A Unified Framework for Trapdoor-Permutation-Based Sequential Aggregate Signatures

Craig Gentry
IBM

Adam O'Neill
Georgetown U.

Leonid Reyzin
Boston U.

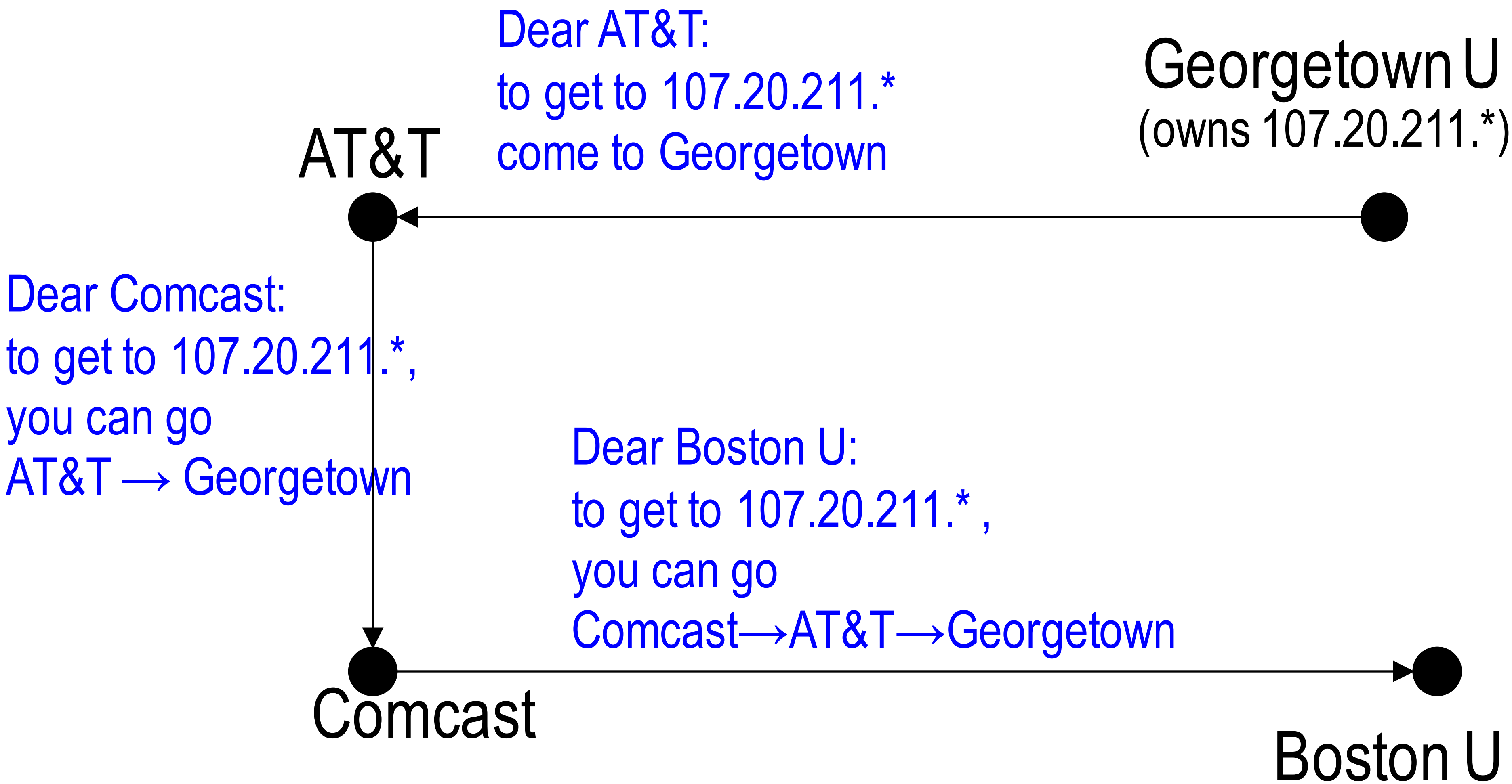
Motivating Example: Border Gateway Protocol (BGP)

- Q: How do you get from here to there on the internet?
- A: BGP [Rekhter, Lougheed, Li, Hares]

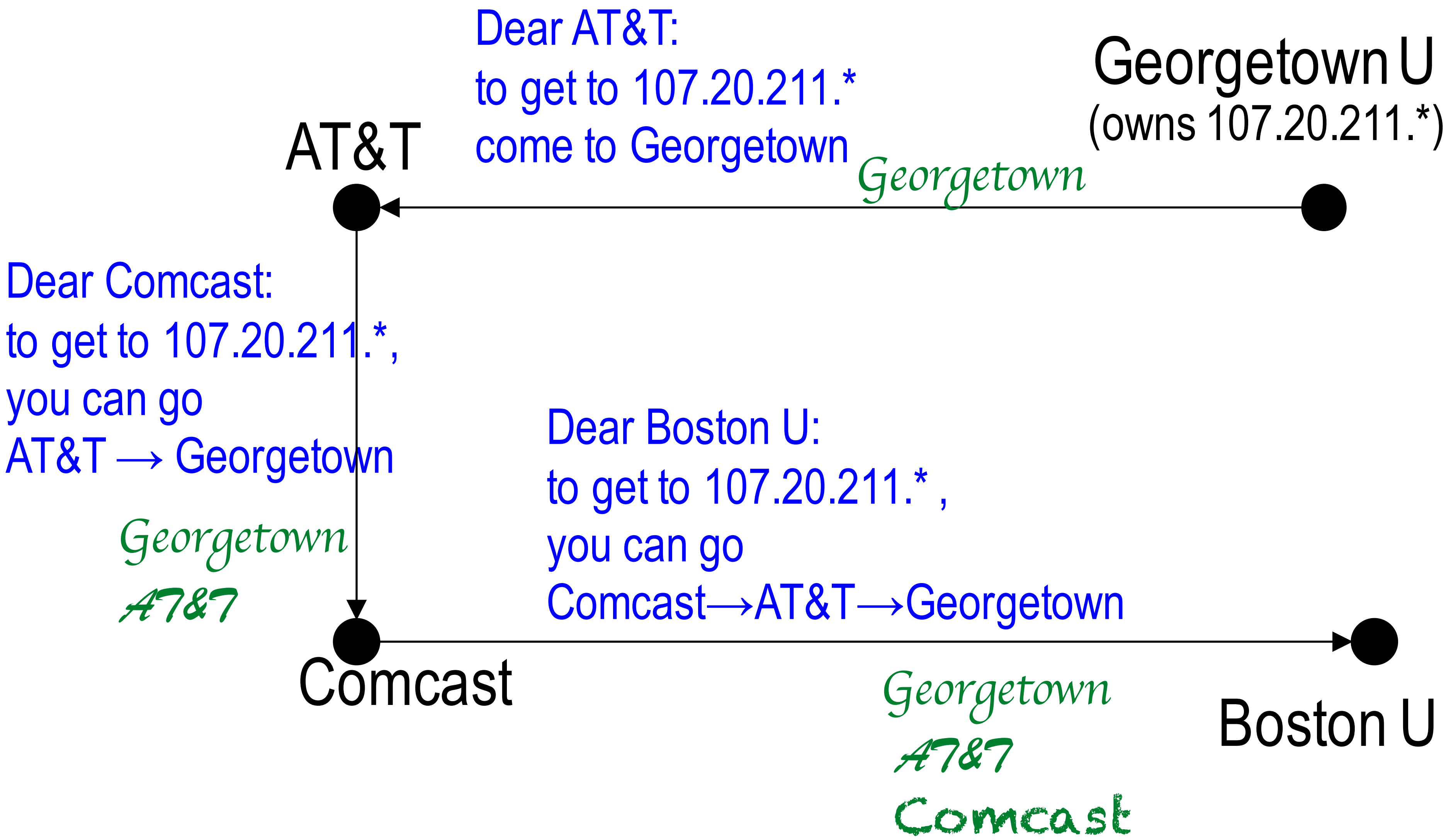
Idea: utilize local knowledge

- Each autonomous system (AS) knows what IP addresses it owns
- Each AS knows its connections (customer-provider, peer)
- Each AS can talk to its neighbors

Border Gateway Protocol (BGP)



Border Gateway Protocol (BGP)

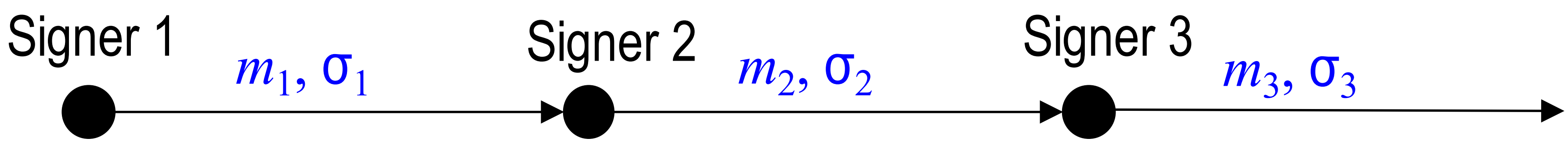


S-BGP [Kent-Lynn-Seo 2000]: Same but with signatures

Sequential Aggregate Signatures (SAS)

- S-BGP requires possibly long signature chains
- Q: Can we compress multiple signatures to save space?
- A: Sequential Aggregate Signatures (SAS)

[Lysyanskaya Micali Reyzin Shacham 04]:



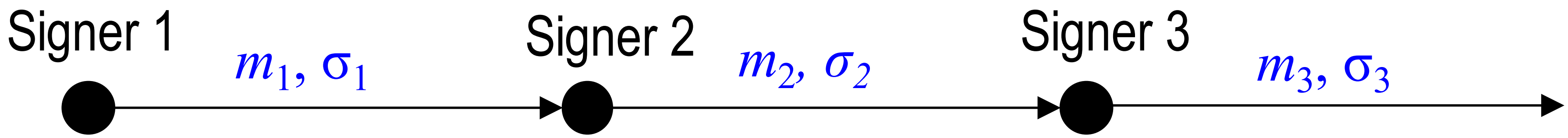
σ_i attests to $m_1\ m_2\ \ldots\ m_i$ on behalf of $PK_1\ PK_2\ \ldots\ PK_i$

- Several prior TDP-based constructions
  - Note: [Boneh Gentry Lynn Shacham 2003] allow non-sequential (even third-party) aggregation post signing, but based on pairings
- <u>This work</u>: understanding + improving TDP-based Sequential Aggregate Signatures

Outline

- Sequential Aggregate Signatures (SAS)
- Security Definition
- Prior Constructions
  - [LMRS]
  - [Neven]
- Our General Construction
  - History-free variants

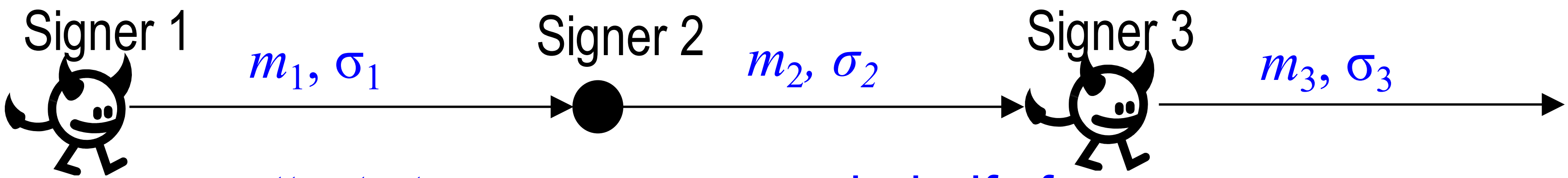
SAS Security



σ_i attests to $m_1\ m_2\ \ldots\ m_i$ on behalf of $PK_1\ PK_2\ \ldots\ PK_i$

- Equivalent to what you get from simply concatenating individual signatures, without any aggregation
- Adversary model: arbitrary subset of adversarial signers

