Hash Functions — Much Ado about Something

Orr Dunkelman

Département d'Informatique
École Normale supérieure

France Telecom Chaire

22nd of September 2008

Outline

1 Introducing Hash Functions
- The Definition War
- Collision Resistance
- (Second) Preimage Security
- Other Security Properties
- Universal One-Way Hash Functions

2 The Merkle-Damgård Construction

3 Why Merkle-Damgård Does Not Offer 2^n Second Preimage Resistance
- Using Fix Points
- Expandable Messages
- Herding Second Preimage Attacks

4 And then Came Prof. Wang

5 New Directions in Cryptographic Hash Functions
- Alternative Modes of Iteration
- New Design Methodologies
- Permutation-Based Hashing
- Proving the Security of the Hash Function
- SHA-3 — the AHS Competition

Outline

1 Introducing Hash Functions
- The Definition War
- Collision Resistance
- (Second) Preimage Security
- Other Security Properties
- Universal One-Way Hash Functions

2 The Merkle-Damgård Construction

3 Why Merkle-Damgård Does Not Offer 2^n Second Preimage Resistance
- Using Fix Points
- Expandable Messages
- Herding Second Preimage Attacks

4 And then Came Prof. Wang

5 New Directions in Cryptographic Hash Functions
- Alternative Modes of Iteration
- New Design Methodologies
- Permutation-Based Hashing
- Proving the Security of the Hash Function
- SHA-3 — the AHS Competition

What is a Hash Function?

[DH76] There is, however, a modification which eliminates the expansion problem when N is roughly a megabit or more. Let g be a one-way mapping from binary N-space to binary n-space where n is approximately 50. Take the N bit message m and operate on it with g to obtain the n bit vector m'. Then use the previous scheme to send m'...

What is a Hash Function? (cont.)

- (Cryptographic) Hash Functions are means to **securely** reduce a string *m* of arbitrarily length into a fixed-length digest.

What is a Hash Function? (cont.)

- (Cryptographic) Hash Functions are means to **securely** reduce a string m of arbitrarily length into a fixed-length digest.
- The main problem is the definition of securely.
- For signature schemes, two basic requirements exist:
  1. Second preimage resistance: given x, it is hard to find x' s.t. $h(x) = h(x')$.
  2. Collision resistance: it is hard to find x_1, x_2 s.t. $h(x_1) = h(x_2)$.

What is a Hash Function? (cont.)

- (Cryptographic) Hash Functions are means to **securely** reduce a string m of arbitrarily length into a fixed-length digest.
- The main problem is the definition of securely.
- For signature schemes, three basic requirements exist:
  1. Preimage resistance: given $y = h(x)$, it is hard to find x (or x', s.t., $h(x') = y$).
  2. Second preimage resistance: given x, it is hard to find x' s.t. $h(x) = h(x')$.
  3. Collision resistance: it is hard to find x_1, x_2 s.t. $h(x_1) = h(x_2)$.

What is a Hash Function? (cont.)

- Hash functions were quickly adopted in other places:
  - Password files (storing $h(pwd, salt)$ instead of pwd).
  - Bit commitments schemes (commit — $h(b, r)$, reveal — b, r).
  - Key derivation functions (take $k = h(g^{xy} \bmod p)$).
  - MACs (long story).
  - Tags of files (to detect changes).
  - Inside PRNGs.
  - Inside protocols (used in many “imaginative” ways).
  - ...

What Do We Want Out of Our Hash Functions?

If two people laid hold of a tallit and one says "it's all mine", and the other one says "it's all mine". What is done?

What Do We Want Out of Our Hash Functions?

If two people laid hold of a tallit and one says "it's all mine", and the other one says "it's all mine". What is done? Division in half.

(Mishna Bava Metsia 1:1)

What Do We Want Out of Our Hash Functions?

If two people laid hold of a tallit and one says "it's all mine", and the other one says "it's all mine". What is done? Division in half.

(Mishna Bava Metsia 1:1)

If two cryptographers defined the security notions of a hash function and one says "it's important to have pseudo-randomness", and the other one says "it's all in the everywhere second preimage resistance". What is done?

What Do We Want Out of Our Hash Functions?

As hash functions are widely used, various requirements are needed to ensure the security of construction based on hash functions:

- Collision resistance — signatures, bit commitment (for binding), MACs.
- Second preimage resistance — signatures.
- Preimage resistance — signatures (RSA, or other TD-OWP), password files, bit commitment (for hiding).
- Pseudo Random Functions — key derivation, MACs.
- Pseudo Random Oracle — protocols, PRNGs.

