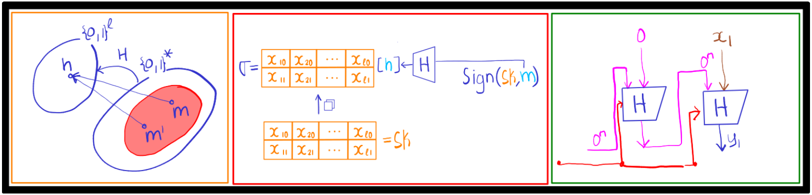


CS783: Theoretical Foundations of Cryptography

Lecture 12 (10/Sep/24)

Instructor: Chethan Kamath

Plan for Today's Lecture

Recall from Last Lecture

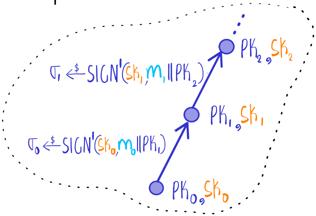
- Introduced digital signatures: public-key analogue of MAC

Recall from Last Lecture

- Introduced digital signatures: public-key analogue of MAC
- Theoretical construction
  - Lamport's one-time DS $\leftarrow$ OWF

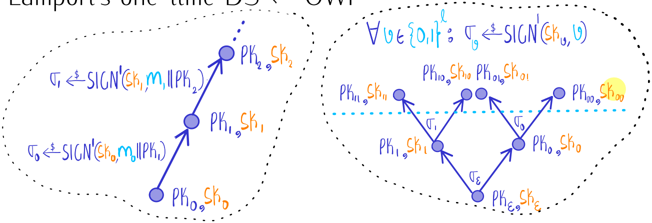
Recall from Last Lecture

- Introduced digital signatures: public-key analogue of MAC
- Theoretical construction
  - Lamport's one-time DS ← OWF



  - One-time DS → (many-time) *stateful* DS: "chain of signatures"

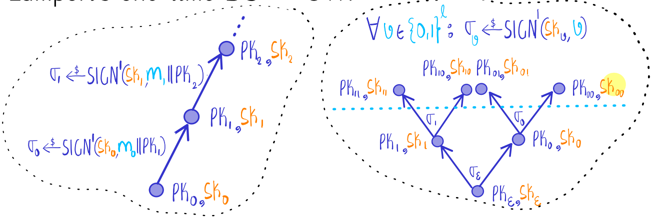
- Introduced digital signatures: public-key analogue of MAC
- Theoretical construction
  - Lamport's one-time DS ← OWF



  - One-time DS → (many-time) *stateful* DS: "chain of signatures"
  - One-time DS → compact *stateful* DS: "tree of signatures"

Recall from Last Lecture

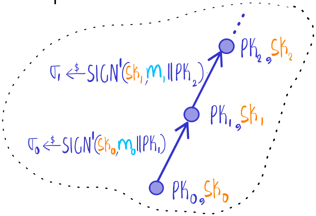
- Introduced digital signatures: public-key analogue of MAC
- Theoretical construction
  - Lamport's one-time DS ← OWF

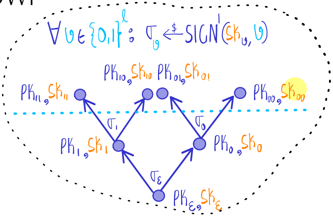


  - One-time DS → (many-time) *stateful* DS: "chain of signatures"
  - One-time DS → compact *stateful* DS: "tree of signatures"
  - *Stateless* DS via *derandomisation* using PRF

Recall from Last Lecture

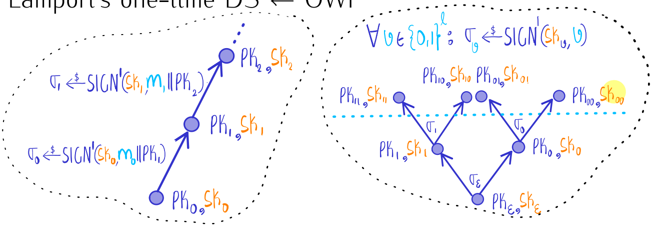
- Introduced digital signatures: public-key analogue of MAC
- Theoretical construction
  - Lamport's one-time DS ← OWF





  - One-time DS → (many-time) *stateful* DS: "chain of signatures"
  - One-time DS → compact *stateful* DS: "tree of signatures"
  - *Stateless* DS via *derandomisation* using PRF
- Efficient DS in "random-oracle model"
  - Lectures 13: from *trapdoor* OWP via hash-then-invert
  - Lecture 15(?): via Fiat-Shamir transform (e.g., Schnorr)

- Introduced digital signatures: public-key analogue of MAC
- Theoretical construction
  - Lamport's one-time DS ← OWF



  - One-time DS → (many-time) *stateful* DS: "chain of signatures"
  - One-time DS → compact *stateful* DS: "tree of signatures"
  - *Stateless* DS via *derandomisation* using PRF
- Efficient DS in "random-oracle model"
  - Lectures 13: from *trapdoor* OWP via hash-then-invert
  - Lecture 15(?): via Fiat-Shamir transform (e.g., Schnorr)
- Takeaways:
  - Constructive: "bootstrapping" one-time to many-time signatures
  - Proof techniques: "plug and pray" $pk^* =$

y_{00}	y_{10}	y_{20}	y_{30}
y_{01}	y^*	y_{21}	y_{31}

$b^* \leftarrow \{0,1\}$

$i^* \leftarrow [\ell]$

Plan for Today's Lecture...

Theorem 1 (Theorem 1, Lecture 11)

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

Plan for Today's Lecture...

Theorem 1 (Theorem 1, Lecture 11)

If f is a OWF then Lamport's scheme is a one-time DS for fixed-length messages $\{0,1\}^{\ell}$.

Plan for Today's Lecture...

Theorem 1 (Theorem 1, Lecture 11)

If f is a OWF then Lamport's scheme is a one-time DS for fixed-length messages $\{0,1\}^{\ell}$. $|PK| = 2\ell n$

Plan for Today's Lecture...

Theorem 1 (Theorem 1, Lecture 11)

If f is a OWF then Lamport's scheme is a one-time DS for fixed-length messages $\{0,1\}^{\ell}$. $|pk| = 2\ell n$

Theorem 2 (PRF→ MAC: Theorem 2, Lecture 7) $\mathrm{Tag}(k,m) := F_k(m)$

If $\{\mathsf{F}_{\mathsf{k}} : \{0,1\}^n \to \{0,1\}^n\}_{\mathsf{k}\in\{0,1\}^n}$ *is a PRF then Construction 3 (Lecture 7) is EU-CMA-secure for fixed-length messages* $\{0,1\}^n$.

Plan for Today's Lecture...

Theorem 1 (Theorem 1, Lecture 11)

If f is a OWF then Lamport's scheme is a one-time DS for fixed-length messages $\{0,1\}^{\ell}$. $|pk| = 2\ell n$

Theorem 2 (PRF→ MAC: Theorem 2, Lecture 7) $\mathrm{Tag}(k,m) := F_k(m)$

If $\{F_k : \{0,1\}^n \to \{0,1\}^n\}_{k \in \{0,1\}^n}$ *is a PRF then Construction 3 (Lecture 7) is EU-CMA-secure for fixed-length messages* $\{0,1\}^n$.

Exercise 1 (Exercise 3, Lecture 11 (Domain Extension)) $\to \boxed{H} \to$

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...

Theorem 1 (Theorem 1, Lecture 11)

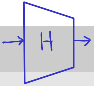
If f is a OWF then Lamport's scheme is a one-time DS for fixed-length messages $\{0,1\}^\ell$.