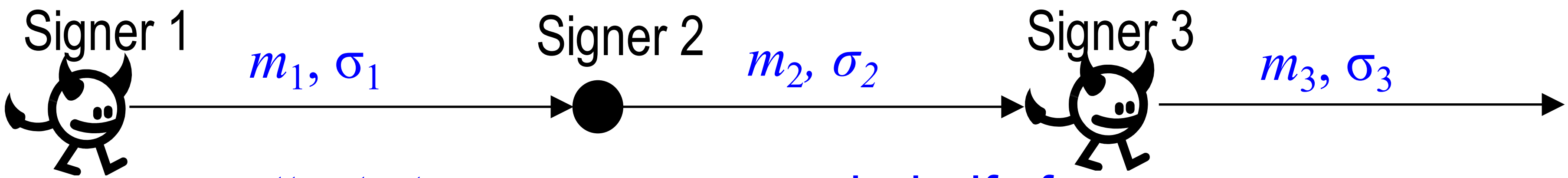
SAS Security



σ_i attests to $m_1\ m_2\ \ldots\ m_i$ on behalf of $PK_1\ PK_2\ \ldots\ PK_i$

- Equivalent to what you get from simply concatenating individual signatures, without any aggregation
- Adversary model: arbitrary subset of adversarial signers

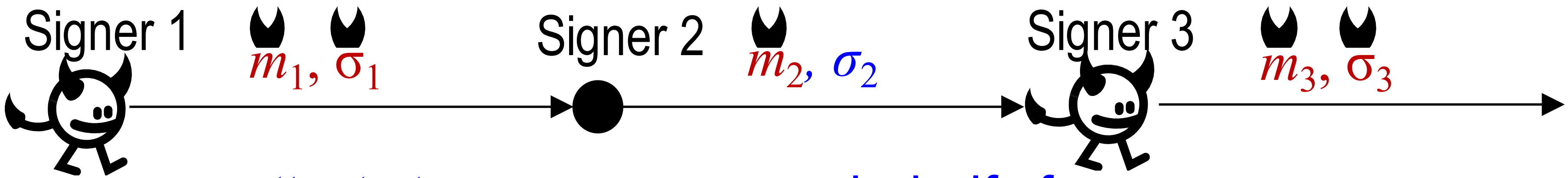
SAS Security



σ_i attests to $m_1\ m_2\ \ldots\ m_i$ on behalf of $PK_1\ PK_2\ \ldots\ PK_i$

- Equivalent to what you get from simply concatenating individual signatures, without any aggregation
- Adversary model: arbitrary subset of adversarial signers
- Chosen Message-and-Aggregate-so-Far attack

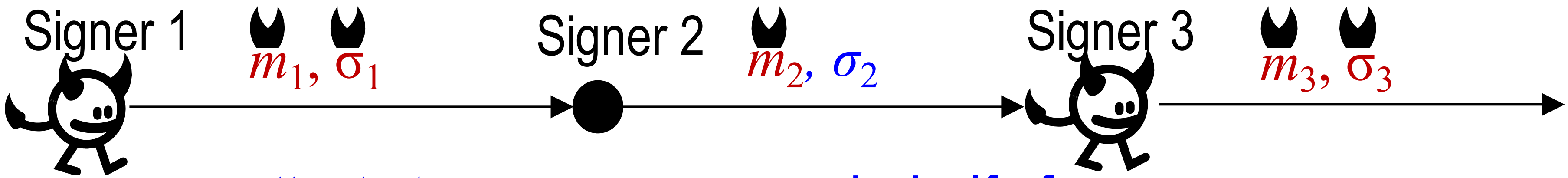
SAS Security



σ_i attests to $m_1\ m_2\ \ldots\ m_i$ on behalf of $PK_1\ PK_2\ \ldots\ PK_i$

- Equivalent to what you get from simply concatenating individual signatures, without any aggregation
- Adversary model: arbitrary subset of adversarial signers
- Chosen Message-and-Aggregate-so-Far attack

SAS Security



σ_i attests to m_1 m_2 … m_i on behalf of PK_1 PK_2 … PK_i

- Equivalent to what you get from simply concatenating individual signatures, without any aggregation
- Adversary model: arbitrary subset of adversarial signers
- Chosen Message-and-Aggregate-so-Far attack
- Even after such an attack,
  adversary can't "frame" the honest parties
  - Adversary can't output any $(m_1^*, m_2^*, m_3^*, \sigma_3^*)$ that verifies as long as Signer 2 never signed m_2^*

Outline

- Sequential Aggregate Signatures (SAS)
- Security Definition
- Prior Constructions
  - [LMRS]
  - [Neven]
- Our General Construction
  - History-free variants

Review: Full-Domain Hash Signatures

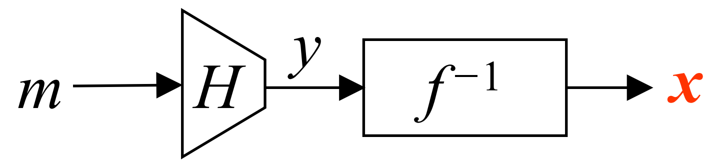
[Bellare-Rogaway 93]

Trapdoor permutation public key $PK=f$, secret key $SK=f^{-1}$

Hash (random oracle) function H (output range equals domain of f)

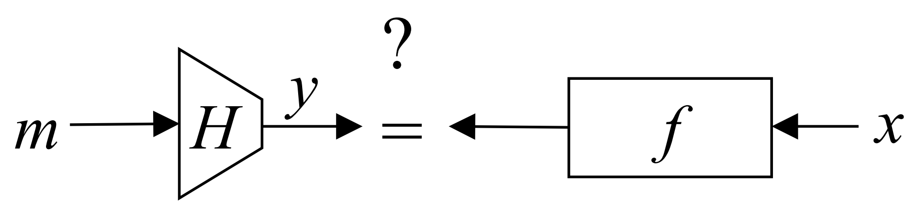
<u>Steps of the Signer:</u>

- $y = H(m)$
- $\boldsymbol{x} = f^{-1}(y)$



<u>Steps of the Verifier:</u>

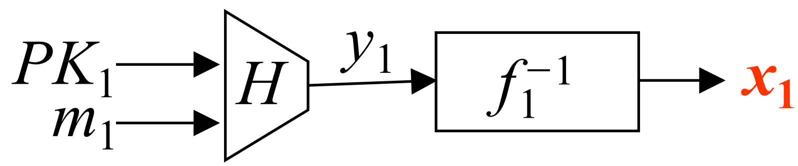
- $y = H(m)$
- $y \stackrel{?}{=} f(x)$



LMRS Aggregate Signature Scheme

[Lysyanskaya-Micali-R-Shacham 04]

<u>Signer 1:</u>

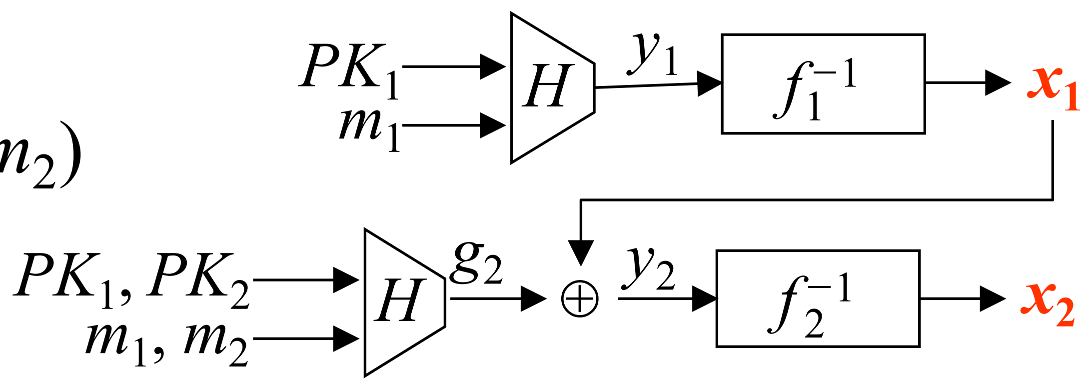


LMRS Aggregate Signature Scheme

[Lysyanskaya-Micali-R-Shacham 04]

Steps of Signer 2:

- Check that $PK_1 = f_1$ specifies a permutation
- Verify x_1 using PK_1, m_1
- $g_2 = H(PK_1, PK_2, m_1, m_2)$
- $y_2 = g_2 \oplus x_1$
- $\mathbf{x_2} = f_2^{-1}(y_2)$



Steps of Signer 3:

- Check that $PK_1 = f_1, PK_2 = f_2$ specify permutations
- Verify x_2 using PK_1, PK_2, m_1, m_2

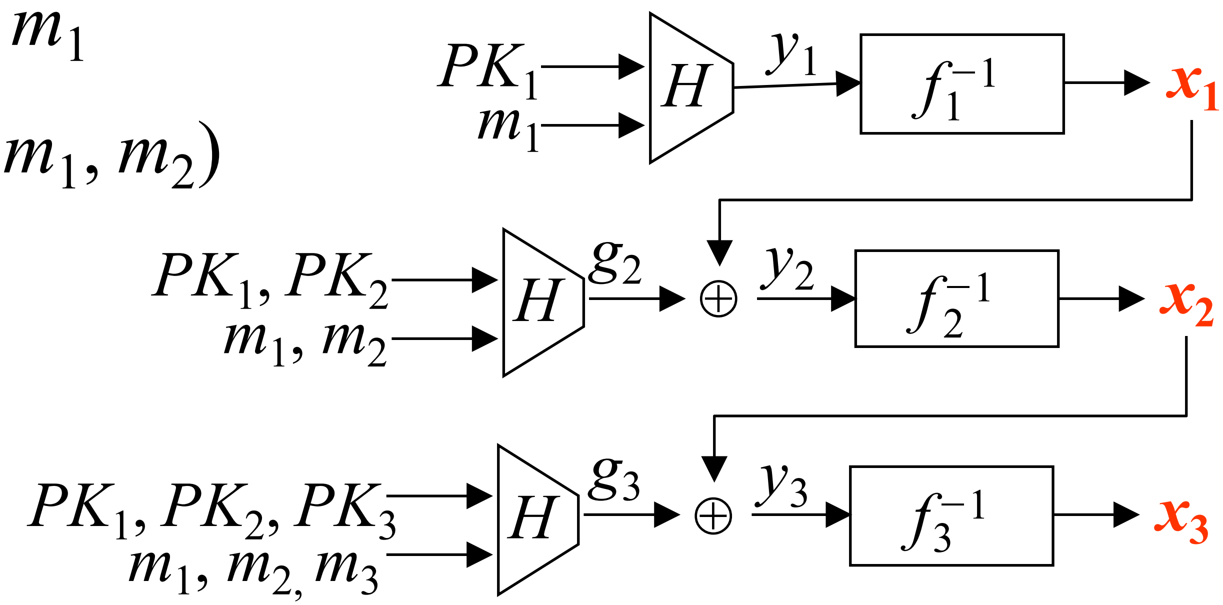
…

LMRS Aggregate Signature Scheme

[Lysyanskaya-Micali-R-Shacham 04]

Steps of Signer 2:

- Check that $PK_1 = f_1$ specifies a permutation
- Verify x_1 using PK_1, m_1
- $g_2 = H\,(PK_1, PK_2, m_1, m_2)$
- $y_2 = g_2 \oplus x_1$
- $\mathbf{x_2} = f_2^{-1}(\, y_2 \,)$



Steps of Signer 3:

- Check that $PK_1 = f_1, PK_2 = f_2$ specify permutations
- Verify x_2 using $PK_1, PK_2,\ m_1, m_2$

…

LMRS Aggregate Signature Scheme

[Lysyanskaya-Micali-R-Shacham 04]

Steps of Signer 2:

- Check that $PK_1 = f_1$ specifies a permutation
- Verify x_1 using PK_1, m_1
- $g_2 = H(PK_1, PK_2, m_1, m_2)$
- $y_2 = g_2 \oplus x_1$
- $\mathbf{x_2} = f_2^{-1}(y_2)$

getting certified TDPs takes work:
for RSA, either extra proofs
[Goldberg-Reyzin-Sagga-Baldimtsi 18]
[Auerbach-Poettering 18]
or long verification exponents

Steps of Signer 3:

- Check that $PK_1 = f_1, PK_2 = f_2$ specify permutations
- Verify x_2 using PK_1, PK_2, m_1, m_2

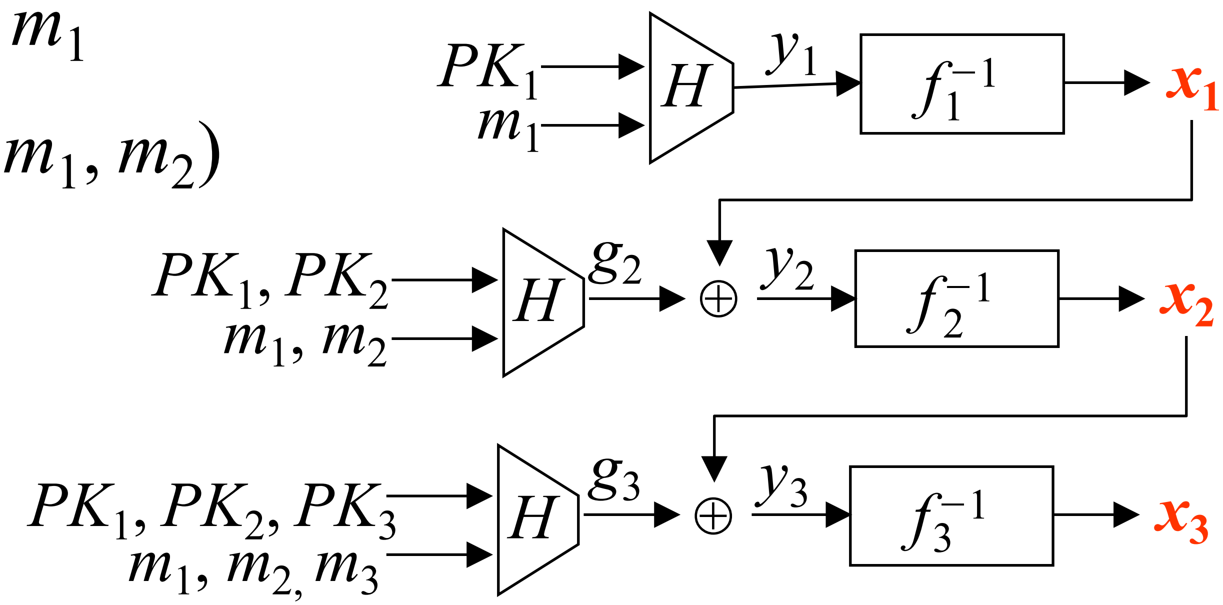
…

LMRS Aggregate Signature Scheme

[Lysyanskaya-Micali-R-Shacham 04]

Steps of Signer 2:

- Check that $PK_1 = f_1$ specifies a permutation
- Verify x_1 using PK_1, m_1
- $g_2 = H\,(PK_1, PK_2, m_1, m_2)$
- $y_2 = g_2 \oplus x_1$
- $\boldsymbol{x_2} = f_2^{-1}(\ y_2\ )$



Steps of Signer 3:

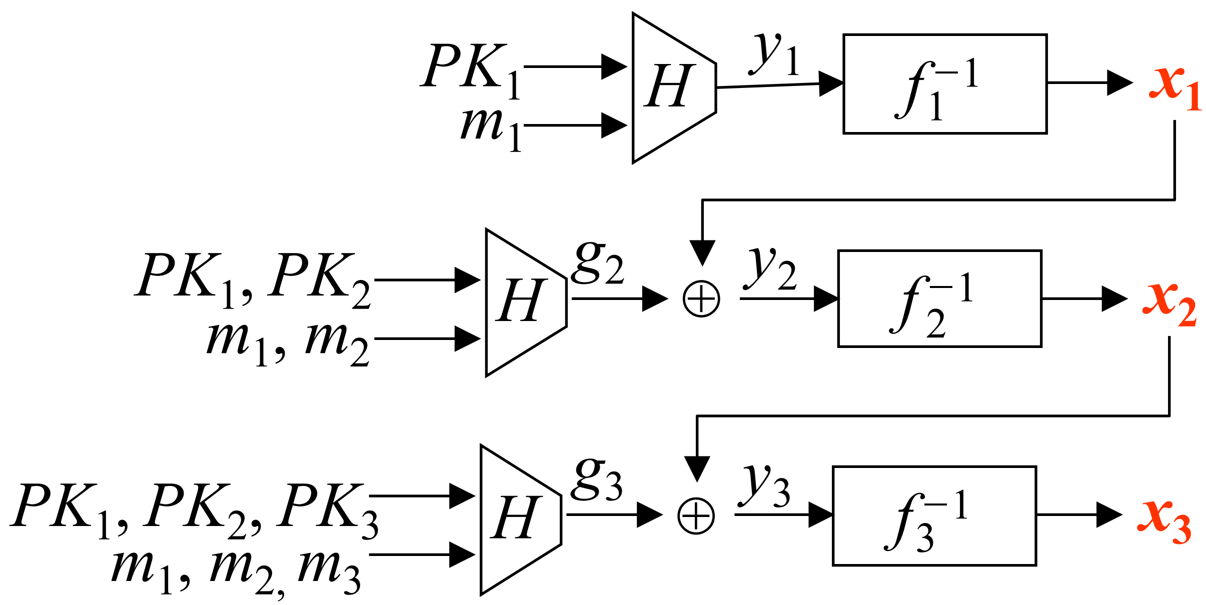
- Check that $PK_1 = f_1, PK_2 = f_2$ specify permutations
- Verify x_2 using $PK_1, PK_2,\ m_1, m_2$

…

LMRS Aggregate Signature Scheme

[Lysyanskaya-Micali-R-Shacham 04]

Q: What happens if f_1 is not a permutation?

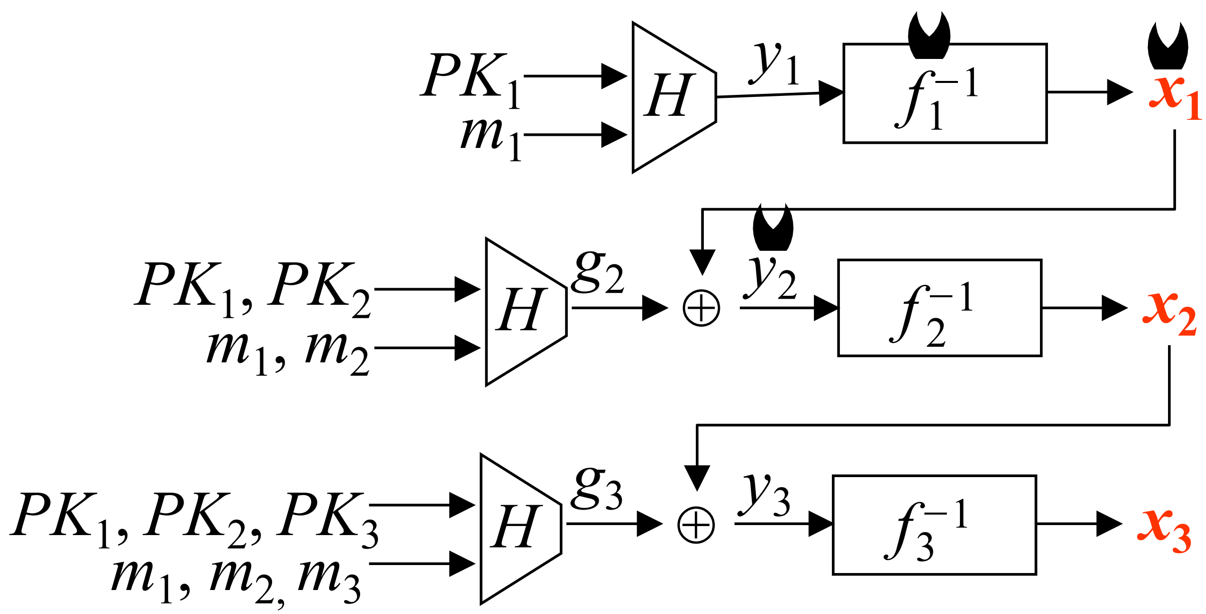


LMRS Aggregate Signature Scheme

[Lysyanskaya-Micali-R-Shacham 04]

Q: What happens if f_1 is not a permutation?

A: Adversary can control input to f_2 and thus attack signer 2!



LMRS Aggregate Signature Scheme

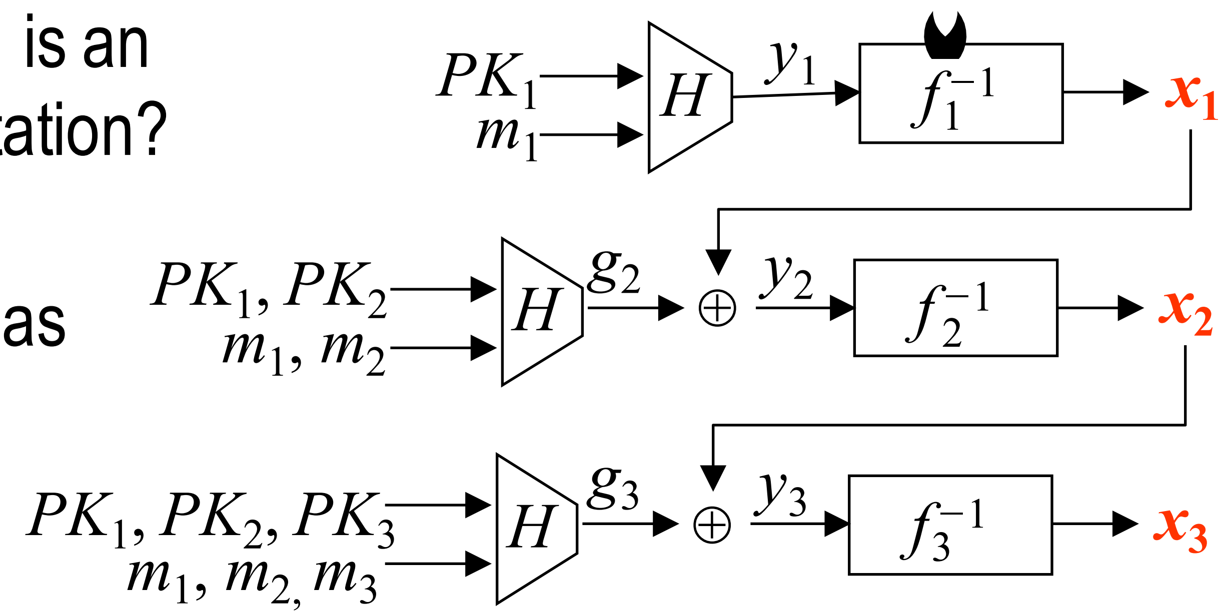
[Lysyanskaya-Micali-R-Shacham 04]

Q: What happens if f_1 is not a permutation?

A: Adversary can control input to f_2 and thus attack signer 2!

Q: What happens if f_1 is an adversarial permutation?