What Do We Really Want Out of Our Hash Functions?

We want the hash function to behave in a manner which would prevent any attacker from doing anything malicious to inputs to the hash function:

- One-wayness (no inversion).
- No collisions (up to the birthday bound).
- No second preimages.
- Outputs which are nicely distributed.
- ...

Therefore, the ideal hash function attaches for each possible message M a random value as $h(M)$. And voilá — a random oracle.

Collision Resistance of Hash Functions

Let us try to define when $h(\cdot)$ is collision resistant.

Collision Resistance of Hash Functions

Let us try to define when $h(\cdot)$ is collision resistant.

- It is computationally infeasible to find a collision.
  Formally: There is no efficient algorithm which given h finds collisions.

Collision Resistance of Hash Functions

Let us try to define when $h(\cdot)$ is collision resistant.

- It is computationally infeasible to find a collision. Formally: There is no efficient algorithm which given h finds collisions.
- $h(\cdot)$ is a hash function. Therefore, necessarily there exist a, b s.t. $h(a) = h(b)$. Consider the algorithm:

  print a, b.

Collision Resistance of Hash Functions

Let us try to define when $h(\cdot)$ is collision resistant.

- It is computationally infeasible to find a collision. Formally: There is no efficient algorithm which given h finds collisions.
- $h(\cdot)$ is a hash function. Therefore, necessarily there exist a, b s.t. $h(a) = h(b)$. Consider the algorithm:

  print a, b.

- What shall we do?

Collision Resistance of Hash Functions (cont.)

- Practical solution — a and b are unknown. For any specific function finding them takes $O(1)$ anyway. So who cares?

Collision Resistance of Hash Functions (cont.)

- Practical solution — *a* and *b* are unknown. For any specific function finding them takes $O(1)$ anyway. So who cares?
- Theoretical solution (I) — let us define a *family* of hash functions, and bundle the collision resistance of one of them to the collision resistance of the family.
- But how?

The Collision Resistance Game [RS04]

- Define a family of hash functions $H = \{h_1, h_2, \ldots\}$.
- The adversary is given a random k, and has to produce a collision for h_k.
- If $|H|$ is exponential, and the adversary has polynomial memory, this prevents him from storing (a_i, b_i) for all h_i.
- The adversary's advantage is then:

$$Adv_H^{Coll} = \Pr\Big[K \overset{\$}{\leftarrow} \mathcal{K}; (M, M') \overset{\$}{\leftarrow} A(K) : M \neq M' \wedge h_K(M) = h_K(M')\Big]$$

Collision Resistance of Hash Functions (cont.)

- Theoretical solution (II) — we do not know the value of a, b for a specific hash function. Thus, let us define a protocol Π, which uses a hash function $h(\cdot)$, such that we can show that every attacker A against Π yields an attack on $h(\cdot)$ [R05].

Collision Resistance of Hash Functions (cont.)

- Theoretical solution (II) — we do not know the value of a, b for a specific hash function. Thus, let us define a protocol Π, which uses a hash function $h(\cdot)$, such that we can show that every attacker A against Π yields an attack on $h(\cdot)$ [R05].
- But how can we construct Π? We should agree in advance on such a Π which is secure assuming $h(\cdot)$ is collision resistant.
- See the paper for some details which constructions we all *assume* to be OK if the underlying hash function is collision resistant.

Other Security Properties

- Second preimage — when the hash function is keyed the game is:
  - Choose K at random, choose M at random.
  - Give the adversary K, M, and ask for a second preimage M'. The formal advantage is

$$Adv_H^{Sec[m]} = \Pr\left[K \xleftarrow{\$} \mathcal{K}; M \xleftarrow{\$} \{0,1\}^m; M' \xleftarrow{\$} A(K,M) : M \neq M' \wedge h_K(M) = h_K(M')\right]$$

  - Note that the length of the message is embedded into definition to ensure that we are not biased towards (too) long messages, and to avoid problems arising from (too) small message spaces.

Other Security Properties (cont.)

- Maybe there are weak "keys"?

Other Security Properties (cont.)

- Maybe there are weak "keys"?
- Always second preimage — the key is chosen to be the "worst" from security point of view (rather than randomly). The advantage:

$$Adv_H^{aSec[m]} = \max_{K \in \mathcal{K}} \left\{ \Pr\left[M \xleftarrow{\$} \{0,1\}^m ; M' \xleftarrow{\$} A(K, M) : M \neq M' \wedge h_K(M) = h_K(M') \right] \right\}$$

Other Security Properties (cont.)

