), apk)$

sk$_1$,pk$_1$

sk$_2$,pk$_2$

$sk_3, pk_3 = G_2^{sk_3}(pk_1)^{-1}(pk_2)^{-1}$

BLS Multi-signatures [BDN18]:

<u>Parameters</u> pp: Groups $\mathbb{G}_1$and $\mathbb{G}_2$ of same prime order q, with generators G_1 and G_2, and bilinear pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and a CRHF $H_1: \{0,1\}^* \to Z_q^*$

KeyGen(pp) → sk randomly picked from Z_q^*
pk= G_2^{sk}

KeyAgg($pk_1,\ldots,pk_k$) → $apk = \prod pk_i^{a_i}$ where
$a_i = H_1(\{pk_1, \ldots, pk_k\}, pk_i)$

Sign(m,sk) → Every validator: $\sigma_i = H(m)^{sk_i a_i}$
Aggregate:$\sigma = \prod \sigma_i$

In PoS, this process repeats for <u>each</u> subset of k validators

Verify(m,apk,σ) → Check if $e(\sigma, G_2) = e(H(m), apk)$

Motivation:

Run KeyAgg **<u>once</u>** per epoch for the **full set** of n committee members.

BLS Multi-signatures – Subset Optimized:

Parameters pp: Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of same prime order q, with generators G_1 and G_2, and bilinear pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and a CRHF $H_1: \{0,1\}^* \rightarrow Z_q^*$

n committee members, **k** validators per block

KeyGen(pp) → sk randomly picked from Z_q^*
pk= G_2^{sk}

KeyAgg($pk_1,\dots,pk_k$) → $apk = \prod pk_i^{a_i}$ where
$a_i = H_1(\{pk_1, \dots, pk_k\}, pk_i)$

Sign(m,sk) → Every validator: $\sigma_i = H(m)^{sk_i a_i}$
Aggregate: $\sigma = \prod \sigma_i$

Verify(m,apk,σ) → Check if $e(\sigma, G_2) = e(H(m), apk)$

[BDN'18]

KeyGen(pp) → sk randomly picked from Z_q^*
pk= G_2^{sk}

Beginning of epoch, all n committee members run:

KeyReRand: $pk_i^* = pk_i^{a_i}$ where $a_i = H_1(\{pk_1, \dots, pk_n\}, pk_i)$
$sk_i^* = sk_i\, a_i$

once

KeyAgg($pk^*_1,\dots,pk^*_k$) → $apk = \prod pk_i^*$

Sign(m,sk_i^*) → Every validator: $\sigma_i = H(m)^{sk_i^*}$
Aggregate: $\sigma = \prod \sigma_i$

Verify(m,apk,σ) → Check if $e(\sigma, G_2) = e(H(m), apk)$

Our scheme

BLS Multi-signatures – Subset Optimized:

Parameters pp: Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of same prime order q, with generators G_1 and G_2, and bilinear pairing $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and a CRHF $H_1: \{0,1\}^* \rightarrow Z_q^*$

n committee members, **k** validators per block

KeyGen(pp) → sk randomly picked from Z_q^*
pk= G_2^{sk}

KeyAgg($pk_1,\ldots,pk_k$) → $apk = \prod pk_i^{a_i}$ where
$a_i = H_1(\{pk_1, \ldots, pk_k\}, pk_i)$

Sign(m,sk) → Every validator: $\sigma_i = H(m)^{sk_i a_i}$
Aggregate: $\sigma = \prod \sigma_i$

Verify(m,apk,σ) → Check if $e(\sigma,G_2) = e(H(m),apk)$

[BDN'18]

KeyGen(pp) → sk randomly picked from Z_q^*
pk= G_2^{sk}

Beginning of epoch, all n committee members run:

KeyReRand: $pk_i^* = pk_i^{a_i}$ where $a_i = H_1(\{pk_1, \ldots, pk_n\}, pk_i)$
$sk_i^* = sk_i\, a_i$

saves k exponentiations per signature

KeyAgg($pk^*_1,\ldots,pk^*_k$) → $apk = \prod pk_i^*$

Sign(m,sk_i^*) → Every validator: $\sigma_i = H(m)^{sk_i^*}$
Aggregate: $\sigma = \prod \sigma_i$

Verify(m,apk,σ) → Check if $e(\sigma,G_2) = e(H(m),apk)$

SMSKR

q-EUF-Chosen Message Attack (EUF-CMA): standard definition

(pk,sk) ← KeyGen(pp)

pk, pp

M

σ

Signing Oracle

$Q_S \leftarrow Q_S \cup \{M\}$

$\sigma \leftarrow \text{Sign}(pp, sk, M)$

σ^*, M^*

Return 1 if:

1. Verify(pk, M^*, σ^*)=1
2. $M^* \notin Q_S$
3. $|Q_S| \leq q$

q-EUF-CMA for SMSKR: Weak and Strong

$(pk_0, sk_0) \leftarrow$ KeyGen(pp)

pk_0, pp

$\overrightarrow{PK_A}, I_0^*$

$PK = \overrightarrow{PK_A} \cup \{pk_0\}$

pk_0^*

(pk_0^*, sk_0^*)
$\leftarrow$ RandKey(pp, sk_0, PK)

