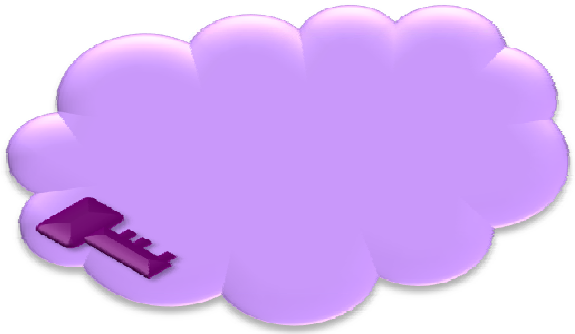
Sequential Aggregate Signatures with Lazy Verification for BGPsec



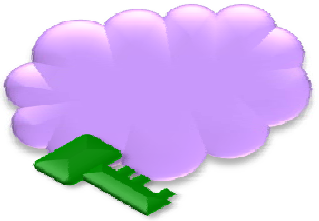
Kyle Brogle **Sharon Goldberg** **Leo Reyzin**
Boston University

http://www.cs.bu.edu/~goldbe/papers/bgpsec-sigs.html

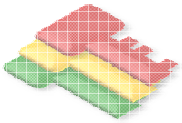
IBM Research, New York
April 4, 2011

This Talk

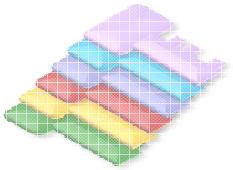
Part I : BGPsec and Our Signature



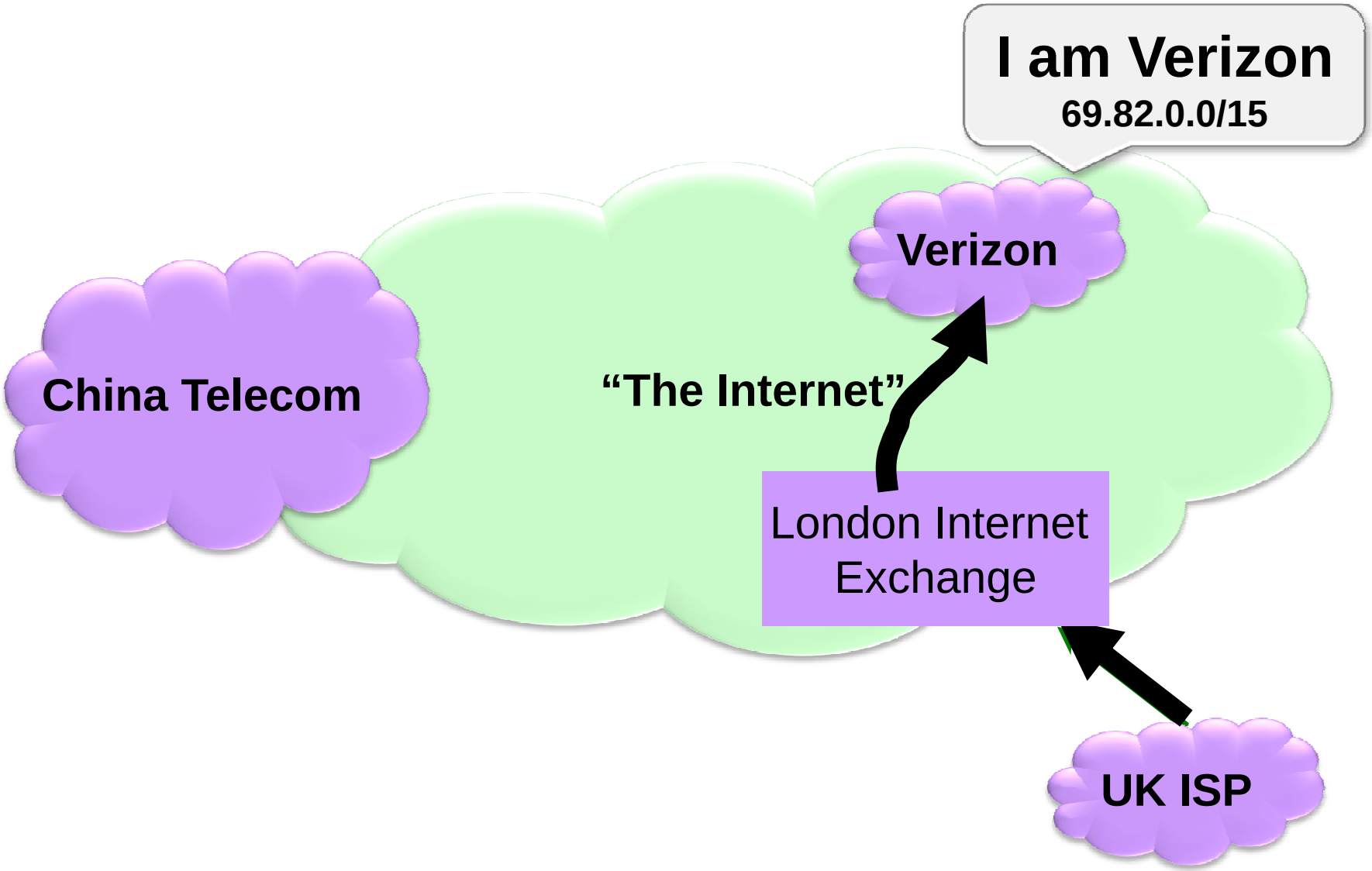
Part 2: RSA-based aggregate signatures



Part 3: Crypto Proofs

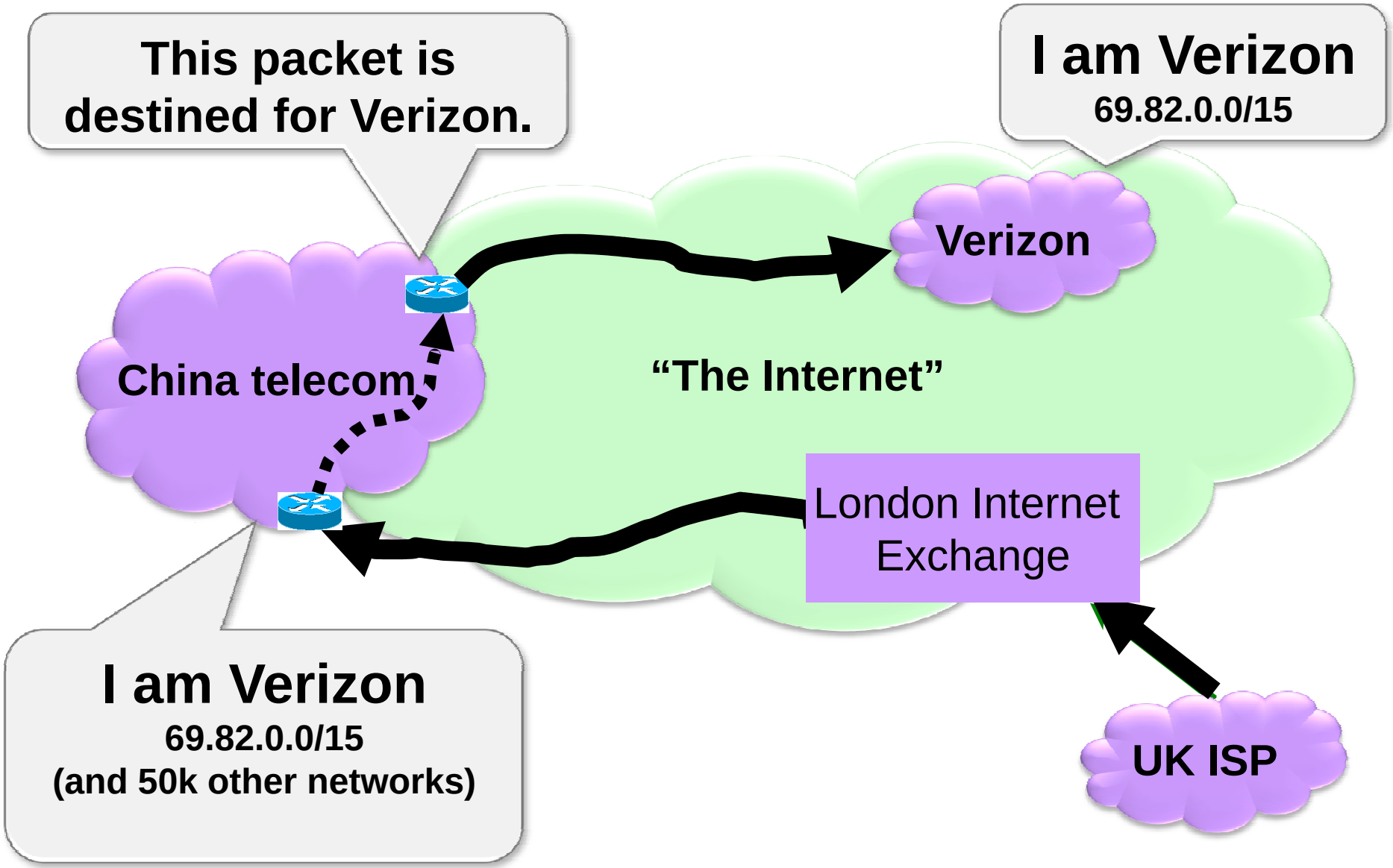


How Secure is Internet Routing Today? (1)



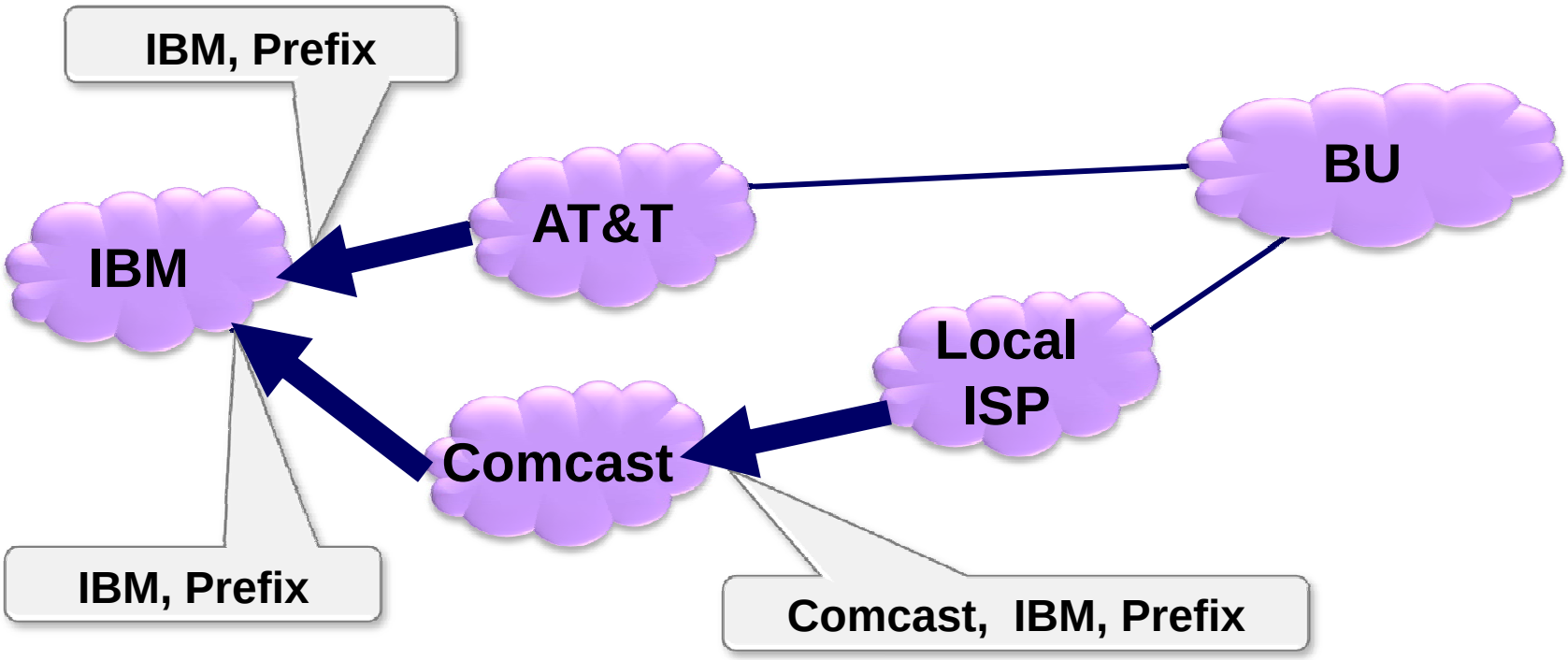
How Secure is Internet Routing Today? (2)

April 2010 : China Telecom intercepts traffic



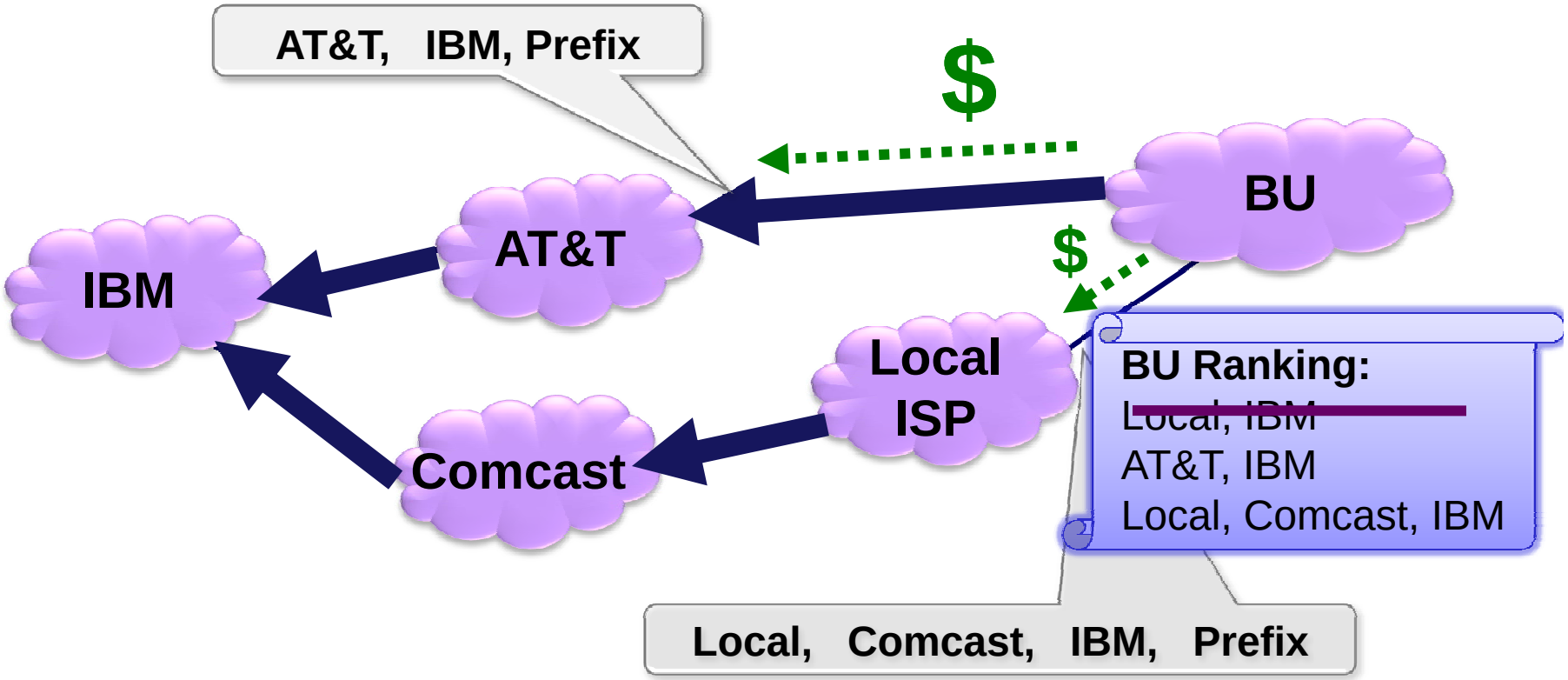
BGP: The Internet Routing Protocol (1)

Paths from Autonomous Systems (ASes) to IP prefixes are set up via the Border Gateway Protocol (BGP).