$|pk| = 2\ell n$

Theorem 2 (PRF→ MAC: Theorem 2, Lecture 7) $\mathrm{Tag}(k,m) := F_k(m)$

If $\{F_k : \{0,1\}^n \to \{0,1\}^n\}_{k \in \{0,1\}^n}$ *is a PRF then Construction 3 (Lecture 7) is EU-CMA-secure for fixed-length messages* $\{0,1\}^n$.

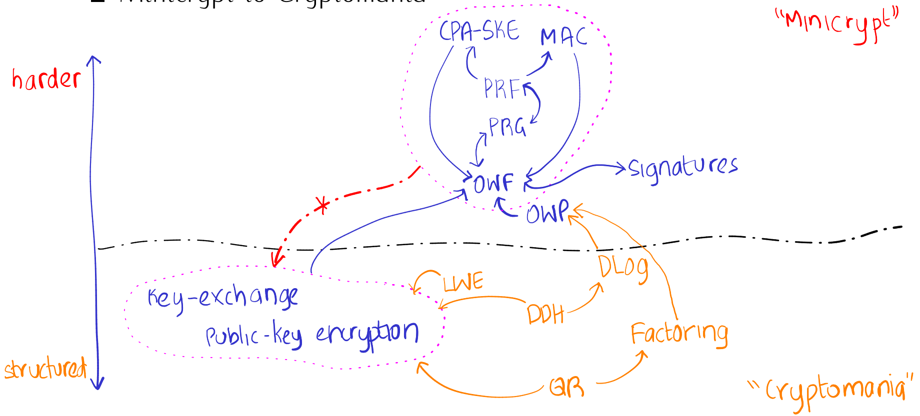
Exercise 1 (Exercise 3, Lecture 11 (Domain Extension))



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?*

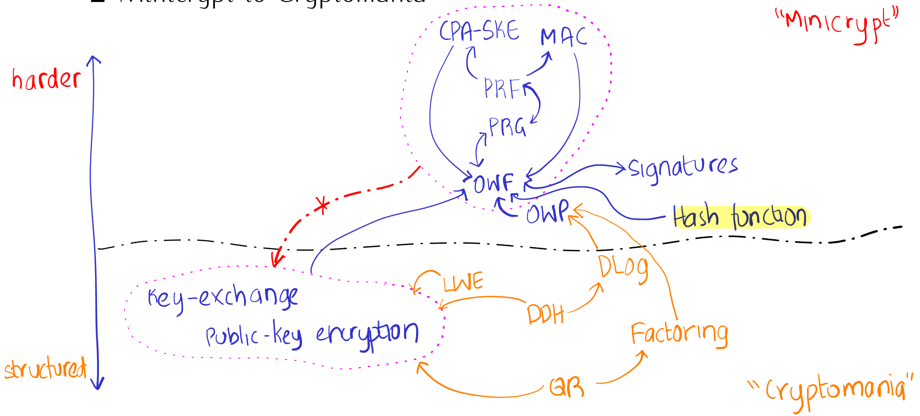
"Hash function"

- Minicrypt to Cryptomania



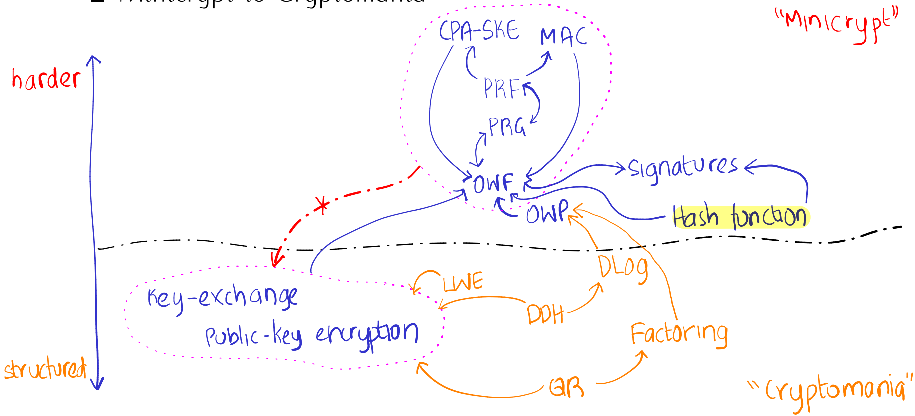
- Sub-task 5.a: domain-extension of digital signature/MAC
  - Sufficient to construct *hash functions* with certain properties

- Minicrypt to Cryptomania



- Sub-task 5.a: domain-extension of digital signature/MAC
  - Sufficient to construct *hash functions* with certain properties

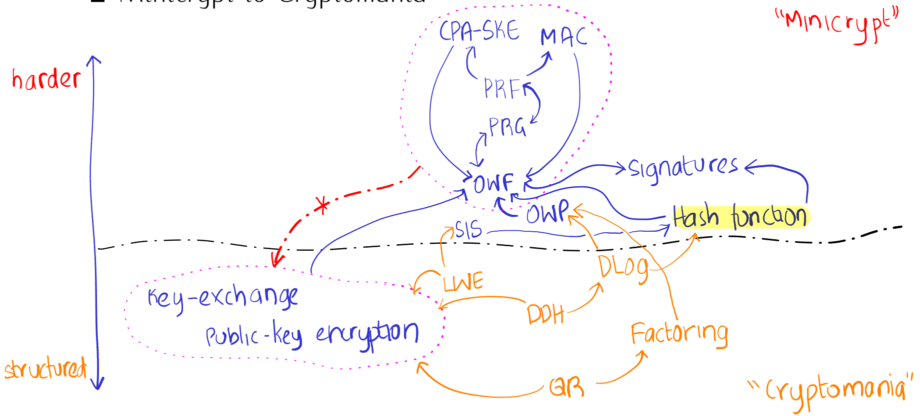
- Minicrypt to Cryptomania



- Sub-task 5.a: domain-extension of digital signature/MAC
  - Sufficient to construct *hash functions* with certain properties

Plan for Today's Lecture...

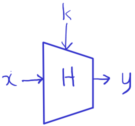
- Minicrypt to Cryptomania



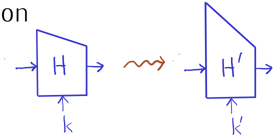
- Sub-task 5.a: domain-extension of digital signature/MAC
  - Sufficient to construct *hash functions* with certain properties

Plan for Today's Lecture...

1 Hash Functions



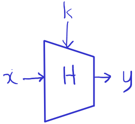
2 Compression Functions and Domain-Extension



3 How to Construct Compression Functions?

SIS DLOG

1 Hash Functions



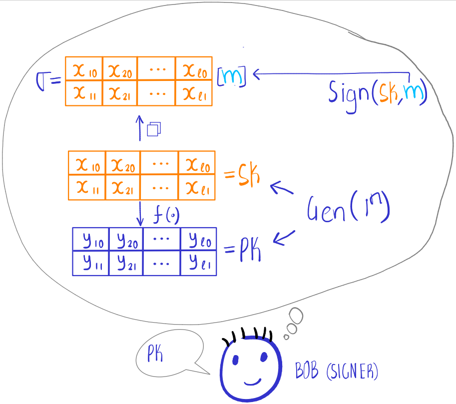
2 Compression Functions and Domain-Extension

3 How to Construct Compression Functions?

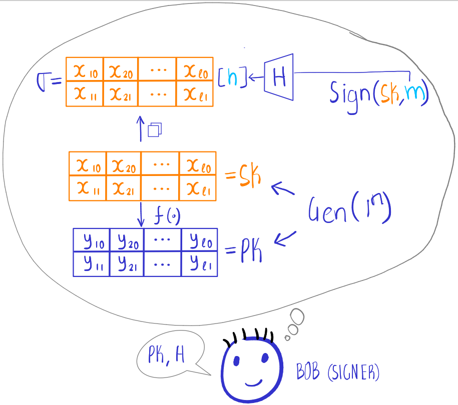
SIS DLOG

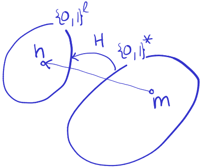
Extending Lamport's One-Time DS for Longer Messages...

