Sequential Aggregate Signatures with Lazy Verification from Trapdoor Permutations

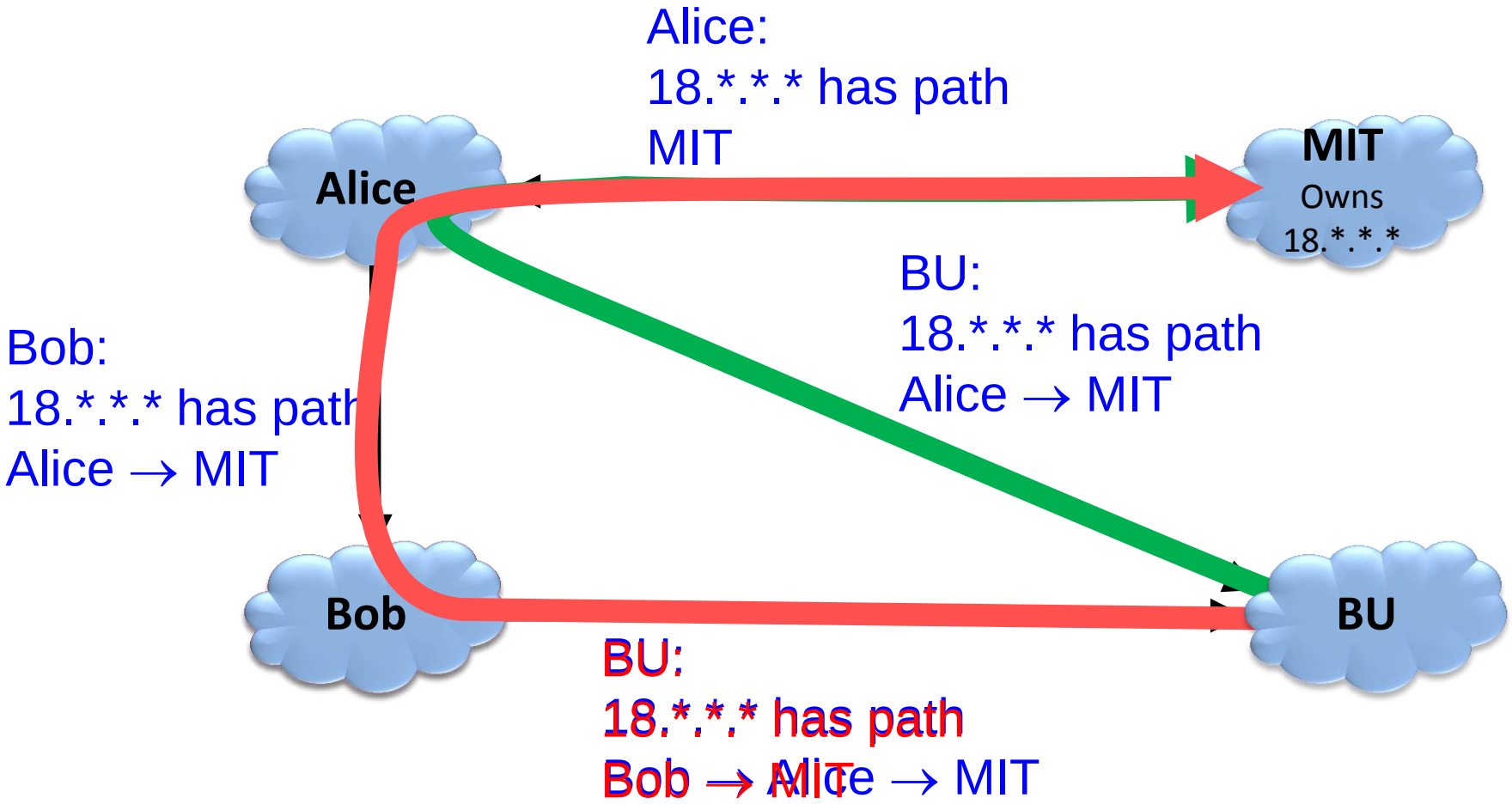
Kyle Brogle[1]
Sharon Goldberg[2]
Leo Reyzin[2]

[1]Stanford University; work done while at Boston University
[2]Boston University