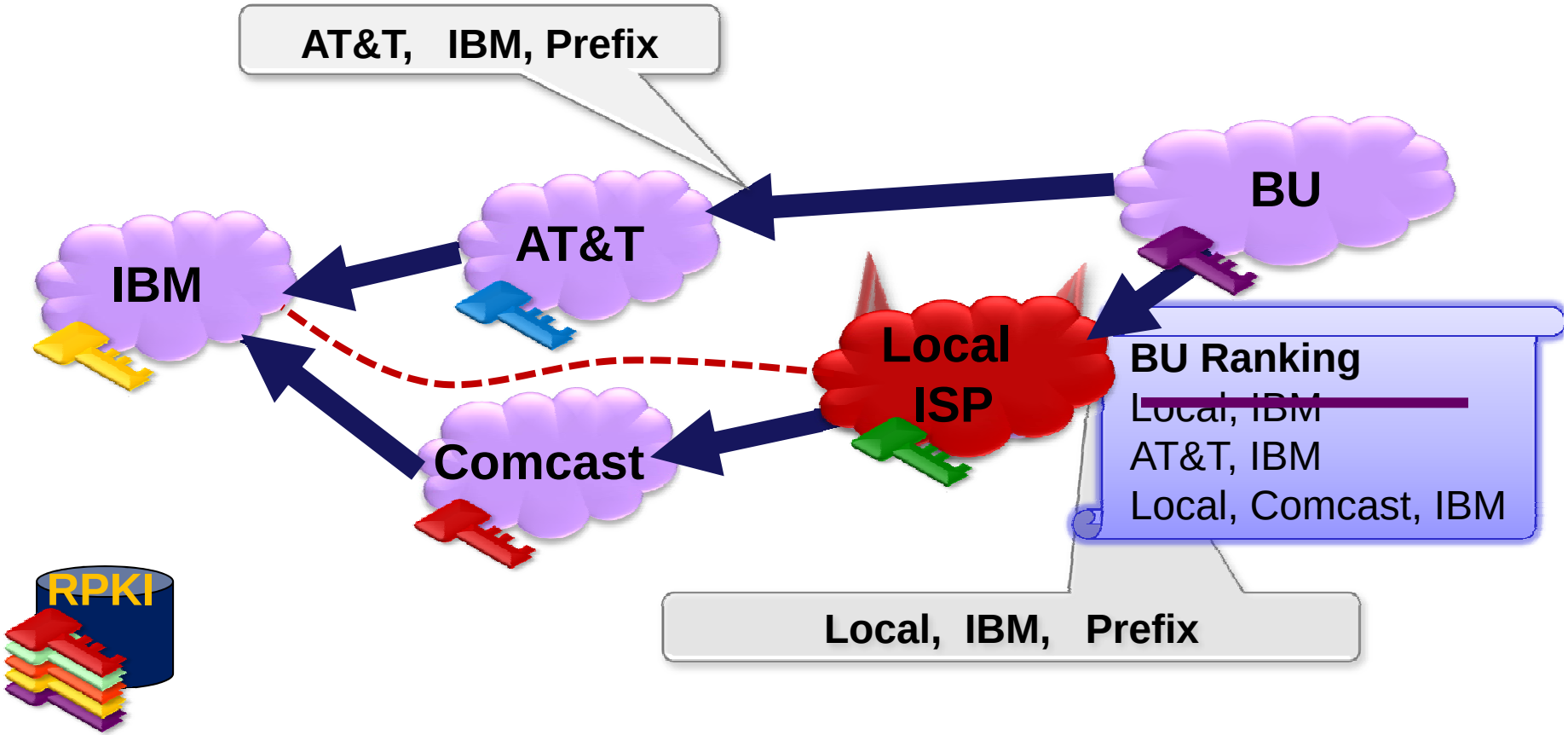
BGP: The Internet Routing Protocol (2)

Paths from Autonomous Systems (ASes) to IP prefixes are set up via the Border Gateway Protocol (BGP).



Attacks on BGP: Announcing false paths

Currently BGP has no mechanism to validate correctness of paths in BGP announcements.



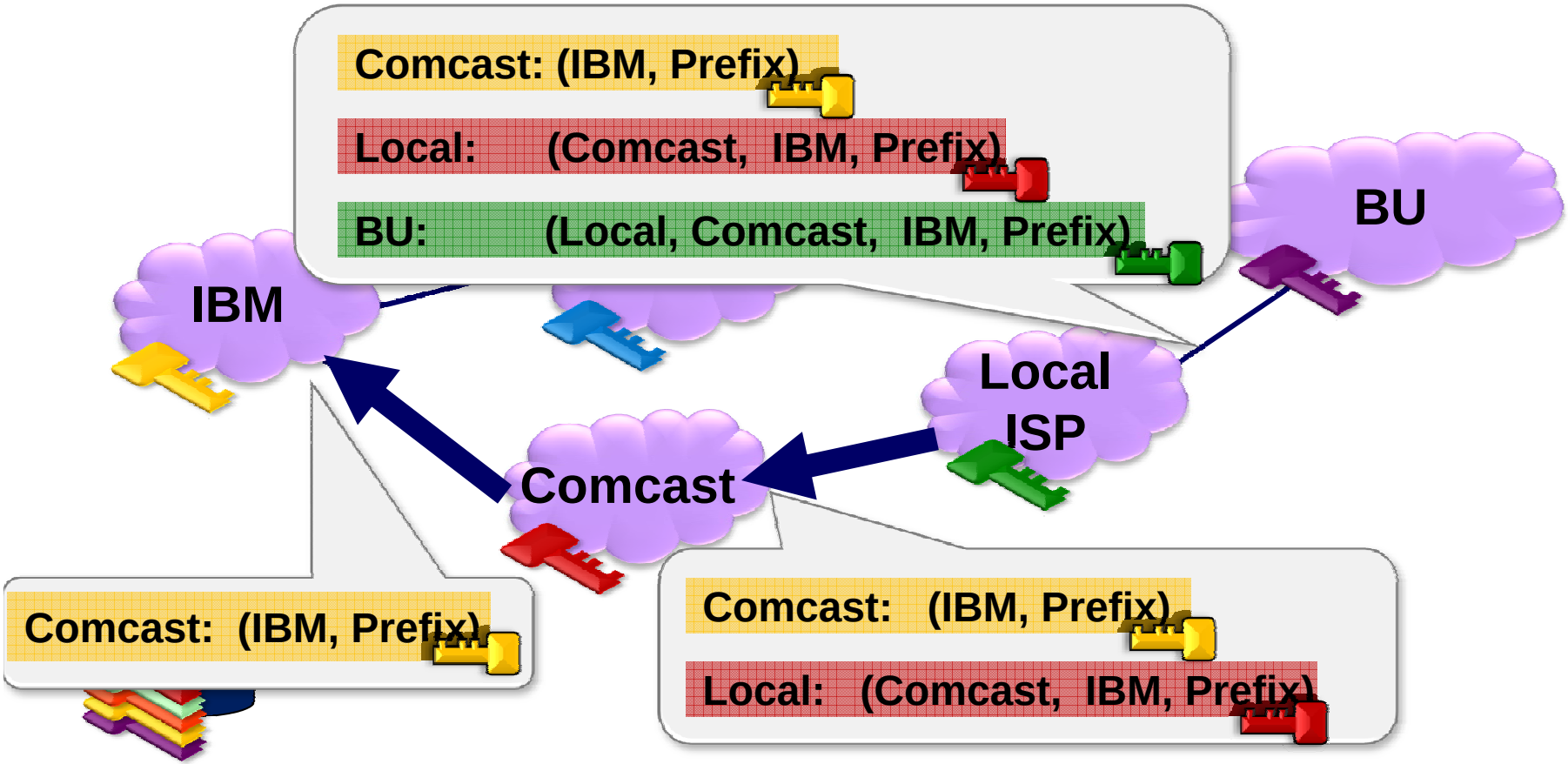
Step 1 of defending against BGP Attacks:
The "RPKI" that certifies mapping of IP Prefixes and PKs to Ases

Operators are actually deploying this now!

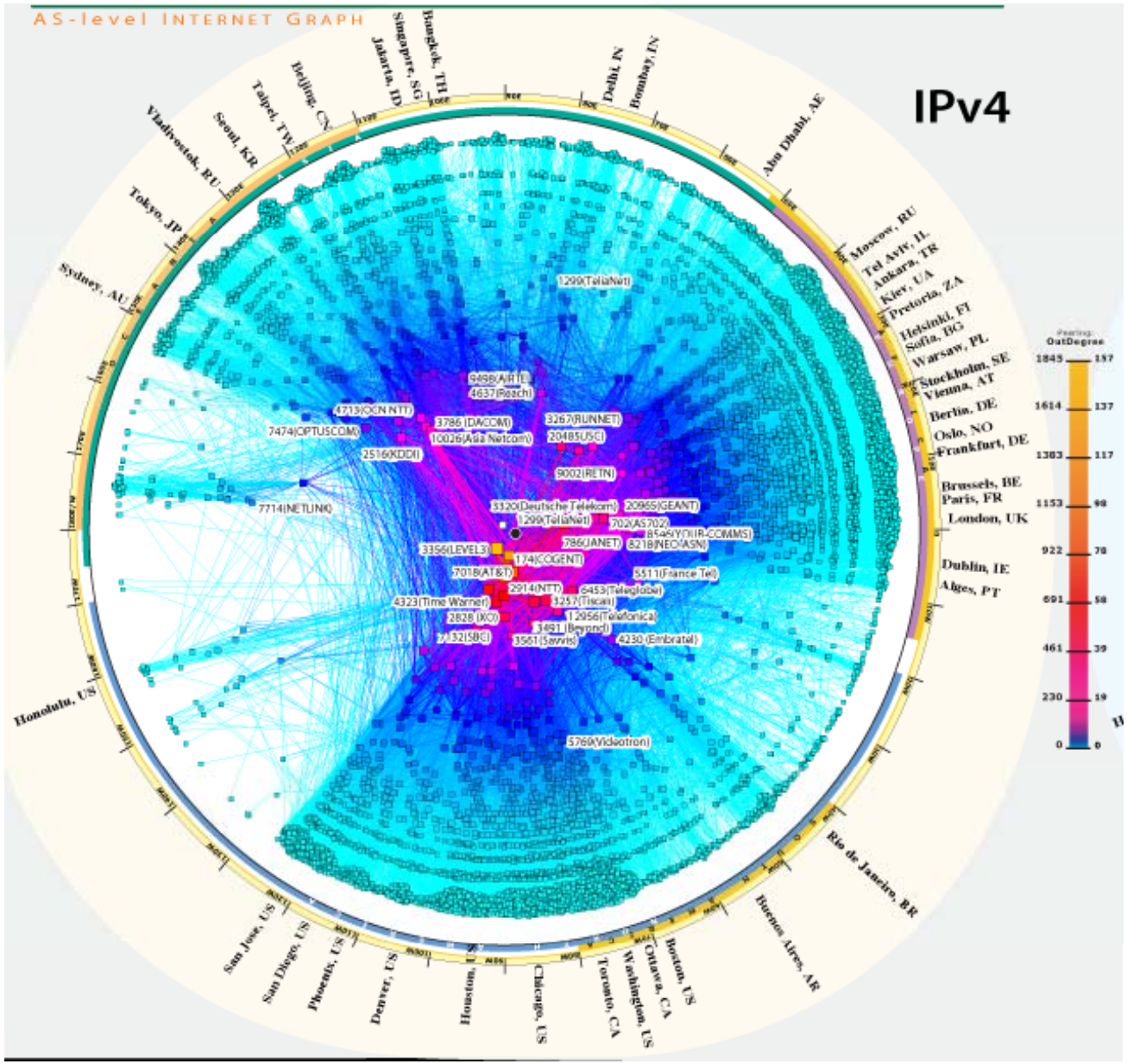
Defending Against Attacks with BGPsec

Step 2: BGPsec to certify correctness of AS-level paths.

New design of this went to the IETF in Prague just last week.



Technical Hurdles for BGPsec



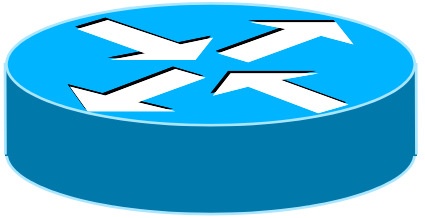
36K ASes

(➔Many PKs)

300K IP prefixes

(➔ Store many BGPsec msgs)

Routers are resource constrained



Source: CAIDA http://www.caida.org/publications/posters/png/ascore-ipv4-ipv6.200902.poster.png

Digital Signatures for BGPsec: Desiderata

Challenge:
Routers must maintain local cache of >36K keys secure and current

Challenge:
Routers must process BGPsec announcements quickly, even when they are overloaded.

Challenge:
Routers must store BGPsec announcements for each prefix.

Signing algorithm
requires no knowledge of previous signers.