Q: Verify-before-sign means adversary has no control over x_1

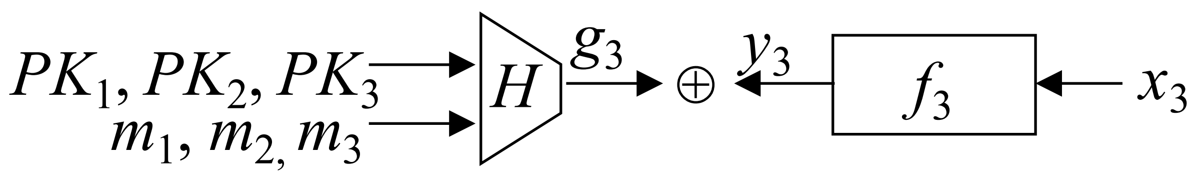


LMRS Verification

Verifier knows: last signature x_3,

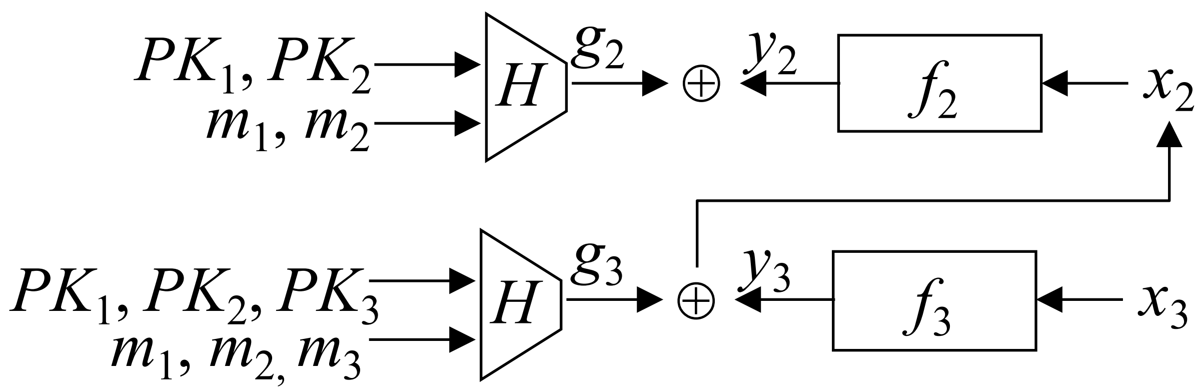
messages m_1, m_2, m_3

public keys $PK_1 = f_1, PK_2 = f_2, PK_3 = f_3$



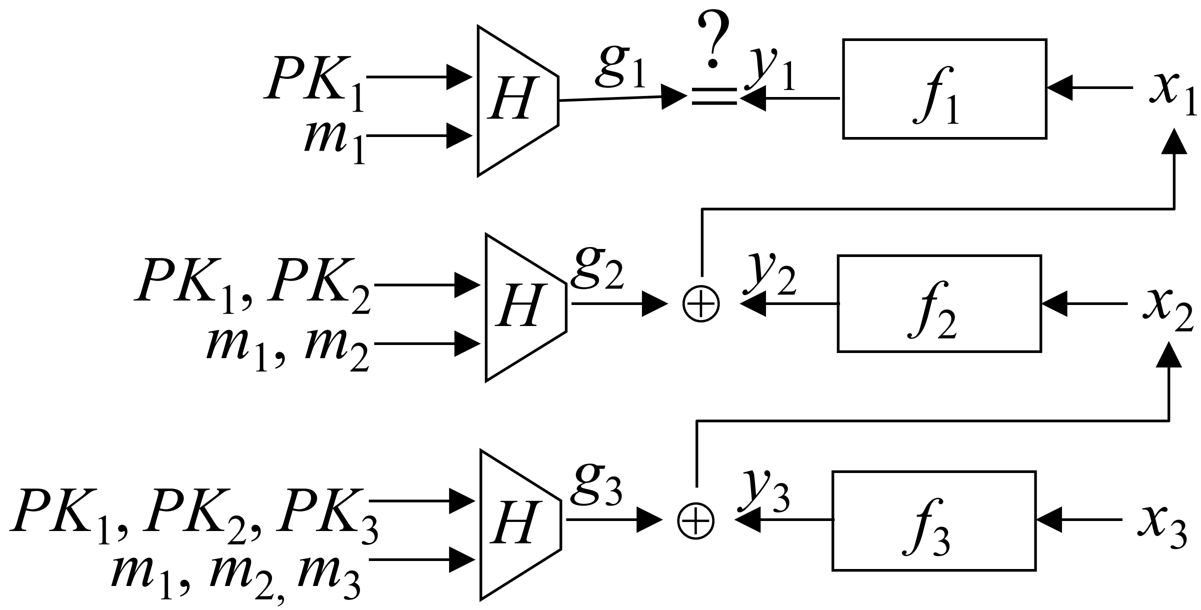
LMRS Verification

Verifier knows: last signature x_3,
messages m_1, m_2, m_3
public keys $PK_1 = f_1, PK_2 = f_2, PK_3 = f_3$



LMRS Verification

Verifier knows: last signature x_3,
messages m_1, m_2, m_3
public keys $PK_1 = f_1, PK_2 = f_2, PK_3 = f_3$



To sum up: scheme works because $\oplus$ can be undone,
but requires <u>certified</u> trapdoor permutations

Outline

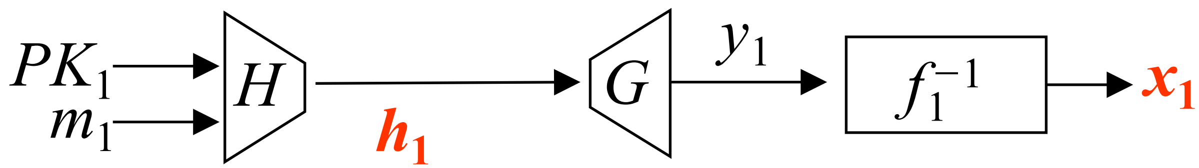
- Sequential Aggregate Signatures (SAS)
- Security Definition
- Prior Constructions
  - [LMRS]: requires certified TDPs
  - [Neven]: works even adversary gives nonpermutations!
- Our General Construction
  - History-free variants

[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

<u>Steps of Signer 1:</u>

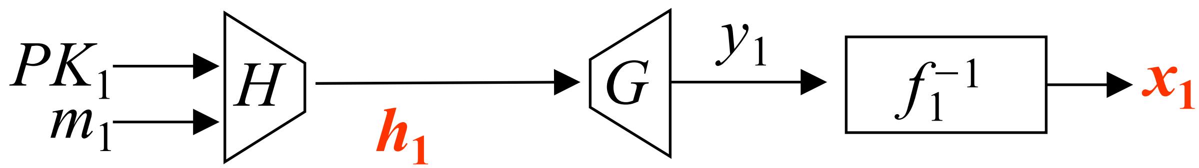


[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

Steps of Signer 2: First, verify (x_1, h_1) using PK_1, m_1



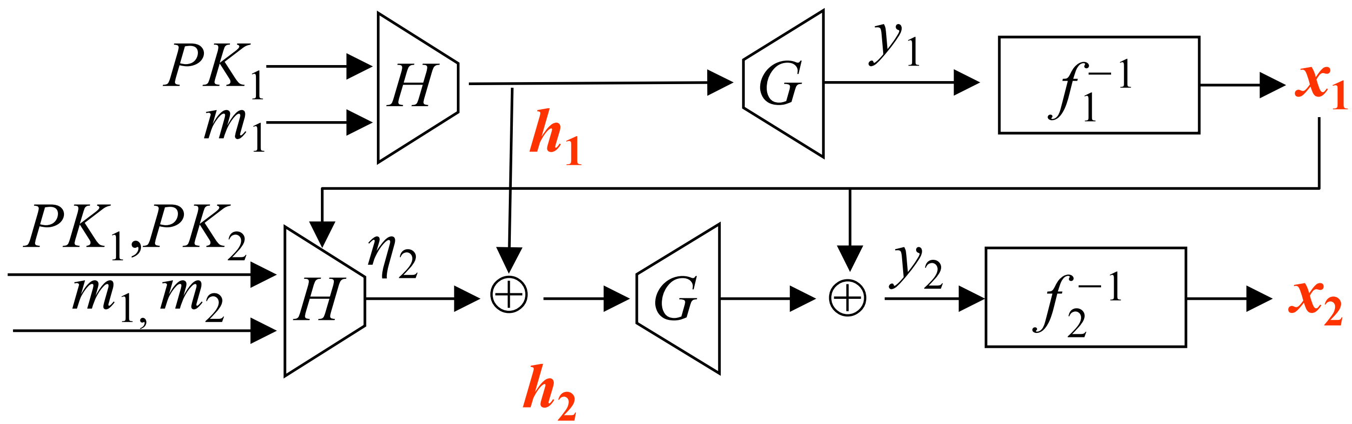
[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

Steps of Signer 2: First, verify (x_1, h_1) using PK_1, m_1

- $\eta_2 = H(PK_1, PK_2, x_1, m_1, m_2)$
- $\boldsymbol{h_2} = \eta_2 \oplus h_1$
- $y_2 = G(h_2) \oplus x_1$
- $\boldsymbol{x_2} = f_2^{-1}(y_2)$



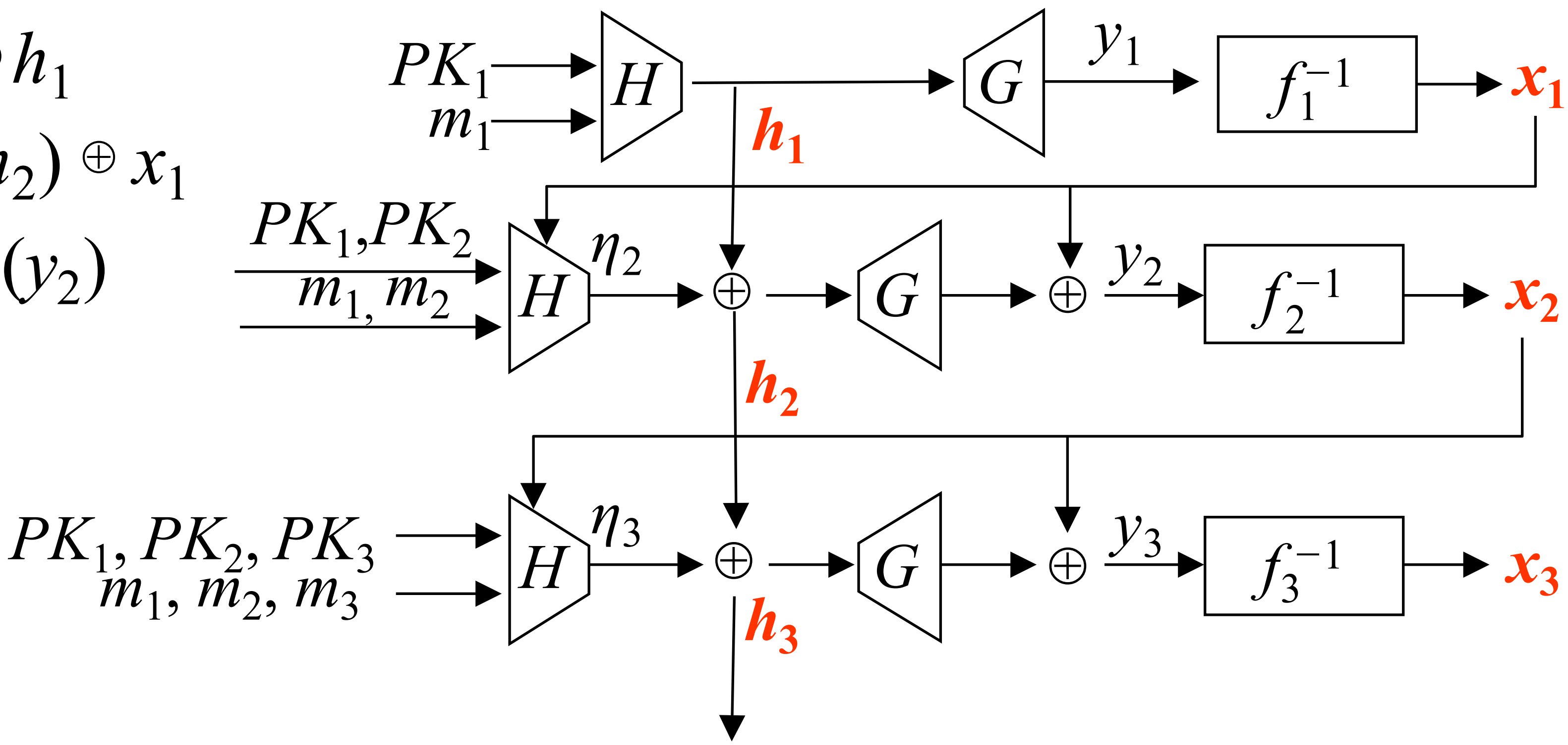
[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