Asiacrypt 2012
Beijing, China
6 December 2012

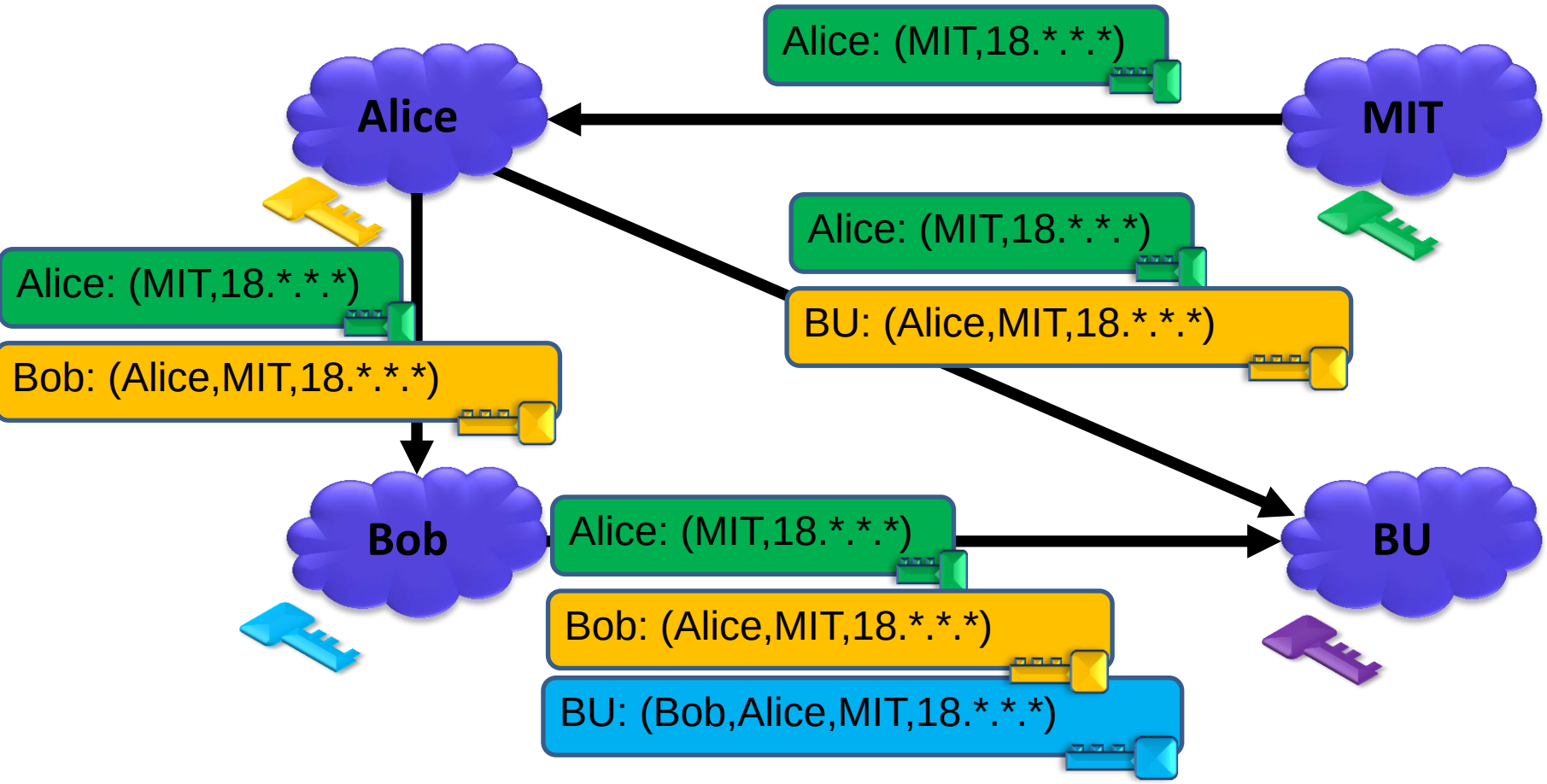
BGP

- Internet Routing between Autonomous Systems (ASes)



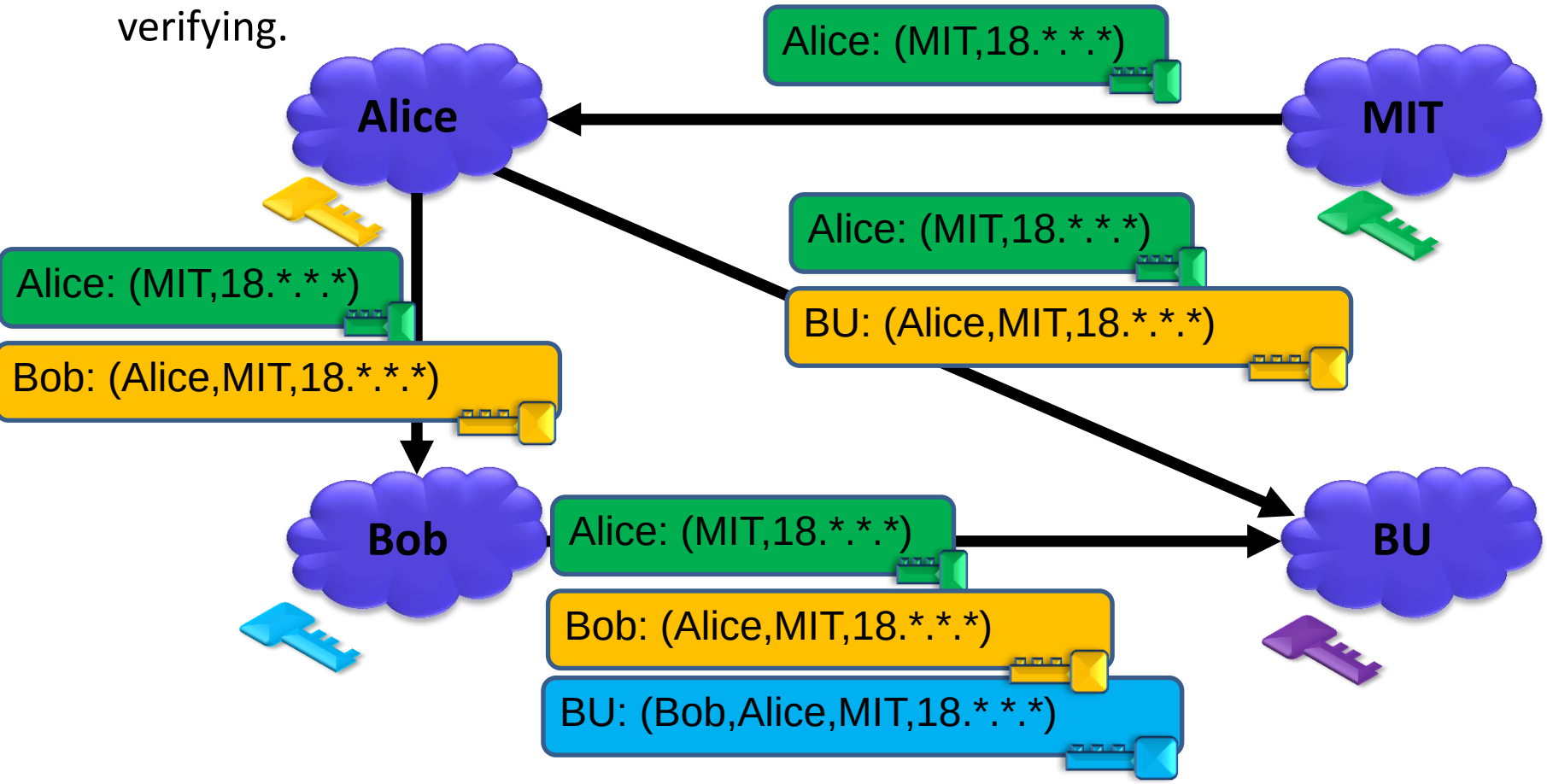
BGPSEC

- Sign announcements sent
- Include announcements you received for the path
- Problem: Announcements get big!