M

σ

$Q_S \leftarrow Q_S \cup \{M\}$

$\sigma \leftarrow \mathrm{Sign}(\mathrm{pp}, sk_0^*, M)$

Signing Oracle

I^*, σ^*, M^*

Return 1 if:

1. Verify($\mathrm{apk}^*_{I^* \cup \{0\}}, M^*, \sigma^*$)=1
2. $M^* \notin Q_S$
3. $|Q_S| \le q$
4. $I^* = I_0^*$

Proving Security of Our Construction:

[BDN'18]: Multi-BLS is secure under CDH in ROM

Our Scheme

Proof 1: secure under CDH in ROM for a weak adversary

Proof 2: secure under DL in AGM+ROM with a 2^n security loss ☹

Proving Security of Our Construction:

[BDN'18]: Multi-BLS is secure under DH in ROM

Our Scheme

Proof 1: secure under DH in ROM for a weaker adversary

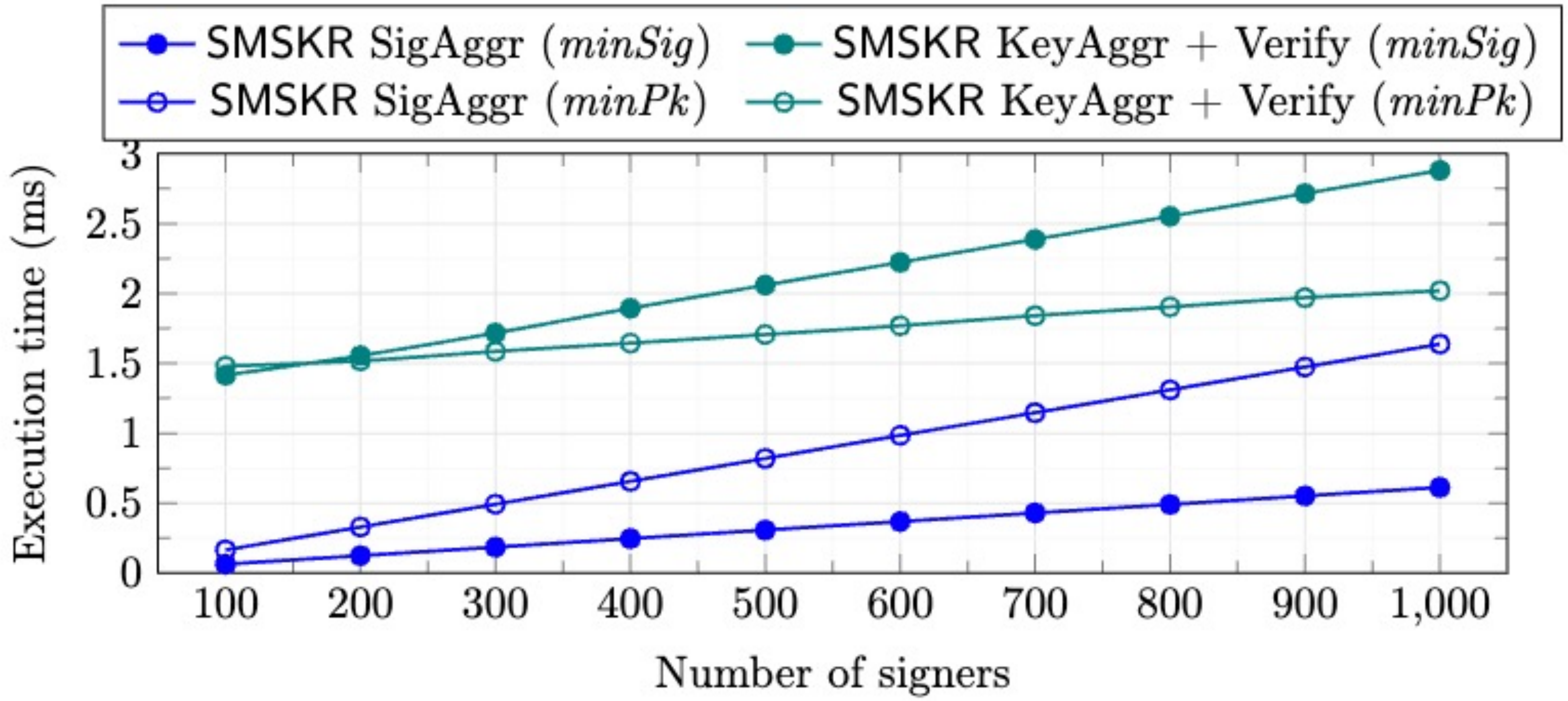
Proof 2: secure under DL+ RMSS in AGM+ROM

Random Modular Subset Sum (RMSS) assumption:
Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of integers and an integer target t, determine if there exists a subset $I \subseteq S$ that sums to the target t.