- Maybe there are weak "keys"?
- Always second preimage — the key is chosen to be the "worst" from security point of view (rather than randomly). The advantage:

$$Adv_H^{aSec[m]} = \max_{K \in \mathcal{K}} \left\{ \Pr \left[M \xleftarrow{\$} \{0,1\}^m; M' \xleftarrow{\$} A(K, M) : M \neq M' \wedge h_K(M) = h_K(M') \right] \right\}$$

- Everywhere second preimage — the message is chosen to be the "worst". The advantage:

$$Adv_H^{eSec[m]} = \max_{M \in \{0,1\}^m} \left\{ \Pr \left[K \xleftarrow{\$} \mathcal{K}; M' \xleftarrow{\$} A(K, M) : M \neq M' \wedge h_K(M) = h_K(M') \right] \right\}$$

Other Security Properties (cont.)

- Preimage resistance — pick K at random, a message M at random, give the adversary $h_K(M)$ and ask for a preimage.
- Always preimage resistance — take the worst K, repeat.
- Everywhere preimage resistance — take the worst possible hash value, repeat.

- When discussing preimage resistance, people might wish to take a random digest. This may lead to a "secure" case becoming insecure (i.e., changing Pre to be ePre).

Even More Security Definition

- Pseudorandom function — If the primitive is keyed, then any adversary cannot distinguish between an instance chosen by a random key, and a random function with the same parameters (input/output size). The advantage:

$$Adv_H^{prf} = \Pr\left[K \xleftarrow{\$} \mathcal{K}; A^{H(K,\cdot)} = 1\right] - \Pr\left[A^{h(\cdot)} = 1\right].$$

The main issue with hash functions is the way to key them (and the compression function). A good mode of iteration would preserve the "PRFness" of its compression function.

Even More Security Definition (cont.)

- Pseudorandom oracle — Does the hash function is indistinguishable from a random oracle?

Even More Security Definition (cont.)

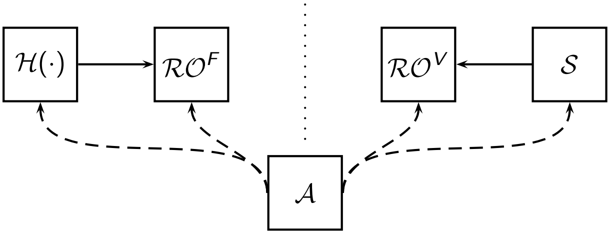
- Pseudorandom oracle — Does the hash function is indistinguishable from a random oracle?
- Of course it is easy to distinguish any hash function from a random oracle.

Even More Security Definition (cont.)

- Pseudorandom oracle — Does the hash function is indistinguishable from a random oracle?
- Of course it is easy to distinguish any hash function from a random oracle.
- But let us assume that we are given a random oracle as a compression function (FIL-RO). Is the hash function **now** is indistinguishable from a random oracle?
- The security game is very different.

Indistinguishability from Random Oracle

- There is the hash function which has access to a FIL-RO.
- There is a simulator which has access to a VIL-RO.
- The adversary can query either the hash and the FIL-RO, or the simulator and the VIL-RO.
- The advantage is the success of the adversary distinguishing between the two cases.



Universal One-Way Hash Functions

- Introduced by Naor & Yung in 1989 to overcome the collision-resistance "problem".
- Let H be a family of hash functions $H = \{h_1, h_2, \ldots, h_k\}$.
- H is UOWHF if for all x:

$$\Pr_{k \xleftarrow{\$} \mathcal{K}} [A(h_k, x) = y | h_k(x) = h_k(y) \wedge x \neq y]$$

- This property is the *Target Collision Resistance* which is the same as eSec.
- This means that for a specific h_i, it might be easy to find collisions, but not for all functions in H.

Outline

1 Introducing Hash Functions
- The Definition War
- Collision Resistance
- (Second) Preimage Security
- Other Security Properties
- Universal One-Way Hash Functions

2 The Merkle-Damgård Construction

3 Why Merkle-Damgård Does Not Offer 2^n Second Preimage Resistance
- Using Fix Points
- Expandable Messages
- Herding Second Preimage Attacks

4 And then Came Prof. Wang

5 New Directions in Cryptographic Hash Functions
- Alternative Modes of Iteration
- New Design Methodologies
- Permutation-Based Hashing
- Proving the Security of the Hash Function
- SHA-3 — the AHS Competition

The Merkle-Damgård Construction

- Presented by Merkle and Damgård independently as an answer to the following problem:
  - Given a compression function $f : \{0,1\}^{m_c} \times \{0,1\}^n \rightarrow \{0,1\}^{m_c}$, how would you generate a hash function $H_f : \{0,1\}^* \rightarrow \{0,1\}^m$.

