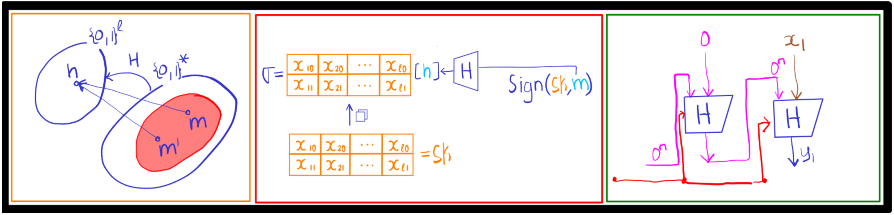


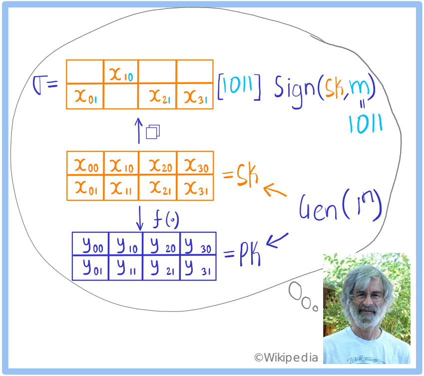
CS409m: Introduction to Cryptography

Lecture 16 (10/Oct/25)

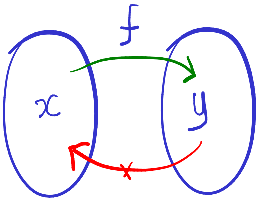
Instructor: Chethan Kamath

- Task: integrity and authentication in the *public-key* setting
- Threat model: EU-CMA

Lamport's Digital Signature



One-Way Function

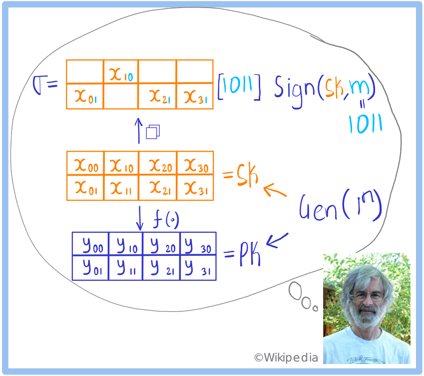


$$\Pr_{x \leftarrow \{0,1\}^n}[\mathsf{Inv}(f_n(x)) \in f_n^{-1}(f_n(x))]$$

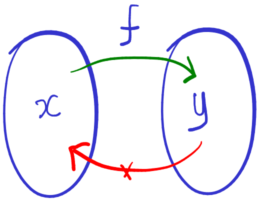
Recall from Last Lecture...

- Task: integrity and authentication in the *public-key* setting
- Threat model: EU-CMA

Lamport's Digital Signature



One-Way Function



$$\Pr_{x \leftarrow \{0,1\}^n}[\mathsf{Inv}(f_n(x)) \in f_n^{-1}(f_n(x))]$$

Proof technique: plug and pray

Lamport's Signature is One-Time Secure...

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Lamport's Signature is One-Time Secure...

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Proof sketch: proof by reduction. Idea: "plug and pray".

Lamport's Signature is One-Time Secure...

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

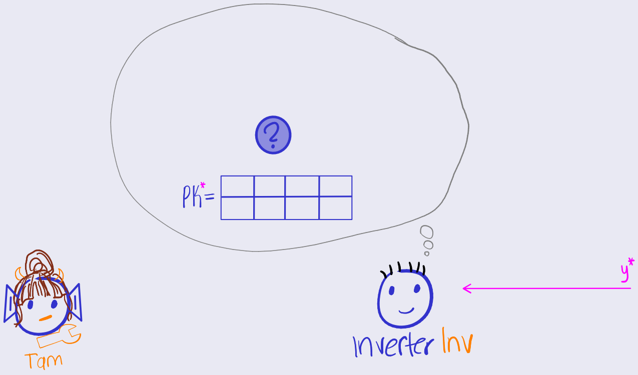
Proof sketch: proof by reduction. Idea: "plug and pray".

Lamport's Signature is One-Time Secure...

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Proof sketch: proof by reduction. Idea: "plug and pray".

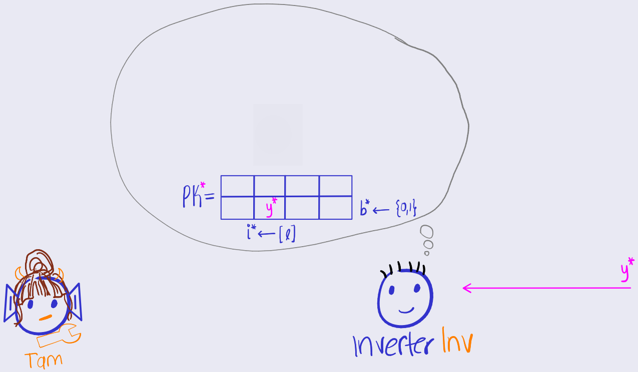


Lamport's Signature is One-Time Secure...

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Proof sketch: proof by reduction. Idea: "plug and pray".

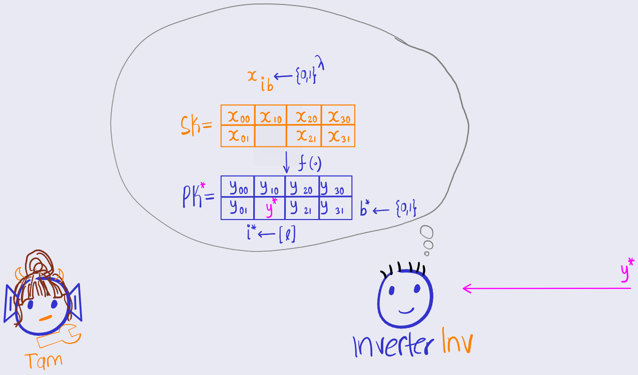


Lamport's Signature is One-Time Secure...

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

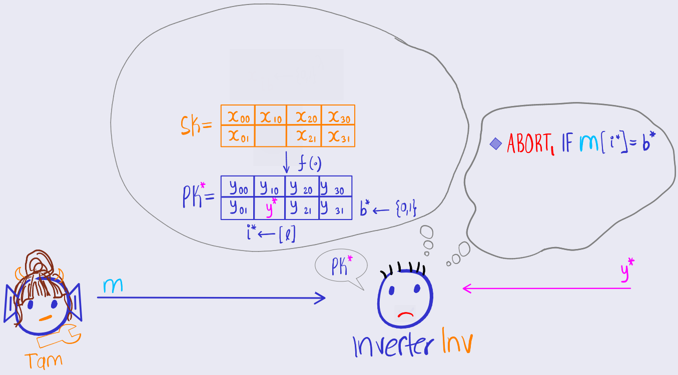
Proof sketch: proof by reduction. Idea: "plug and pray".



Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Proof sketch: proof by reduction. Idea: "plug and pray".

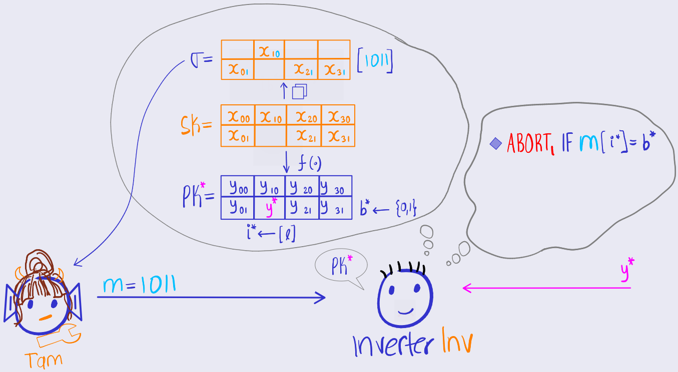


Lamport's Signature is One-Time Secure...