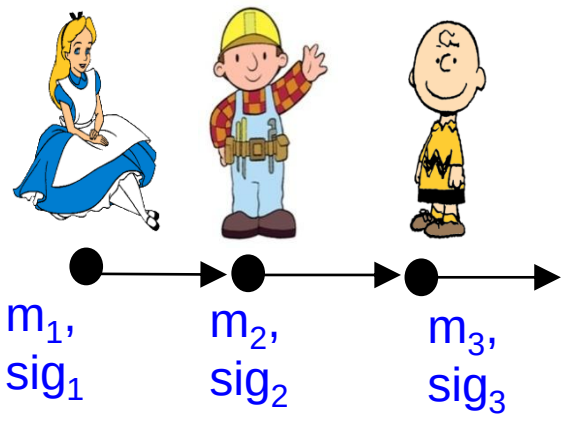


Lazy Verification

- Problem: To verify these signatures, routers need to retrieve and maintain ~40,000 public keys. Hard to do with low latency packet forwarding.
- Want to be able to defer verification of signatures time permits (but can't afford to defer sending announcements) Must be able to sign before verifying.



Sequential Aggregate Signatures



Best known aggregate signature scheme is BGLS
[Boneh-Gentry-Lynn-Shacham 03]

Based on Pairings over Elliptic Curves

- Has desired properties, but would be nice to have alternative from different assumptions.

[Fischlin-Lehmann-Schröder 11]

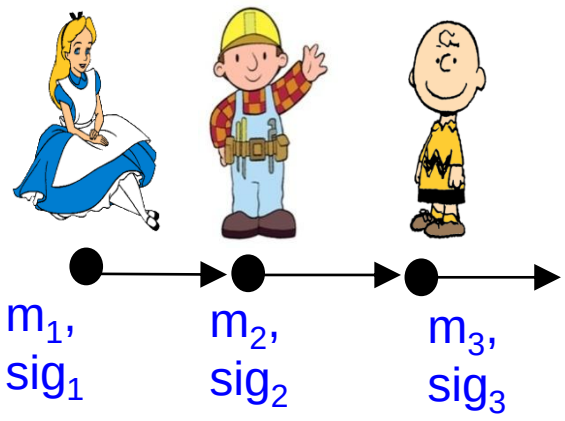
- Variant of BGLS with stronger security guarantees.
- Guarantees aren't needed in BGP, but are interesting in other contexts.

What about an alternative built from TDPs?

All known constructions without pairings:

- Don't allow for lazy verification
- Some operations using other signers' public keys

Sequential Aggregate Signatures



Sequential Aggregate Sigs can solve the problem of large announcements

- One signature instead of n

But can Sequential Aggregate Sigs also Handle lazy verification?

Two prior schemes from TDPs:

- By Lysyanskaya-Micali-Reyzin-Shacham (LMRS)
- By Neven

Both require verifying the aggregate-so-far before signing (no lazy verification), and both use other signer's public keys in signing operation.

Goal: Build a scheme from TDPs, removing requirement for verify before sign to allow for lazy verification.

- To do so, we will have to let sig grow by a small amount per signer (much less than growth in msg length)

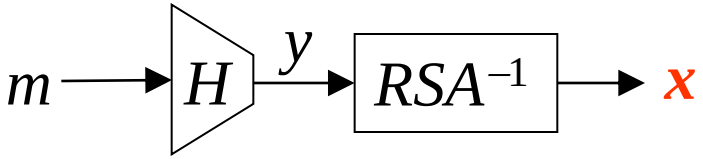
Previous Sequential Aggregate Signature Schemes

Full-Domain Hash RSA [Rivest-Shamir-Adleman 78, Bellare-Rogaway 93]

- Going to use RSA as an example for today, but can be done with any TDP.
- Hash function H (full RSA domain outputs; "random oracle").
- Public key $PK = (n, e)$. Secret key $SK = (n, d)$.

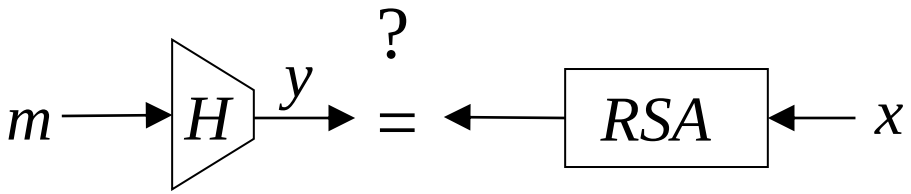
Signer:

- $y = H(m)$
- $\boldsymbol{x} = y^d \bmod n$



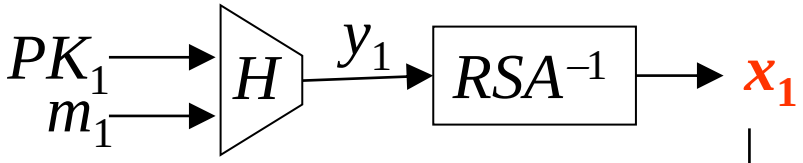
Verifier:

- $y = H(m)$
- $y \stackrel{?}{=} x^e \bmod n$

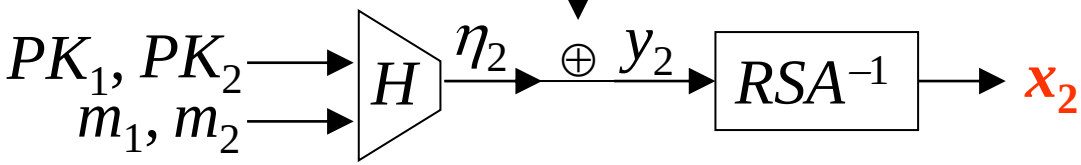


LMRS Signature Scheme [LMRS 04]

Signer 1:



Signer 2:



Steps for Signer 2:

- Check that PK_1 specifies permutation
- Verify x_1 using PK_1, m_1
- $\eta_2 = H\,(PK_1, PK_2, m_1, m_2)$
- $y_2 = \eta_2 \oplus x_1$
- $x_2 = y_2^{d_2} \bmod n_2$