The Merkle-Damgård Construction

- Presented by Merkle and Damgård independently as an answer to the following problem:
  - Given a compression function $f : \{0,1\}^{m_c} \times \{0,1\}^n \rightarrow \{0,1\}^{m_c}$, how would you generate a hash function $H_f : \{0,1\}^* \rightarrow \{0,1\}^m$.
- The solution is as follows:
  1. Pad the message M to a multiple of b (with 1, and many 0's as needed and the length of the message).
  2. Divided the padded message into l blocks $m_1 m_2 \ldots m_l$.
  3. Set $h_0 = IV$.
  4. For $i = 1$ to l, do $h_i = f(h_{i-1}, m_i)$.
  5. Output h_l (or some function of it).

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.
- Also, finding a second preimage in H_f means finding a collision in f.

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.

- Also, finding a second preimage in H_f means finding a collision in f.
- The same is true for finding a preimage (because you can use it to find a second preimage).

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.

- Also, finding a second preimage in H_f means finding a collision in f.
- The same is true for finding a preimage (because you can use it to find a second preimage).

To conclude, if f is collision resistant (i.e., it takes $O(2^{m_c/2})$ invocations to find a collision), then H_f is collision resistant and (second) preimage resistant with security level of $O(2^{m_c/2})$.

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.

- Also, finding a second preimage in H_f means finding a collision in f.
- The same is true for finding a preimage (because you can use it to find a second preimage).

To conclude, if f is collision resistant (i.e., it takes $O(2^{m_c/2})$ invocations to find a collision), then H_f is collision resistant and (second) preimage resistant with security level of $O(2^{m_c/2})$. But we want better security guarantees, (of $O(2^{m_c})$) for (second) preimage!

Outline

1 Introducing Hash Functions
- The Definition War
- Collision Resistance
- (Second) Preimage Security
- Other Security Properties
- Universal One-Way Hash Functions

2 The Merkle-Damgård Construction

3 Why Merkle-Damgård Does Not Offer 2^n Second Preimage Resistance
- Using Fix Points
- Expandable Messages
- Herding Second Preimage Attacks

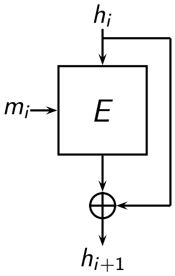
4 And then Came Prof. Wang

5 New Directions in Cryptographic Hash Functions
- Alternative Modes of Iteration
- New Design Methodologies
- Permutation-Based Hashing
- Proving the Security of the Hash Function
- SHA-3 — the AHS Competition

Second Preimage Attack on Merkle-Damgård

- If a fix-point can be easily found, a second preimage attack on a 2^l-block message takes — $\min\{O(2^{m_c-l}), O(2^{m_c/2})\}$ [D99]

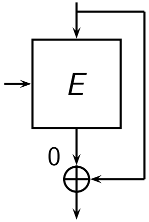
- Find $O(2^{m_c/2})$ fix-points denoted by $A = (h, m)$.
- Select $O(2^{m_c/2})$ single blocks and compute $B = (C_{MD}(IV, \tilde{m}), \tilde{m})$.
- Find a collision between A and B.
- Voilà — an **expandable message** $\tilde{m}||m^t$ for all t lead to the same chaining value h.



Second Preimage Attack on Merkle-Damgård

- If a fix-point can be easily found, a second preimage attack on a 2^l-block message takes — $\min\{O(2^{m_c-l}), O(2^{m_c/2})\}$ [D99]

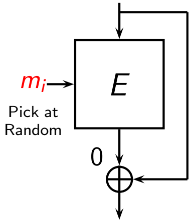
- Find $O(2^{m_c/2})$ fix-points denoted by $A = (h, m)$.
- Select $O(2^{m_c/2})$ single blocks and compute $B = (C_{MD}(IV, \tilde{m}), \tilde{m})$.
- Find a collision between A and B.
- Voilà — an **expandable message** $\tilde{m}||m^t$ for all t lead to the same chaining value h.



Second Preimage Attack on Merkle-Damgård

- If a fix-point can be easily found, a second preimage attack on a 2^l-block message takes — $\min\{O(2^{m_c-l}), O(2^{m_c/2})\}$ [D99]

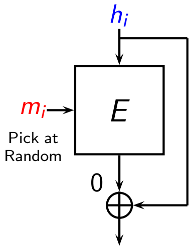
- Find $O(2^{m_c/2})$ fix-points denoted by $A = (h, m)$.
- Select $O(2^{m_c/2})$ single blocks and compute $B = (C_{MD}(IV, \tilde{m}), \tilde{m})$.
- Find a collision between A and B.
- Voilà — an **expandable message** $\tilde{m}||m^t$ for all t lead to the same chaining value h.