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

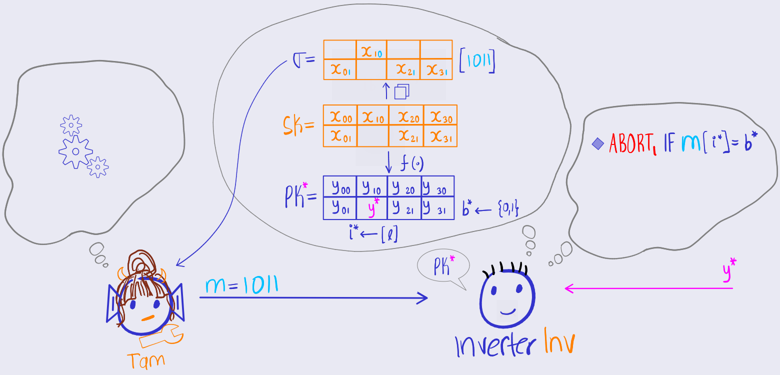
Proof sketch: proof by reduction. Idea: "plug and pray".



Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Proof sketch: proof by reduction. Idea: "plug and pray".

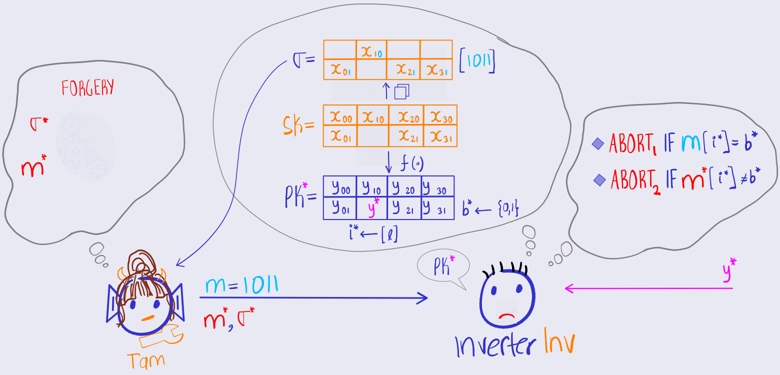


Lamport's Signature is One-Time Secure...

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Proof sketch: proof by reduction. Idea: "plug and pray".

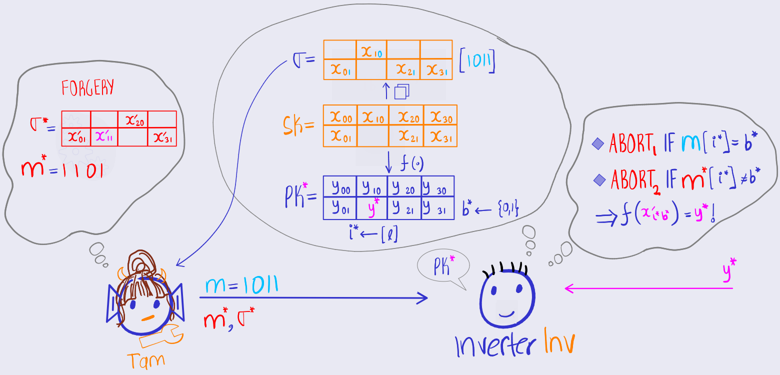


Lamport's Signature is One-Time Secure . . .

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

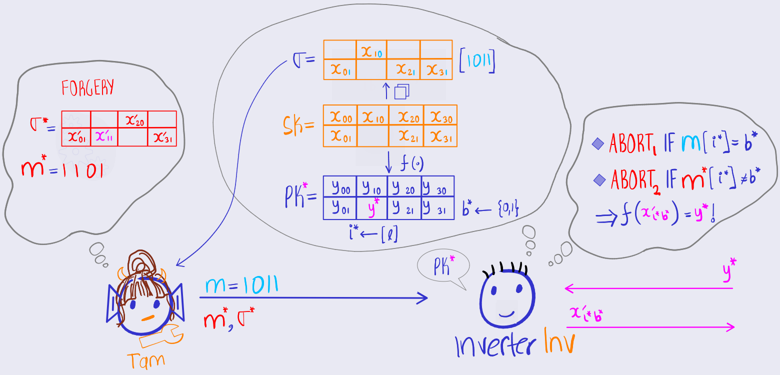
Proof sketch: proof by reduction. Idea: "plug and pray".



Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

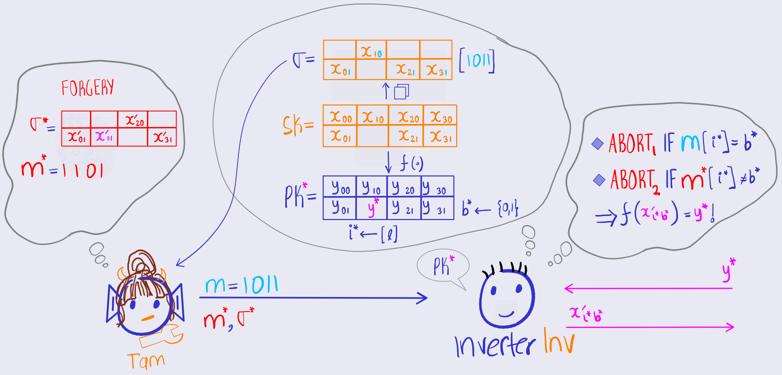
Proof sketch: proof by reduction. Idea: "plug and pray".



Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Proof sketch: proof by reduction. Idea: "plug and pray".



(Analysis on whiteboard)

Lamport's Signature is One-Time Secure...

Theorem 2 ([Mer90a, Gol87])

If one-time DS and PRFs exists then many-time DS exists

Lamport's Signature is One-Time Secure...

Theorem 2 ([Mer90a, Gol87])

If one-time DS and PRFs exists then many-time DS exists

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS

Lamport's Signature is One-Time Secure...

Theorem 2 ([Mer90a, Gol87])

If one-time DS and PRFs exists then many-time DS exists

Theorem 1

If f is a OWF then Lamport's scheme is a one-time DS for fixed-length messages!

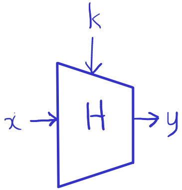
Exercise 1 (Domain Extension)

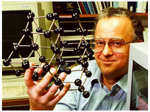
Given a compressing function $H : \{0,1\}^{2\ell} \to \{0,1\}^{\ell}$, construct a one-time DS for *arbitrary-length* messages. What are the properties you need from H to ensure that the one-time DS is secure?

Plan for Today's Lecture...

- Task: sign arbitrarily long messages
- Threat model: EU-CMA

NEW Hash Function

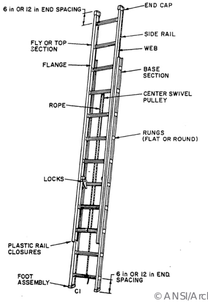




© Prof. Bill Buchanan/Medium

© cs.au.dk/~ivan

Domain Extension NEW

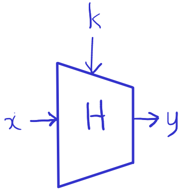


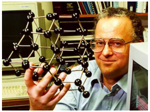
© ANSI/Archive.org

Plan for Today's Lecture...