Lazy Verification
to defer verification until load permits or public keys are retrieved

Aggregate signatures
shorten length of signature chains

[BGR] RSA sequential aggregate signature w. lazy verification.
(or, a randomized version of [Neven'08])

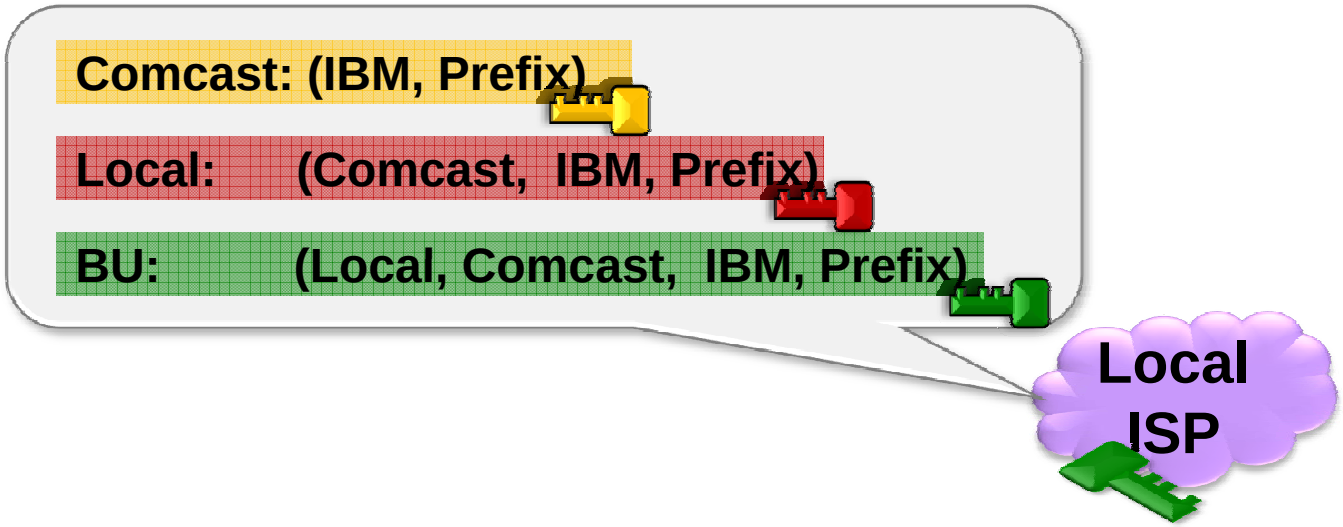
[BGR11] Instantiation with RSA, SHA-1, and HMAC

To Sign

1. $r_n = HMAC_k(msg_n \mid x_{n-1} \mid h_{n-1})$ — 128 bits
2. $h_n = h_{n-1} + H(PK_n \mid msg_n \mid r_n \mid x_{n-1})$ — 256 bits
3. $x_n = RSA(G(h_n) + x_{n-1})$ — 2048 bits
4. Remove 1st bit of x_n and save it as b_n — 1 bits

- Signer has a local (unshared) key **k** used to compute **HMAC**
- **In our implementation: k=256 bits, and 2048 bit RSA**
- **H = SHA-256; HMAC uses SHA-256; G** is **MGF** with **SHA-256**

For our target application of BGPsec



The real competition is nothing fancy:

- **Trivial RSA**
- **Trivial ECDSA**

Both of which allow:

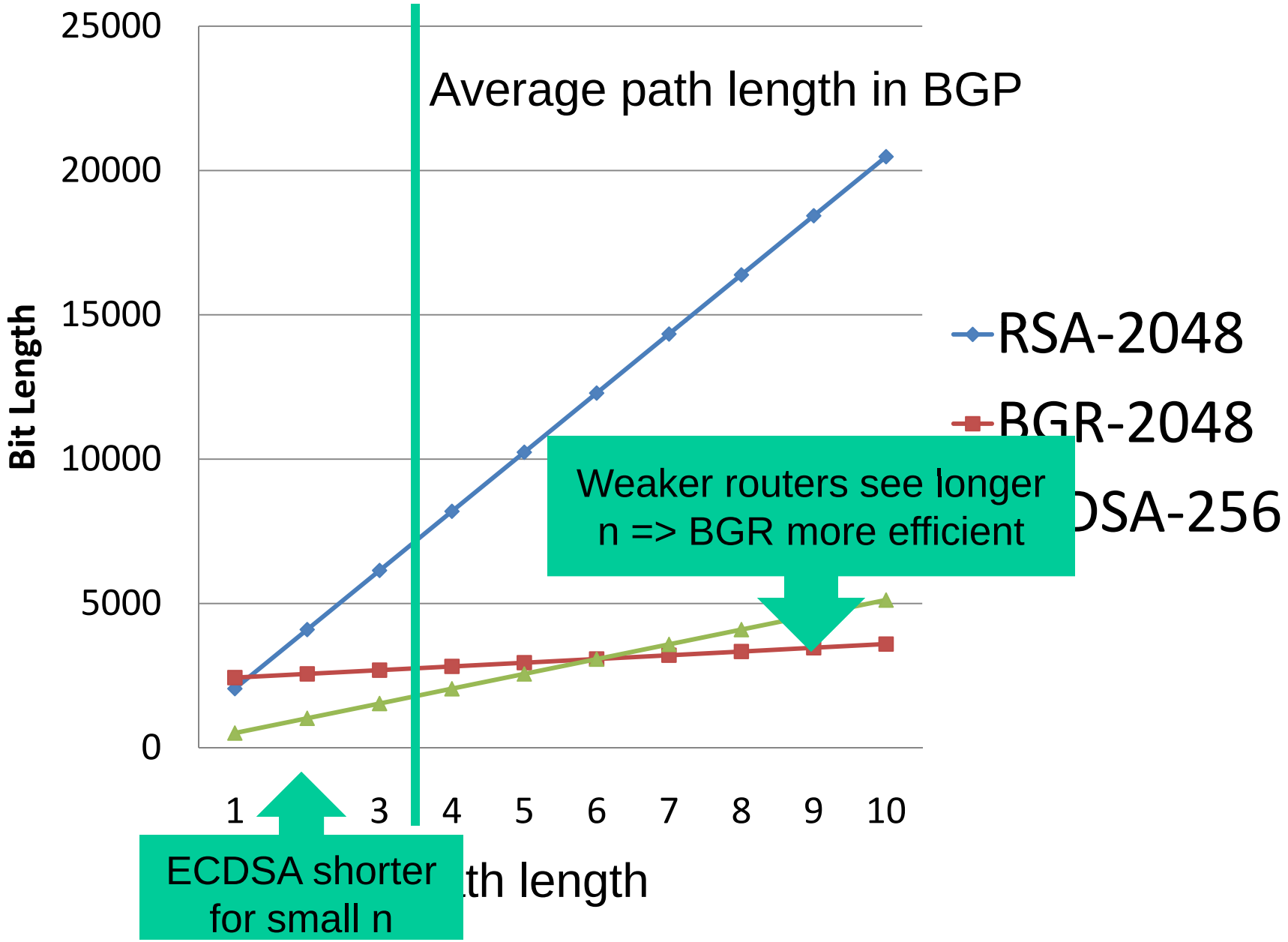
1) **Lazy verification,**
2) **Sign without knowing others keys**

Benchmarks and Performance Comparison

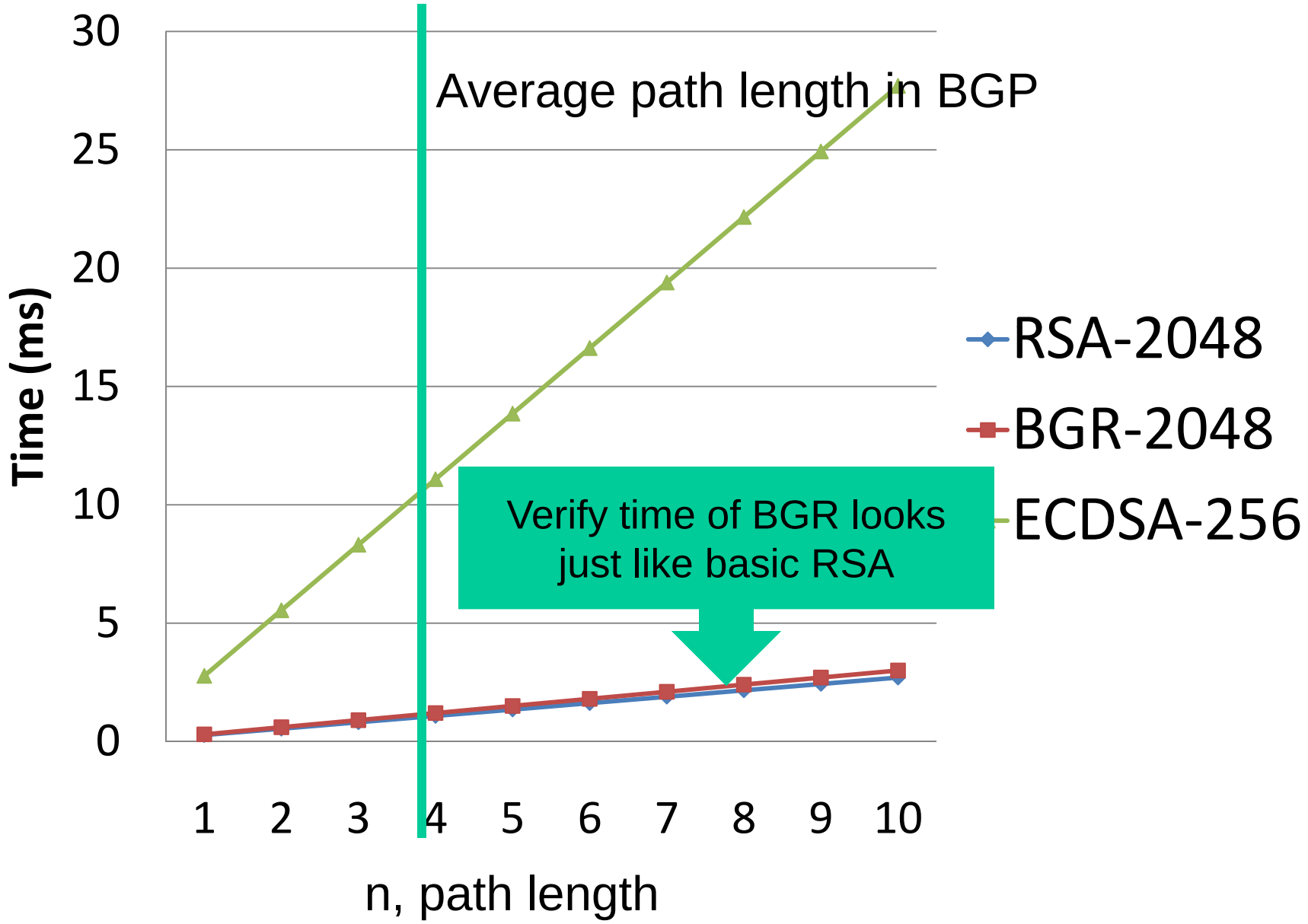
	RSA-2048 PKCS, SHA-256 exp 65537	BGR - RSA-2048 SHA-256 and RSA exp 65537	ECDSA-256
Total Sig Length	n * 2048 bits	2048 + 256 + 129*n	2n * 256 bits
Average Sig Length n = 3.5	7168 bits	2756 bits	**1792 bits**
Signing Time	11.8 ms	11.9 ms	**2.33 ms**
Verification Time	**n * 0.27 ms**	n * 0.30 ms	n*2.77 ms

Benchmarks computed using OpenSSL & (our implementation) on a laptop: 2GB Ram, Core i3 at 2.4GHz running Linux Ubuntu

Signature Lengths

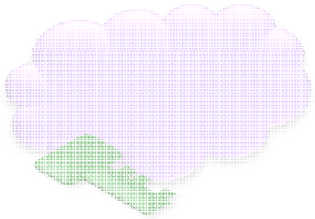


Verify Time


30
25
20
15
10
5
0
Time (ms)
1
2
3
4
5
6
7
8
9
10
n, path length
Average path length in BGP
Verify time of BGR looks just like basic RSA
RSA-2048
BGR-2048
ECDSA-256

This Talk

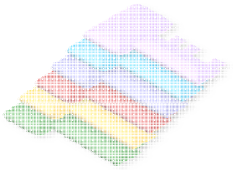
Part I : BGPsec and Our Signature



Part 2: RSA-based aggregate signatures

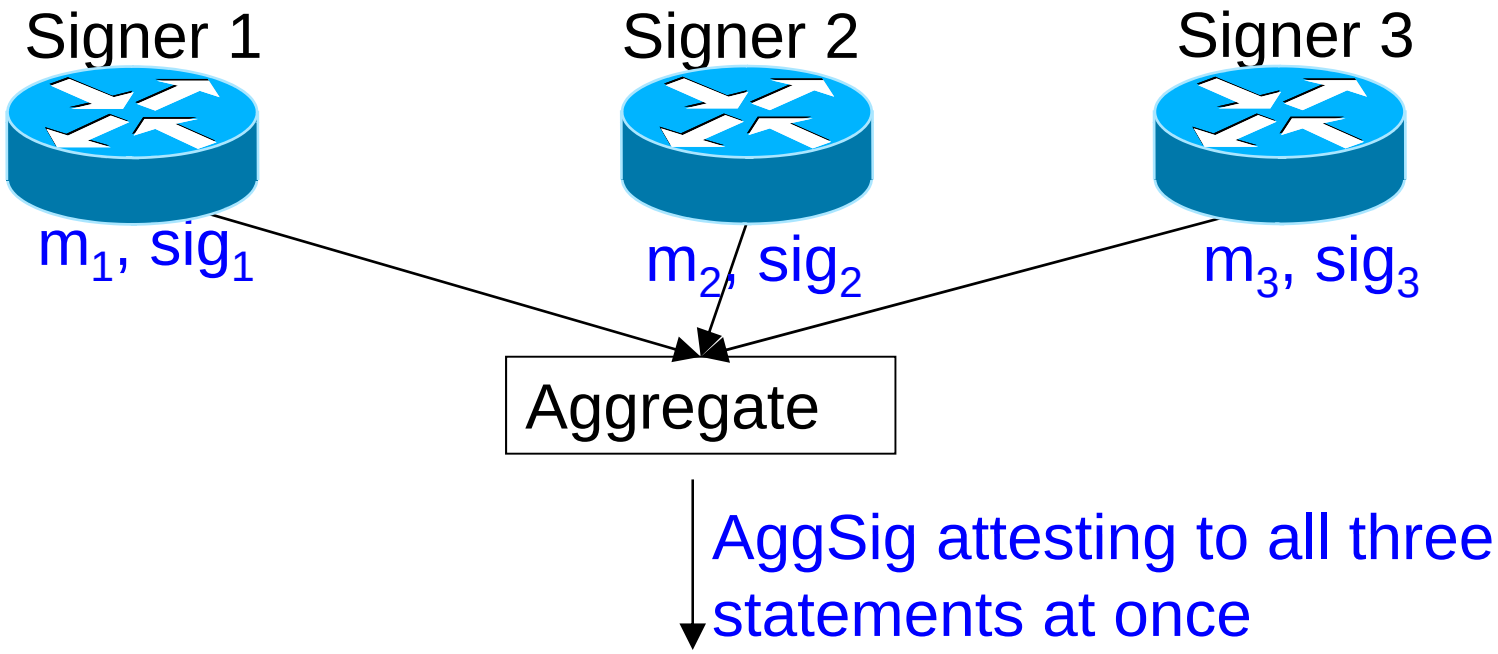
[BGLS-03] → [LMRS-04] → [Neven-08] → [Our Scheme]

Part 3: Crypto Proofs



Aggregate Signatures

- Can we compress multiple signatures to save space?
- Aggregate signature: **[Boneh-Gentry-Lynn-Shacham 2003]**



The Aggregate Signature Tradeoff

- **[BGLS]** Based on Pairings over Elliptic Curves
  - less standard assumptions
- What about something like RSA or Discrete Log?
- All known constructions without pairings require:
  - signers to know each others keys
  - have a prescribed order of operations
  - some operations using other signers' public keys
- Sequential Aggregate Signatures: signers to go in order
  - Not a big problem for BGP, since they do this anyway

Full-Domain Hash RSA

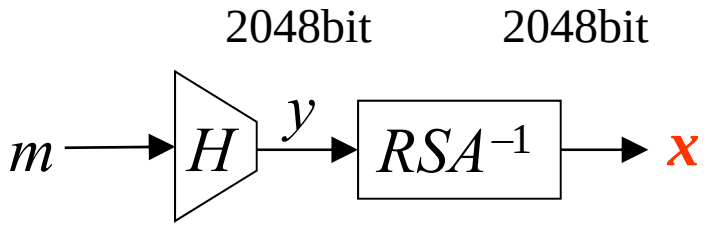
[Rivest-Shamir-Adleman 78, Bellare-Rogaway 93]

Hash function H (full RSA domain outputs; "random oracle").