<u>Steps of Signer 2:</u> First, verify (x_1, h_1) using PK_1, m_1

- $\eta_2 = H\,(PK_1, PK_2, x_1, m_1, m_2)$
- $\boldsymbol{h_2} = \eta_2 \oplus h_1$
- $y_2 = G(h_2) \oplus x_1$
- $\boldsymbol{x_2} = f_2^{-1}(y_2)$



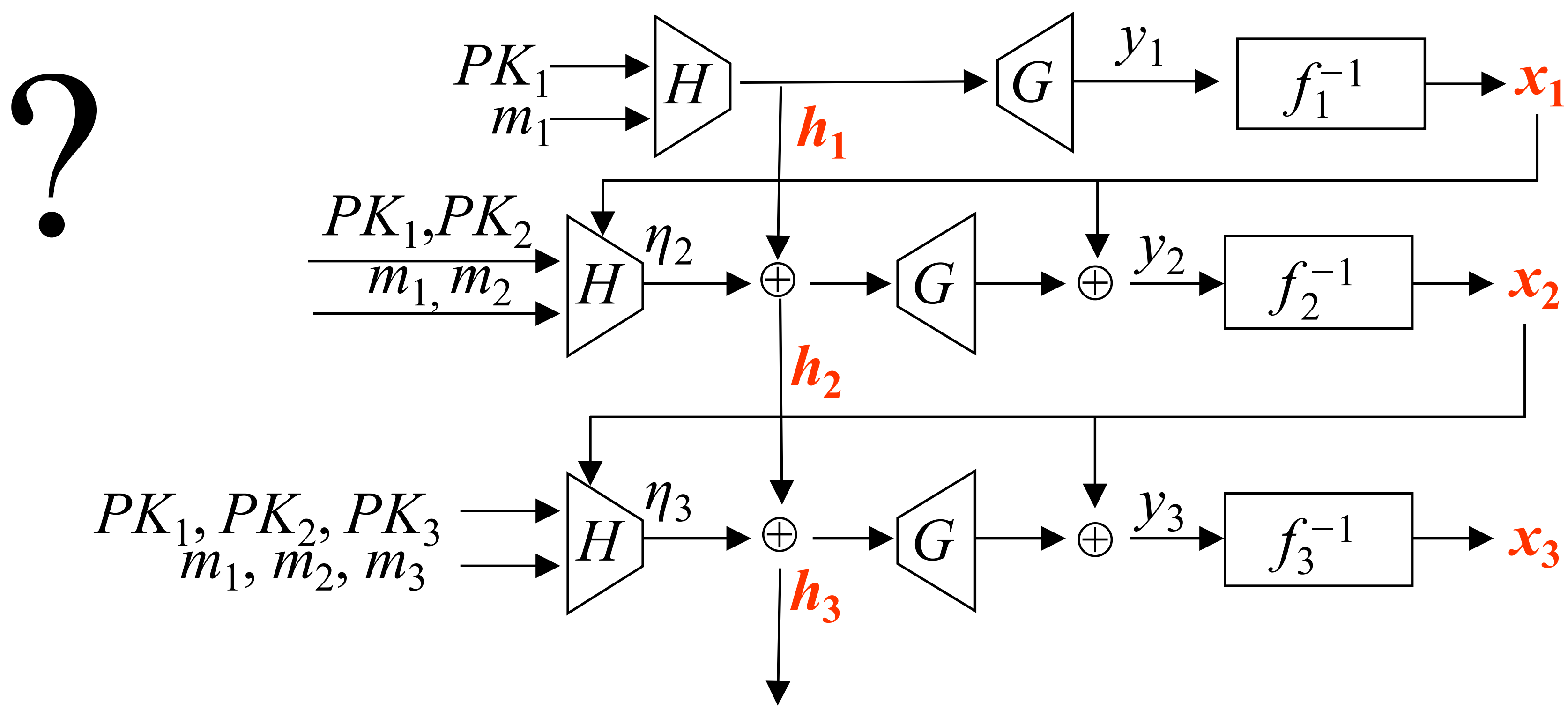
<u>Steps of Signer 3:</u> First, verify (x_2, h_2) using $PK_1, PK_2,\ m_1, m_2$

…

[Neven08] Aggregate Signature Scheme

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

Signature has two components: (x, h)



Q: How do even verify?

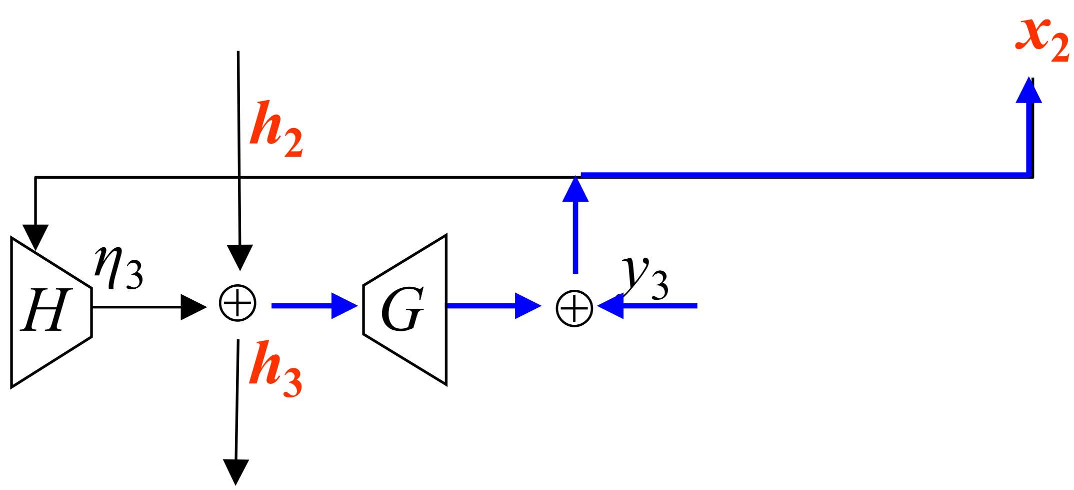
[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

The transformation from (x_2, h_2) to (y_3, h_3) is invertible!

$$x_2 = G(h_3) \oplus y_3$$



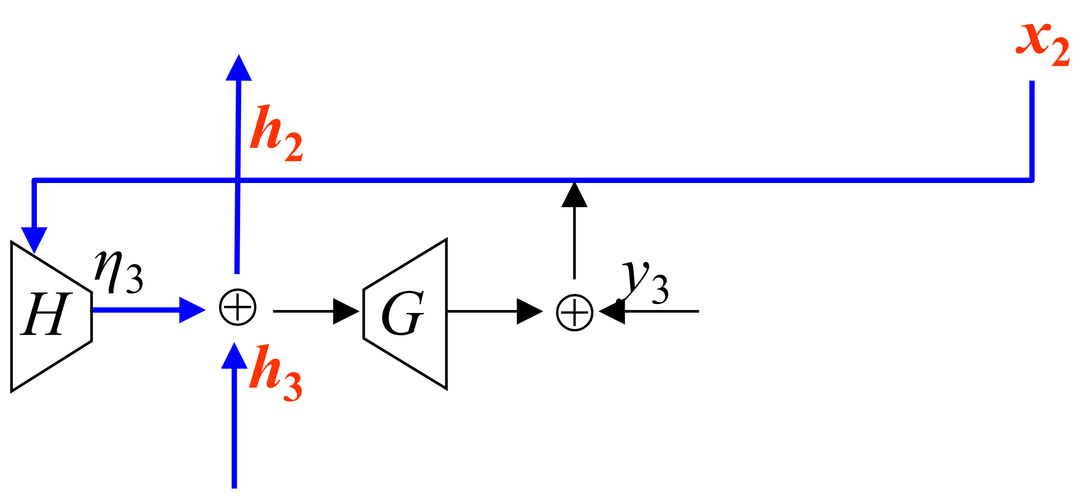
[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

The transformation from (x_2, h_2) to (y_3, h_3) is invertible!

$$x_2 = G(h_3) \oplus y_3$$
$$h_2 = H(x_2) \oplus h_3$$



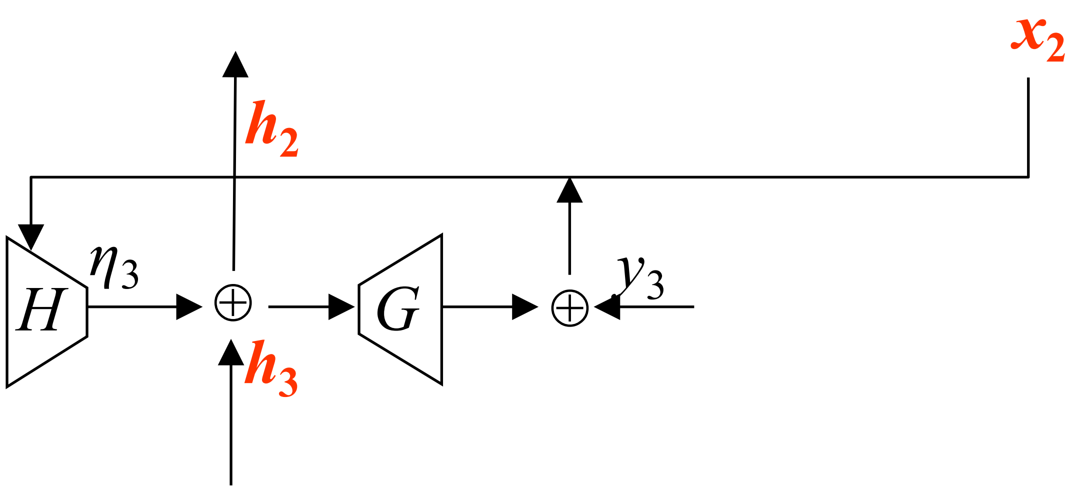
[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

The transformation from (x_2, h_2) to (y_3, h_3) is invertible!

$$\left.\begin{aligned} x_2 &= G(h_3) \oplus y_3 \\ h_2 &= H(x_2) \oplus h_3 \end{aligned}\right\}$$ This is just 2 rounds of (unbalanced) Feistel



[Neven08] Aggregate Signature Scheme

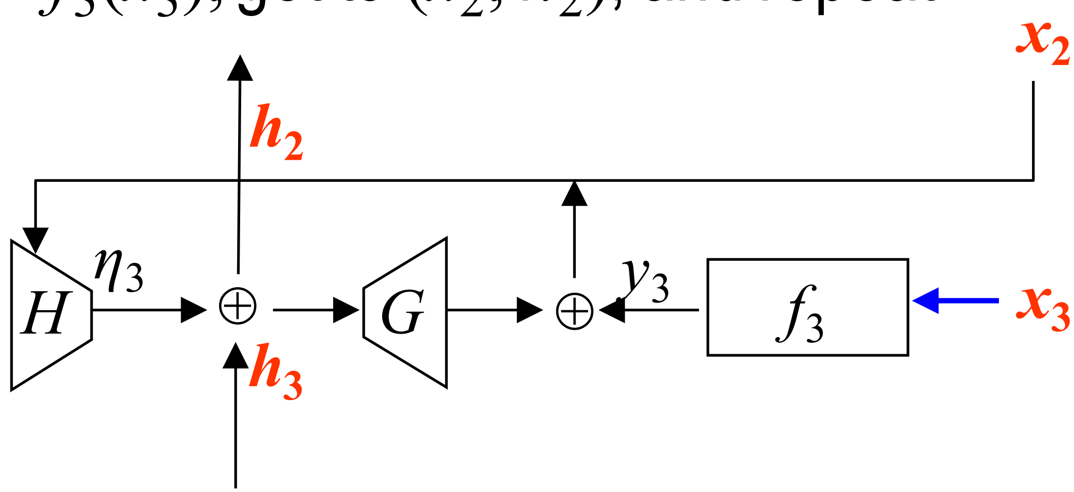
Hash function H (short outputs), G (full domain outputs)