- Task: sign arbitrarily long messages
- Threat model: EU-CMA

NEW Hash Function

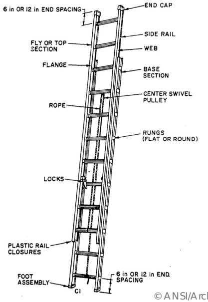




© Prof. Bill Buchanan/Medium

© cs.au.dk/~ivan

Domain Extension NEW



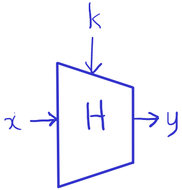
© ANSI/Archive.org

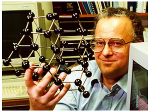
Old tricks: chain, tree-based constructions

Plan for Today's Lecture...

- Task: sign arbitrarily long messages
- Threat model: EU-CMA

NEW **Hash Function**

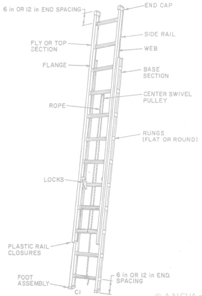




© Prof. Bill Buchanan/Medium

© cs.au.dk/~ivan

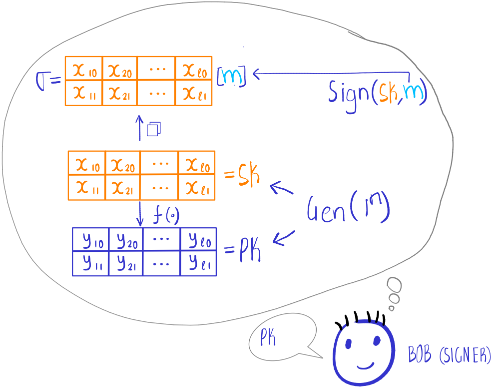
Domain Extension NEW



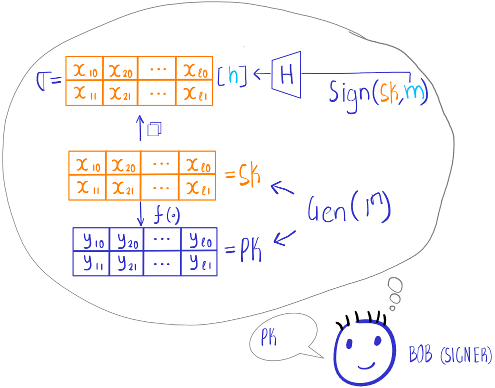
© ANSI/Archive.org

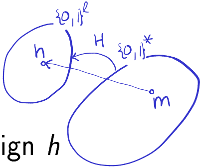
Old tricks: chain, tree-based constructions

Extending Lamport's One-Time DS for Long Messages...

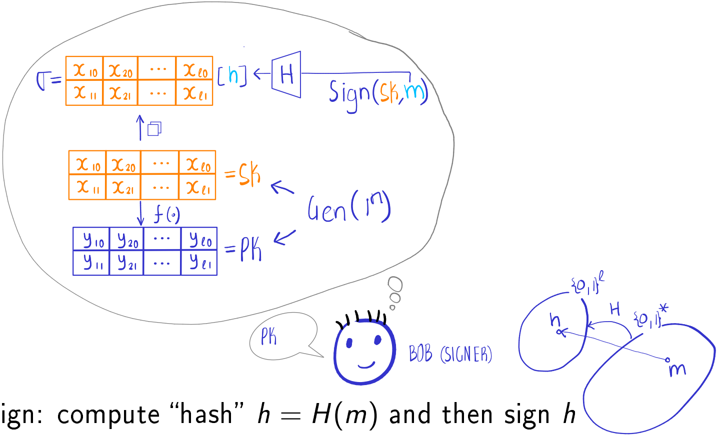


Extending Lamport's One-Time DS for Long Messages...

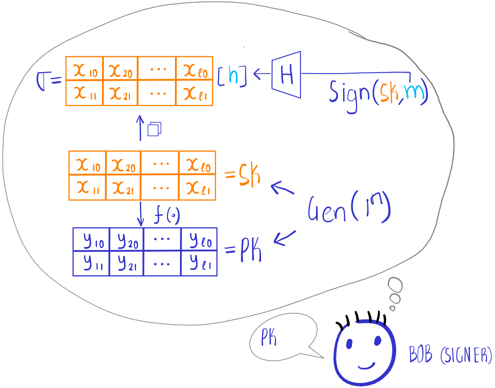


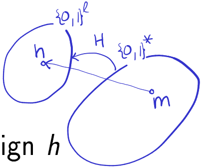


- Hash-then-sign: compute "hash" $h = H(m)$ and then sign h



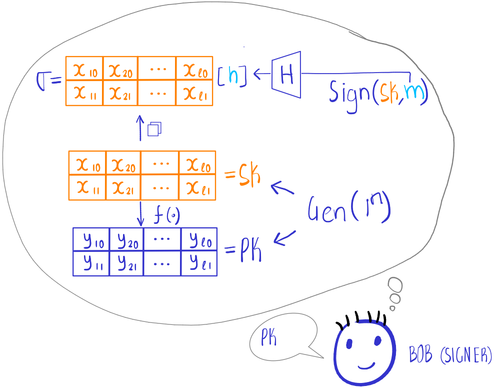
- Hash-then-sign: compute "hash" $h = H(m)$ and then sign h
- What are the requirements from H? When can Tam forge?

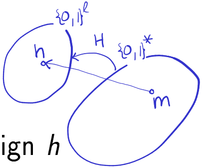




- Hash-then-sign: compute "hash" $h = H(m)$ and then sign h
- What are the requirements from H? When can Tam forge?
  - Must be one-way. Is one-wayness sufficient?

Extending Lamport's One-Time DS for Long Messages...





- Hash-then-sign: compute "hash" $h = H(m)$ and then sign h

What are the requirements from H? When can Tam forge?

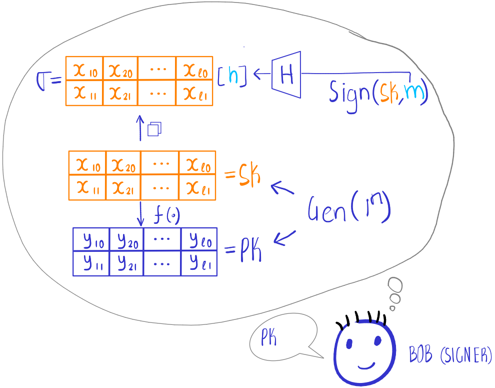
- Must be one-way. Is one-wayness sufficient?

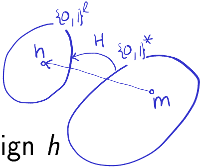
No, it must be hard to find inputs that "collide"

  - Collisions are guaranteed to exist (pigeonhole principle)

- Is "collision-resistance" sufficient?

Extending Lamport's One-Time DS for Long Messages...





- Hash-then-sign: compute "hash" $h = H(m)$ and then sign h

? What are the requirements from H? When can Tam forge?

  - Must be one-way. Is one-wayness sufficient?

  ⚠ No, it must be hard to find inputs that "collide"

    - Collisions are guaranteed to exist (pigeonhole principle)

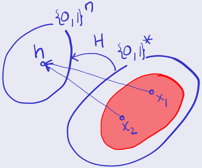
  - Is "collision-resistance" sufficient? Yes, as we'll see.