Public key $PK = (n, e)$. Secret key $SK = (n, d)$.

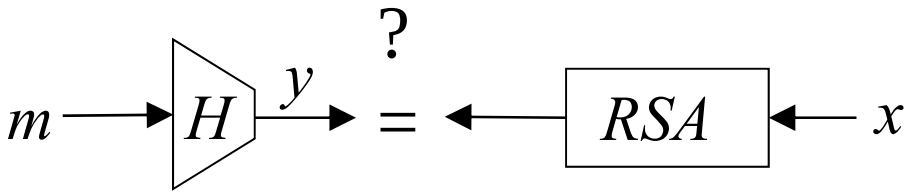
Steps of the Signer:

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



Steps of the Verifier:

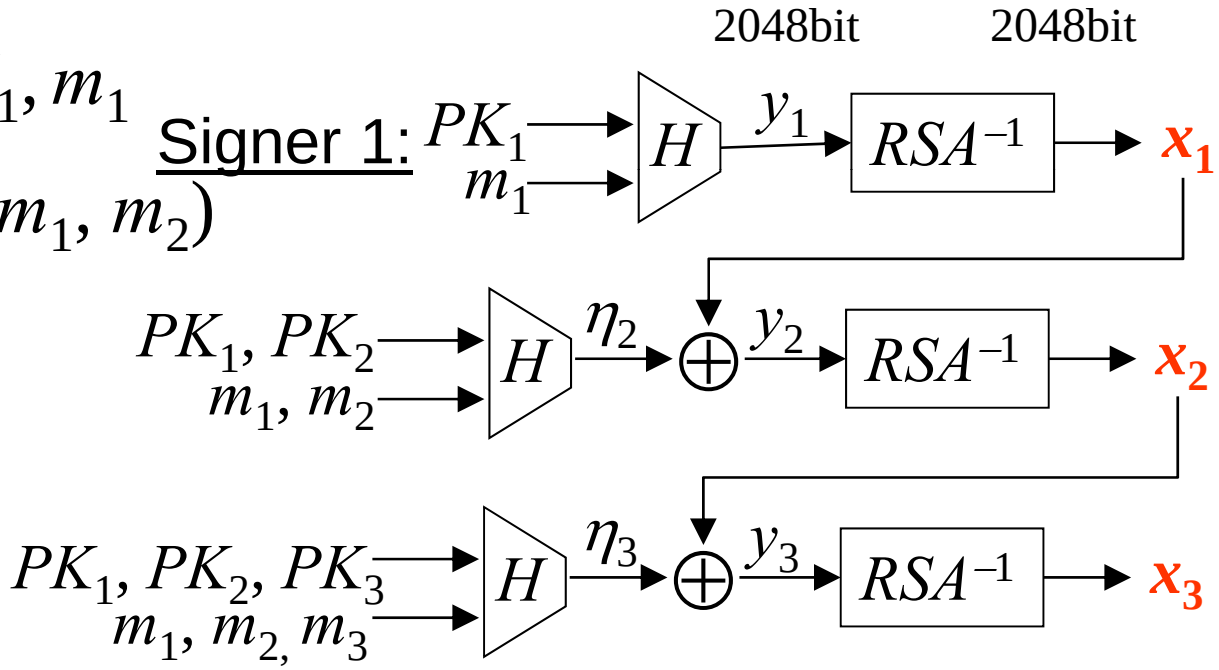
- $y = H(m)$
- $y \stackrel{?}{=} x^e \bmod n$



[Lysyanskaya-Micali-Reyzin-Shacham 04]: Sign

Steps of Signer 2:

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



Steps of Signer 3:

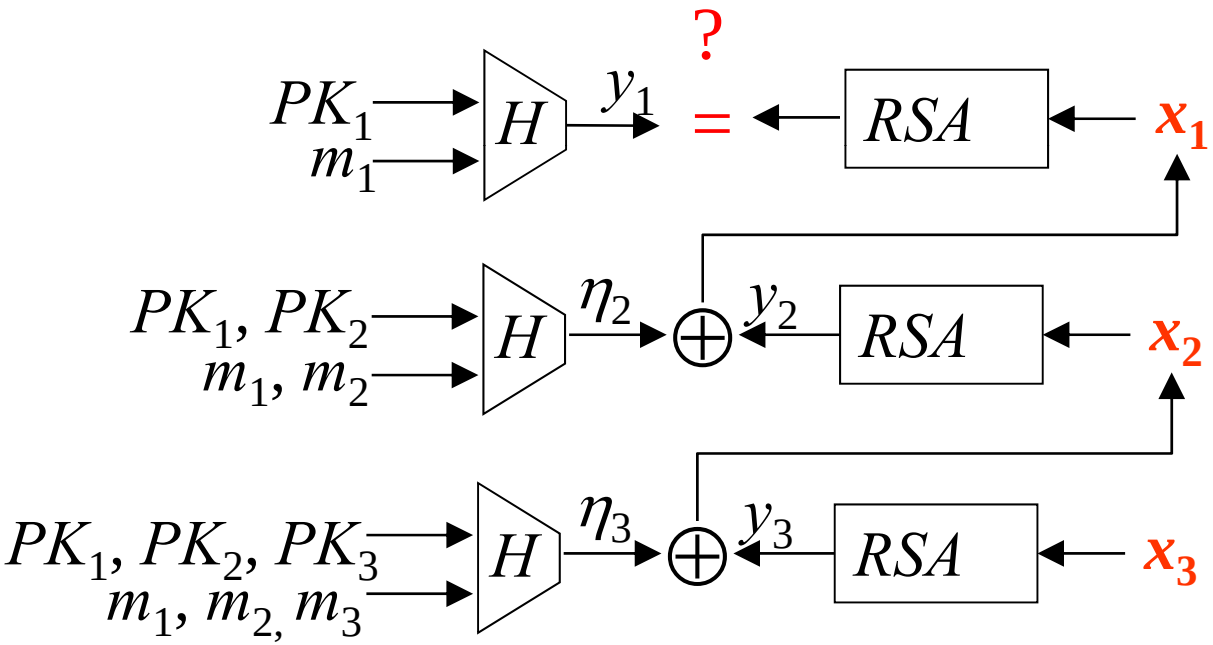
- Check that PK_1, PK_2 specify permutations
- Verify x_2 using PK_1, PK_2, m_1, m_2

• • •

[Lysyanskaya-Micali-Reyzin-Shacham 04]: Verify

Verification Algorithm

- Check that PK_1, PK_2,PK_3 specify permutations



[Lysyanskaya-Micali-Reyzin-Shacham 04]: Issues

Steps of Signer 2:

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

☹ Either proofs, or long verification exponents

☹ Prevents lazy verification

☹ Requires other signers PK's

Steps of Signer 3:

- Check that PK_1, PK_2 specify permutations
- Verify x_2 using PK_1, PK_2, m_1, m_2

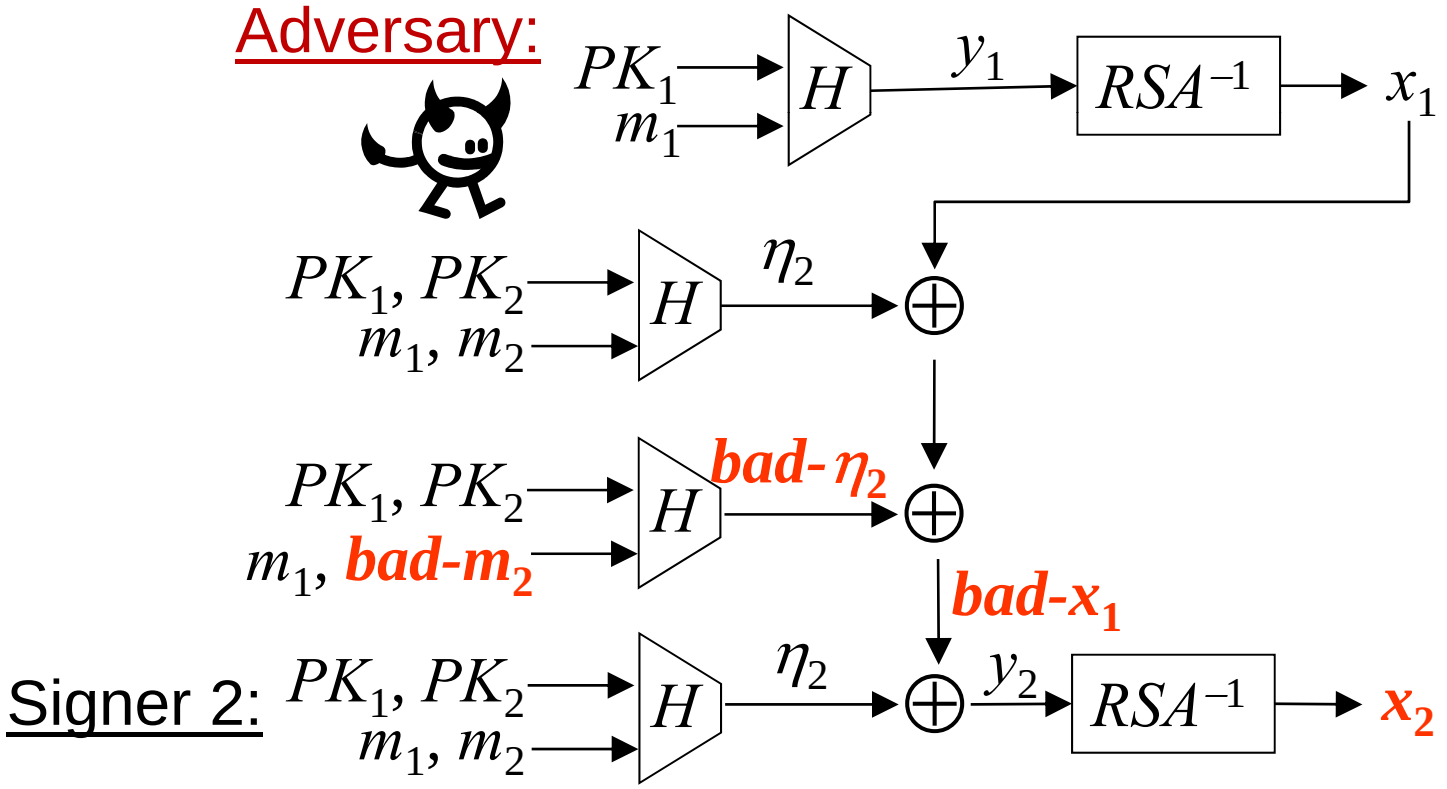
• • •

[LMRS] Fails under Lazy Verification.

Suppose an adversary wants to attack Signer 2

Adversary knows Signer 2 wants to sign m_2.

But adversary wants to get a sig on $bad\text{-}m_2$.



Valid aggregate sig on $(m_1, bad\text{-}m_2)$

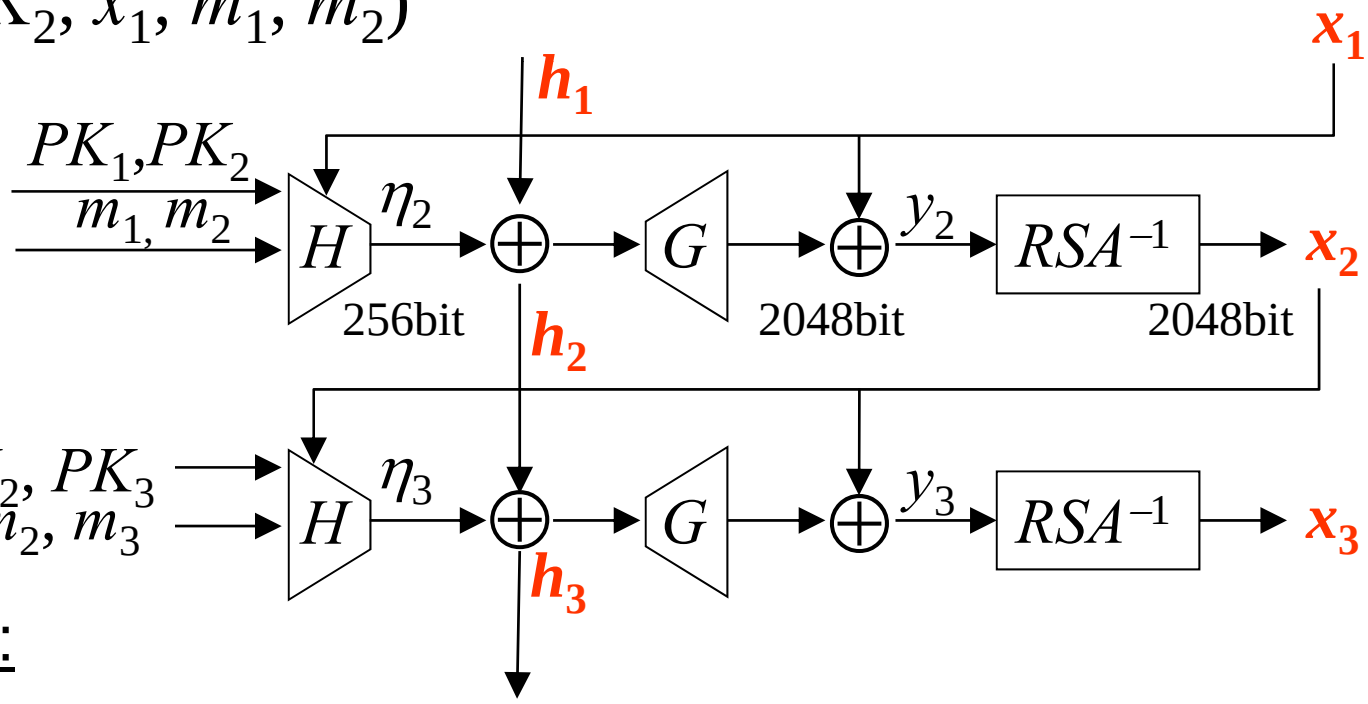
[Neven08]: Sign

No more certified TDP, but sig now has two components: (x, h).