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

The transformation from (x_2, h_2) to (y_3, h_3) is invertible!

$$\left.\begin{aligned} x_2 &= G(h_3) \oplus y_3 \\ h_2 &= H(x_2) \oplus h_3 \end{aligned}\right\} \text{This is just 2 rounds of (unbalanced) Feistel}$$

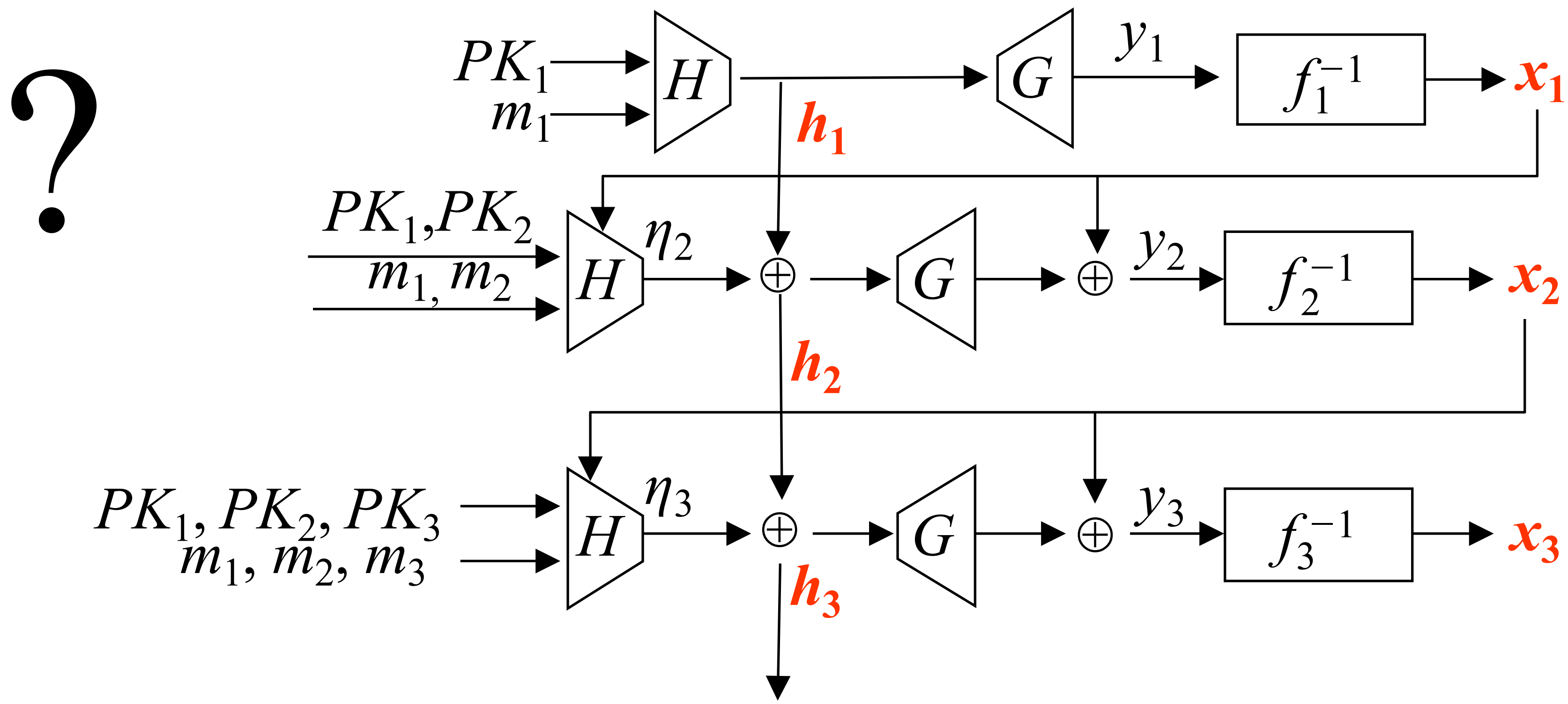
So verifier can compute $y_3 = f_3(x_3)$, get to (x_2, h_2), and repeat



[Neven08] Aggregate Signature Scheme

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

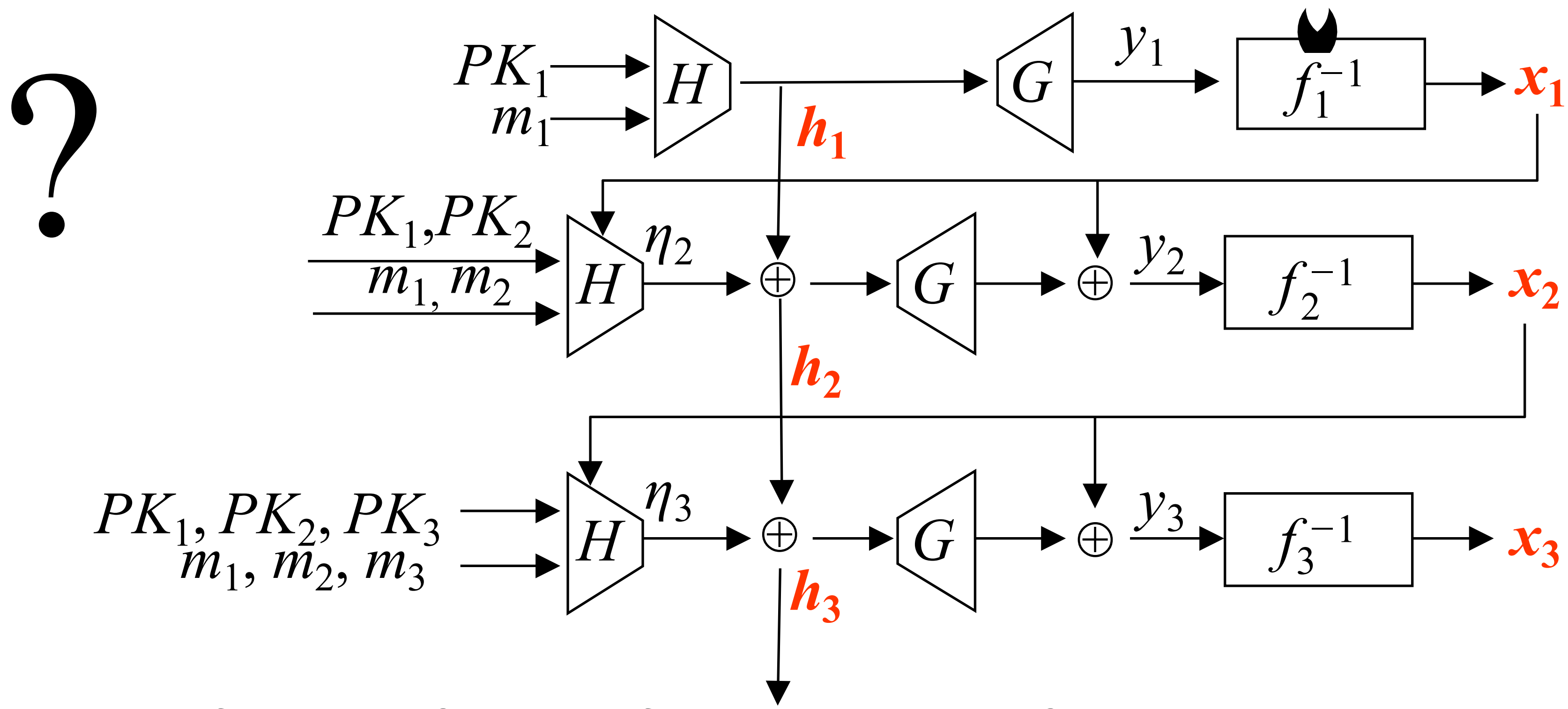
Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$



[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$



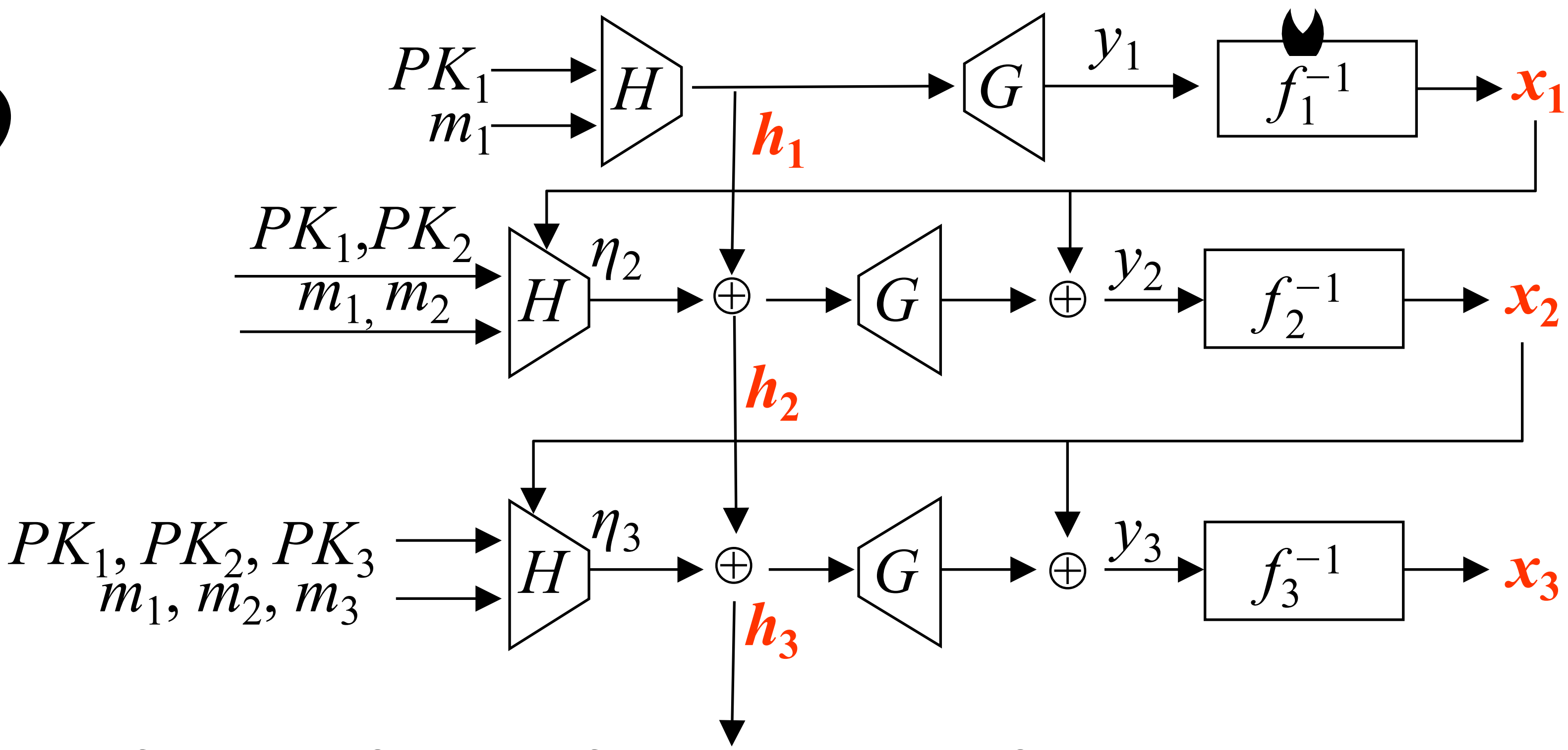
Q: Why no certified TDP? What if f_1 is not a TDP?