Second Preimage Attack on Merkle-Damgård

- If a fix-point can be easily found, a second preimage attack on a 2^l-block message takes — $\min\{O(2^{m_c-l}), O(2^{m_c/2})\}$ [D99]

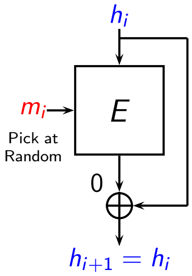
- Find $O(2^{m_c/2})$ fix-points denoted by $A = (h, m)$.
- Select $O(2^{m_c/2})$ single blocks and compute $B = (C_{MD}(IV, \tilde{m}), \tilde{m})$.
- Find a collision between A and B.
- Voilà — an **expandable message** $\tilde{m}||m^t$ for all t lead to the same chaining value h.



Second Preimage Attack on Merkle-Damgård

- If a fix-point can be easily found, a second preimage attack on a 2^l-block message takes — $\min\{O(2^{m_c-l}), O(2^{m_c/2})\}$ [D99]

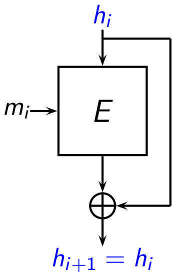
- Find $O(2^{m_c/2})$ fix-points denoted by $A = (h, m)$.
- Select $O(2^{m_c/2})$ single blocks and compute $B = (C_{MD}(IV, \tilde{m}), \tilde{m})$.
- Find a collision between A and B.
- Voilà — an **expandable message** $\tilde{m}||m^t$ for all t lead to the same chaining value h.



Second Preimage Attack on Merkle-Damgård (cont.)

- If a fix-point can be easily found, a second preimage attack on a 2^l-block message takes — $\min\{O(2^{m_c-l}), O(2^{m_c/2})\}$ [D99]

- Take the message M.
- Starting from h, try to find a message block x s.t., $f(h,x) = h_i$, for one of the chaining values of M.
- If succeeded, pad the message to the right length and obtain a second preimage.

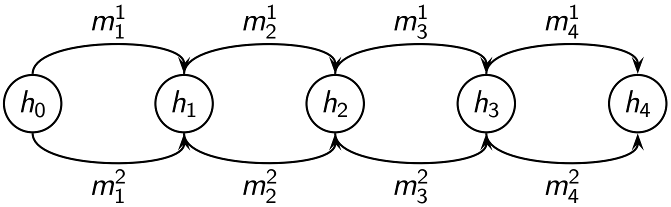


Multi-collision Attacks on Iterative Hashing

- Finding 2^t collisions in iterative hash function with chaining value length m_c, takes $O(t \cdot 2^{m_c/2})$ [J04]

Multi-collision Attacks on Iterative Hashing

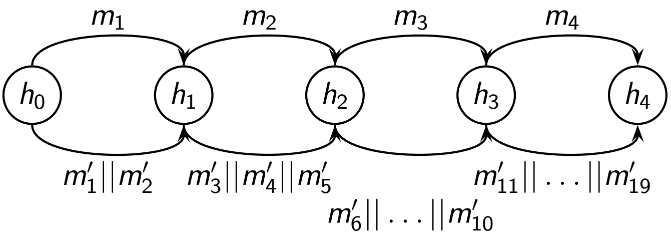
- Finding 2^t collisions in iterative hash function with chaining value length m_c, takes $O(t \cdot 2^{m_c/2})$ [J04]


$$\begin{tikzcd}
h_0 \arrow[r, bend left, "m_1^1"] \arrow[r, bend right, "m_1^2"'] & h_1 \arrow[r, bend left, "m_2^1"] \arrow[r, bend right, "m_2^2"'] & h_2 \arrow[r, bend left, "m_3^1"] \arrow[r, bend right, "m_3^2"'] & h_3 \arrow[r, bend left, "m_4^1"] \arrow[r, bend right, "m_4^2"'] & h_4
\end{tikzcd}$$

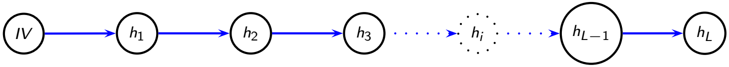
In an ideal hash function the time complexity should be $O(2^{\frac{2^t-1}{2^t} \cdot m_c})$.

Another Way to Generate Expandable Messages

- In [KS05] the expandable message is constructed as a multi-collision. In the first block between a message of one block and a message of two blocks, then between one block and three blocks, one and five, etc.