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

Steps of Signer 2:

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

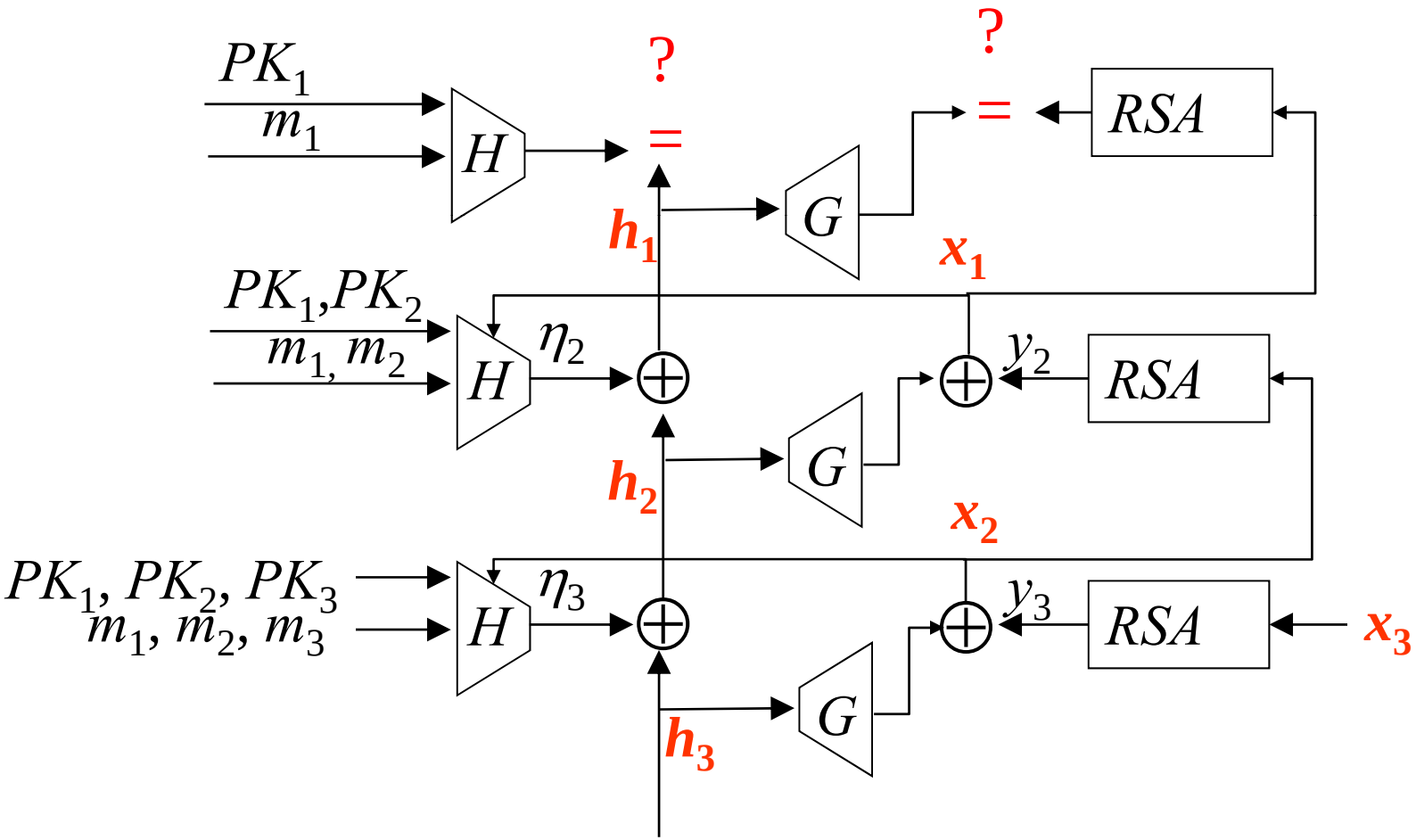


Steps of Signer 3:

- Verify (x_2, h_2) using PK_1, PK_2, m_1, m_2
- • •

[Neven08]: Verify

Signature has two components: (x_3, h_3)



[Neven08]: Issues

No more certified TDP, but sig now has two components: (x, h).

Hash function H (short outputs), G (full RS

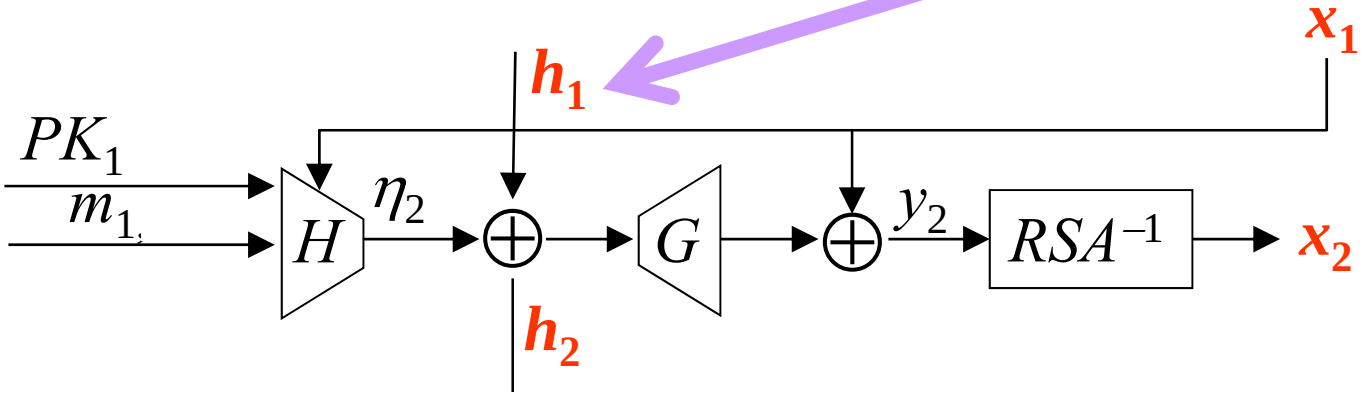
☹ No certified TDP, but still prevents lazy verification

Steps of Signer 2:

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

☺ Only an artifact of Neven's proof. We get rid of this!

Can break lazy verification in a similar way: inject $h_1 \oplus \eta_2 \oplus$ **bad-**η_2



Our Signature Scheme

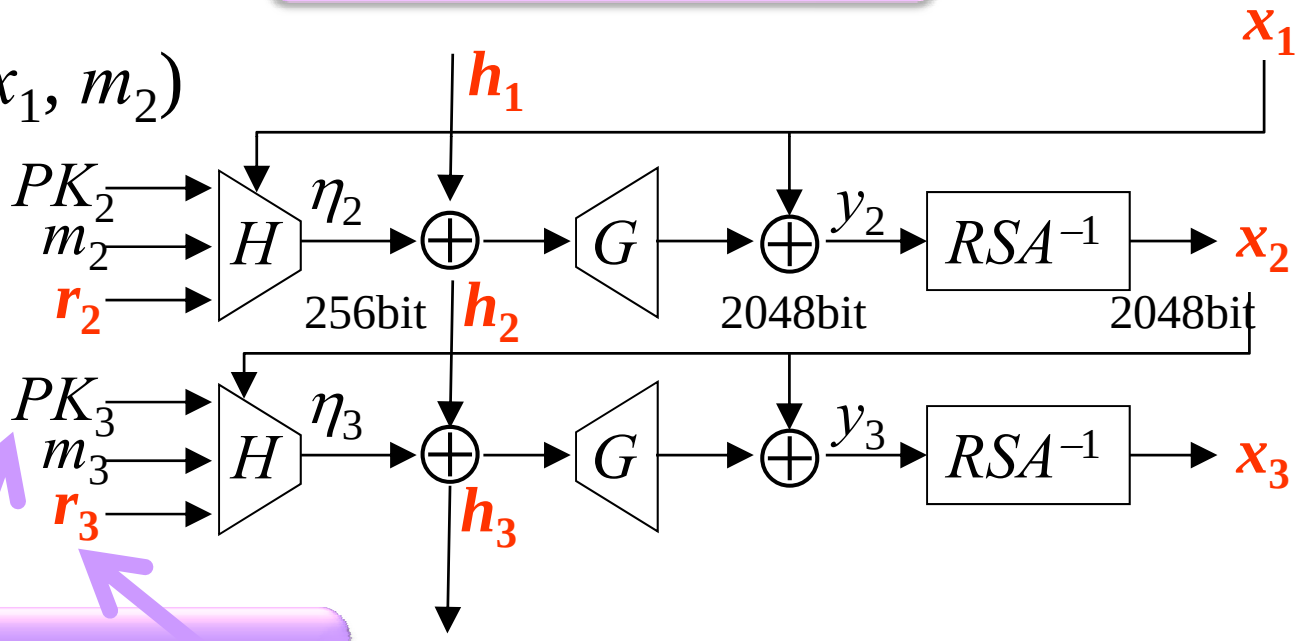
Randomize hash. Sig has (x, h) an a randomness r per signer.

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

Steps of Signer 2:

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

☺ Lazy Verification



☺ Signing depends only on your own public key!

☹ Signature grows (~128 bits / signer if r psuedorandom.)

Comparison of (Some) Aggregate Signatures

	[BGLS-03]	[LMRS-04]	[Neven-08]	[BGR-11]
Assumption	**Pairings**	**RSA**	**RSA**	**RSA**
Non-Sequential?	**Yes**	**No**	**No**	**No**
Uncertified TDP?	**N/A**	**No**	**Yes**	**Yes**
Lazy Verify?	**Yes**	**No**	**No**	**Yes**
Signature Length (bits)	128	2048	2048 + 256	2048 + 256 + **128n**

Wins on all counts except for assumption

Wins on all counts except for sig length… But best for our target app of BGPsec

This Talk

Part I : BGPsec and Our Signature

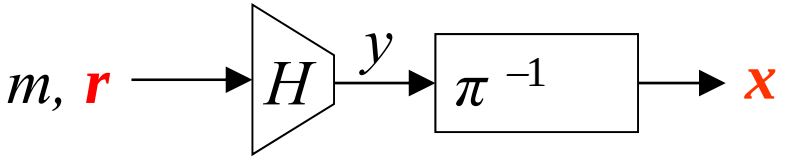
Part 2: RSA-based aggregate signatures

Part 3: Crypto Proofs

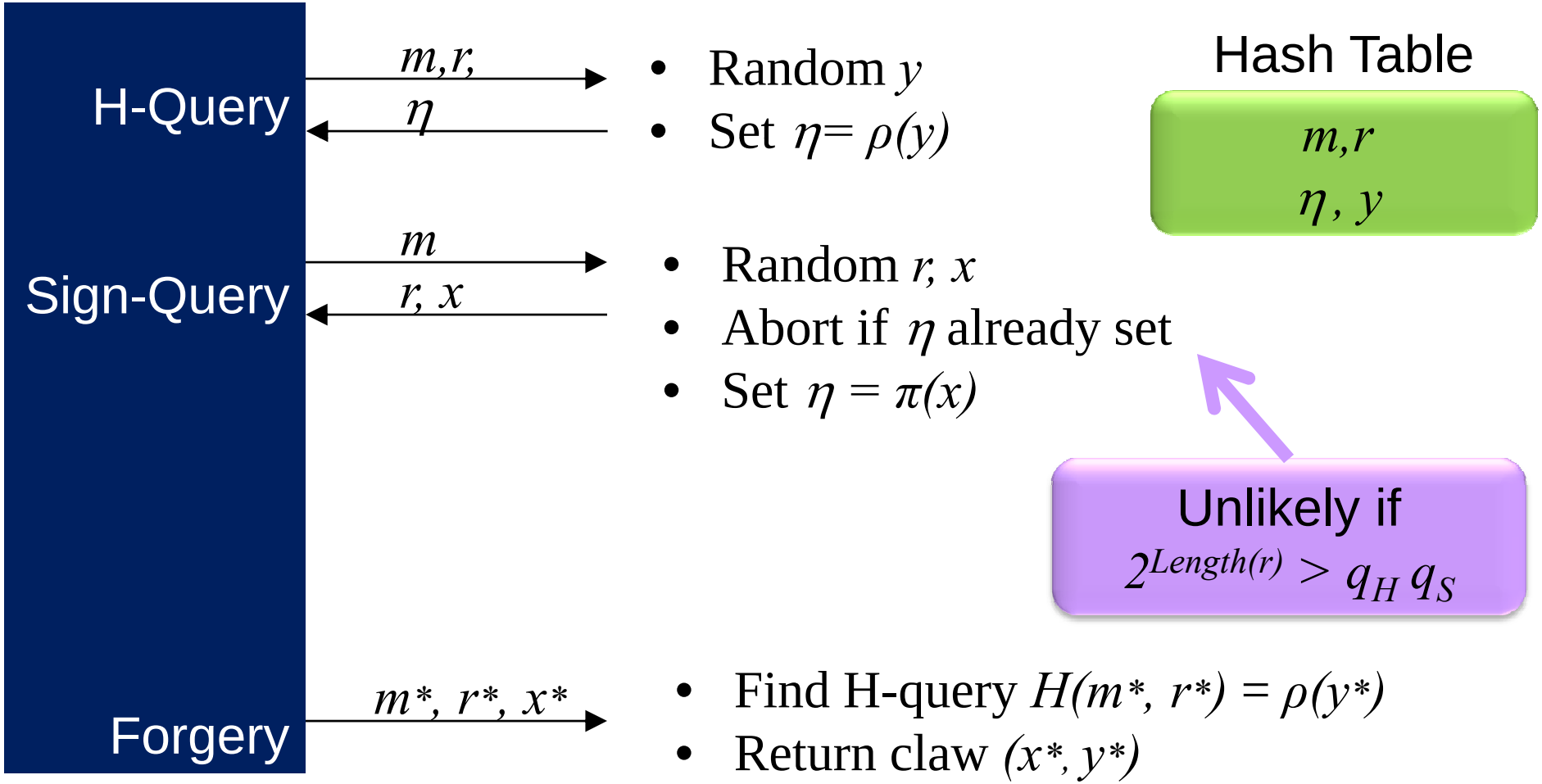
Randomized Full Domain Hash → [LMRS-04] → [Neven-08] → Improved [Neven-08] → [Our Scheme]

Proof Warm Up A: Randomized Full Domain Hash

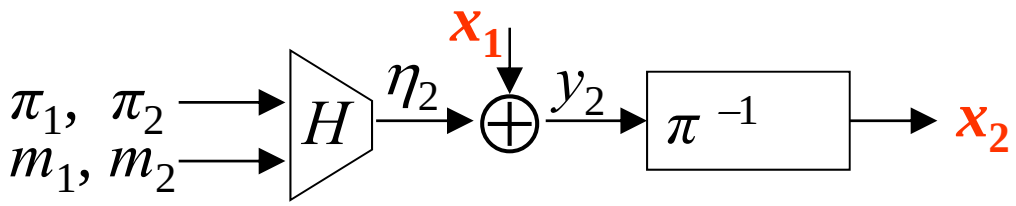
RSA is claw-free.
H is FDH Random Oracle.