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

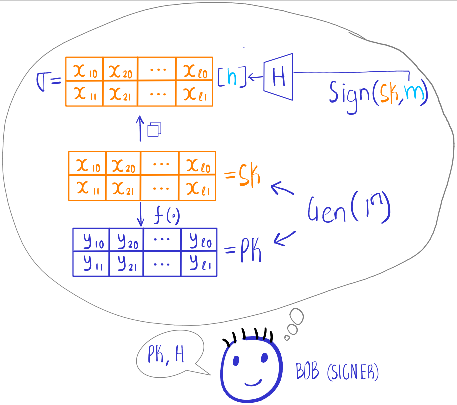


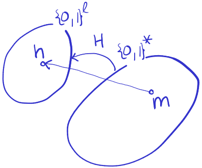
Extending Lamport's One-Time DS for Longer Messages...



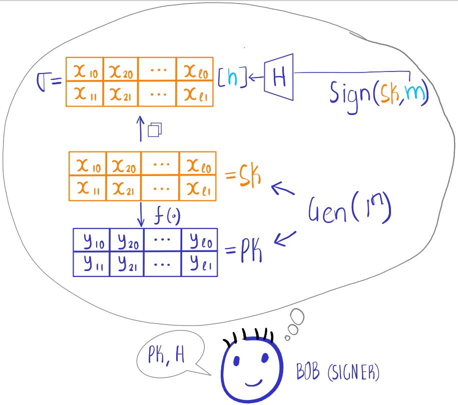


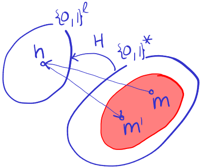
- Hash-then-sign: compute "hash" $h = \mathrm{H}(m)$ and then sign h





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



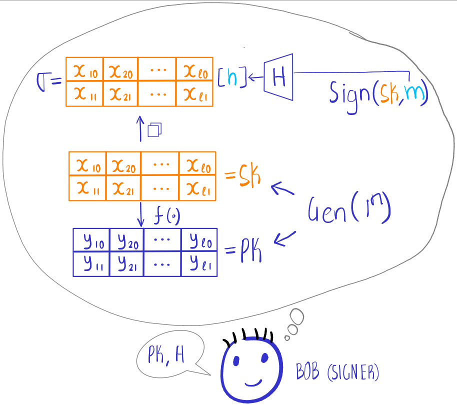


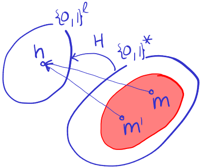
- Hash-then-sign: compute "hash" $h = \mathrm{H}(m)$ and then sign h

? 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 Longer Messages...

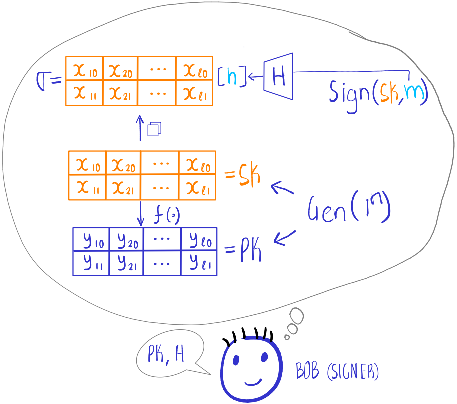


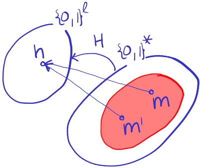


- Hash-then-sign: compute "hash" $h = \mathrm{H}(m)$ and then sign h

? 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?





- Hash-then-sign: compute "hash" $h = \mathrm{H}(m)$ 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.

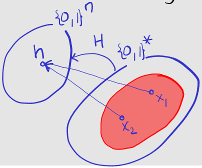
Collision-Resistant Hash Function (CRHF)...

Collision-Resistant Hash Function (CRHF)...

Defintion 1 (*Keyless* CRHF)

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

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



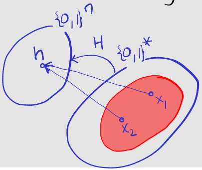
Collision-Resistant Hash Function (CRHF)...

Defintion 1 (*Keyless* CRHF)

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

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

Need not be same length



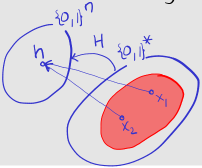
Collision-Resistant Hash Function (CRHF)...

Defintion 1 (*Keyless* CRHF)

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

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

Need not be same length

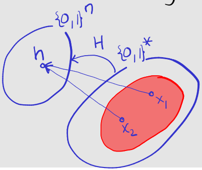


- Problem: trivial for *non-uniform* adversaries

Collision-Resistant Hash Function (CRHF)...

Defintion 1 (*Keyless* CRHF)

A function (family) $\left\{ \mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^n \right\}$ *is a CRHF if for every PPT collision-finder* F*, the following is negligible.*



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

Need not be same length

- Problem: trivial for *non-uniform* adversaries

Defintion 2 (CRHF, with key generation algorithm Gen)

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

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

Collision-Resistant Hash Function (CRHF)...

If H_1 and H_2 are CRHFs is H?

Collision-Resistant Hash Function (CRHF)...

If H_1 and H_2 are CRHFs is H?

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

Collision-Resistant Hash Function (CRHF)...

If H_1 and H_2 are CRHFs is H?

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

Collision-Resistant Hash Function (CRHF)...

If H_1 and H_2 are CRHFs is H?

1. Hash-then-append: $\mathsf{H}(\mathsf{k}, x) := \mathsf{H}_1(\mathsf{k}, x)0$
2. Hash-then-truncate: $\mathsf{H}(\mathsf{k}, x) := y_1 \ldots y_{n-1}$, where $y_1 \ldots y_n := \mathsf{H}_1(\mathsf{k}, x)$

Collision-Resistant Hash Function (CRHF)...

If H_1 and H_2 are CRHFs 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. Double hash: $H(k_1 k_2, x) := H_1(k_1, x) H_2(k_2, x)$

Collision-Resistant Hash Function (CRHF)...

If H_1 and H_2 are CRHFs 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. Double hash: $H(k_1 k_2, x) := H_1(k_1, x) H_2(k_2, x)$
4. 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)...

If H_1 and H_2 are CRHFs is H?

1. Hash-then-append: $\mathsf{H}(\mathsf{k}, x) := \mathsf{H}_1(\mathsf{k}, x)0$
2. Hash-then-truncate: $\mathsf{H}(\mathsf{k}, x) := y_1 \ldots y_{n-1}$, where $y_1 \ldots y_n := \mathsf{H}_1(\mathsf{k}, x)$
3. Double hash: $\mathsf{H}(\mathsf{k}_1\mathsf{k}_2, x) := \mathsf{H}_1(\mathsf{k}_1, x)\mathsf{H}_2(\mathsf{k}_2, x)$
4. Hash-then-XOR: $\mathsf{H}(\mathsf{k}_1\mathsf{k}_2, x) := \mathsf{H}_1(\mathsf{k}_1, x) \oplus \mathsf{H}_2(\mathsf{k}_2, x)$
5. Hash-of-hash: $\mathsf{H}(\mathsf{k}, x_1x_2) := \mathsf{H}_1(\mathsf{k}, \mathsf{H}_2(\mathsf{k}, x_2))$

If H_1 and H_2 are CRHFs is H?