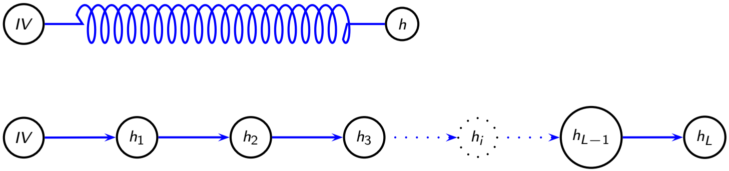


Expandable Message $\longrightarrow$ a Second Preimage Attack


$$\begin{tikzcd}
IV \arrow[r] & h_1 \arrow[r] & h_2 \arrow[r] & h_3 \arrow[r, dotted] & h_i \arrow[r, dotted] & h_{L-1} \arrow[r] & h_L
\end{tikzcd}$$

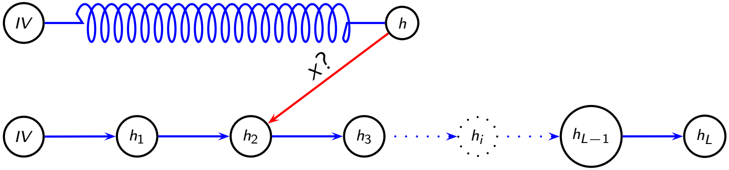
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers lengths from l to $2^l + l - 1$, whose output chaining value is h.



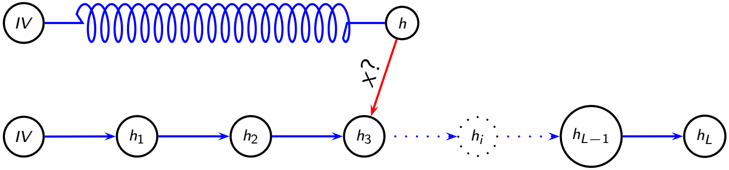
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers lengths from l to $2^l + l - 1$, whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).



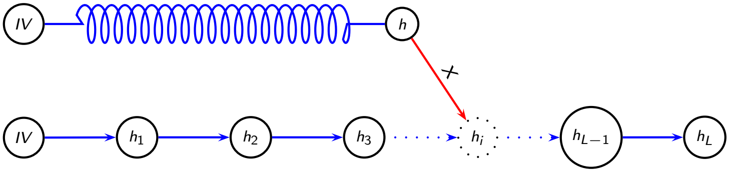
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers lengths from l to $2^l + l - 1$, whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).



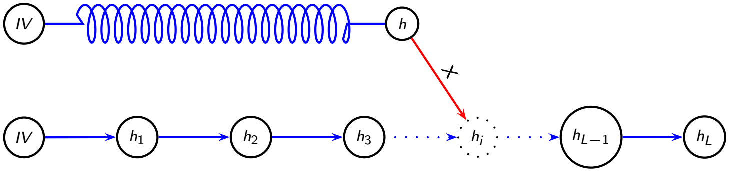
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers lengths from l to $2^l + l - 1$, whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).



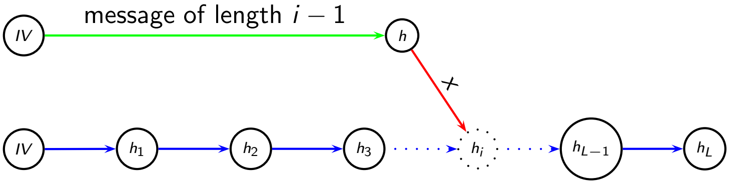
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers lengths from l to $2^l + l - 1$, whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).
- Once the "connection" step succeeds, fix the length using the precomputed expandable message.



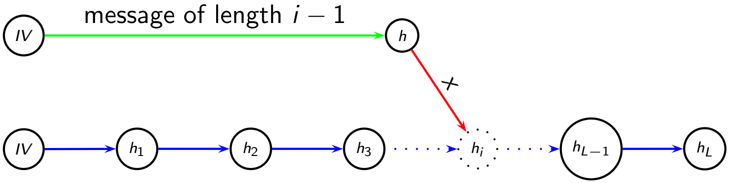
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers lengths from l to $2^l + l - 1$, whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).
- Once the "connection" step succeeds, fix the length using the precomputed expandable message.