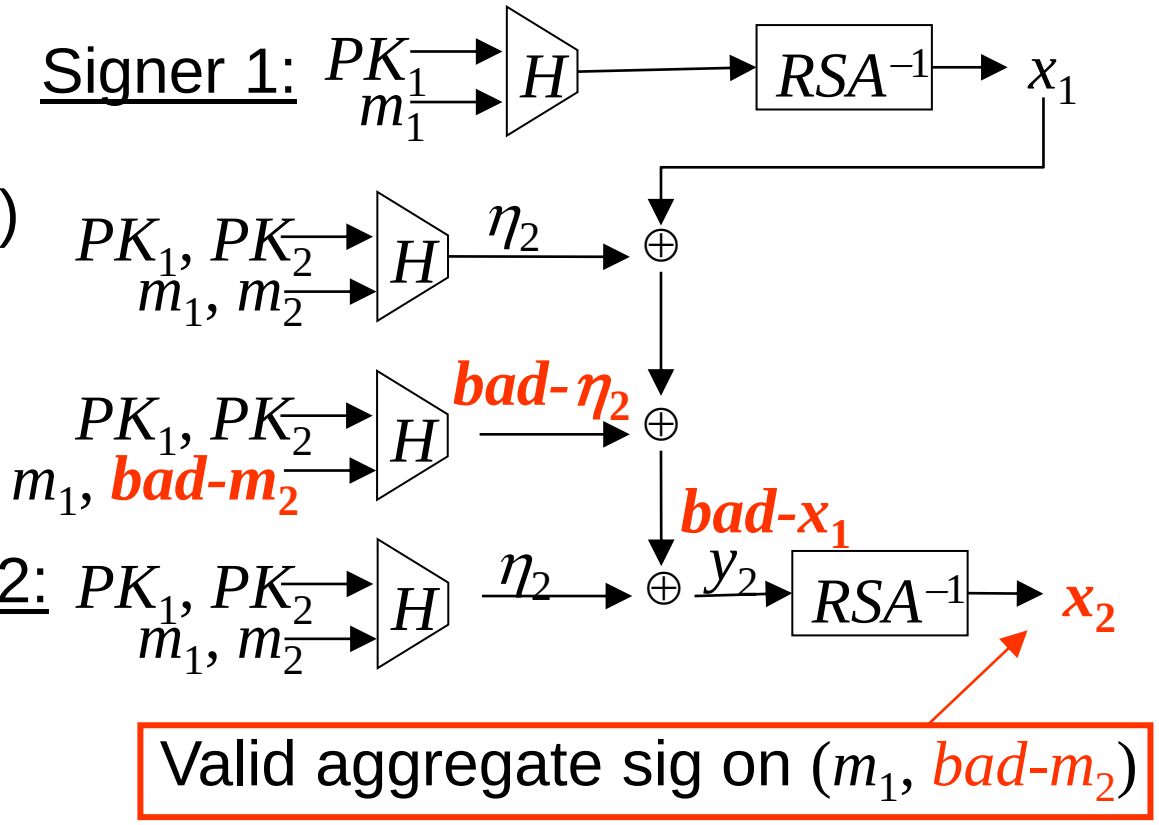
Possible to generate a malicious PK that doesn't specify a permutation.

Prevents Lazy Verification (Need to verify aggregate-so-far before you add your sig)

LMRS Fails Under Lazy Verification

Signer 2 wants to sign m_2

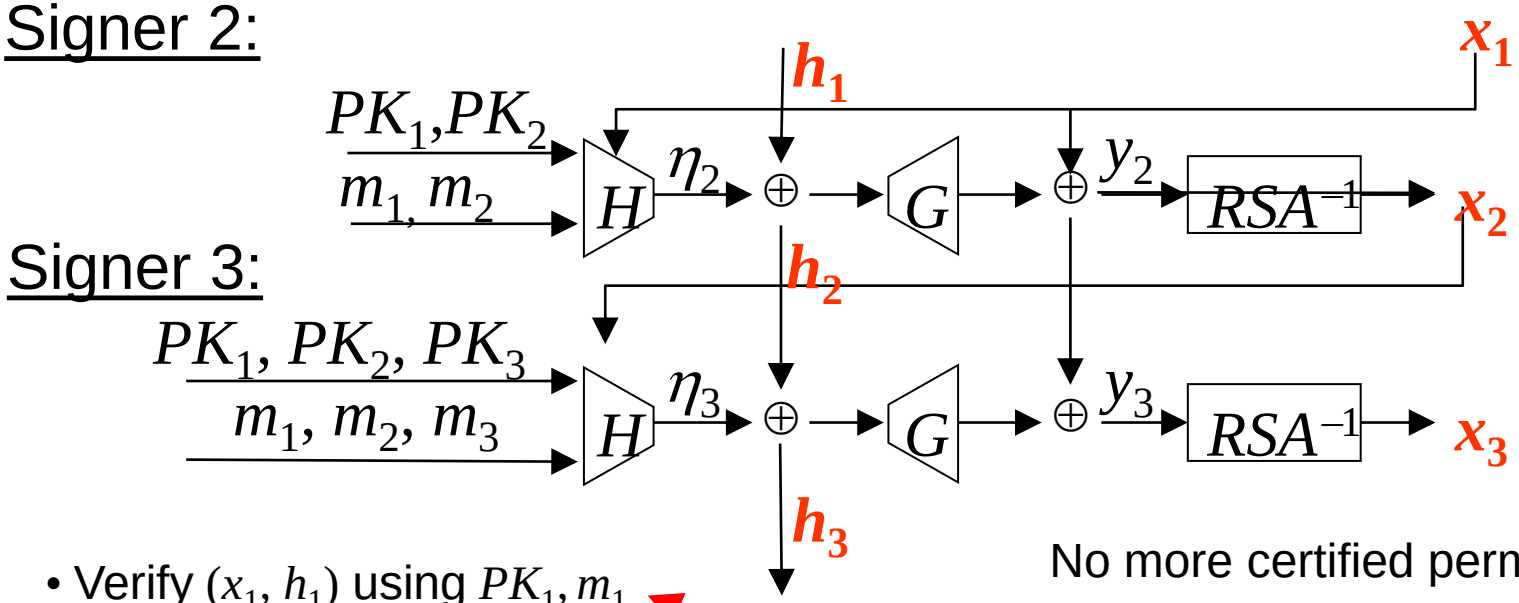
But Signer 1 wants to get a sig on *bad*-m_2 (Chosen Message Attack)



Neven Signature Scheme [Neven 08]

Hash functions H (short outputs), G (full RSA domain outputs)

Signature has two components: (x, h)



- Verify (x_1, h_1) using PK_1, m_1
- $\eta_2 = H\,(PK_1, PK_2, x_1, m_1, m_2)$
- $h_2 = \eta_2 \oplus h_1$
- $y_2 = G(h_2) \oplus x_1$
- $x_2 = y_2^{d_2} \bmod n_2$

No more certified permutations

Without verification, same "bad-m_2" attack works!
Will always work if signer i knows exactly what goes into RSA^{-1} for signer i+1

Need something to be out of previous signer's control!