Definition 1 (CRHF, with key generation algorithm Gen)

A keyed function (family) $\{H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n\}$ is a CRHF if for every PPT collision-finder F, the following is negligible.

$$\Pr_{\substack{k \leftarrow \mathrm{Gen}(1^n) \\ (x_1 \overset{\neq}{,} x_2) \leftarrow \mathsf{F}(k)}} [H(k, x_1) = H(k, x_2)]$$

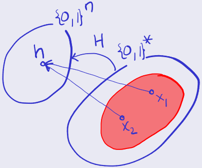
Need not be same length



Collision-Resistant Hash Function (CRHF)

Definition 1 (CRHF, with key generation algorithm Gen)

A keyed function (family) $\{H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n\}$ is a CRHF if for every PPT collision-finder F, the following is negligible.



$$\Pr_{\substack{k \leftarrow \mathsf{Gen}(1^n) \\ (x_1 \overset{\neq}{,} x_2) \leftarrow \mathsf{F}(k)}} [H(k, x_1) = H(k, x_2)]$$

Need not be same length

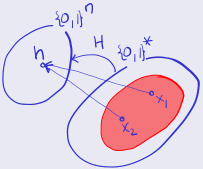
? If H_1 and H_2 are CRHFs then is H?

Definition 1 (CRHF, with key generation algorithm Gen)

A keyed function (family) $\{H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n\}$ is a CRHF if for every PPT collision-finder F, the following is negligible.

$$\Pr_{\substack{k \leftarrow \mathsf{Gen}(1^n) \\ (x_1, x_2) \leftarrow \mathsf{F}(k)}} [H(k, x_1) = H(k, x_2)]$$

Need not be same length



? If H_1 and H_2 are CRHFs then is H?

1. Hash-then-append: $H(k, x) := H_1(k, x) \| 0$

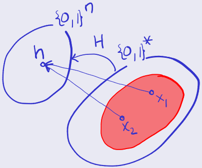
Collision-Resistant Hash Function (CRHF)

Definition 1 (CRHF, with key generation algorithm Gen)

A keyed function (family) $\{H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n\}$ is a CRHF if for every PPT collision-finder F, the following is negligible.

$$\Pr_{\substack{k \leftarrow \mathsf{Gen}(1^n) \\ (x_1, x_2) \leftarrow \mathsf{F}(k)}} [H(k, x_1) = H(k, x_2)]$$

Need not be same length



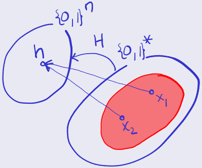
If H_1 and H_2 are CRHFs then is H?

1. Hash-then-append: $H(k, x) := H_1(k, x) \| 0$
2. Hash-then-truncate: $H(k, x) := y_1 \| \ldots \| y_{n-1}$, where $y_1 \| \ldots \| y_n := H_1(k, x)$

Definition 1 (CRHF, with key generation algorithm Gen)

A keyed function (family) $\{H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n\}$ is a CRHF if for every PPT collision-finder F, the following is negligible.

$$\Pr_{\substack{k \leftarrow \mathsf{Gen}(1^n) \\ (x_1^*, x_2) \leftarrow \mathsf{F}(k)}} [H(k, x_1) = H(k, x_2)]$$



Need not be same length

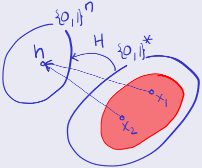
If H_1 and H_2 are CRHFs then is H?

1. Hash-then-append: $H(k, x) := H_1(k, x) \| 0$
2. Hash-then-truncate: $H(k, x) := y_1 \| \ldots \| y_{n-1}$, where $y_1 \| \ldots \| y_n := H_1(k, x)$
3. Hash-then-XOR: $H(k_1 \| k_2, x) := H_1(k_1, x) \oplus H_2(k_2, x)$

Collision-Resistant Hash Function (CRHF)

Definition 1 (CRHF, with key generation algorithm Gen)

A keyed function (family) $\{H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n\}$ is a CRHF if for every PPT collision-finder F, the following is negligible.



$$\Pr_{\substack{k \leftarrow \mathrm{Gen}(1^n) \\ (x_1^*, x_2) \leftarrow F(k)}} [H(k, x_1) = H(k, x_2)]$$

Need not be same length

? If H_1 and H_2 are CRHFs then is H?

1. Hash-then-append: $H(k, x) := H_1(k, x) \| 0$
2. Hash-then-truncate: $H(k, x) := y_1 \| \ldots \| y_{n-1}$, where $y_1 \| \ldots \| y_n := H_1(k, x)$
3. ? Hash-then-XOR: $H(k_1 \| k_2, x) := H_1(k_1, x) \oplus H_2(k_2, x)$

Exercise 2

Prove formally cases where H is CRHF; describe counter-e.g. otherwise

Let's (Slowly) Find Collisions in H!

What about a *deterministic* $O(2^n)$-time collision-finder?

Let's (Slowly) Find Collisions in H!

What about a *deterministic* $O(2^n)$-time collision-finder?

- Exploit pigeonhole principle

Let's (Slowly) Find Collisions in H!

What about a *deterministic* $O(2^n)$-time collision-finder?

- Exploit pigeonhole principle
- Compute (e.g.) hash of inputs $0\|0^n, \ldots, 0\|1^n, 1\|0^n$
  - There *must exist* colliding pair of inputs

Can we do better?

Let's (Slowly) Find Collisions in H!

What about a *deterministic* $O(2^n)$-time collision-finder?

Exploit pigeonhole principle

- Compute (e.g.) hash of inputs $0\|0^n, \ldots, 0\|1^n, 1\|0^n$
  - There *must exist* colliding pair of inputs

Can we do better? Yes, recall *birthday paradox*:

Theorem 3 (Lecture 2)

Let $q \leq \sqrt{2 \cdot 2^n}$ elements $(y_1, \ldots, y_q)$ be chosen uniformly and independently at random from $\{0,1\}^n$, then

$$\Pr[\exists i \neq j \ s.t. \ y_i = t_j] \geq q(q-1)/42^n$$

How does a *randomised* $O(2^{n/2})$-time collision-finder work?

Let's (Slowly) Find Collisions in H!

What about a *deterministic* $O(2^n)$-time collision-finder?

Exploit pigeonhole principle

- Compute (e.g.) hash of inputs $0\|0^n, \ldots, 0\|1^n, 1\|0^n$
  - There *must exist* colliding pair of inputs

Can we do better? Yes, recall *birthday paradox*:

Theorem 3 (Lecture 2)

Let $q \leq \sqrt{2 \cdot 2^n}$ elements $(y_1, \ldots, y_q)$ be chosen uniformly and independently at random from $\{0,1\}^n$, then

$$\Pr[\exists i \neq j \ s.t. \ y_i = t_j] \geq q(q-1)/42^n$$

How does a *randomised* $O(2^{n/2})$-time collision-finder work?

Let's (Slowly) Find Collisions in H!

What about a *deterministic* $O(2^n)$-time collision-finder?

Exploit pigeonhole principle

- Compute (e.g.) hash of inputs $0\|0^n, \ldots, 0\|1^n, 1\|0^n$
  - There *must exist* colliding pair of inputs

Can we do better? Yes, recall *birthday paradox*:

Theorem 3 (Lecture 2)

Let $q \leq \sqrt{2 \cdot 2^n}$ elements $(y_1, \ldots, y_q)$ be chosen uniformly and independently at random from $\{0,1\}^n$, then

$$\Pr[\exists i \neq j \; s.t. \; y_i = t_j] \geq q(q-1)/42^n$$

How does a *randomised* $O(2^{n/2})$-time collision-finder work?

- Compute hash of $q := O(2^{n/2})$ *random* inputs $x_1, \ldots, x_q \leftarrow \{0,1\}^n$
- By Theorem 3, with *noticeable probability* there exists a colliding pair

Let's (Slowly) Find Collisions in H!

What about a *deterministic* $O(2^n)$-time collision-finder?