Proof: If forger *F* succeeds, we find a claw $\pi(x)=\rho(y)$



Proof Warm Up B: [LMRS-04] (1)



Proof: If forger *F* succeeds, we find a claim $\pi^*(x)=\rho^*(y)$

H-Query

Sign-Query

Forgery

$\vec{\pi_i}, \vec{m_i}$

η_i

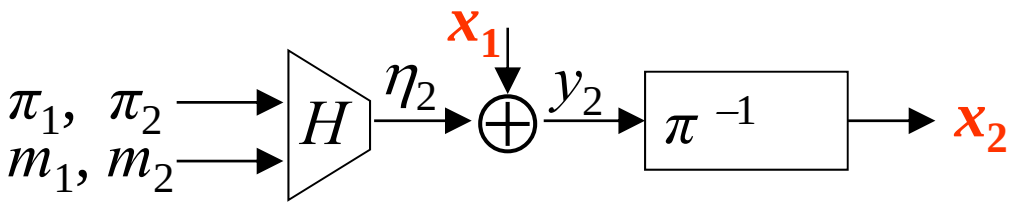
- Get x_{i-1} associated with $\vec{\pi_{i-1}}, \vec{m_{i-1}}$
- If $\pi_i \neq \pi^*$
  - Random x_i
  - $\eta_i = \pi_i(x_i) \oplus x_{i-1}$
- If $\pi_i = \pi^*$
  - Random z_i
  - $\eta_i = \rho^*(z_i) \oplus x_{i-1}$

Hash Table

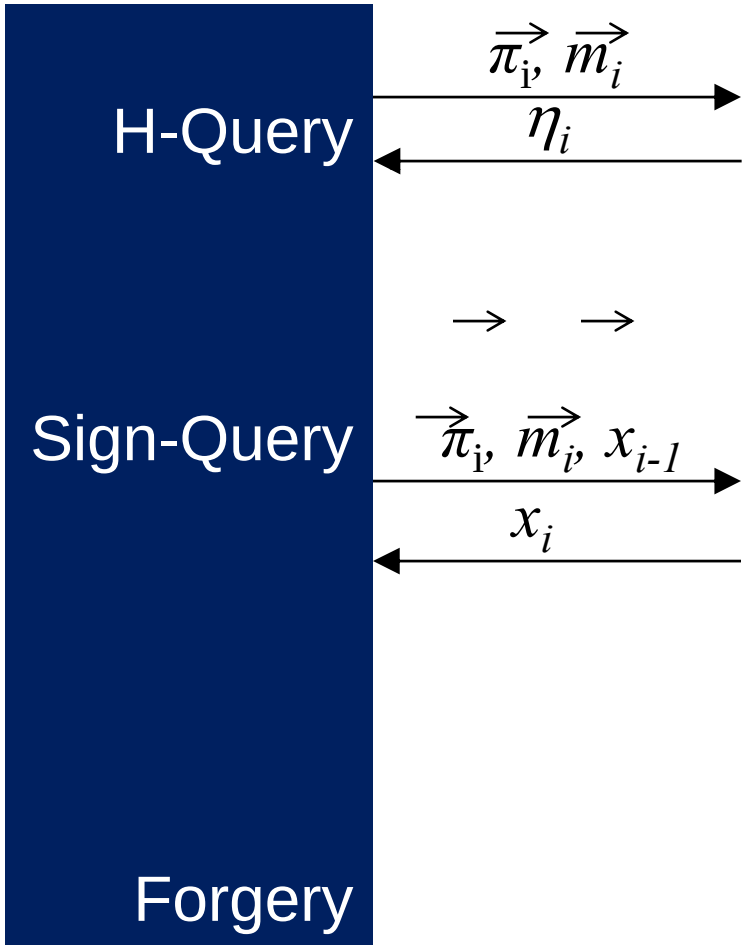
$\vec{\pi_i}, \vec{m_i}$

$\eta_1, x_i,$

Proof Warm Up B: [LMRS-04] (2A)



Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$



- Get x_{i-1} associated with $\vec{\pi}_{i-1}, \vec{m}_{i-1}$
- If $\pi_i \neq \pi^*$
  - Random x_i
  - $\eta_i = \pi_i(x_i) \oplus x_{i-1}$

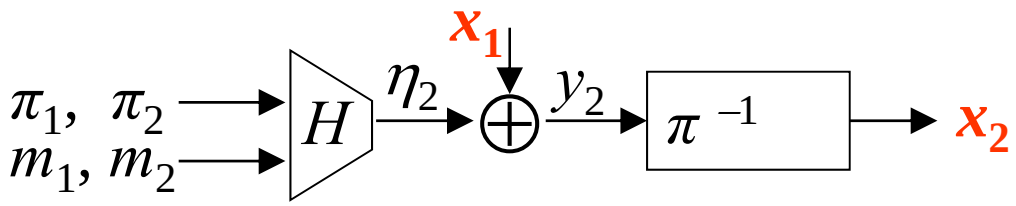
Hash Table

$\vec{\pi}_i, \vec{m}_i$

$\eta_1, x_i,$

- Verify x_{i-1}
- Verify π_i are permutations
- Random x_i
- $\eta_i = \pi_i(x_i) \oplus x_{i-1}$

Proof Warm Up B: [LMRS-04] (2B)



Proof: If forger *F* succeeds, we find a claim $\pi^*(x)=\rho^*(y)$

H-Query

$\vec{\pi}_i, \vec{m}_i$

η_i

- Get x_{i-1} associated with $\vec{\pi}_{i-1}, \vec{m}_{i-1}$
- If $\pi_i \neq \pi^*$
  - Random x_i
  - $\eta_i = \pi_i(x_i) \oplus x_{i-1}$

There is only one x_{i-1} for each $\vec{\pi}_{i-1}, \vec{m}_{i-1}$

Hash Table

$\vec{\pi}_i, \vec{m}_i$

$\eta_1, x_i,$

Sign-Query

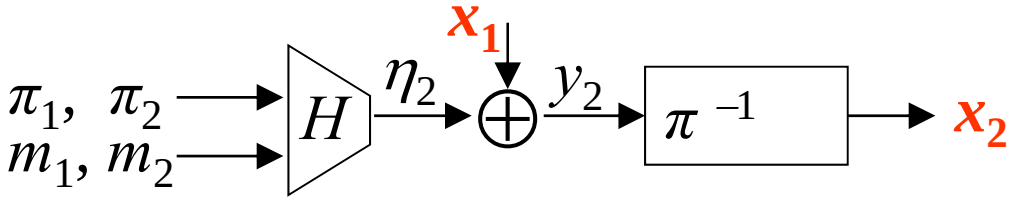
$\vec{\pi}_i, \vec{m}_i, x_{i-1}$

x_i

- Verify x_{i-1}
- Verify π_i are permutations
- Random x_i
- $\eta_i = \pi_i(x_i) \oplus x_{i-1}$

So that F can't set η_i to the wrong value on an earlier H-Query

Proof Warm Up B: [LMRS-04] (3)



Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$

H-Query

Sign-Query

Forgery

$\vec{\pi_i}, \vec{m_i}$

η_i

- Get x_{i-1} associated with $\vec{\pi_{i-1}}, \vec{m_{i-1}}$
- If $\pi_i = \pi^*$
  - Random z_i
  - $\eta_i = \rho^*(z_i) \oplus x_{i-1}$

Hash Table

$\vec{\pi_i}, \vec{m_i}$

$\eta_1, x_i,$

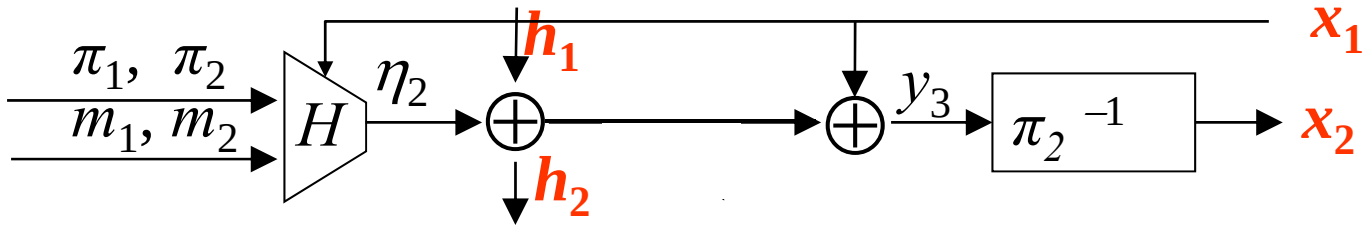
$\vec{\pi^*}, \vec{m^*_i}, x_i$

- Get z_i, x_{i-1} η_i associated with $\vec{\pi_i}, \vec{m_i}$
- Return claw (x_i, z_i) because

$\rho^*(z_i) = \eta_i \oplus x_{i-1} = \pi_i(x_i)$

Proof Warm Up C: [Neven-08] (1A)

To simplify, let H be FDH



Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$.

H-Query

Sign-Query

Forgery

$\overrightarrow{\pi_i}, \overrightarrow{m_i}, x_{i-1}$

η_i

- Get h_{i-1} associated with $\overrightarrow{\pi_{i-1}}, \overrightarrow{m_{i-1}}, x_{i-1}$
- If $\pi_i \neq \pi^*$
  - Random x_i
  - $h_i = \pi_i(x_i) \oplus x_{i-1}$
  - $\eta_i = h_{i-1} \oplus h_i$
- If $\pi_i = \pi^*$
  - Random z_i
  - $h_i = \rho^*(z_i) \oplus x_{i-1}$
  - $\eta_i = h_{i-1} \oplus h_{i-1}$

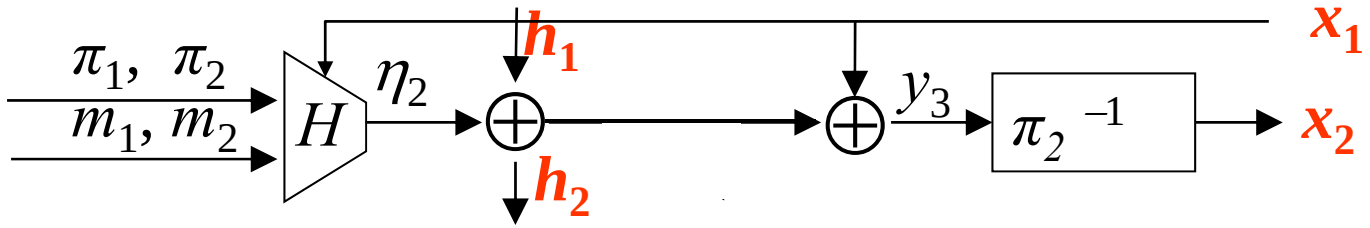
Hash Table

$\overrightarrow{\pi_i}, \overrightarrow{m_i}$

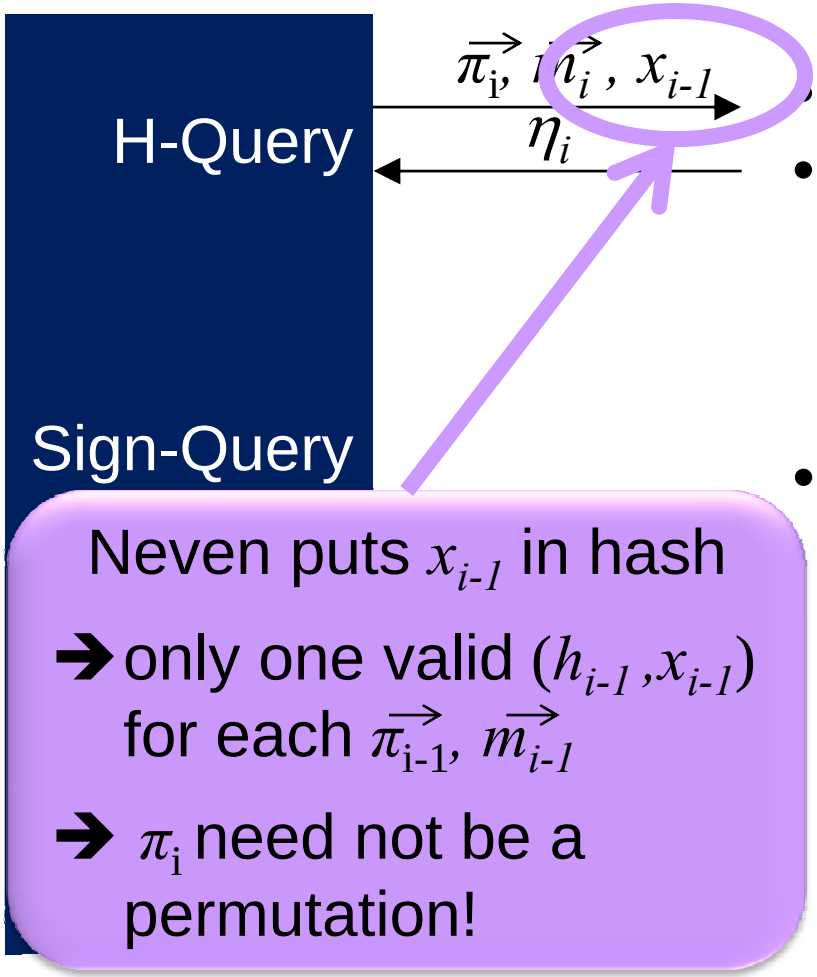
η_i, h_i, x_i

Proof Warm Up C: [Neven-08] (1B)

To simplify, let H be FDH



Proof: If forger *F* succeeds, we find a claim $\pi^*(x)=\rho^*(y)$.