If the number of possible subsets is negligibly smaller than the size of the output space of H_1, then the probability of existence of a subset sum solution is negligible ☺

Implementation of our SMSKR:

- less than 0.2 ms to aggregate signatures
- and less than 1.5 ms to verify signatures in a setting with less than 100 signers



A product-ready implementation.
Over bls12-381 written in Rust
using blst library.

On a t3.medium AWS instance with 2 virtual CPUs (1 physical core) on a 2.5 GHz Intel Xeon Platinum 8259 and 4GB of RAM.

Conclusion and Open Problems:

- Multi-Signatures and applications to Proof-of-Stake.
- Subset-Optimized Multi-Signature with Key Randomization.
- Security properties and used proof techniques.
- Performance analysis.

Potential open questions and subsequent works:

1) Extend the concept of SMSKR to other multi-signatures like Schnorr, Musig, Musig2, PS.
2) Remove the RMSS assumption.

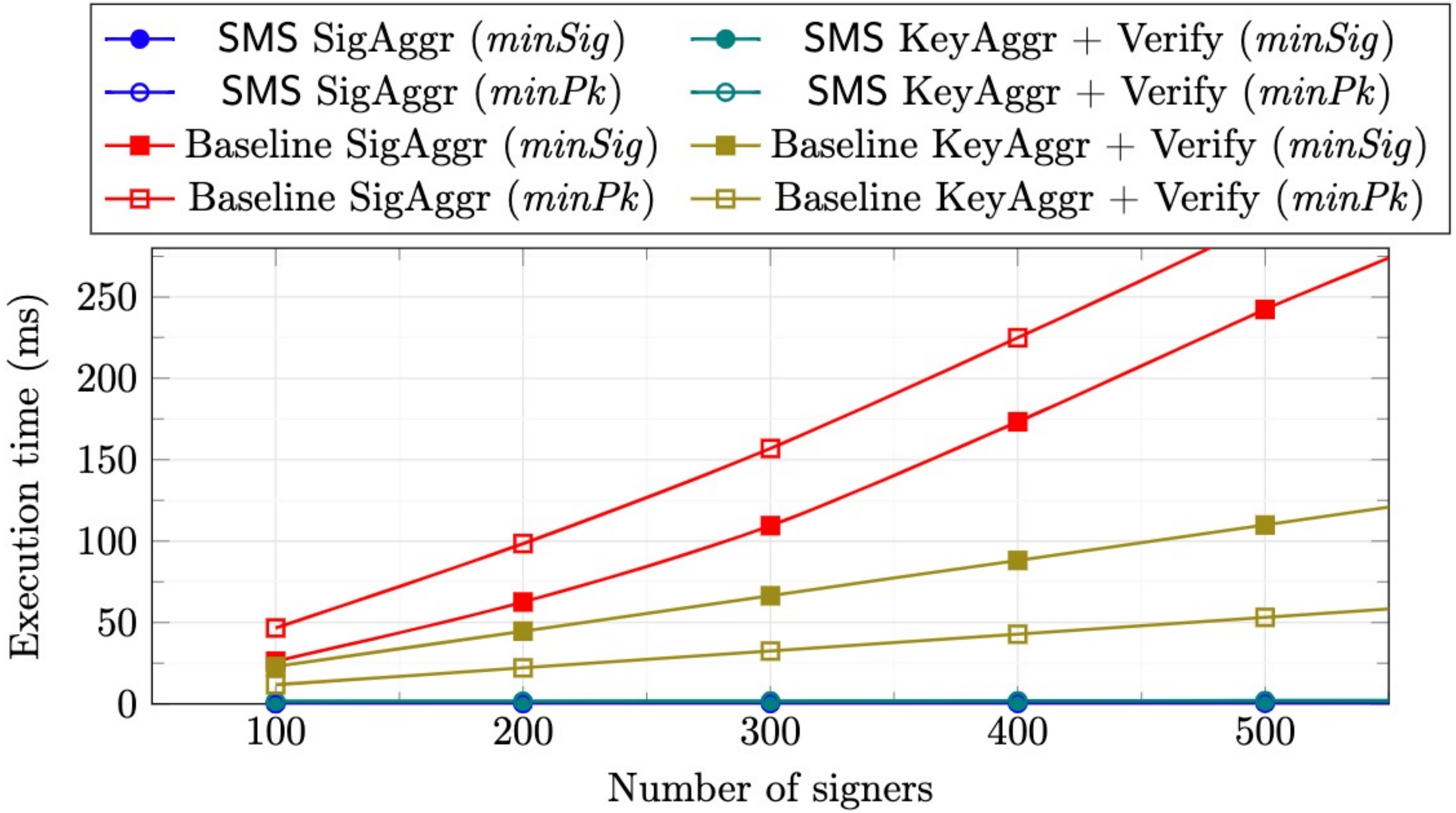
Thank you!

https://eprint.iacr.org/2023/498

The illustrations are credited to Disneyclips.

Baseline Comparisons:

Our SMSKR minSig and minPk implementations respectively, save 25 ms and 50 ms when compared to the baseline for aggregating 100 signatures!



Comparative performance of SMSKR with the baseline [BDN'18] on a low-end t3.medium AWS instance