[Neven08] Aggregate Signature Scheme

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

Signature has two components: $(\boldsymbol{x}, \boldsymbol{h})$

?



Q: Why no certified TDP? What if f_1 is not a TDP?

A: Adversary can't control y_2, because now x_1 gets hashed before $\oplus$

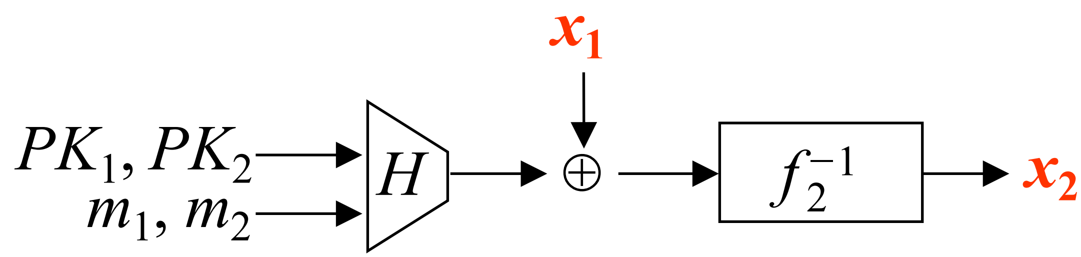
Outline

- Sequential Aggregate Signatures (SAS)
- Security Definition
- Prior Constructions
  - [LMRS]: requires certified TDPs
  - [Neven]: works even adversary gives nonpermutations!
- Our General Construction
  - History-free variants

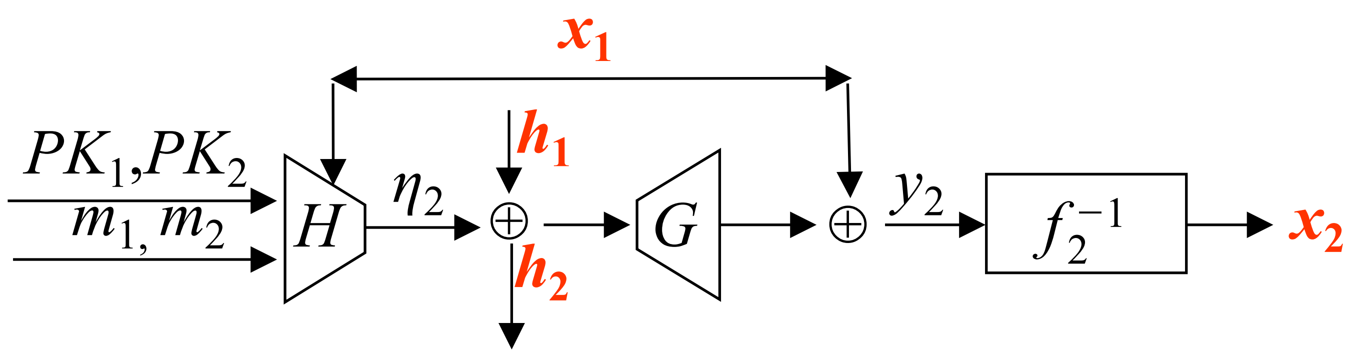
Our Aggregate Signature Scheme

LMRS:

(certified TDPs)

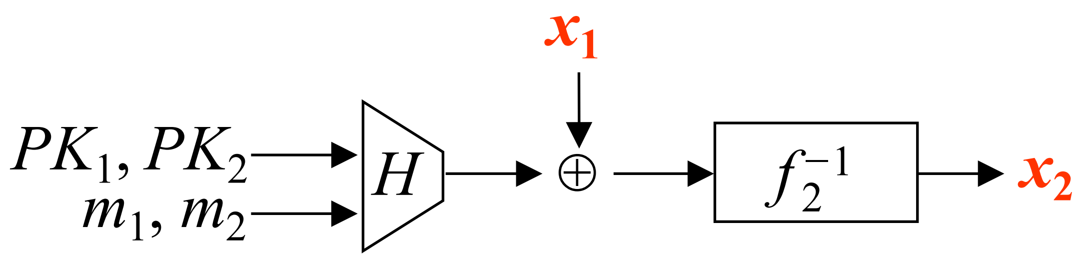


Neven:

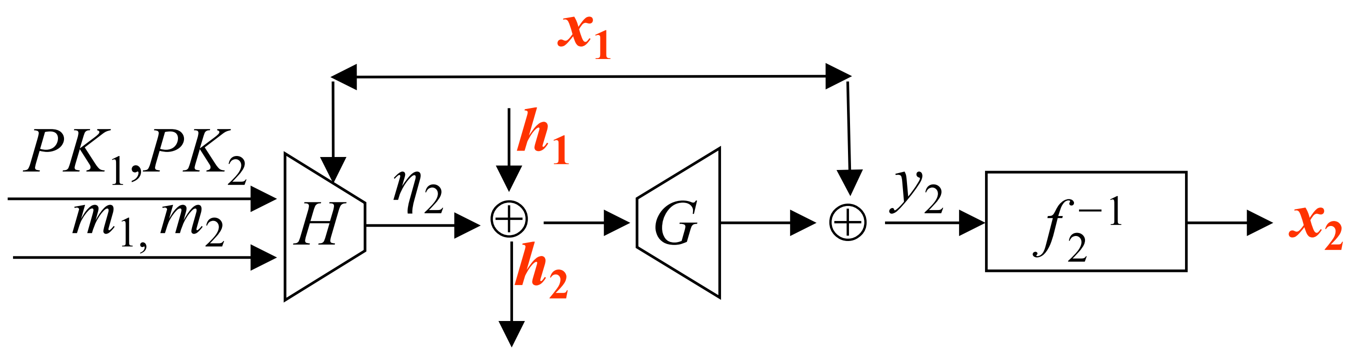


Our Aggregate Signature Scheme

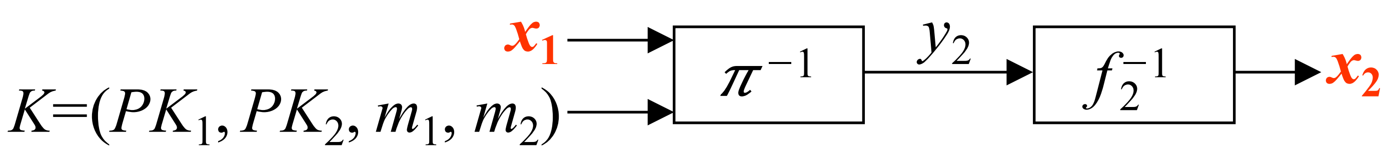
LMRS:
(certified TDPs)



Neven:

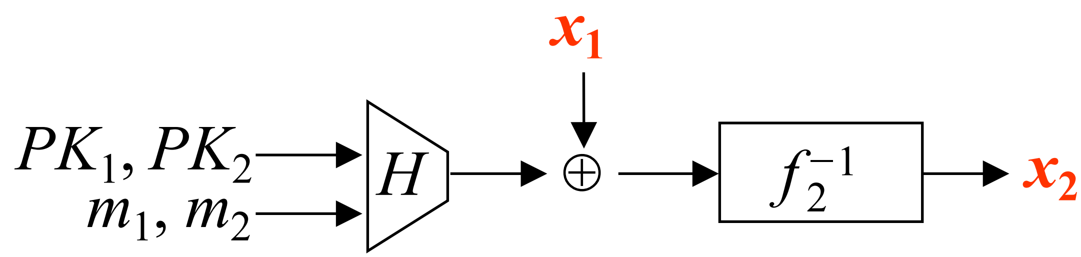


This Work:

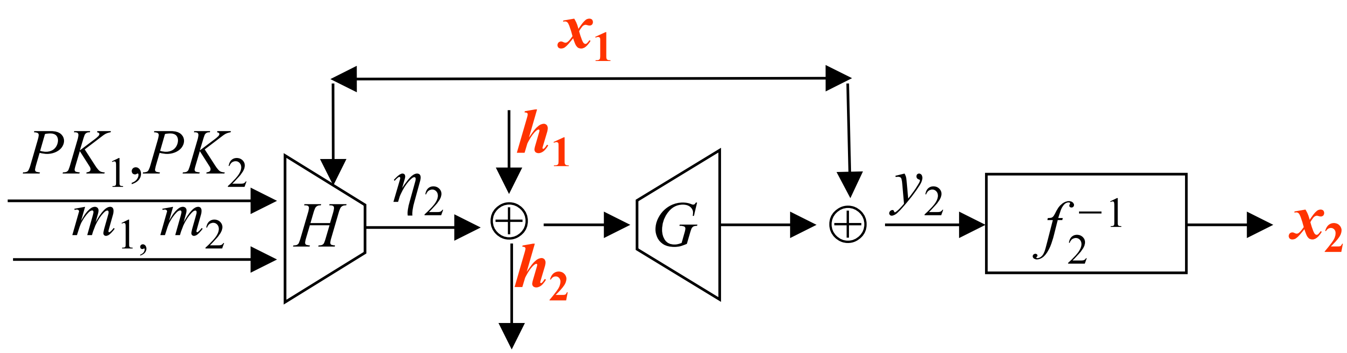


Our Aggregate Signature Scheme

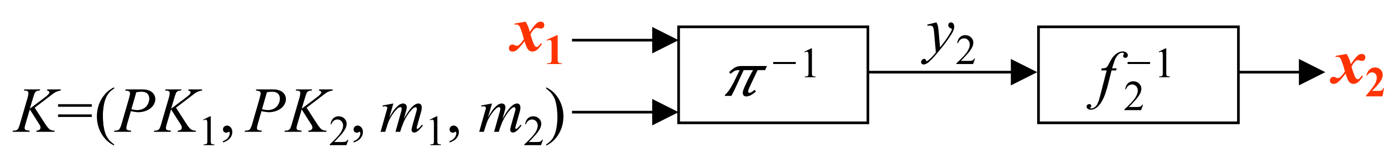
LMRS:
(certified TDPs)



Neven:



This Work:



π is an ideal cipher (keyed public random permutation, like AES)

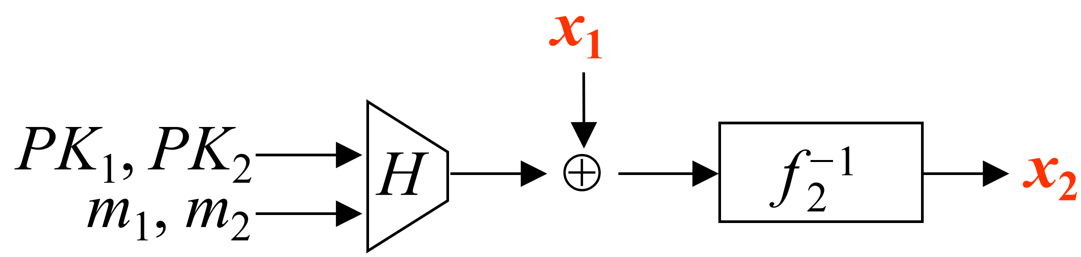
π can't be AES, because need bigger domain (at least for f = RSA)

But: π can be built from random oracle via 8-round Feistel

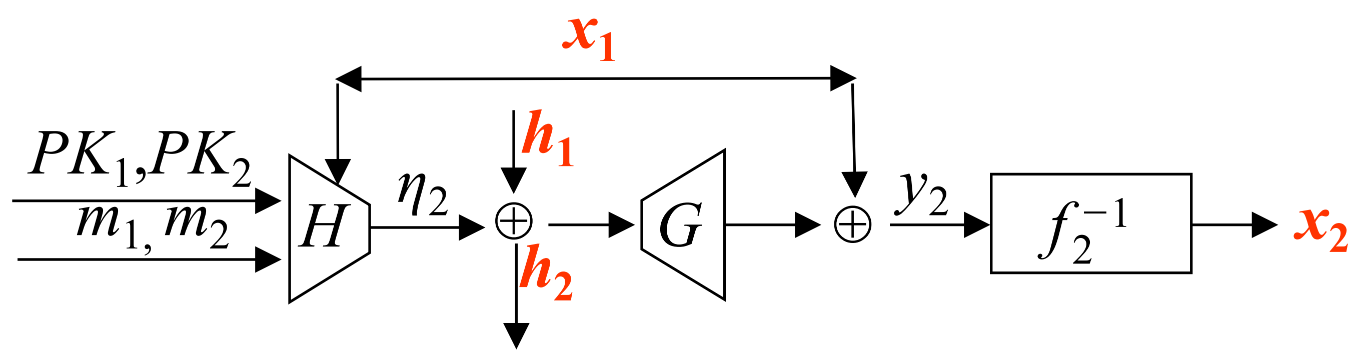
[Coron, Holenstein, Künzler, Patarin, Seurin, Tessaro;
Dachman-Soled, Katz, Thiruvengadam; Dai-Steinberger 16]