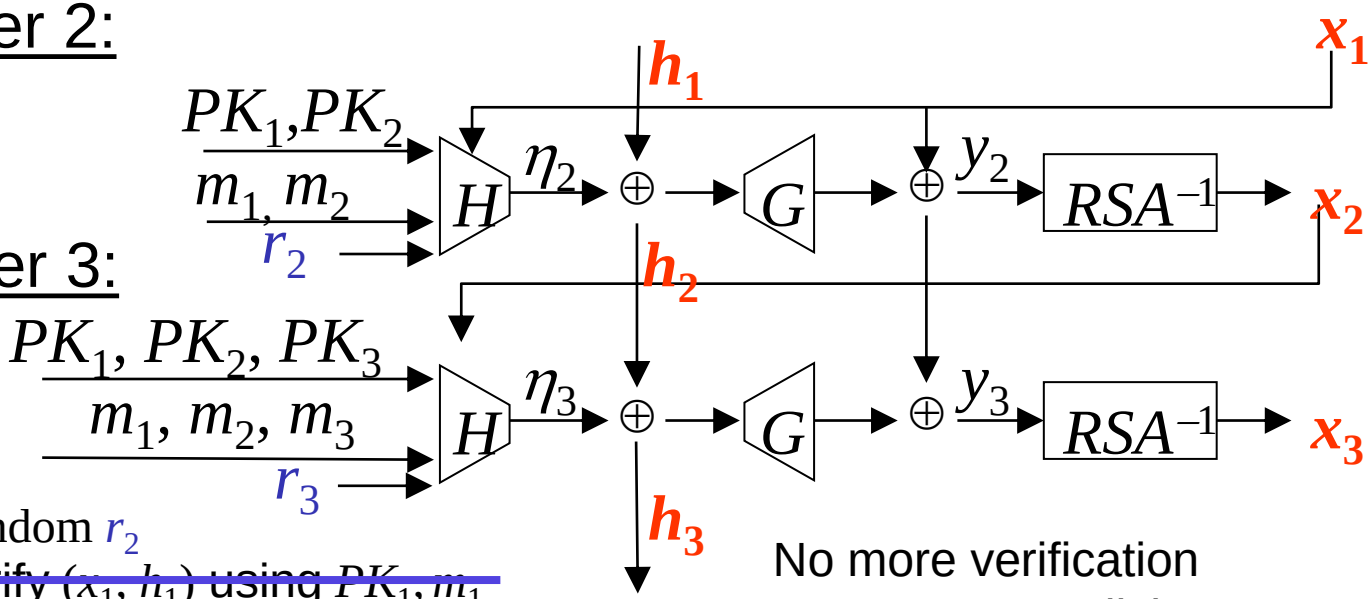
Our Scheme [BGR 12]

Hash functions H (short outputs), G (full RSA domain outputs)

Signature has two components: (x, h) plus an r value per signer

Signer 2:

Signer 3:



- Random r_2
- ~~Verify (x_1, h_1) using PK_1, m_1~~
- $\eta_2 = H(PK_1, PK_2, x_1, m_1, m_2), r_2)$
- $h_2 = \eta_2 \oplus h_1$
- $y_2 = G(h_2) \oplus x_1$
- $x_2 = y_2^{d_2} \bmod n_2$

No more verification necessary…malicious signer i cannot predict input to RSA^{-1} for signer i+1

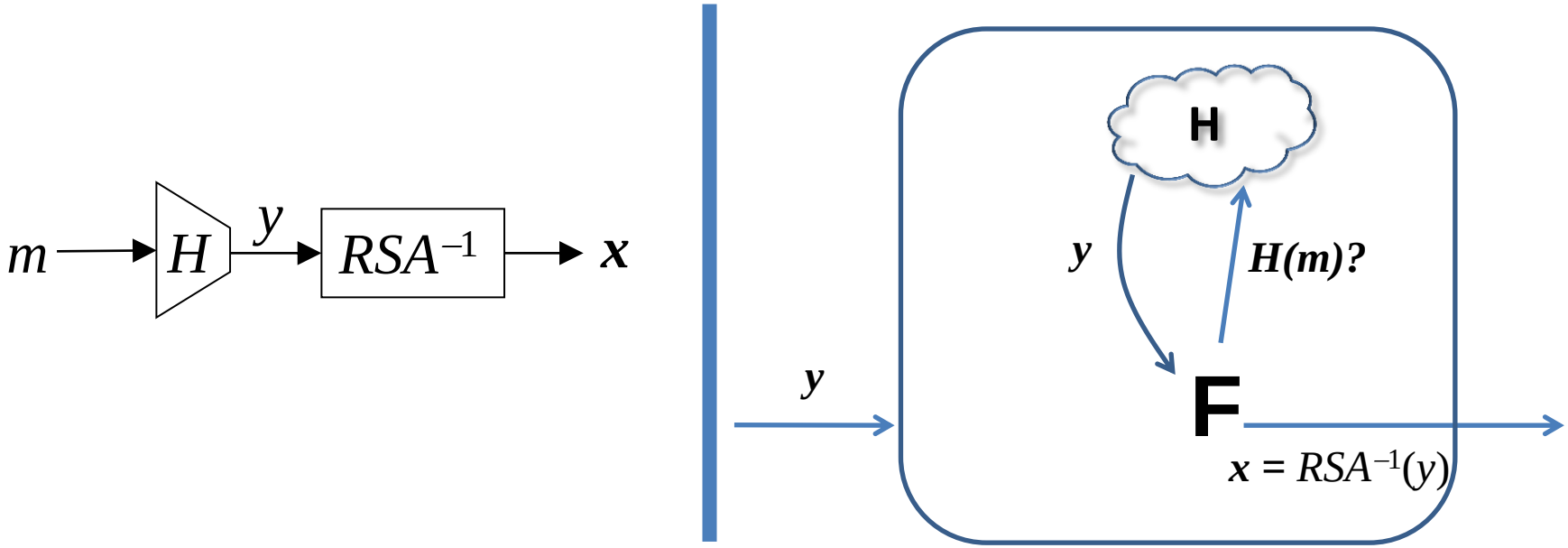
Lazy Verification Achieved!