Exploit pigeonhole principle

- Compute (e.g.) hash of inputs $0\|0^n, \ldots, 0\|1^n, 1\|0^n$
  - There *must exist* colliding pair of inputs

Can we do better? Yes, recall *birthday paradox*:

Theorem 3 (Lecture 2)

Let $q \leq \sqrt{2 \cdot 2^n}$ elements $(y_1, \ldots, y_q)$ be chosen uniformly and independently at random from $\{0,1\}^n$, then

$$\Pr[\exists i \neq j \ s.t. \ y_i = t_j] \geq q(q-1)/42^n$$

How does a *randomised* $O(2^{n/2})$-time collision-finder work?

- Compute hash of $q := O(2^{n/2})$ *random* inputs $x_1, \ldots, x_q \leftarrow \{0,1\}^n$
- By Theorem 3, with *noticeable probability* there exists a colliding pair

Consequence: key-size/output length must be $2\times$ security level

Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for $H \Leftarrow \exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case CAL: $H(k,m) = H(k,m^*)$

Case $\overline{\text{CAL}}$: $H(k,m) \neq H(k,m^*) = h^*$

Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case CAL: $H(k,m) = H(k,m^*)$

$\Downarrow$

(m, m^*) collision for H

Case $\overline{\text{CAL}}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f

Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case (α): $H(k,m) = H(k,m^*)$

$\Downarrow$

(m, m^*) collision for H

Case $\overline{(\alpha)}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f

Collision finder F

Tam

Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

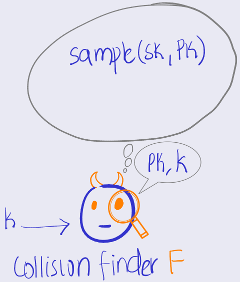
Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case $\textcircled{a}$: $H(k,m) = H(k,m^*)$

$\Downarrow$

(m, m^*) collision for H



Case $\textcircled{b}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

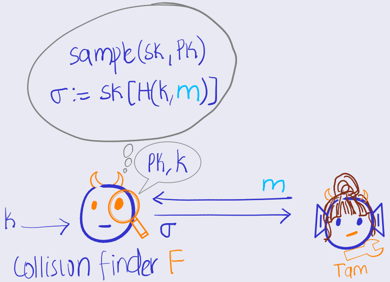
Proof sketch: $\exists$Inv for f *or* $\exists$F for $H \Leftarrow \exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case COLL: $H(k,m) = H(k,m^*)$

$\Downarrow$

(m, m^*) collision for H



Case $\overline{\text{COLL}}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f

Extending Lamport's One-Time DS for Long Messages...

Theorem 4

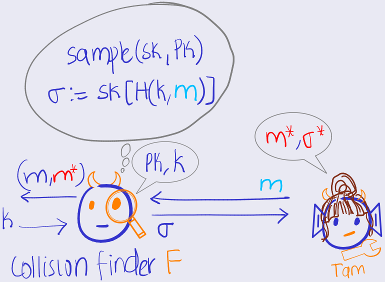
If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case COL: $H(k,m) = H(k,m^*)$
$\Downarrow$
(m, m^*) collision for H

Case $\overline{\text{COL}}$: $H(k,m) \neq H(k,m^*) = h^*$
$\Downarrow$
(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f



Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case (a): $H(k,m) = H(k,m^*)$

$\Downarrow$

(m, m^*) collision for H

Case $\overline{(a)}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f

Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case (a): $H(k,m) = H(k,m^*)$

$\Downarrow$

(m,m^*) collision for H

Case (b): $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f


Inverter Inv


Tam

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

◆ Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case COL: $H(k,m) = H(k,m^*)$

$\Downarrow$

(m, m^*) collision for H

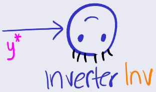
sample(sk, PK)

$\sigma := sk[H(k,m)]$

Case $\overline{\text{COL}}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f



Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

◆ Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case COL: $H(k,m) = H(k,m^*)$

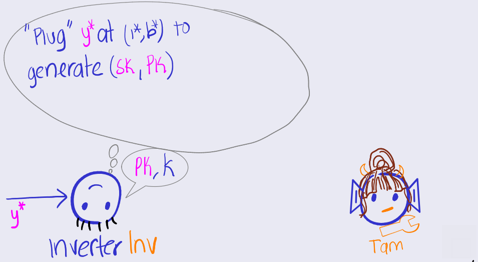
$\Downarrow$

(m, m^*) collision for H

Case $\overline{\text{COL}}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f



Extending Lamport's One-Time DS for Long Messages...

Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

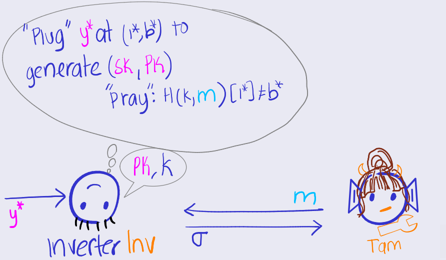
Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case ~~coll~~ : $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f



Theorem 4

If f is a OWF and H is CRHF then the "hash-then-sign" scheme is a one-time DS for arbitrarily-long messages.

Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

◆ Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

Case COL: $H(k,m) = H(k,m^*)$

$\Downarrow$

(m, m^*) collision for H

sample(sk, pk)

$\sigma := sk[H(k,m)]$

Case $\overline{\text{COL}}$: $H(k,m) \neq H(k,m^*) = h^*$

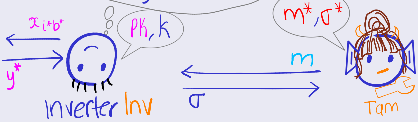
$\Downarrow$

(h^*, σ^*) forgery for Lamport's $\Rightarrow$ invert f

"Plug" y^* at (i^*, b^*) to generate (sk, pk)

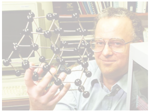
"Pray": $H(k,m)[i^*] \neq b^*$

"Pray": $H(k,m^*)[i^*] = b^*$



- Task: sign arbitrarily long messages
- Threat model: EU-CMA

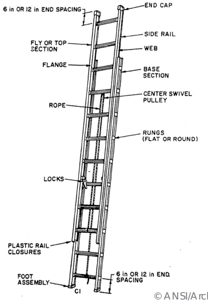
NEW Hash Function



© Prof. Bill Buchanan/Medium

© cs.au.dk/~ivan

Domain Extension NEW

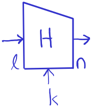


© ANSI/Archive.org

Old tricks: chain, tree-based constructions

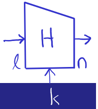
Compression Functions and Domain-Extension

- *Compression function*: hash function for fixed input length $\ell(n) > n$
  - Easier to construct in practice: e.g., MD5, SHA2 (unkeyed) compression function of certain block-size



Compression Functions and Domain-Extension

- *Compression function*: hash function for fixed input length $\ell(n) > n$
  - Easier to construct in practice: e.g., MD5, SHA2 (unkeyed) compression function of certain block-size



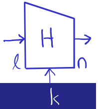
Definition 2 ($\ell(n)$-compression function)

A keyed function (family) $\{H : \mathcal{K} \times \{0,1\}^{\ell(n)} \to \{0,1\}^n\}$ is an $\ell(n)$-compression function if for every PPT collision-finder F, the following is negligible.

$$\Pr_{\substack{k \leftarrow \mathsf{Gen}(1^n) \\ (x_1 \overset{\neq}{,} x_2) \leftarrow \mathsf{F}(k)}} [H(k, x_1) = H(k, x_2)]$$

same length

- *Compression function*: hash function for fixed input length $\ell(n) > n$
  - Easier to construct in practice: e.g., MD5, SHA2 (unkeyed) compression function of certain block-size