1. Hash-then-append: $\mathsf{H}(\mathsf{k}, x) := \mathsf{H}_1(\mathsf{k}, x)0$
2. Hash-then-truncate: $\mathsf{H}(\mathsf{k}, x) := y_1 \ldots y_{n-1}$, where $y_1 \ldots y_n := \mathsf{H}_1(\mathsf{k}, x)$
3. Double hash: $\mathsf{H}(\mathsf{k}_1\mathsf{k}_2, x) := \mathsf{H}_1(\mathsf{k}_1, x)\mathsf{H}_2(\mathsf{k}_2, x)$
4. Hash-then-XOR: $\mathsf{H}(\mathsf{k}_1\mathsf{k}_2, x) := \mathsf{H}_1(\mathsf{k}_1, x) \oplus \mathsf{H}_2(\mathsf{k}_2, x)$
5. Hash-of-hash: $\mathsf{H}(\mathsf{k}, x_1x_2) := \mathsf{H}_1(\mathsf{k}, \mathsf{H}_2(\mathsf{k}, x_2))$

Exercise 2

Prove formally the cases where H is a CRHF; describe counter-example otherwise.

Let's (Slowly) Find Collisions in H!

$\{K \times \{0,1\}^* \to \{0,1\}^n\}$

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

Let's (Slowly) Find Collisions in H!

$\{K \times \{0,1\}^* \to \{0,1\}^n\}$

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

- Exploit pigeonhole principle

Let's (Slowly) Find Collisions in H!

$\{K \times \{0,1\}^* \to \{0,1\}^n\}$

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

- Exploit pigeonhole principle
- Compute (e.g.) hash of inputs $00^n, \ldots, 01^n, 10^n$
  - There *must exist* colliding pair of inputs
- What is the amount of space required? Naïvely, $O(n2^n)$

Let's (Slowly) Find Collisions in H!

$\{K \times \{0,1\}^* \to \{0,1\}^n\}$

- What about a deterministic $O(2^n)$-time collision-finder?
  - Exploit pigeonhole principle
  - Compute (e.g.) hash of inputs $00^n, \ldots, 01^n, 10^n$
    - There *must exist* colliding pair of inputs
  - What is the amount of space required? Naïvely, $O(n2^n)$
- Randomised $O(2^{n/2})$-time$+O(n2^{n/2})$-space collision-finder

Let's (Slowly) Find Collisions in H!

$\{K \times \{0,1\}^* \to \{0,1\}^n\}$

- What about a deterministic $O(2^n)$-time collision-finder?
  - Exploit pigeonhole principle
  - Compute (e.g.) hash of inputs $00^n, \ldots, 01^n, 10^n$
    - There *must exist* colliding pair of inputs
  - What is the amount of space required? Naïvely, $O(n2^n)$
- Randomised $O(2^{n/2})$-time+$O(n2^{n/2})$-space collision-finder
  - Exploit birthday paradox

Let's (Slowly) Find Collisions in H!

$\{K \times \{0,1\}^* \to \{0,1\}^n\}$

- What about a deterministic $O(2^n)$-time collision-finder?
  - Exploit pigeonhole principle
  - Compute (e.g.) hash of inputs $00^n, \ldots, 01^n, 10^n$
    - There *must exist* colliding pair of inputs
  - What is the amount of space required? Naïvely, $O(n2^n)$
- Randomised $O(2^{n/2})$-time+$O(n2^{n/2})$-space collision-finder

  - Exploit birthday paradox
  - Compute hash of $N := O(2^{n/2})$ *random* inputs $x_1, \ldots, x_N$
    - With *noticeable probability*, there exist colliding pairs of inputs
  - Consequence: key-size/output length must be $2\times$ security level

Let's (Slowly) Find Collisions in H!

$\{K \times \{0,1\}^* \to \{0,1\}^n\}$

- What about a deterministic $O(2^n)$-time collision-finder?
  - Exploit pigeonhole principle
  - Compute (e.g.) hash of inputs $00^n, \ldots, 01^n, 10^n$
    - There *must exist* colliding pair of inputs
  - What is the amount of space required? Naïvely, $O(n2^n)$
- Randomised $O(2^{n/2})$-time+$O(n2^{n/2})$-space collision-finder
  - Exploit birthday paradox
  - Compute hash of $N := O(2^{n/2})$ *random* inputs $x_1, \ldots, x_N$
    - With *noticeable probability*, there exist colliding pairs of inputs
  - Consequence: key-size/output length must be $2\times$ security level

Exercise 3

1. *Is deterministic $O(2^{n/2})$-time+$O(n2^{n/2})$-space collision-finder possible?*
2. *Is rand. $O(2^{n/2})$-time+$O(n)$-space collision-finder possible?*

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

Theorem 3

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 Longer Messages...

Theorem 3

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: ∃Inv for f *or* ∃F for H ⇐ ∃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 Longer Messages...

Theorem 3

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: $\exists\mathsf{Inv}$ for f *or* $\exists\mathsf{F}$ for H $\Leftarrow$ $\exists\mathsf{Tam}$ 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 $\overline{\mathsf{CAL}}$: $H(k,m) \neq H(k,m^*) = h^*$

□

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

Theorem 3

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

$\square$

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

Theorem 3

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

$\square$

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

Theorem 3

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

sample(sk, PK)

k

Collision finder F

Tam

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

$\square$

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

Theorem 3

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".

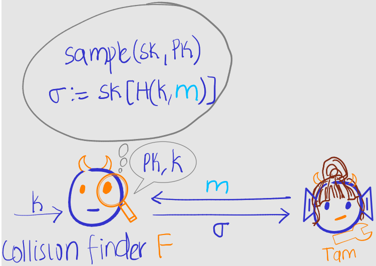
- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

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

$\Downarrow$

(m, m^*) collision for H

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



□

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

Theorem 3

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".

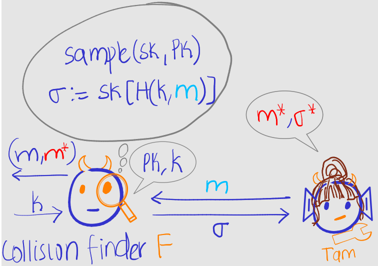
- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

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

$\Downarrow$

(m, m^*) collision for H

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



$\square$

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

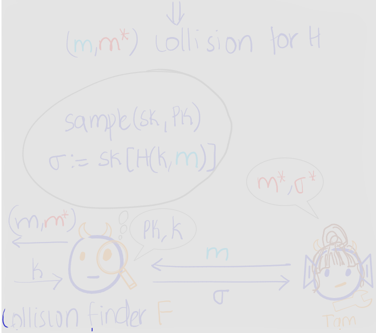
Theorem 3

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 ① : $H(k, m) = H(k, m^*)$



Case $\overline{①}$: $H(k, m) \neq H(k, m^*) = h^*$

$\Downarrow$

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

□

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

Theorem 3

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

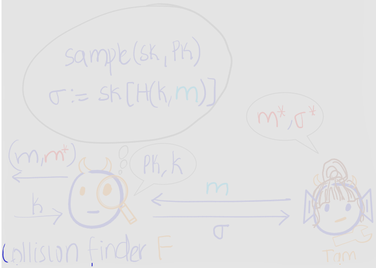
Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