Note: Security proof improves if r is pseudorandom; see paper for interesting combinatorial tricks.

- Need for Lazy Verification
- Sequential Aggregate Signatures
- Our Scheme
- Proof
- Benchmarks

Security Proof

Warm Up: Full-Domain-Hash Proof [Bellare-Rogaway 93]

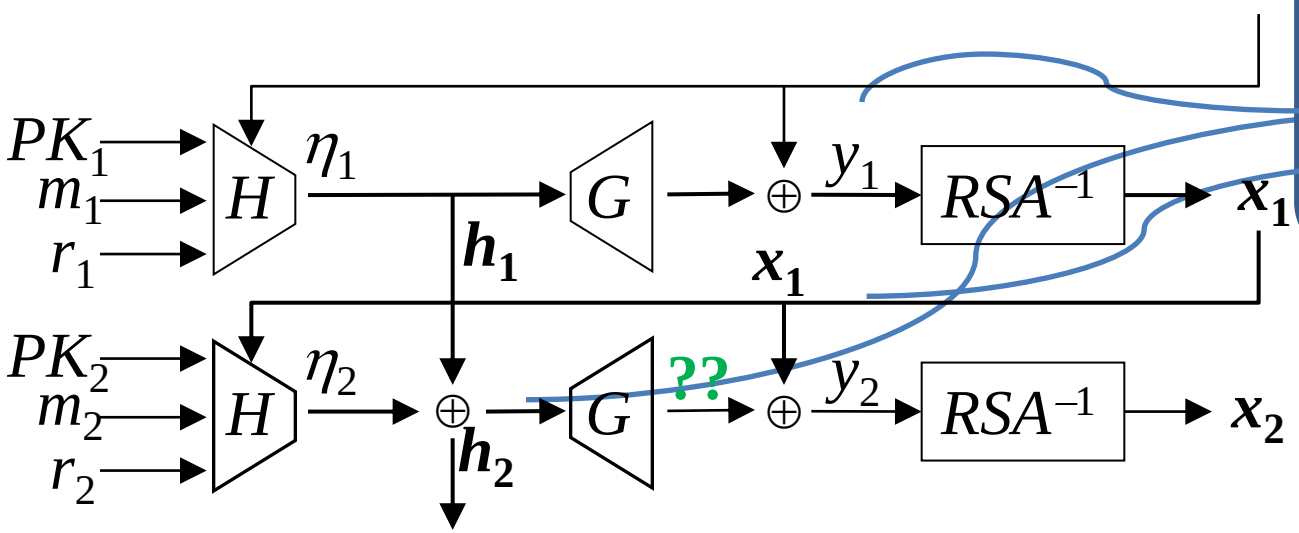


Proof logic: if forger F succeeds, we can invert RSA on a given y

H is a random oracle $\Rightarrow$ F has to query it $\Rightarrow$ answer one query with y

By *programming* the random oracle *H* to respond with *y*, we can ensure that if the forger succeeds, we will have inverted RSA on a given *y*.

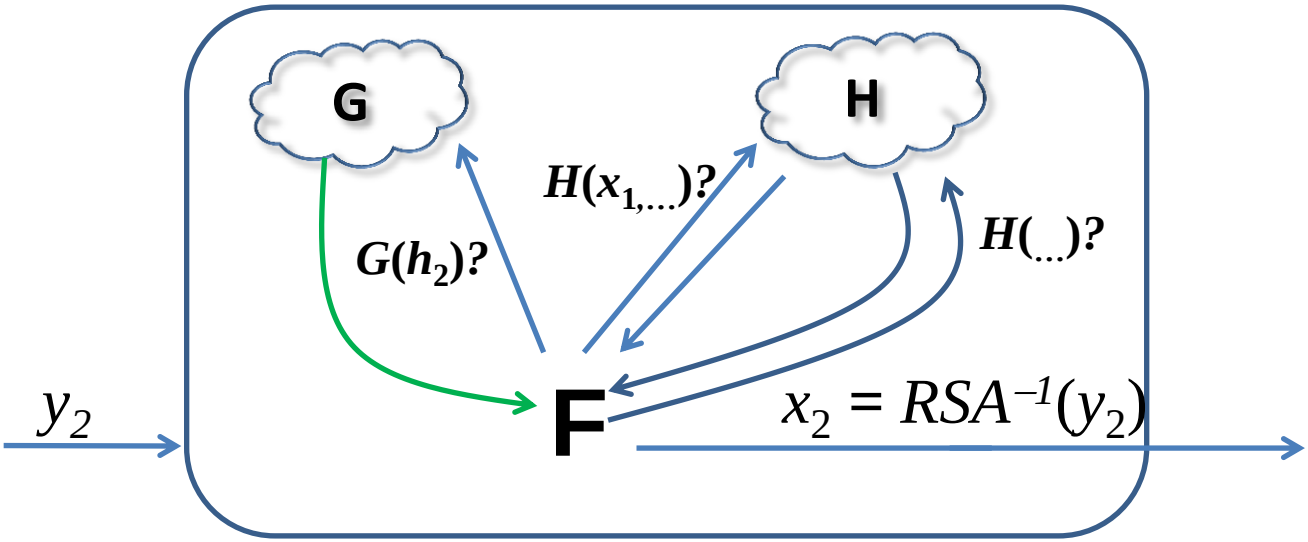
Security Proof cont'd



Note $G(h_1) = RSA_{x_1}(x_1)$. This can be used to pair the two queries.

Reduction sees all queries to H. Need to find one to match to x_1.

Now we have a pair of matched queries, so we've found x_1!