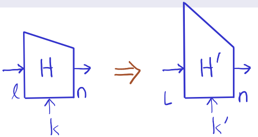
Definition 2 ($\ell(n)$-compression function)

A keyed function (family) $\{H : \mathcal{K} \times \{0,1\}^{\ell(n)} \to \{0,1\}^n\}$ is an $\ell(n)$-compression function if for every PPT collision-finder F, the following is negligible.

$$\Pr_{\substack{k \leftarrow \mathsf{Gen}(1^n) \\ (x_1, x_2) \leftarrow \mathsf{F}(k)}} [H(k, x_1) = H(k, x_2)]$$

same length

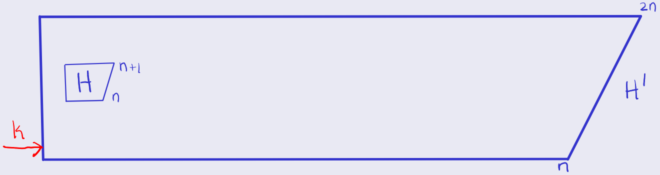
- Domain extension: $\ell(n)$-compression function $\Rightarrow L(n)$-compression function for $L(n) > \ell(n)$



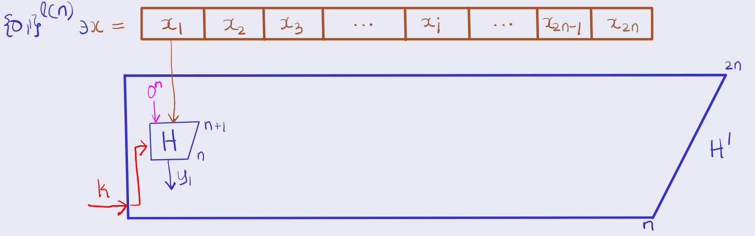
Approach I: Merkle-Damgård Construction (Chaining)...

Construction 1 ($(n+1)$-compression fn. $H \Rightarrow 2n$-compression fn. H')

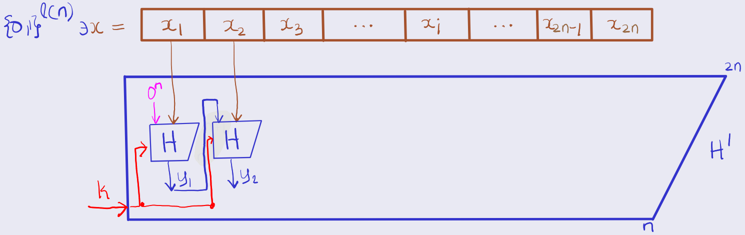
$\{0,1\}^{\ell(n)} \ni x = \boxed{x_1 \mid x_2 \mid x_3 \mid \cdots \mid x_i \mid \cdots \mid x_{2n-1} \mid x_{2n}}$



Construction 1 ($(n+1)$-compression fn. $H \Rightarrow 2n$-compression fn. H')

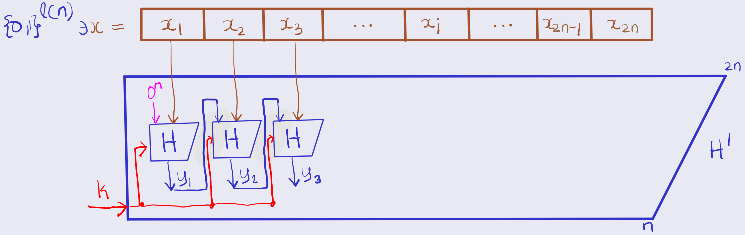


Construction 1 ($(n+1)$-compression fn. $H \Rightarrow 2n$-compression fn. H')

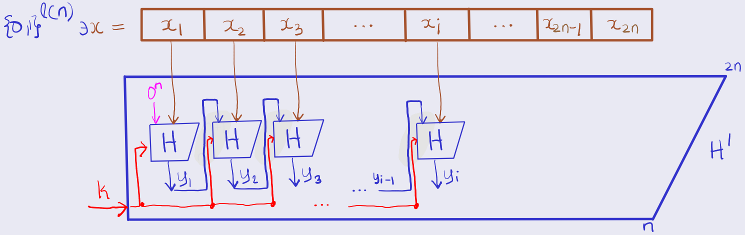


Approach I: Merkle-Damgård Construction (Chaining)...

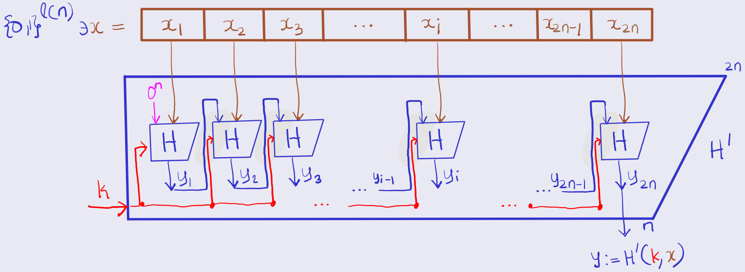
Construction 1 ($(n+1)$-compression fn. $H \Rightarrow 2n$-compression fn. H')



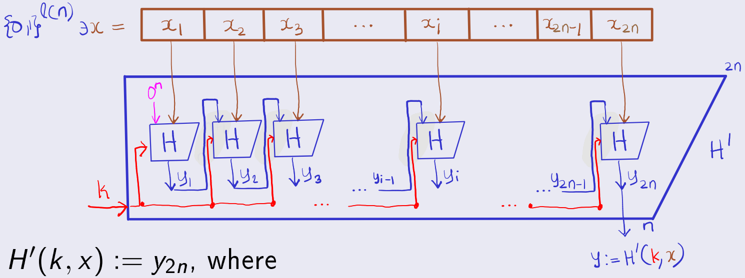
Construction 1 (($n+1$)-compression fn. $H \Rightarrow 2n$-compression fn. H')



Construction 1 (($n+1$)-compression fn. $H \Rightarrow 2n$-compression fn. H')



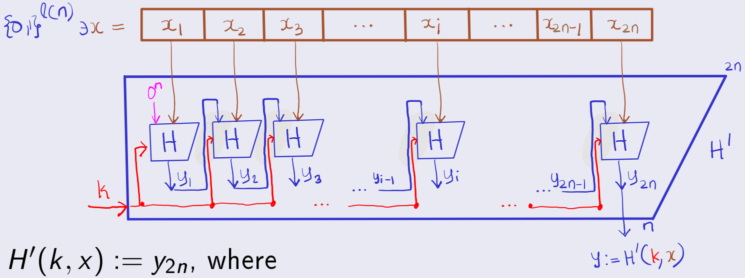
Construction 1 ($(n+1)$-compression fn. $H \Rightarrow 2n$-compression fn. H')



- $H'(k, x) := y_{2n}$, where
  - $y_1 := H(k, 0^n \| x_1)$ and $y_i := H(k, y_{i-1} \| x_i)$ for $i \in [2, 2n]$

Approach I: Merkle-Damgård Construction (Chaining)...