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

$\Downarrow$

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

□

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

Theorem 3

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

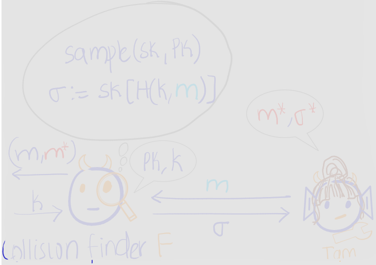
Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

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

Case ①: $H(k,m) = H(k,m^*)$

$\Downarrow$

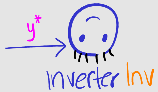
(m,m^*) collision for H



Case ②: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

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



$\square$

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

Theorem 3

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^*)

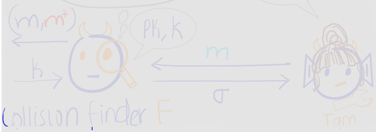
Case (a): $H(k,m) = H(k,m^*)$

$\Downarrow$

(m, m^*) collision for H

sample(sk, PK)

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

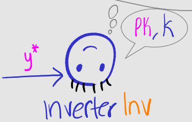


Case $\overline{(a)}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

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

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



$\square$

Theorem 3

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: ∃Inv for f *or* ∃F for H ⇐ ∃Tam for "hash-then-sign".

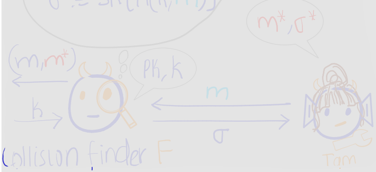
- Suppose Tam queries some $m \in \{0,1\}^*$ and outputs forgery (m^*, σ^*)

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

$\Downarrow$

(m, m^*) collision for H

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

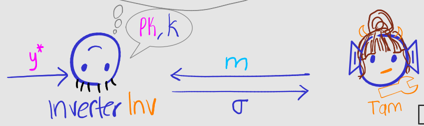


Case $\overline{\textcircled{a}}$: $H(k,m) \neq H(k,m^*) = h^*$

$\Downarrow$

(h^*, σ^*) forgery for Lamport's ⇒ invert f

"Plug" y^* at (i^*, b^*) to
generate (sk, PK)
"Pray": $H(k,m)[i^*] \neq b^*$



□

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

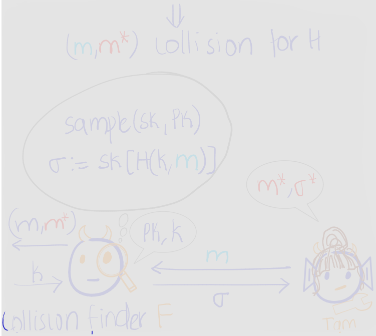
Theorem 3

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^*)

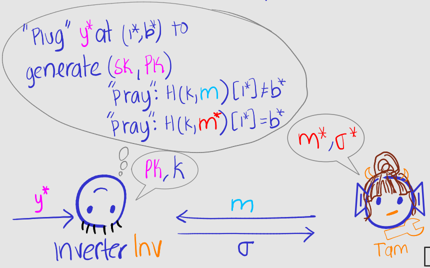
Case Cal: $H(k,m) = H(k,m^*)$



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

$\Downarrow$

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



□

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

Theorem 3

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

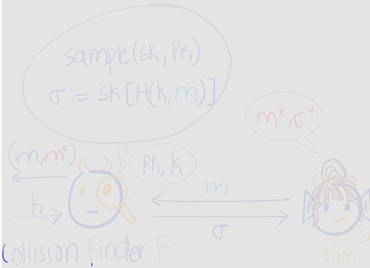
Proof sketch: $\exists$Inv for f *or* $\exists$F for H $\Leftarrow$ $\exists$Tam for "hash-then-sign".

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

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

$\Downarrow$

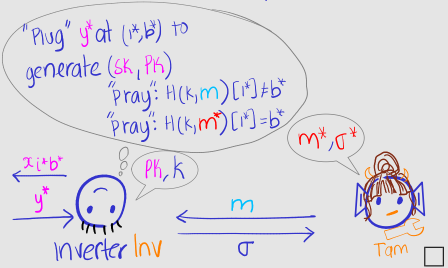
(m, m^*) collision for H



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

$\Downarrow$

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

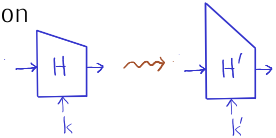


□

Plan for Today's Lecture

1 Hash Functions

2 Compression Functions and Domain-Extension

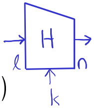


3 How to Construct Compression Functions?

SIS DLOG

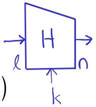
Compression Functions and Domain-Extension

- (Collision-resistant) *compression function*: CRHF 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

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



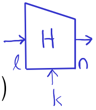
Defintion 3 ($\ell(n)$-compression function)

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

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

same length

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



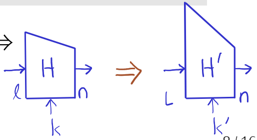
Defintion 3 ($\ell(n)$-compression function)

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

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

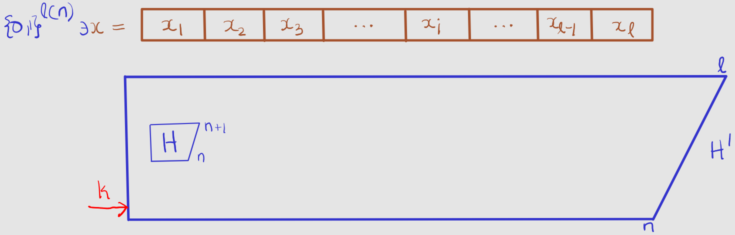
same length

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

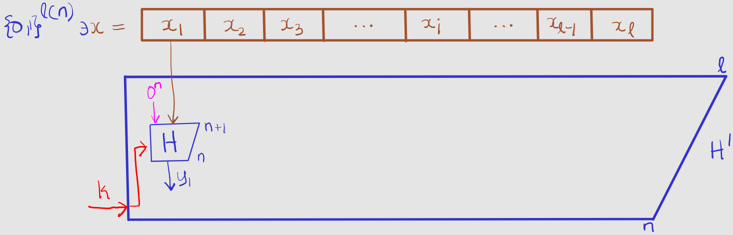


Merkle–Damgård Transform (Chaining)...

Construction 1 (($n+1$)-compression function H $\Rightarrow$ $\ell(n)$-compression function H′, for any polynomial $\ell(n)$)

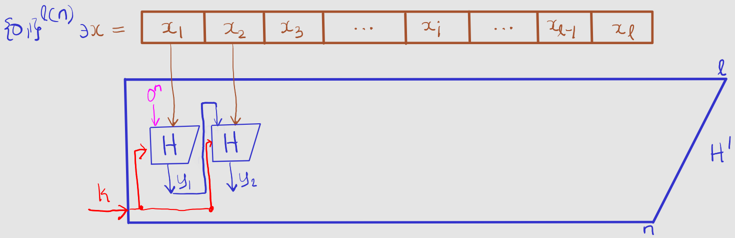


Construction 1 (($n+1$)-compression function H $\Rightarrow$ $\ell(n)$-compression function H′, for any polynomial $\ell(n)$)



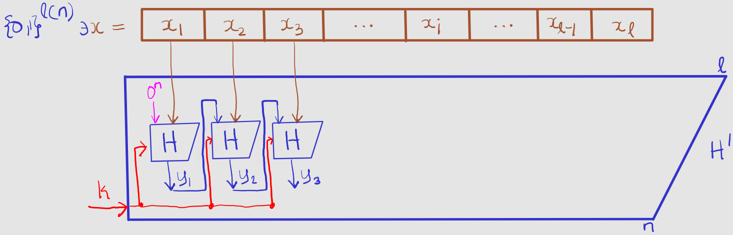
Merkle-Damgård Transform (Chaining)...