- Get h_{i-1} associated with $\vec{\pi}_{i-1}$, $\vec{m}_{i-1}$, x_{i-1}
- If $\pi_i \neq \pi^*$
  - Random x_i
  - $h_i = \pi_i(x_i) \oplus x_{i-1}$
  - $\eta_i = h_{i-1} \oplus h_i$
- If $\pi_i = \pi^*$
  - Random z_i
  - $h_i = \rho^*(z_i) \oplus$
  - $\eta_i = h_{i-1} \oplus h_{i-}$

Hash Table

$\vec{\pi}_i$, $\vec{m}_i$
η_i, h_i, x_i

Neven puts x_{i-1} in hash

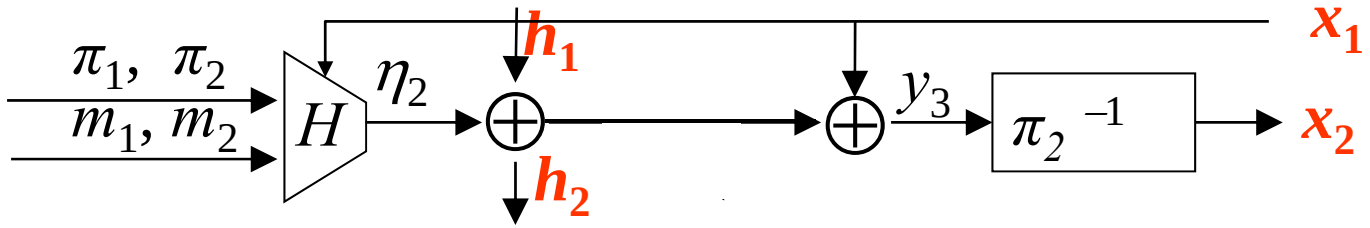
➔ only one valid (h_{i-1}, x_{i-1}) for each $\vec{\pi}_{i-1}$, $\vec{m}_{i-1}$

➔ π_i need not be a permutation!

π_i is a function ➔ there is only one h_i for each x_i and **valid** x_{i-1}

Proof Warm Up C: [Neven-08] (2A)

To simplify, let H be FDH



Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$.

H-Query

$\vec{\pi}_i, \vec{m}_i, x_{i-1}$

η_i

- Get h_{i-1} associated with $\vec{\pi}_{i-1}, \vec{m}_{i-1}, x_{i-1}$
- If $\pi_i \neq \pi^*$
  - Random x_i
  - $h_i = \pi_i(x_i) \oplus x_{i-1}$
  - $\eta_i = h_{i-1} \oplus h_i$

Hash Table

$\vec{\pi}_i, \vec{m}_i$

η_i, h_i, x_i

Sign-Query

$\vec{\pi}_i, \vec{m}_i, x_{i-1}, h_{i-1}$

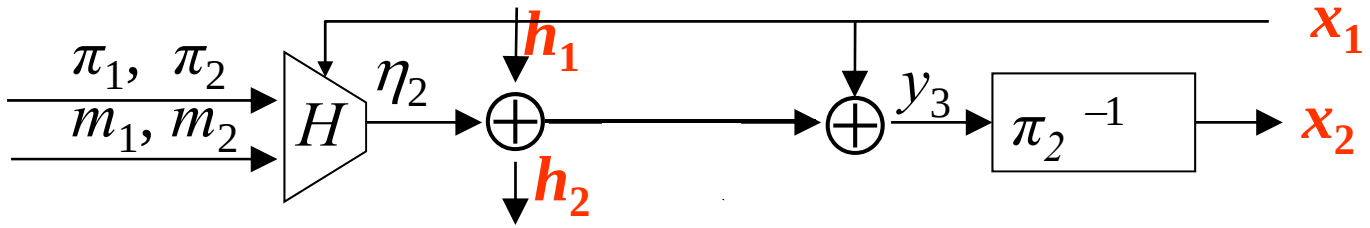
x_i, h_i

- Verify x_{i-1} h_{i-1}
- Random x_i
- $h_i = \pi_i(x_i) \oplus x_{i-1}$
- $\eta_i = h_{i-1} \oplus h_i$

Forgery

Proof Warm Up C: [Neven-08] (2B)

To simplify, let H be FDH



Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$.

H-Query

$\vec{\pi}_i, \vec{m}_i, x_{i-1}$

η_i

- Get h_{i-1} associated with $\vec{\pi}_{i-1}, \vec{m}_{i-1}, x_{i-1}$
- If $\pi_i \neq \pi^*$
  - Random x_i
  - $h_i = \pi_i(x_i) \oplus x_{i-1}$
  - $\eta_i = h_{i-1} \oplus h_i$

Hash Table

$\vec{\pi}_i, \vec{m}_i$
η_i, h_i, x_i

Sign-Query

$\vec{\pi}_i, \vec{m}_i, x_{i-1}, h_{i-1}$

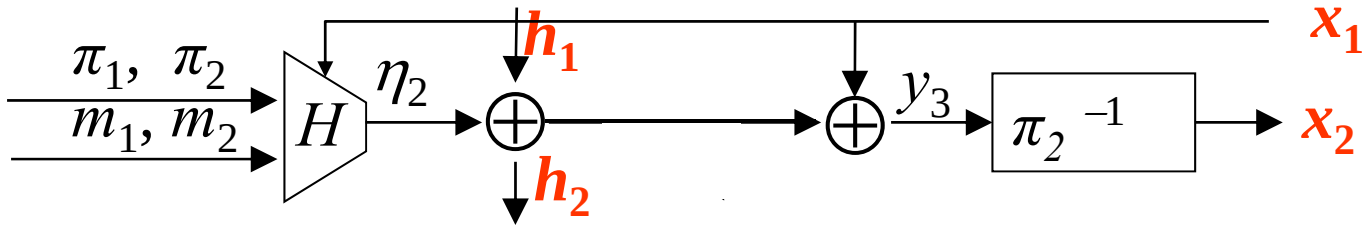
x_i, h_i

- Verify x_{i-1} h_{i-1}
- Random x_i
- $h_i = \pi_i(x_i) \oplus x_{i-1}$
- $\eta_i = h_{i-1} \oplus h_i$

Still need to make sure F can't set h_i, η_i to the wrong value on an earlier H-Query

Proof Warm Up C: [Neven-08] (3)

To simplify, let H be FDH



Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$.

H-Query

Sign-Query

Forgery

$\vec{\pi}_i, \vec{m}_i, x_{i-1}$

η_i

- Get h_{i-1} associated with $\vec{\pi}_{i-1}, \vec{m}_{i-1}, x_{i-1}$
- If $\pi_i = \pi^*$
  - Random z_i
  - $h_i = \rho^*(z_i) \oplus x_{i-1}$
  - $\eta_i = h_{i-1} \oplus h_{i-1}$

Hash Table

$\vec{\pi}_i, \vec{m}_i$

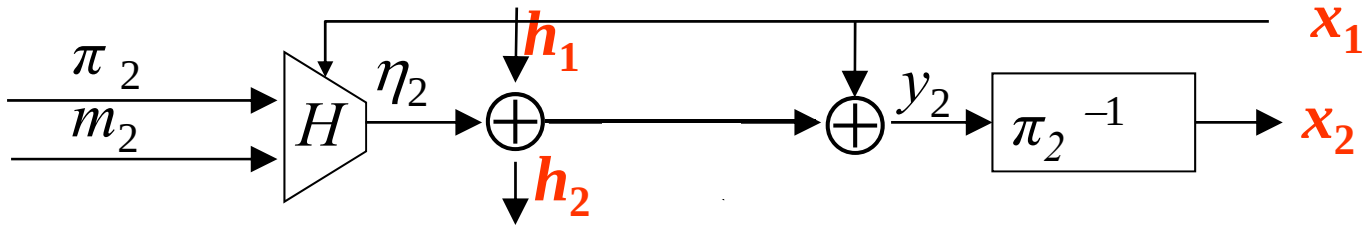
η_i, h_i, x_i

$\vec{\pi}^*, \vec{m}^*, h_i, x_i$

- Get z_i, x_{i-1} associated with $\vec{\pi}^*, \vec{m}^*$
- Return claw (x_i, z_i) because

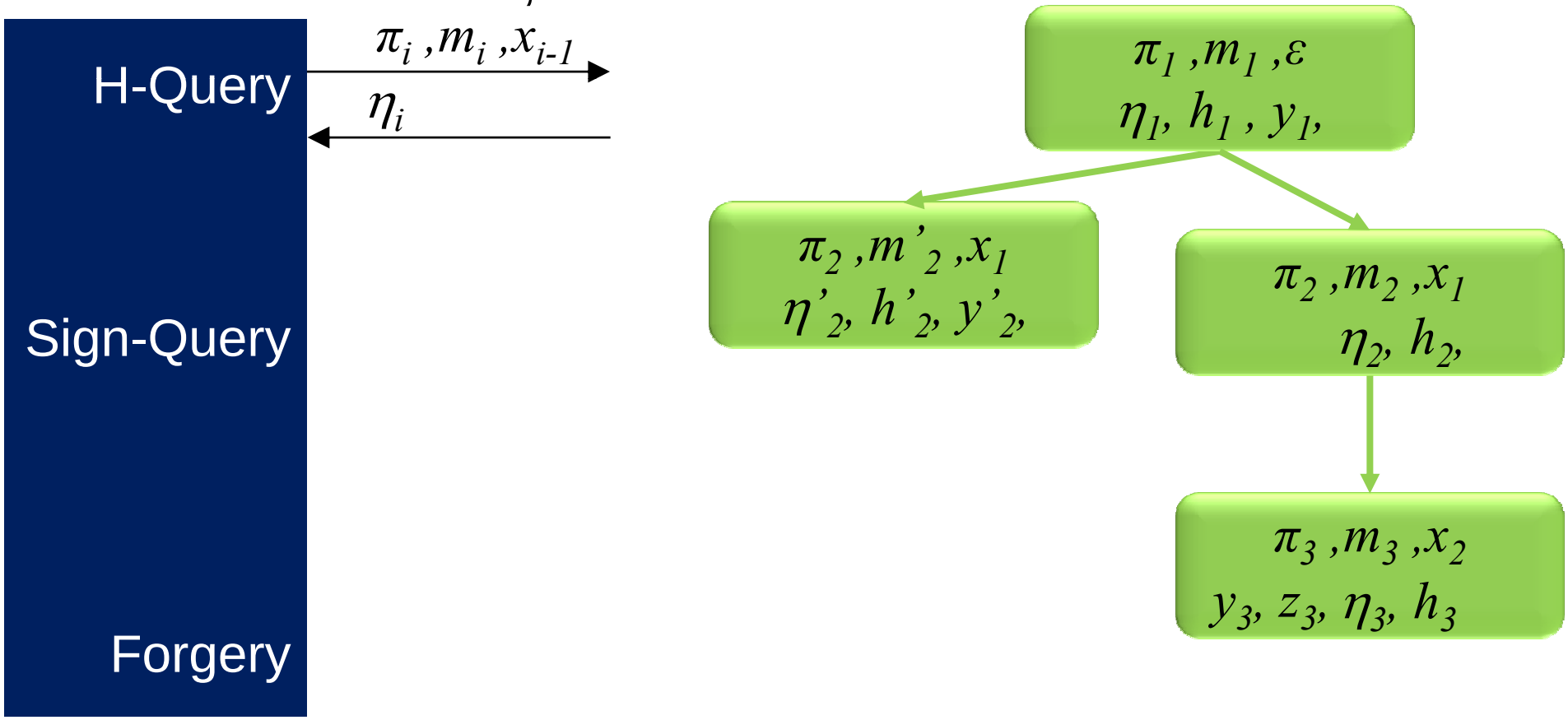
$$\rho^*(z_i) = h_i \oplus x_{i-1} = \pi_i(x_i)$$

Proof Warm Up D: Our version of [Neven-08] (1A)

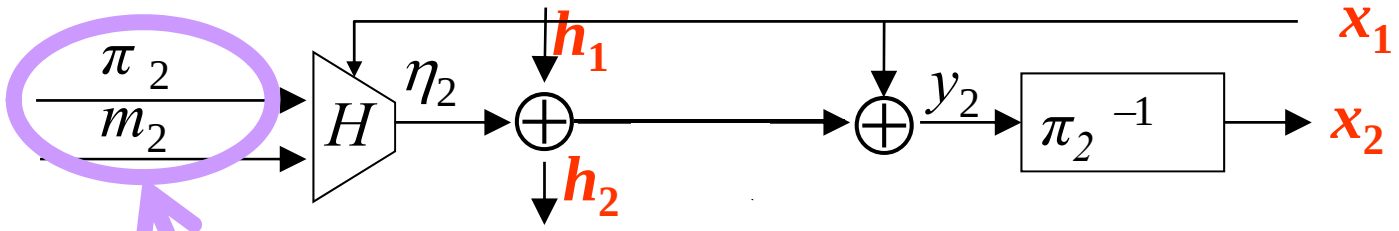


Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$.