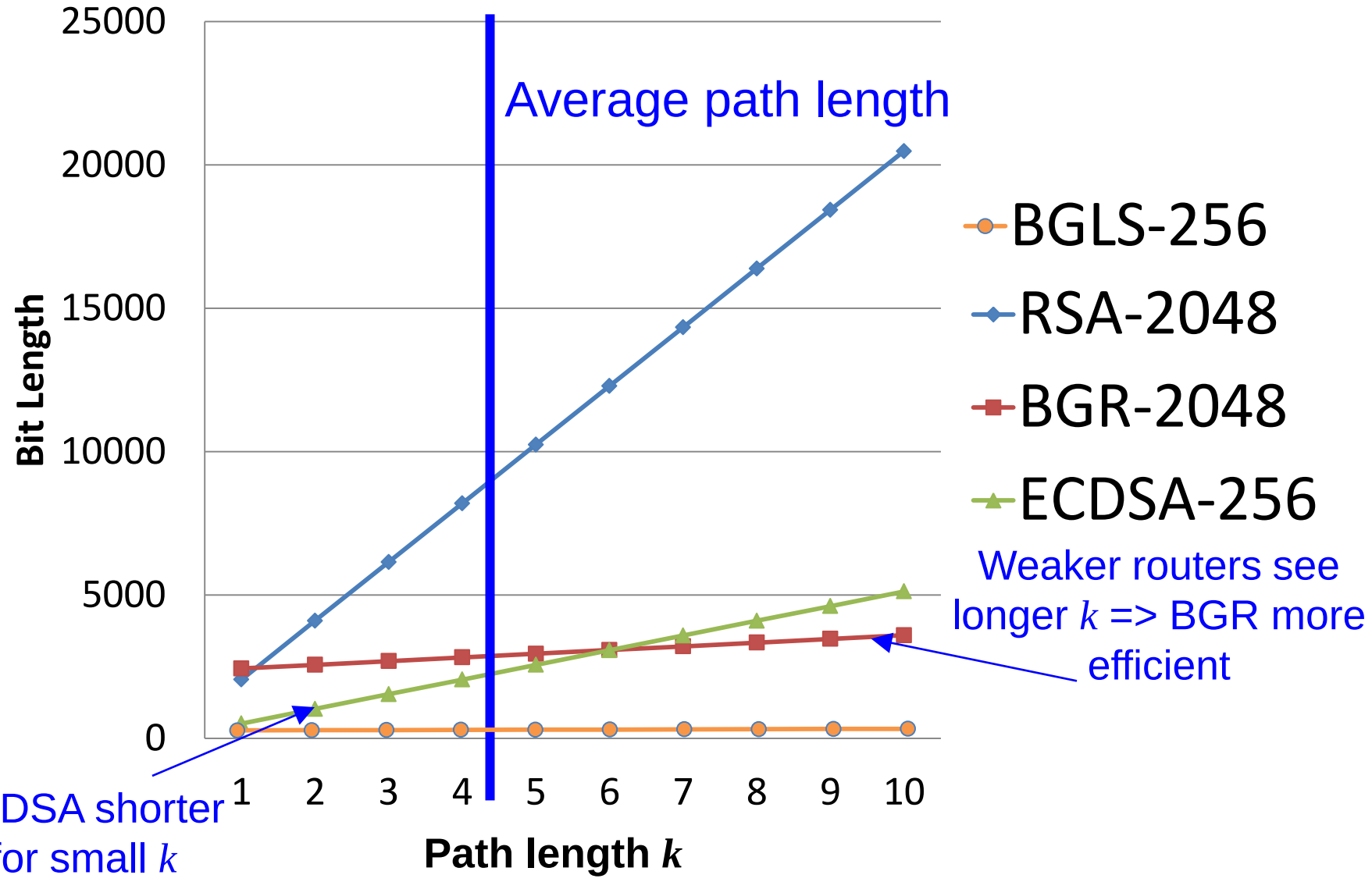


- Need for Lazy Verification
- Sequential Aggregate Signatures
- Our Scheme
- Proof
- Benchmarks