Construction 1 (($n+1$)-compression function H $\Rightarrow$ $\ell(n)$-compression function H′, for any polynomial $\ell(n)$)

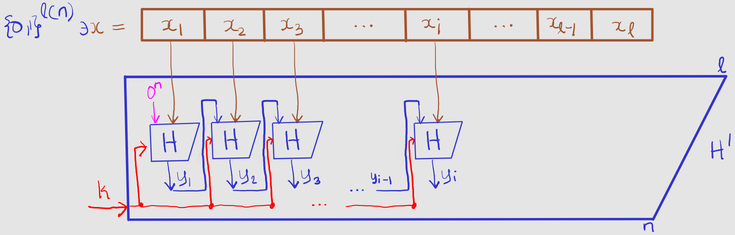


Merkle–Damgård Transform (Chaining)...

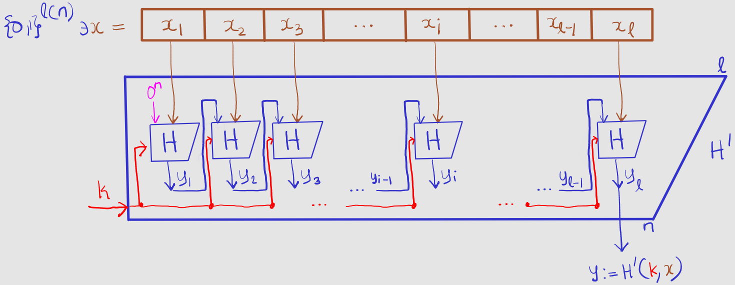
Construction 1 (($n+1$)-compression function H $\Rightarrow$ $\ell(n)$-compression function H$'$, for any polynomial $\ell(n)$)



Construction 1 (($n+1$)-compression function H $\Rightarrow$ $\ell(n)$-compression function H′, for any polynomial $\ell(n)$)

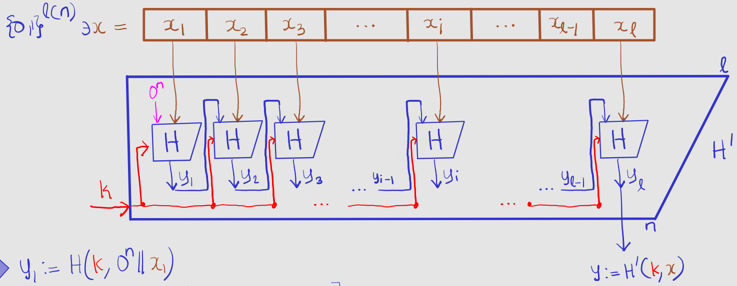


Construction 1 ($(n+1)$-compression function H $\Rightarrow$ $\ell(n)$-compression function H′, for any polynomial $\ell(n)$)



Merkle–Damgård Transform (Chaining)

Construction 1 (($n+1$)-compression function H $\Rightarrow$ $\ell(n)$-compression function H′, for any polynomial $\ell(n)$)



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

Merkle-Damgård Transform (Chaining)...

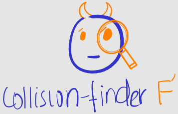
Theorem 4

If H *is a compression function then so is* H'

Theorem 4

If H *is a compression function then so is* H′

Proof sketch: ∃ collision finder F for H ⇐ ∃ coll. finder F′ for H′.



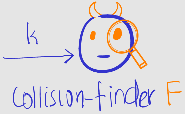
□

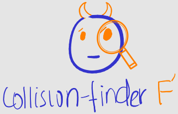
Merkle-Damgård Transform (Chaining)...

Theorem 4

If H *is a compression function then so is* H′

Proof sketch: ∃ collision finder F for H ⇐ ∃ coll. finder F′ for H′.





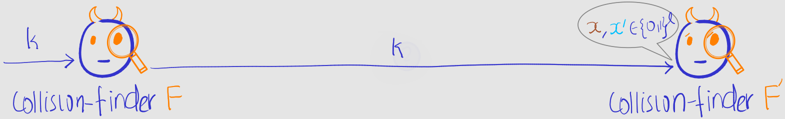
□

Merkle-Damgård Transform (Chaining)...

Theorem 4

If H *is a compression function then so is* H′

Proof sketch: ∃ collision finder F for H ⇐ ∃ coll. finder F′ for H′.



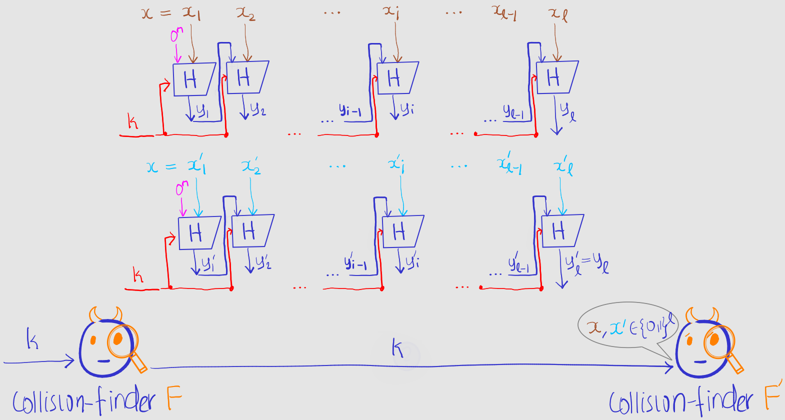
□

Merkle–Damgård Transform (Chaining)...

Theorem 4

If H *is a compression function then so is* H′

Proof sketch: ∃ collision finder F for H ⇐ ∃ coll. finder F′ for H′.



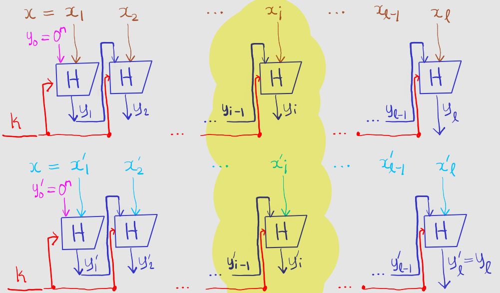
□

Merkle-Damgård Transform (Chaining)...

Theorem 4

If H *is a compression function then so is* H′

Proof sketch: ∃ collision finder F for H ⇐ ∃ coll. finder F′ for H′.