Expandable Message $\longrightarrow$ a Second Preimage Attack

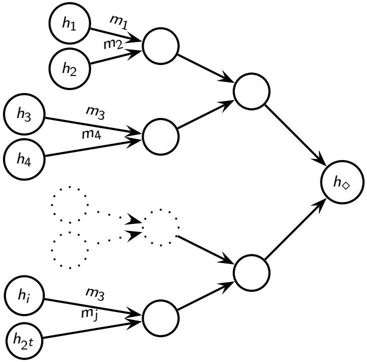
- Generate an expandable message that covers lengths from l to $2^l + l - 1$, whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).
- Once the "connection" step succeeds, fix the length using the precomputed expandable message.
- Time complexity: offline $O(l \cdot 2^{m_c/2} + 2^l)$. Online $O(2^{m_c - l})$.



The Herding Attack — Targeted Preimage Attack

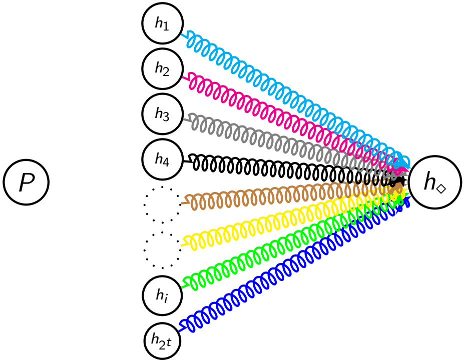
- Presented in [KK06] – the attacker fixes h_T, and given a challenge P, generates a message $m = P||S$, such that $h(m) = h_T$ in time $O(2^{m_c-t} + 2^{(m_c+t)/2})$.

Precomputation — generation of a **diamond structure**.



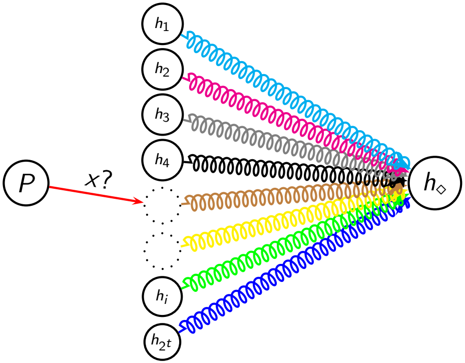
The Herding Attack — Targeted Preimage Attack

- The attacker tries 2^{m_c-t} possible x's until $H(P||x)$ is one of the precomputed h_i's in the diamond structure.
- Then, by concatenating the path in the diamond structure to $P||x$ it is possible to find a preimage of $h_\diamond$.



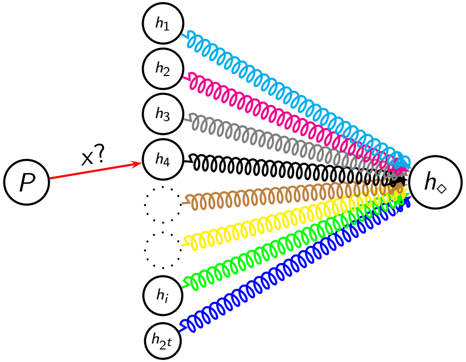
The Herding Attack — Targeted Preimage Attack

- The attacker tries $2^{m_c - t}$ possible x's until $H(P||x)$ is one of the precomputed h_i's in the diamond structure.
- Then, by concatenating the path in the diamond structure to $P||x$ it is possible to find a preimage of $h_\diamond$.



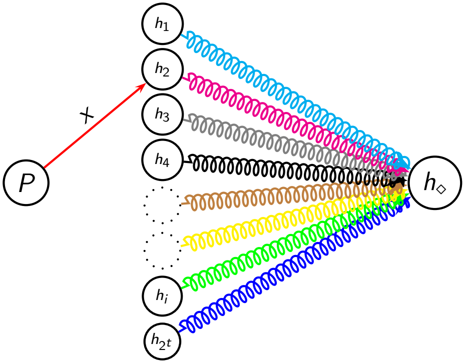
The Herding Attack — Targeted Preimage Attack

- The attacker tries $2^{m_c - t}$ possible x's until $H(P||x)$ is one of the precomputed h_i's in the diamond structure.
- Then, by concatenating the path in the diamond structure to $P||x$ it is possible to find a preimage of $h_\diamond$.



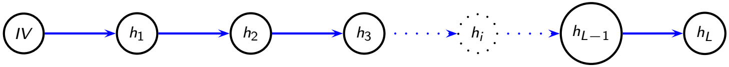
The Herding Attack — Targeted Preimage Attack

- The attacker tries $2^{m_c - t}$ possible x's until $H(P||x)$ is one of the precomputed h_i's in the diamond structure.
- Then, by concatenating the path in the diamond structure to $P||x$ it is possible to find a preimage of $h_\diamond$.