Construction 1 (($(n+1)$-compression fn. $H \Rightarrow 2n$-compression fn. H')



- $H'(k,x) := y_{2n}$, where
  - $y_1 := H(k, 0^n \| x_1)$ and $y_i := H(k, y_{i-1} \| x_i)$ for $i \in [2, 2n]$

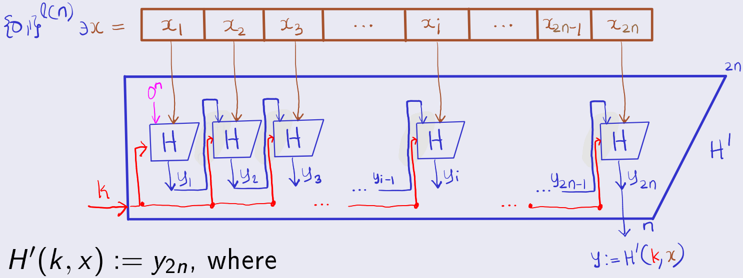
Exercise 3

Show that if H is a compression function then so is H'

- Is H' parallelisable?
- Can parts of input can be locally verified?

Approach I: Merkle-Damgård Construction (Chaining)...

Construction 1 ($(n+1)$-compression fn. $H \Rightarrow 2n$-compression fn. H')



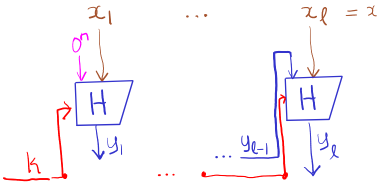
- $H'(k,x) := y_{2n}$, where
  - $y_1 := H(k, 0^n \| x_1)$ and $y_i := H(k, y_{i-1} \| x_i)$ for $i \in [2, 2n]$

Exercise 3

Show that if H is a compression function then so is H'

- Is H' parallelisable?
- Can parts of input can be *locally* verified?

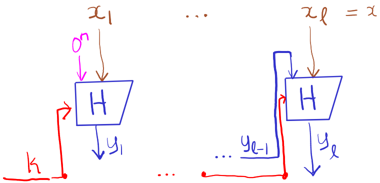
What happens if we use Construction 1 for $\{0,1\}^*$?



Approach I: Merkle-Damgård Construction (Chaining)...

What happens if we use Construction 1 for $\{0,1\}^*$?

Is it possible to find collisions of *different* length?

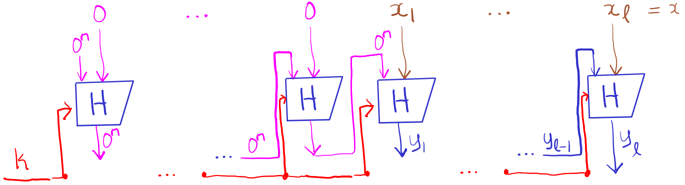


Approach I: Merkle-Damgård Construction (Chaining)...

What happens if we use Construction 1 for $\{0,1\}^*$?

Is it possible to find collisions of *different* length?

- Yes, consider H for which $H(k, 0^{n+1}) = 0^n$ (for all k)
  - For H' instantiated with above H: $H'(k, 0^n \| x) = H'(k, x)$

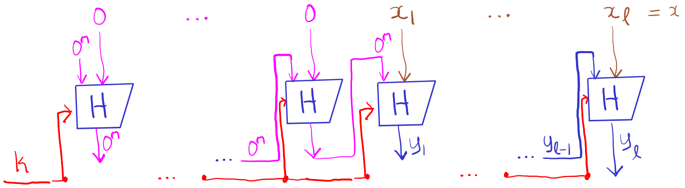


Approach I: Merkle-Damgård Construction (Chaining)...

What happens if we use Construction 1 for $\{0,1\}^*$?

Is it possible to find collisions of *different* length?

- Yes, consider H for which $H(k, 0^{n+1}) = 0^n$ (for all k)
  - For H' instantiated with above H: $H'(k, 0^n \| x) = H'(k, x)$



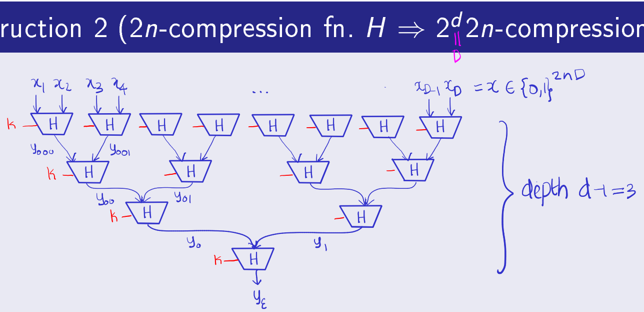
Exercise 4

Tweak Construction 1 to obtain CRHF (i.e., for domain $\{0,1\}^*$)

- Hint: add appropriate padding in the end

Approach II: Merkle's Construction (Tree-Based)

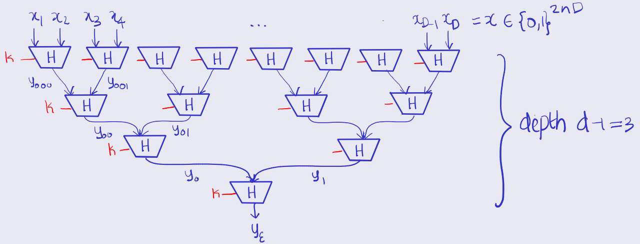
Construction 2 ($2n$-compression fn. $H \Rightarrow 2^d 2n$-compression fn. H')



Approach II: Merkle's Construction (Tree-Based)

Construction 2 ($2n$-compression fn. $H \Rightarrow 2^d 2n$-compression fn. H')

$\overset{\|}{D}$



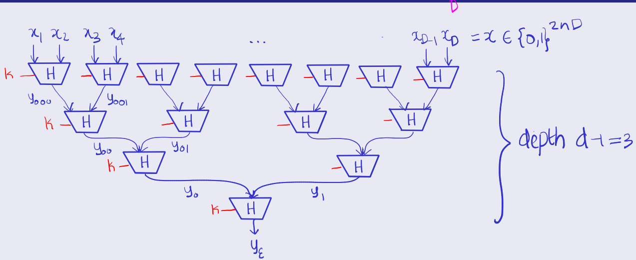
- $\forall v \in \{0,1\}^d : y_v := x_v$ and $\forall v \in \{0,1\}^{<d} : y_v := H(k, y_{v\|0} \| y_{v\|1})$

Exercise 5

Show that if H is a compression function then so is H'

Approach II: Merkle's Construction (Tree-Based)

Construction 2 ($2n$-compression fn. $H \Rightarrow 2^d 2n$-compression fn. H')



- $\forall v \in \{0,1\}^d : y_v := x_v$ and $\forall v \in \{0,1\}^{<d} : y_v := H(k, y_{v\|0} \| y_{v\|1})$

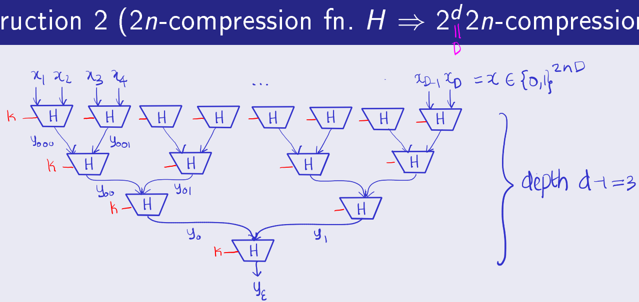
Exercise 5

Show that if H is a compression function then so is H'

- Is H' parallelisable?
- Can parts of input can be *locally* verified?

Approach II: Merkle's Construction (Tree-Based)

Construction 2 ($2n$-compression fn. $H \Rightarrow 2^d 2n$-compression fn. H')



- $\forall v \in \{0,1\}^d : y_v := x_v$ and $\forall v \in \{0,1\}^{<d} : y_v := H(k, y_{v\|0} \| y_{v\|1})$

Exercise 5

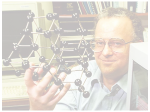
Show that if H is a compression function then so is H'

Is H' parallelisable?