Instead of the H-Table, we use an **H-Tree.**

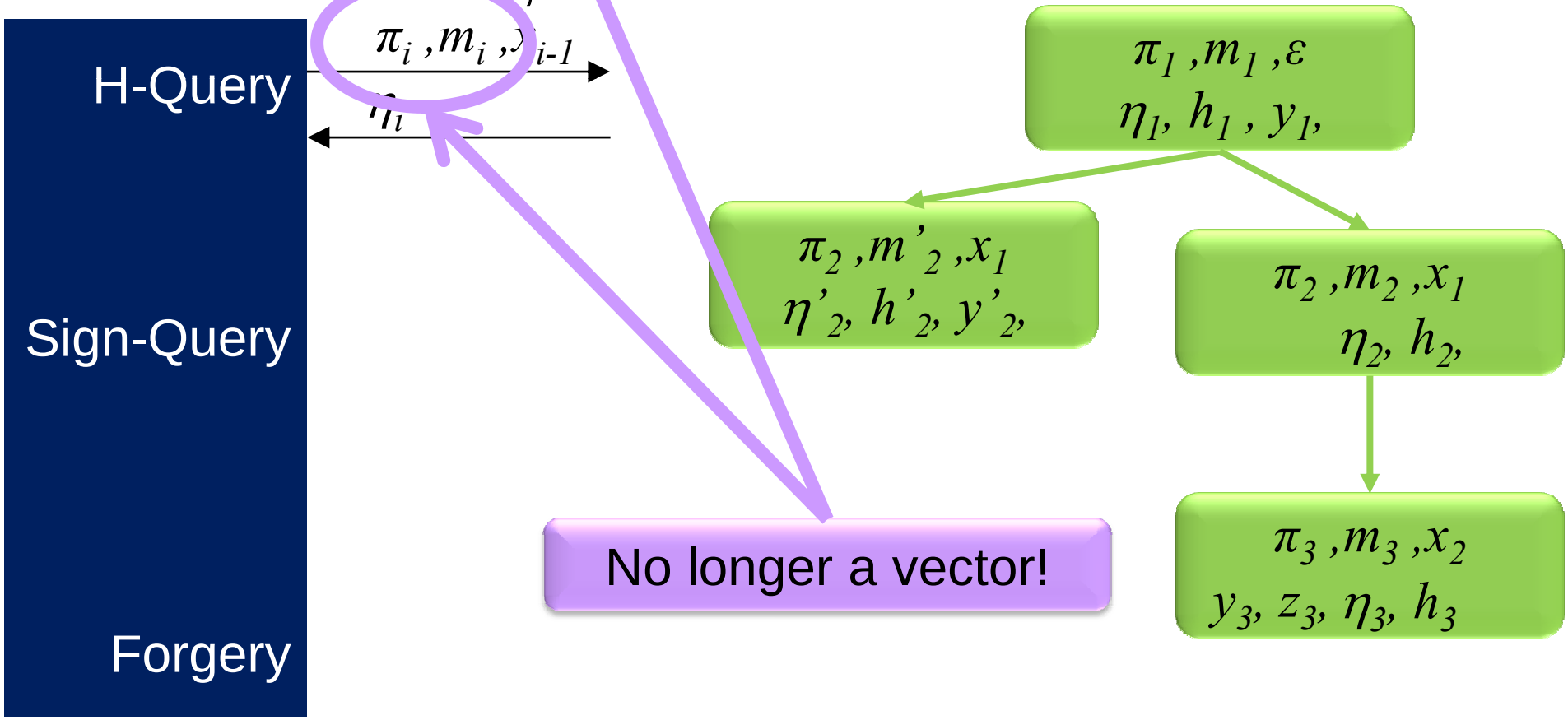


Proof Warm Up D: Our version of [Neven-08] (1B)

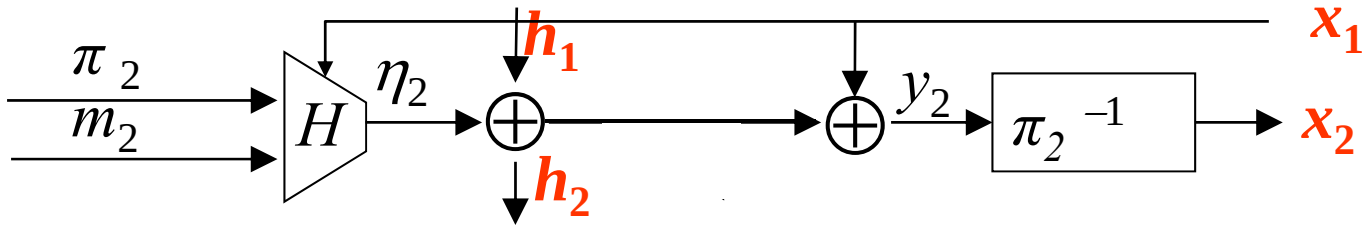


Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$.

Instead of the H-Table, we use an **H-Tree.**



Proof Warm Up D: Our version of [Neven-08] (2)



Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$.

Instead of the H-Table, we use an **H-Tree.**

H-Query

π_i, m_i, x_{i-1}

η_i

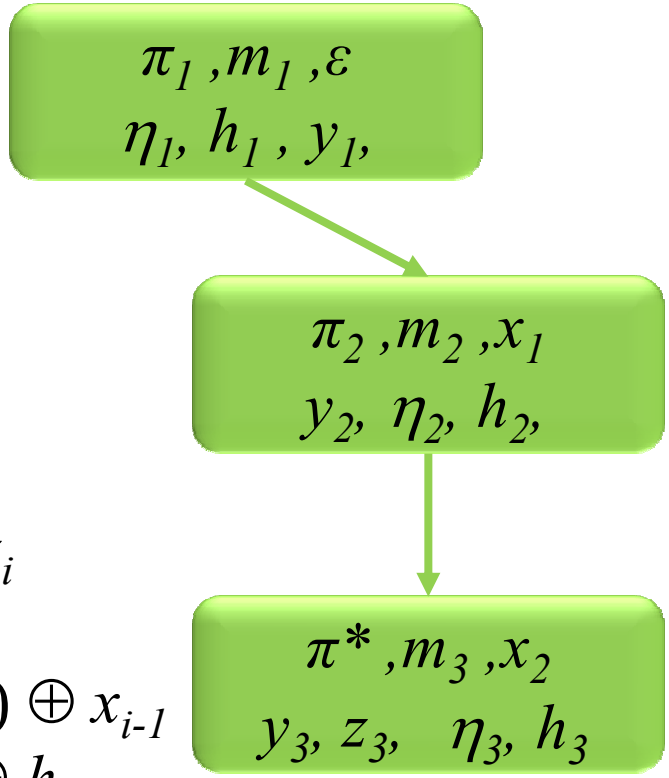
Sign-Query

- Tether to parent with (π, y) st.

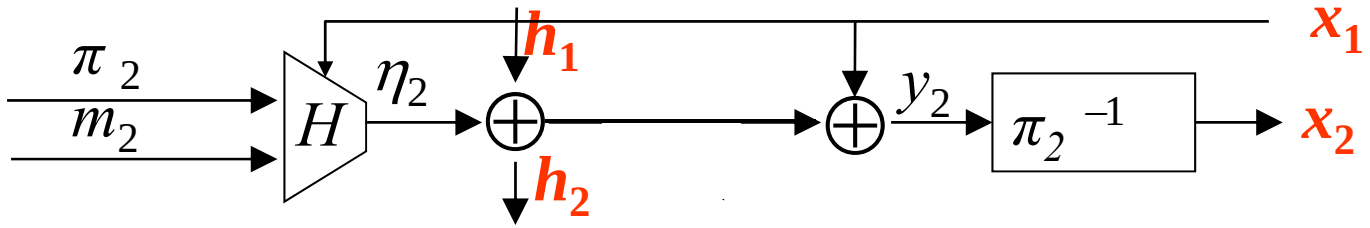
$$\pi(x_{i-1}) = y$$

- Retrieve h_{i-1} from parent
- If $\pi_i \neq \pi^*$
  - Random x_i
  - $y_i = \pi_i(x_i)$
  - $h_i = y_i \oplus x_{i-1}$
  - $\eta_i = h_{i-1} \oplus h_i$
- If $\pi_i = \pi^*$
  - Random z_i
  - $y_i = \pi_i(z_i)$
  - $h_i = \rho^*(z_i) \oplus x_{i-1}$
  - $\eta_i = h_{i-1} \oplus h_{i-1}$

Forgery



Proof Warm Up D: Our version of [Neven-08] (3)



Proof: If forger *F* succeeds, we find a claw $\pi^*(x)=\rho^*(y)$.

Instead of the H-Table, we use an **H-Tree.**

H-Query

Sign-Query

Forgery

π_i, m_i, x_{i-1}

η_i

- Tether to parent with (π, y) st. $\pi(x_{i-1}) = y$

Claim: Probability < $2^{-Length(y)}$ that $\pi^{-1}_1(y_1) = \pi^{-1}_2(y_2)$

"Proof": π_{parent} is a function. y_{parent} is random.

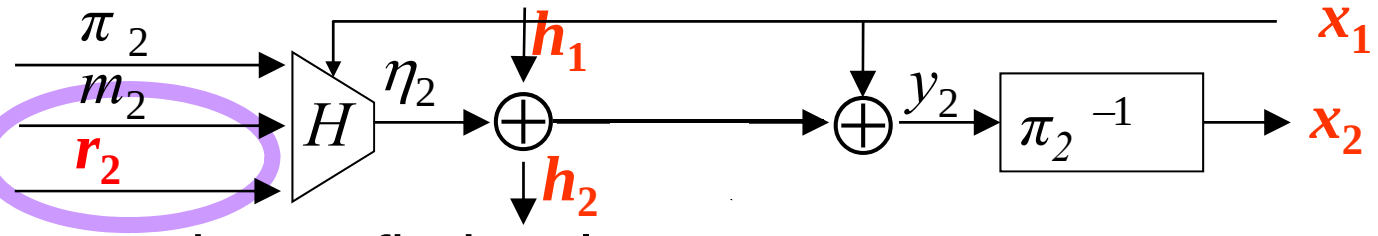
➔w.h.p only 1 parent.

π_1, m_1, ε
$\eta_1, h_1, y_1,$

π_2, m_2, x_1
$y_2, \eta_2, h_2,$

π^*, m_3, x_2
y_3, z_3, η_3, h_3

Finally! Proof of our scheme (1)



Proof: If forger F succeeds, we find a claim $\pi^*(x)=\rho^*(y)$.

Instead of the H-Table, we use an **H-Tree.**

H-Query

Sign-Query

Forgery

π_i, m_i, r_i, x_{i-1}

η_i

Randomized Hash

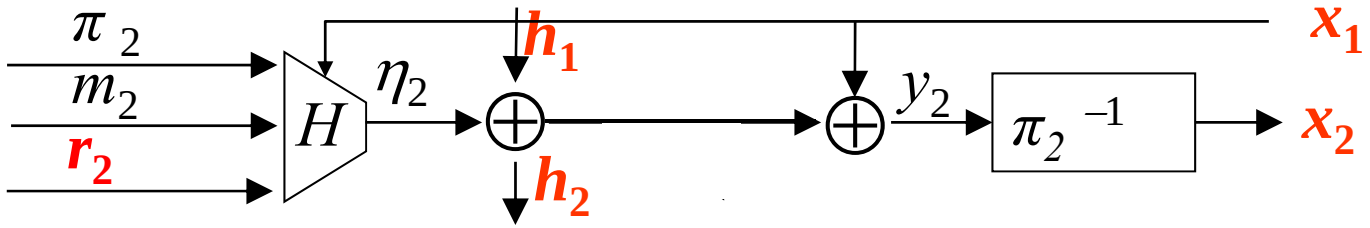
$\pi_1, m_1, r_1, \varepsilon$
$\eta_1, h_1, y_1,$

π_2, m_2, r_2, x_1
$y_2, \eta_2, h_2,$

π^*, m_3, r_3, x_2
y_3, z_3, η_3, h_3

H-Queries are the same, just add r to each node

Finally! Proof of our scheme (2)



Proof: If forger *F* succeeds, we find a claim $\pi^*(x)=\rho^*(y)$.

Instead of the H-Table, we use an **H-Tree.**

H-Query

π_i, m_i, r_i, x_{i-1}

η_i

Sign-Query

$\vec{\pi}_i, \vec{m}_i, x_{i-1}, h_{i-1}$

x_i, h_i

Abort unlikely if $2^{Length(r)} > q_H q_S$

$\pi_1, m_1, r_1, \varepsilon$
$\eta_1, h_1, y_1,$

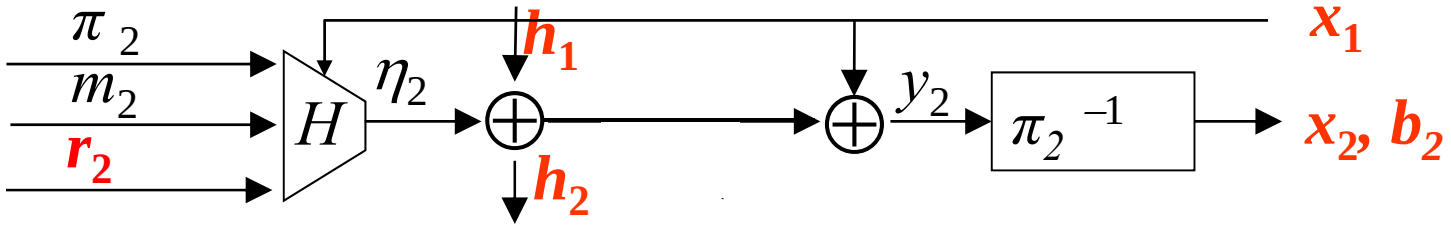
π_2, m_2, r_2, x_1
$y_2, \eta_2, h_2,$

- Random r_i x_i
- Abort if π_i, m_i, r_i, x_{i-1} in H-Tree!
- $h_i = \pi_i(x_i) \oplus x_{i-1}$
- $\eta_i = h_{i-1} \oplus h_i$

No verify!
Just abort if F set h_i, η_i
in an earlier H-Query

QED ☺

Finishing Up the Design of our Scheme



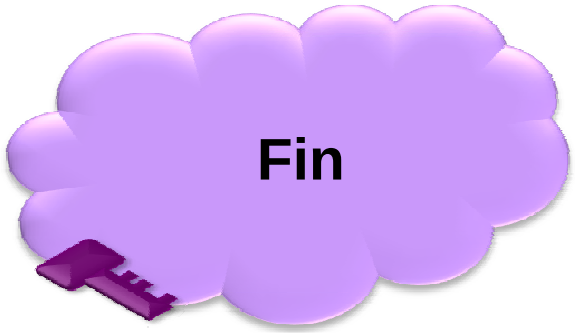
- Make H be a hash and G be a full domain hash
  - So h_i is 128 bits, not 2048 bits (i.e. $Length(h_i) = 2log(q_H)$, not $Length(\pi)$)
- Compute the randomness as $r_i = PRF(m_i, x_{i-1}, h_{i-1})$
  - Combinatorial argument reduces $Length(r)$ from $log(q_H\ q_S)$ to $2log(q_S)$
  - Because of G, a r-collision in the H-Query (i.e. on π_i, m_i, r_i, x_{i-1})
  - … is not always a collision on an G-query! (which also takes in h_{i-1})
- Remove and carry around one bit of x_i
  - To deal with permutations over different domains (different RSA keys)
  - Now signature grows by 129 bits / signer, not 128 bits / signer.

Conclusions and Open Questions

- Sequential Aggregate Signatures
  - ☺ Based on claw-free permutation in the random oracle model
  - ☺ Lazy verification
  - ☺ No knowledge of other's public keys required to sign
  - ☺ Fully spec'd and implemented in OpenSSL
  - ☹ Sig length grows ~128 bits/ signer

Can we improve on this?

- But: pairings are getting faster and faster
  - With pairings, shorter signatures and can be non-sequential
  - Still far from RSA verification with short exponent e
  - Speed requires very special curves, how secure are they?



http://www.cs.bu.edu/~goldbe/papers/bgpsec-sigs.html