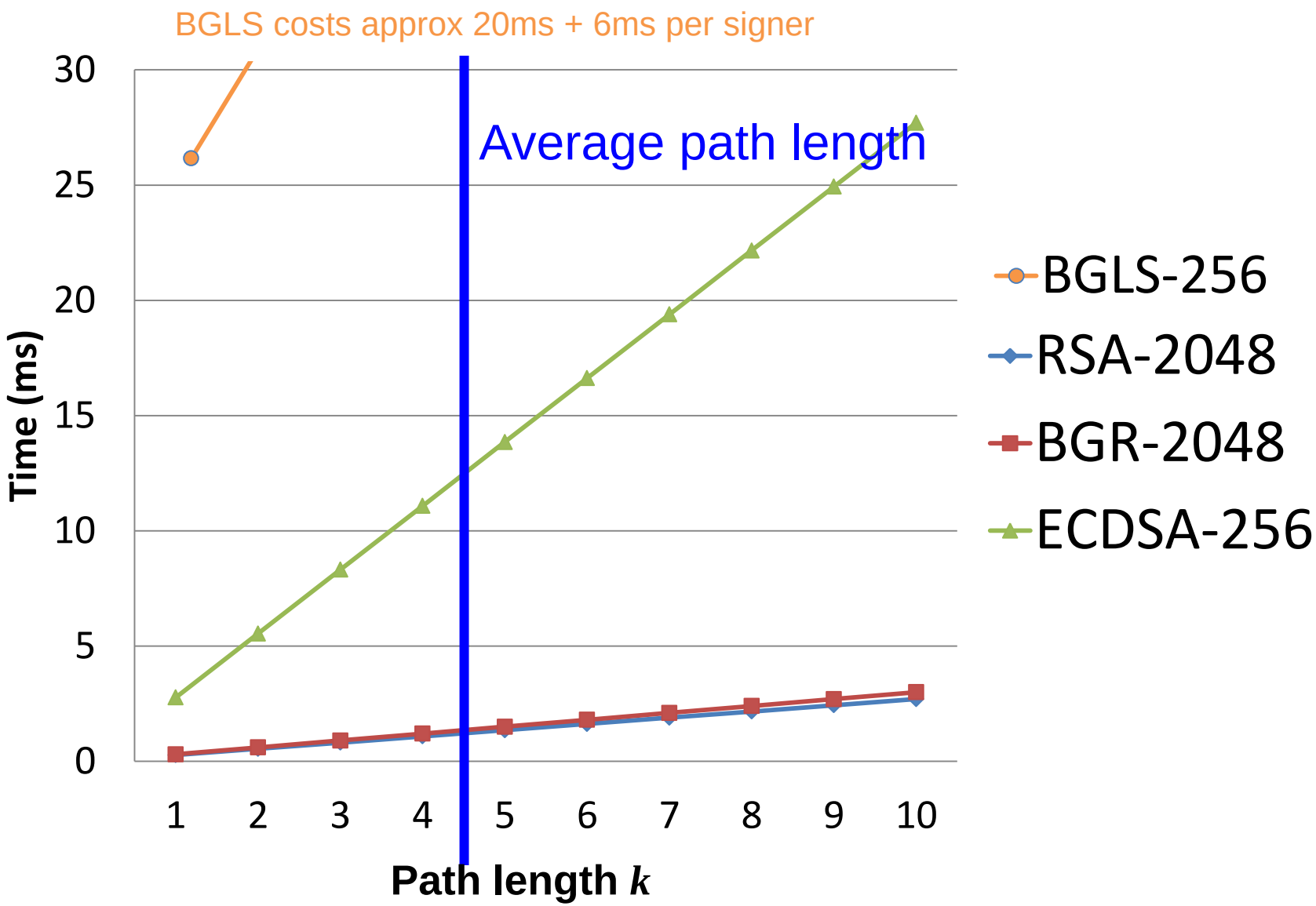
Benchmarks

- Implemented our scheme with OpenSSL primitives
- Benchmarks computed with software implementations.
  - Things may look different in hardware.
- Benchmarks computed using OpenSSL:
  - 2GB Ram, 2.4GHz Core i3
  - BLGS benchmark computed with MIRACL crypto library, as OpenSSL did not have an implementation.
- Benchmarks considered were signature length, verify time, and sign time.

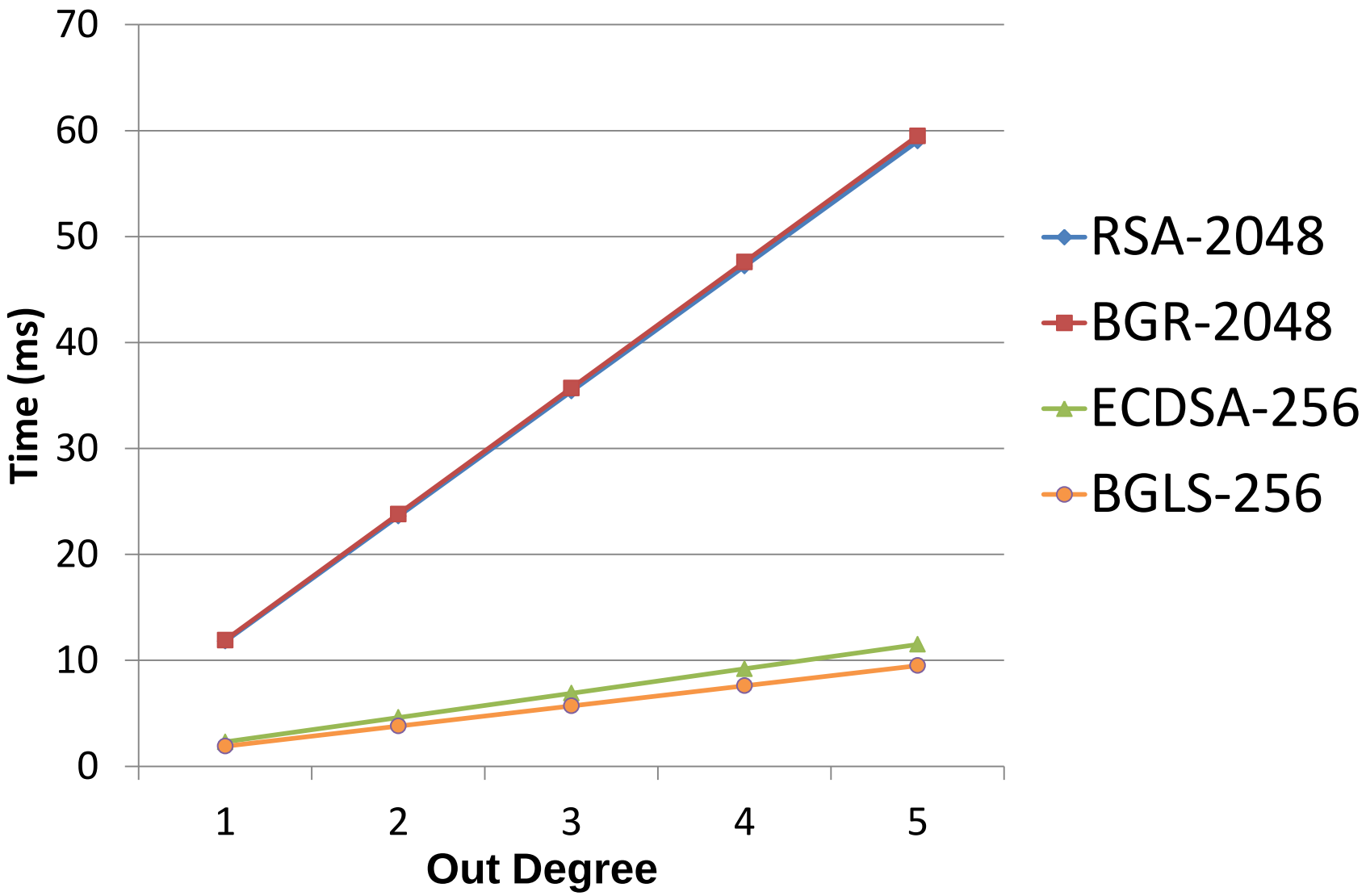
Signature Length



Verify Time


BGLS costs approx 20ms + 6ms per signer
Average path length
Time (ms)
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
9
10
Path length k
BGLS-256
RSA-2048
BGR-2048
ECDSA-256

Sign Time


70
60
50
40
30
20
10
0
Time (ms)
1
2
3
4
5
Out Degree
RSA-2048
BGR-2048
ECDSA-256
BGLS-256

Conclusions

Sequential Aggregate Signatures

- + From any TDP (in RO model)
- + Lazy Verification (In fact, don't need to know previous signers at all)
- – Signature grows ~128 bits/signer
  - Already have linear growth due to messages, which are on average longer than 128 bytes.
- ± Speed comparable to RSA (fast verify, slower sign).