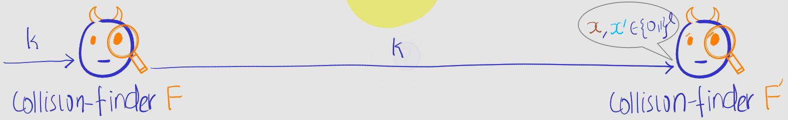
Claim: since $x \neq x'$

$\exists i \in [1,\ell]$ s.t.

i) $y_{i-1} \| x_i \neq y'_{i-1} \| x'_i$

and

ii) $H(k, y_{i-1} \| x_i) = H(k, y'_{i-1} \| x'_i)$



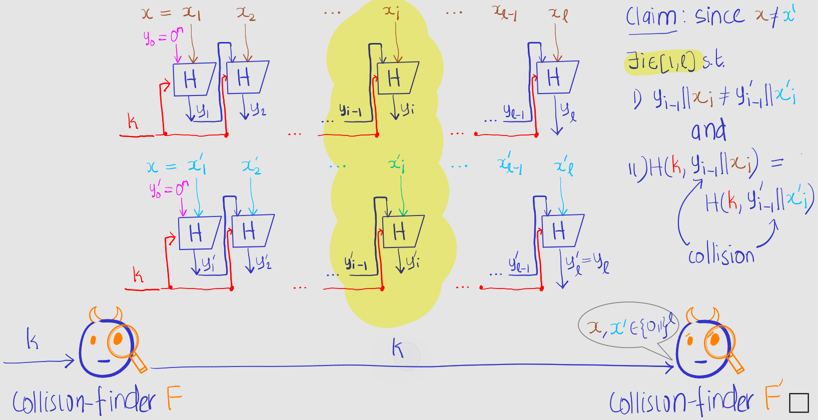
□

Merkle-Damgård Transform (Chaining)...

Theorem 4

If H *is a compression function then so is* H′

Proof sketch: ∃ collision finder F for H ⇐ ∃ coll. finder F′ for H′.

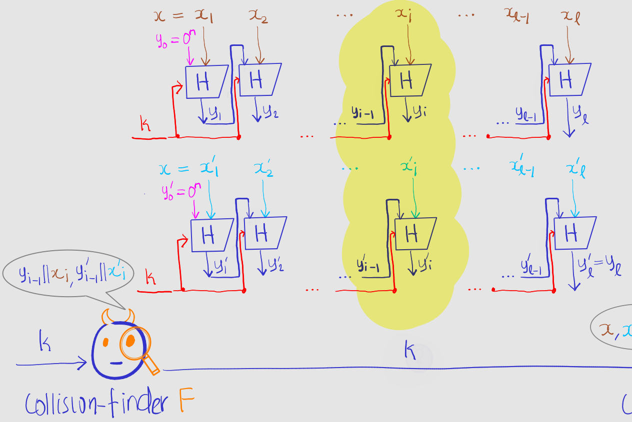


Merkle–Damgård Transform (Chaining)...

Theorem 4

If H *is a compression function then so is* H′

Proof sketch: $\exists$ collision finder F for H $\Leftarrow$ $\exists$ coll. finder F′ for H′.



Claim: since $x \neq x'$

$\exists i \in [1,\ell]$ s.t.

I) $y_{i-1}\|x_i \neq y'_{i-1}\|x'_i$

and

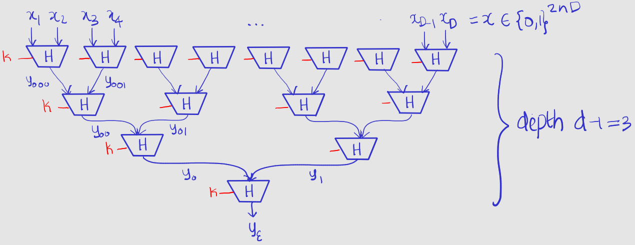
II) $H(k, y_{i-1}\|x_i) = H(k, y'_{i-1}\|x'_i)$

collision

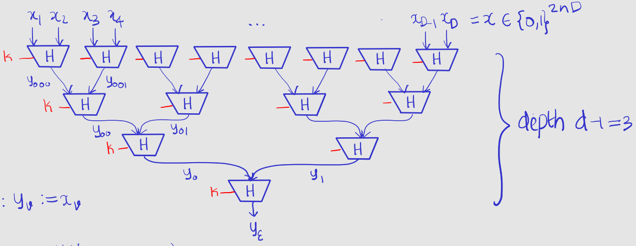
$\square$

Construction 2 (2n-compression function H $\Rightarrow$ $2^d 2n$-compression function H′, for any $d \in \mathbb{N}$)

$\overset{\Vert}{D}$

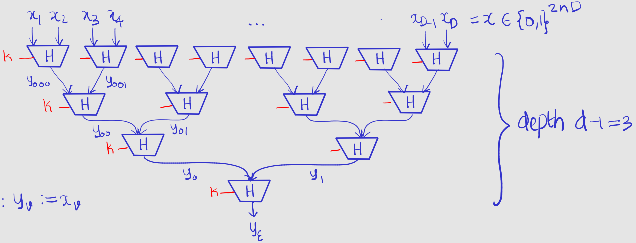


Construction 2 ($2n$-compression function H $\Rightarrow$ $\underset{\overset{\|}{D}}{2^d}2n$-compression function H′, for any $d \in \mathbb{N}$)



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

Construction 2 (2n-compression function H $\Rightarrow$ $\underset{\substack{\| \\ D}}{2^d}2n$-compression function H′, for any $d \in \mathbb{N}$)



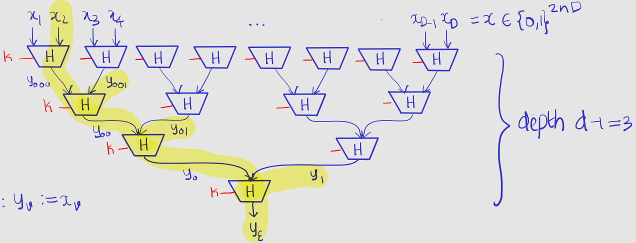
- $\forall v \in \{0,1\}^d : y_v := x_v$
- $\forall v \in \{0,1\}^{<d} : y_v := H(k, y_{v\|0}, y_{v\|1})$

Exercise 4

Show that if H *is a compression function then so is* H′

Construction 2 ($2n$-compression function H $\Rightarrow$ $2^d 2n$-compression function H′, for any $d \in \mathbb{N}$)

$2^d = D$



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

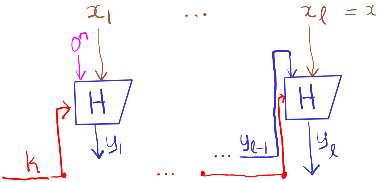
Exercise 4

Show that if H *is a compression function then so is* H′

- Has several interesting properties:
  1. Parallelisable: computable in depth $O(d)$
  2. Locally verifiable: parts of input can be verified

What If We Use Construction 1 for $\{0,1\}^*$?

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

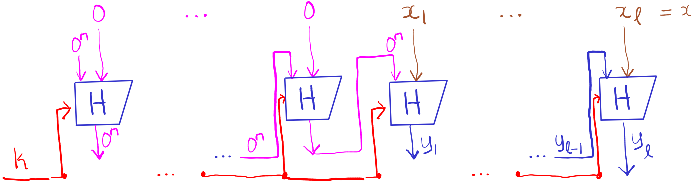


What If We Use Construction 1 for $\{0,1\}^*$?

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

Yes, consider H for which $\mathsf{H}(\mathsf{k}, \mathbf{0}^{n+1}) = \mathbf{0}^{n}$ (for all k)

- For H′ instantiated with above H: $\mathsf{H}'(\mathsf{k}, \mathbf{0}^{n}\boldsymbol{x}) = \mathsf{H}'(\mathsf{k}, \boldsymbol{x})$

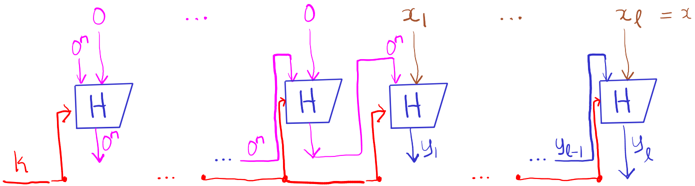


What If We Use Construction 1 for $\{0,1\}^*$?