Can parts of input can be *locally* verified?

- Task: sign arbitrarily long messages
- Threat model: EU-CMA

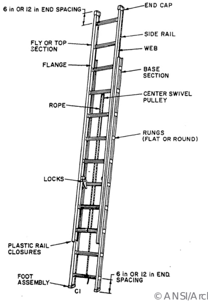
Hash Function



© Prof. Bill Buchanan/Medium

© cs.au.dk/~ivan

Domain Extension



© ANSI/Archive.org

Old tricks: chain, tree-based constructions

How to Construct Compression Functions in Practice?

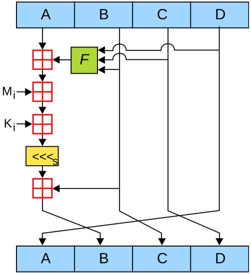
- *Unkeyed* compression fn. for fixed input (block)/output length

How to Construct Compression Functions in Practice?

- *Unkeyed* compression fn. for fixed input (block)/output length
  - Message Digest (MD) family
    - ⚠ MD5 (512/128): collisions have been found!

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab58712467eab 4004583eb8fb7f89
55ad340609f4b302 83e488832571415a 085125e8f7cdc99f d91dbdf280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e2b487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080a80d1e c69821bcb6a88393 96f9652b6ff72a70
```

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab50712467eab 4004583eb8fb7f89
55ad340609f4b302 83e4888325f1415a 085125e8f7cdc99f d91dbd7280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e23487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080280d1e c69821bcb6a88393 96f965ab6ff72a70
```



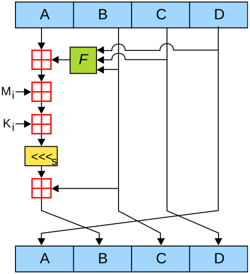
© Surachit/Wikipedia

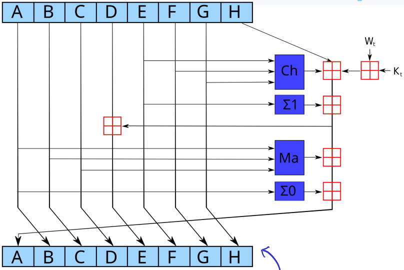
How to Construct Compression Functions in Practice?

- *Unkeyed* compression fn. for fixed input (block)/output length
  - Message Digest (MD) family
    - ⚠ MD5 (512/128): collisions have been found!

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab58712467eab 4004583eb8fb7f89
55ad340609f4b302 83e488832571415a 085125e8f7cdc99f d91dbdf280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e2b487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080a80d1e c69821bcb6a88393 96f9652b6ff72a70
```

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab50712467eab 4004583eb8fb7f89
55ad340609f4b302 83e4888325f1415a 085125e8f7cdc99f d91dbd7280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e23487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080280d1e c69821bcb6a88393 96f965ab6ff72a70
```





  - Secure-Hashing Algorithm (SHA) family
    - SHA2 (512/256,1024/512...): Davis-Meyer compression function
    - SHA3 (1152/224,576,512): “Sponge”-based compression function

How to Construct Compression Functions in Theory?

- Based on DLP in $\mathbb{Z}_p^\times$: $\{H : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$, where

$$H((g,h),(a,b)) := g^a h^b \bmod p$$

How to Construct Compression Functions in Theory?

- Based on DLP in $\mathbb{Z}_p^\times$: $\{H : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$, where

$$H((g,h),(a,b)) := g^a h^b \bmod p$$

? How to solve DLP given a collision $((a,b),(a',b'))$?

How to Construct Compression Functions in Theory?

- Based on DLP in $\mathbb{Z}_p^\times$: $\{H : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$, where

$$H((g,h),(a,b)) := g^a h^b \bmod p$$

? How to solve DLP given a collision $((a,b),(a',b'))$?

- Based on subset-sum problem:

$$H((a_1,\ldots,a_n), x_1\|\ldots\|x_n) := \sum_{i\in[1,n]} x_i a_i \bmod p$$

? When is H compressing?

How to Construct Compression Functions in Theory?

- Based on DLP in $\mathbb{Z}_p^\times$: $\{H : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$, where

$$H((g,h),(a,b)) := g^a h^b \bmod p$$

  ? How to solve DLP given a collision $((a,b),(a',b'))$?

- Based on subset-sum problem:

$$H((a_1,\ldots,a_n), x_1\|\ldots\|x_n) := \sum_{i\in[1,n]} x_i a_i \bmod p$$

  ? When is H compressing? When we set $p < 2^n$

  ? How to solve subset-sum given a collision?

Recap/Next Lecture

- Introduced a new primitive: collision-resistant hash function
  - Application: sign long messages
  - Also yields MAC for long messages! Refer to “HMAC”

Recap/Next Lecture

- Introduced a new primitive: collision-resistant hash function
  - Application: sign long messages
  - Also yields MAC for long messages! Refer to “HMAC”
- Domain extension
  - Merkle-Damgård transform
  - Merkle trees

Recap/Next Lecture

- Introduced a new primitive: collision-resistant hash function
  - Application: sign long messages
  - Also yields MAC for long messages! Refer to "HMAC"
- Domain extension
  - Merkle-Damgård transform
  - Merkle trees
- Some constructions:
  - Practical/unkeyed: SHA2, MD5
  - Theoretical/keyed: based on DLP and subset-sum problem

Recap/Next Lecture

- Introduced a new primitive: collision-resistant hash function
  - Application: sign long messages
  - Also yields MAC for long messages! Refer to "HMAC"
- Domain extension
  - Merkle-Damgård transform
  - Merkle trees
- Some constructions:
  - Practical/unkeyed: SHA2, MD5
  - Theoretical/keyed: based on DLP and subset-sum problem

- Next lecture:
  - Efficient many-time signatures
  - New primitive: trap-door (one-way) permutation (TDP)
  - Proof in random oracle model (ROM)

References

1. You can read about hash functions and collision resistance in [KL14, Chapter 6].
2. Hash functions were first studied in [WC81], but they considered pairwise-independence/universal hashing
3. Collision resistance, and other cryptographic properties of hash functions were studied later [Dam88, Dam90, NY89, Mer90b] a thorough historical perspective can be found in [RS04]

Ivan Damgård.
Collision free hash functions and public key signature schemes.
In David Chaum and Wyn L. Price, editors, *EUROCRYPT'87*, volume 304 of *LNCS*, pages 203–216. Springer, Berlin, Heidelberg, April 1988.

Ivan Damgård.
A design principle for hash functions.
In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer, New York, August 1990.

Oded Goldreich.
Two remarks concerning the Goldwasser-Micali-Rivest signature scheme.
In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 104–110. Springer, Berlin, Heidelberg, August 1987.

Jonathan Katz and Yehuda Lindell.
Introduction to Modern Cryptography (3rd ed.).
Chapman and Hall/CRC, 2014.

Ralph C. Merkle.
A certified digital signature.
In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, New York, August 1990.

Ralph C. Merkle.
One way hash functions and DES.
In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 428–446. Springer, New York, August 1990.

Moni Naor and Moti Yung.
Universal one-way hash functions and their cryptographic applications.
In *21st ACM STOC*, pages 33–43. ACM Press, May 1989.

Phillip Rogaway and Thomas Shrimpton.
Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance.
In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 371–388. Springer, Berlin, Heidelberg, February 2004.

Mark N. Wegman and J.Lawrence Carter.
New hash functions and their use in authentication and set equality.
Journal of Computer and System Sciences, 22(3):265–279, 1981.