Our Aggregate Signature Scheme

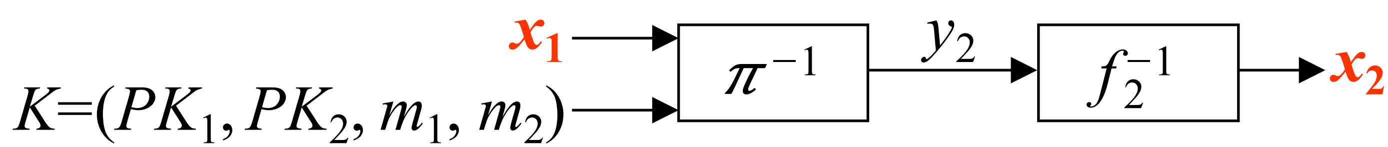
LMRS:
(certified TDPs)



Neven:



This Work:



- Simpler and easier to analyze (proofs in the paper)
- Doesn't require certified TDPs (same as Neven)
- Aggregate signature has only one component
(shorter than Neven if you believe in ideal ciphers)

Outline

- Sequential Aggregate Signatures (SAS)
- Security Definition
- Prior Constructions
  - [LMRS]: requires certified TDPs
  - [Neven]: works even adversary gives nonpermutations!
- Our General Construction
  - History-free variants

Why History-Free?

LMRS, Neven, and our scheme: all ***require*** verify-before-sign

Devastating attack if you use your f^{-1}

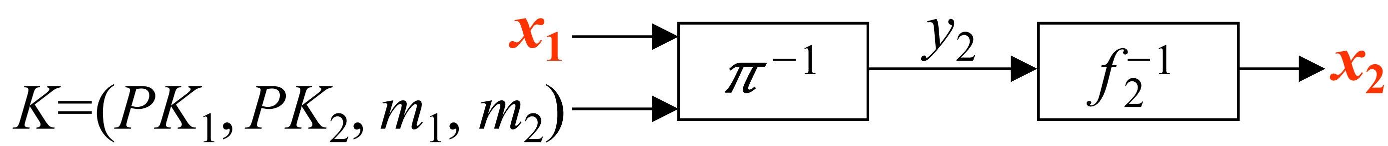
before verifying what you put into it!

Why History-Free?

LMRS, Neven, and our scheme: all ***require*** verify-before-sign

Devastating attack if you use your f^{-1} before verifying what you put into it!

Why History-Free?



LMRS, Neven, and our scheme: all ***require*** verify-before-sign

Devastating attack if you use your f^{-1} before verifying what you put into it!

(Chosen-aggregate attack using a bogus x_1 to get a y_2 collision)

Why History-Free?

LMRS, Neven, and our scheme: all ***require*** verify-before-sign

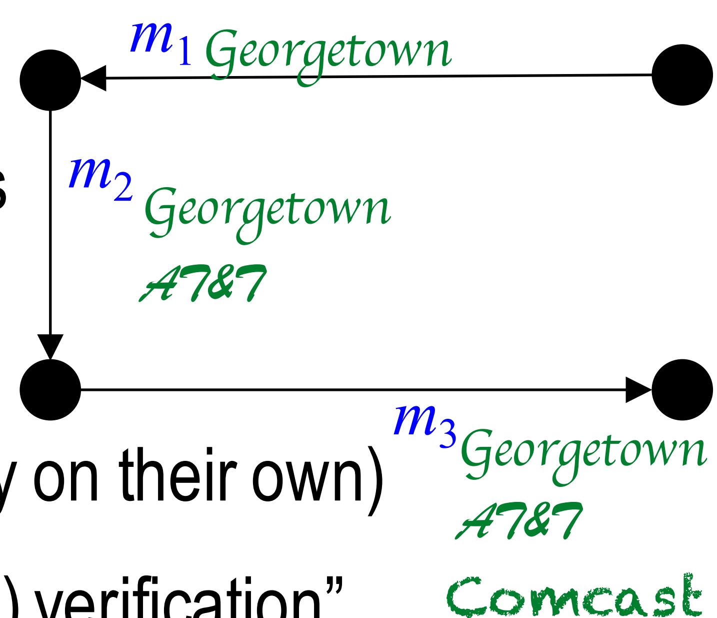
Devastating attack if you use your f^{-1} before verifying what you put into it!

Problem with verify-before-sign:

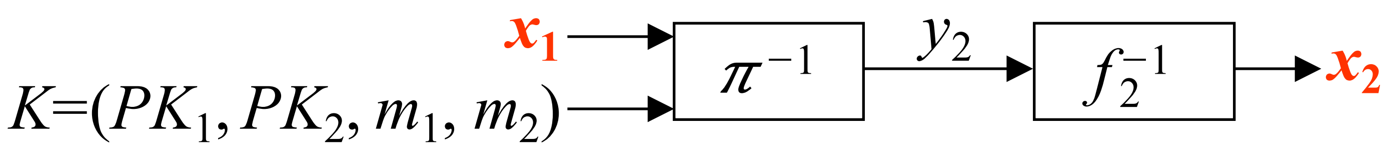
Verification requires retrieving current PKs (out of 85000 ASes on the internet)

If you wait to verify before forwarding, you'll delay others (who can anyway verify on their own)

At times of high load, need "lazy (delayed) verification"



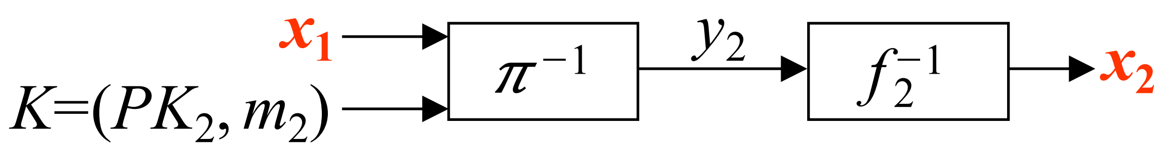
History-Free Variants



History-Free Variants

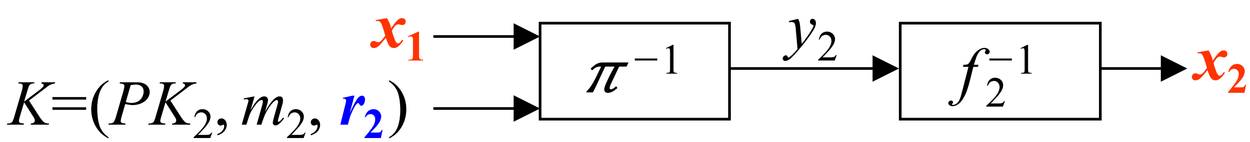
$$\begin{tikzcd}
 & x_1 \arrow[r] & \boxed{\pi^{-1}} \arrow[r, "y_2"] & \boxed{f_2^{-1}} \arrow[r] & x_2 \\
K=(\cancel{PK_1}, PK_2, \cancel{m_1}, m_2) \arrow[urr] & & & &
\end{tikzcd}$$

History-Free Variants



Problem: not secure!

Randomized History-Free Variant



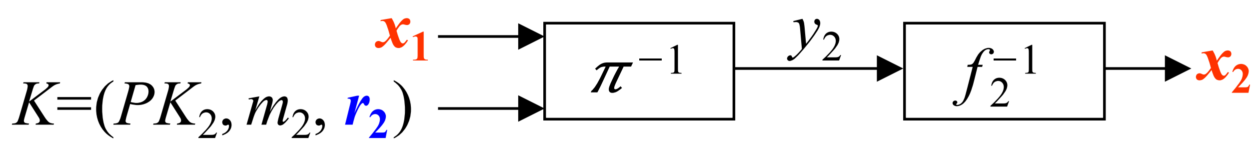
Just add fresh randomness to the key for π [Brogle-Goldberg-Reyzin ‘12]

Drawback: final aggregate is $r_1\, r_2\, \ldots\, r_n\, x_n$ — not constant size

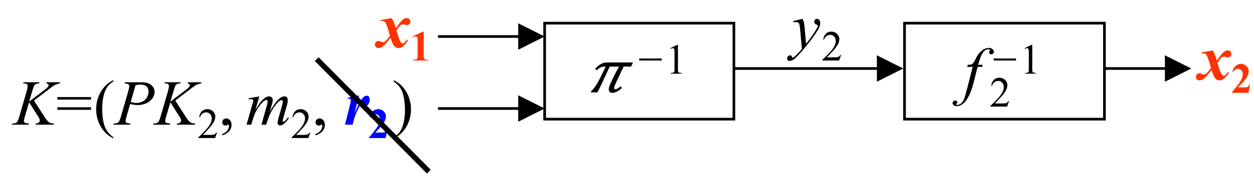
but still better than n individual sigs because each r_i is short

Intuition why it works: Adversary can’t predict y_2, so this is like FDH

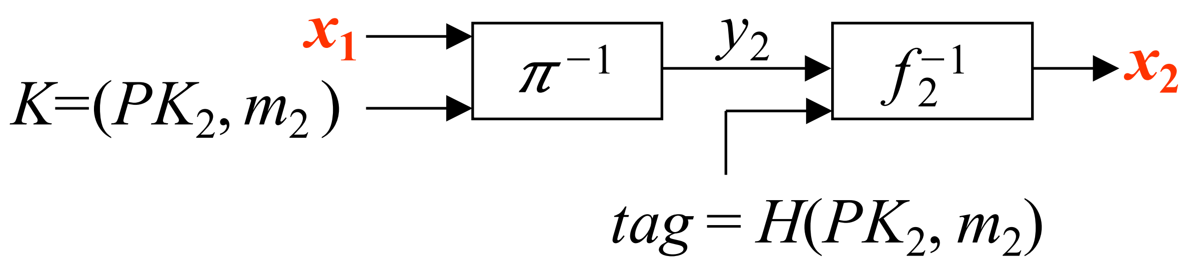
Deterministic History-Free Variant



Deterministic History-Free Variant



Deterministic History-Free Variant



Use "tag-based TDP" (tag is a public input that defines a fresh TDP)

Tag-based TDP can be built on a variant of strong RSA

[Kiltz-Mohassel-O'Neill '10]

Intuition why it works: chosen message attack will hit the wrong tag

Outline

- Sequential Aggregate Signatures (SAS)
- Security Definition
- Prior Constructions
  - [LMRS]: requires certified TDPs
  - [Neven]: works even adversary gives nonpermutations!
- Our General Construction
  - History-free variants (randomness or stronger assumption)

Conclusion

This Work: $K=(PK_1, PK_2, m_1, m_2)$

$$x_1,\ K \longrightarrow \boxed{\pi^{-1}} \xrightarrow{y_2} \boxed{f_2^{-1}} \longrightarrow x_2$$

- Simpler and easier to analyze
- Unfortunately, current techniques for building π have a large security loss, so parameters not practical (while [Neven 08] is practical assuming RO)
- Let's build ideal ciphers with good parameters!
- <u>Question</u>: if you build π using RO, you need 8 rounds of Feistel. Neven works with 2 rounds of Feistel, but ends up with longer sigs. Do you really need an ideal cipher for the shorter sigs?