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

Yes, consider H for which $\mathsf{H}(\mathsf{k}, \mathbf{0}^{n+1}) = \mathbf{0}^n$ (for all k)

- For H′ instantiated with above H: $\mathsf{H}'(\mathsf{k}, \mathbf{0}^n x) = \mathsf{H}'(\mathsf{k}, x)$



Exercise 5

1. *Find similar "length-extension" attack for Construction* 2
2. *Tweak Constructions* 1 *and* 2 *to obtain CRHF (i.e., for domain* $\{0,1\}^*$*)*
   - *Hint: add appropriate padding in the end*

Plan for Today's Lecture

1. Hash Functions
2. Compression Functions and Domain-Extension
3. How to Construct Compression Functions?

How to Construct Compression Functions in Practice?

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

How to Construct Compression Functions in Practice?

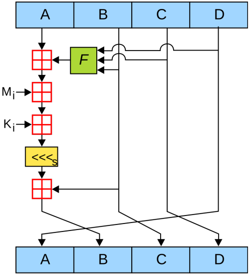
- *Unkeyed* compression function for fixed input (block) length/output length

How to Construct Compression Functions in Practice?

- *Unkeyed* compression function for fixed input (block) length/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
```

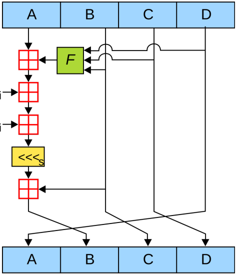


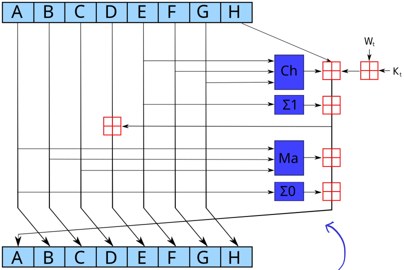
How to Construct Compression Functions in Practice?

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

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab58712467eab 4004583eb8fb7f89
55ad340609f4b302 83e4888325 71415a 085125e8f7cdc99f d91dbdf280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e2b487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b67080a80d1e c69821bcb6a88393 96f9652b6ff72a70
```

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab50712467eab 4004583eb8fb7f89
55ad340609f4b302 83e4888325f1415a 085125e8f7cdc99f d91dbd7280373c5b
d8823e3156348f5b ae6dacd436c919c6 dd53e23487da03fd 02396306d248cda0
e99f33420f577ee8 ce54b670802 80d1e 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

Image credit: Wikipedia (Surachit,kockmeyer)

How to Construct Compression Functions in Theory?

- Discrete-logarithm-based compression function $\{\mathsf{H} : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$:

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

How to Construct Compression Functions in Theory?

- Discrete-logarithm-based compression function $\{\mathsf{H} : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$:

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

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

How to Construct Compression Functions in Theory?

- Discrete-logarithm-based compression function $\{\mathsf{H} : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$:

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

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

- Lattice-based compression function $\{\mathsf{H} : \mathbb{Z}_p^{n\times m} \times \{0, 1\}^m \to \mathbb{Z}_p^n\}$ for $m \geq \lceil n \log(p) \rceil$:

$$\mathsf{H}(\bar{A}, \bar{x}) := \bar{A}\bar{x} \bmod p$$

How to Construct Compression Functions in Theory?

- Discrete-logarithm-based compression function $\{\mathsf{H} : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$:

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

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

- Lattice-based compression function $\{\mathsf{H} : \mathbb{Z}_p^{n\times m} \times \{0,1\}^m \to \mathbb{Z}_p^n\}$ for $m \geq \lceil n\log(p) \rceil$:

$$\mathsf{H}(\bar{A},\bar{x}) := \bar{A}\bar{x} \bmod p$$

  - Based on short integer solution (SIS) problem:
    - Input: $\bar{A} \leftarrow \mathbb{Z}_p^{n\times m}$, with $m \geq \lceil n\log(p) \rceil$
    - Solution: non-zero vector $\bar{x} \in \{0,\pm 1\}^m$ in $\bar{A}$'s kernel, i.e., $\bar{A}\bar{x} = \bar{0} \bmod p$

How to Construct Compression Functions in Theory?

- Discrete-logarithm-based compression function $\{\mathsf{H} : (\mathbb{Z}_p^\times)^2 \times \mathbb{Z}_p^2 \to \mathbb{Z}_p^\times\}$:

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

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

- Lattice-based compression function $\{\mathsf{H} : \mathbb{Z}_p^{n\times m} \times \{0, 1\}^m \to \mathbb{Z}_p^n\}$ for $m \geq \lceil n \log(p) \rceil$:

$$\mathsf{H}(\bar{A}, \bar{x}) := \bar{A}\bar{x} \bmod p$$

  - Based on short integer solution (SIS) problem:
    - Input: $\bar{A} \leftarrow \mathbb{Z}_p^{n\times m}$, with $m \geq \lceil n \log(p) \rceil$
    - Solution: non-zero vector $\bar{x} \in \{0, \pm 1\}^m$ in $\bar{A}$'s kernel, i.e., $\bar{A}\bar{x} = \bar{0} \bmod p$

? How to solve SIS given a collision $(\bar{x}, \bar{x}')$?

To Recap Today's Lecture

- Introduced a new primitive: collision-resistant hash function
  - Motivation: domain-extension for MAC/DS

To Recap Today's Lecture

- Introduced a new primitive: collision-resistant hash function
    - Motivation: domain-extension for MAC/DS
- Generic attack via birthday bound

To Recap Today's Lecture

- Introduced a new primitive: collision-resistant hash function
  - Motivation: domain-extension for MAC/DS
- Generic attack via birthday bound
- Domain extension for compression functions
  - Merkle-Damgård transform
  - Merkle trees

To Recap Today's Lecture

- Introduced a new primitive: collision-resistant hash function
    - Motivation: domain-extension for MAC/DS

- Generic attack via birthday bound
- Domain extension for compression functions
    - Merkle-Damgård transform
    - Merkle trees

- Some constructions:
    - Practical/unkeyed: SHA2, MD5
    - Theoretical/keyed: DLog- and SIS-based

Next Lecture

- New cryptographic primitive: *trap-door (one-way) permutation* (TDP)

Next Lecture

- New cryptographic primitive: *trap-door (one-way) permutation* (TDP)
  - OWP that is easy to invert given "trapdoor" information
  - Candidates
    - RSA TDP
    - Rabin TDP

Next Lecture

- New cryptographic primitive: *trap-door (one-way) permutation* (TDP)
  - OWP that is easy to invert given “trapdoor” information
  - Candidates
    - RSA TDP
    - Rabin TDP

- Efficient digital signatures in “random-oracle model”
  - (RSA) Full-domain hash

Next Lecture

- New cryptographic primitive: *trap-door (one-way) permutation* (TDP)
  - OWP that is easy to invert given "trapdoor" information
  - Candidates
    - RSA TDP
    - Rabin TDP
- Efficient digital signatures in "random-oracle model"
  - (RSA) Full-domain hash
- TDP $\rightarrow$ PKE
  - New constructions of PKE: RSA

References

1. As discussed in Lecture 7, hash functions were first studied in [WC81], but they considered pairwise-independence/universal hashing
2. Collision resistance, and other cryptographic properties of hash functions were studied later [Dam88, Dam90, NY89, Mer90] 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, 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, Heidelberg, 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, Heidelberg, 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, 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.