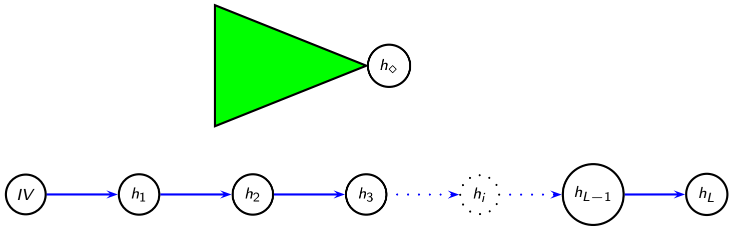
Second Preimage Attack Based on Herding

- Using the herding attack to allow short "patches" to messages $O(2^{m_c-t} + 2^{(m_c+t)/2} + 2^{m_c-l})$ [A+08].



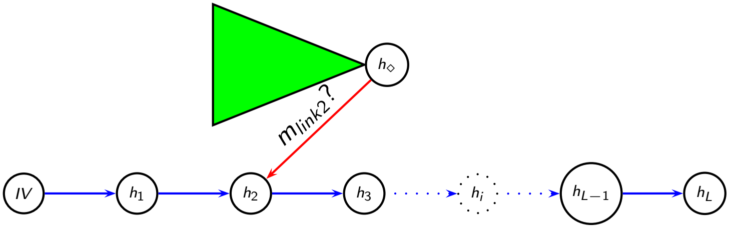
Second Preimage Attack Based on Herding

- Using the herding attack to allow short “patches” to messages $O(2^{m_c-t} + 2^{(m_c+t)/2} + 2^{m_c-l})$ [A+08].
- Generate a diamond structure.



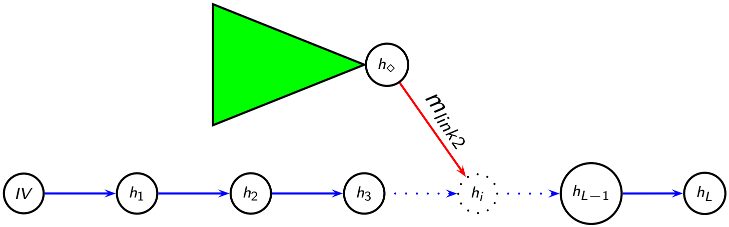
Second Preimage Attack Based on Herding

- Using the herding attack to allow short "patches" to messages $O(2^{m_c-t} + 2^{(m_c+t)/2} + 2^{m_c-l})$ [A+08].
- Generate a diamond structure.
- Try random m_{link2}, until $f(h_\diamond, m_{link2}) = h_i$, for some h_i obtained during the computation of $h(M)$.



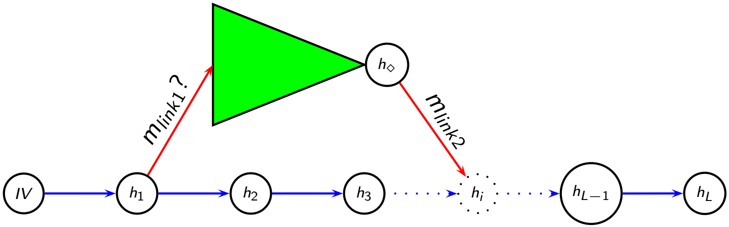
Second Preimage Attack Based on Herding

- Using the herding attack to allow short "patches" to messages $O(2^{m_c-t} + 2^{(m_c+t)/2} + 2^{m_c-l})$ [A+08].
- Generate a diamond structure.
- Try random m_{link2}, until $f(h_\diamond, m_{link2}) = h_i$, for some h_i obtained during the computation of $h(M)$.



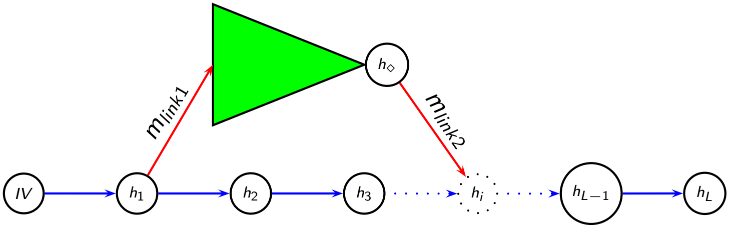
Second Preimage Attack Based on Herding

- Using the herding attack to allow short "patches" to messages $O(2^{m_c-t} + 2^{(m_c+t)/2} + 2^{m_c-l})$ [A+08].
- Generate a diamond structure.
- Try random m_{link2}, until $f(h_\diamond, m_{link2}) = h_i$, for some h_i obtained during the computation of $h(M)$.
- So starting from h_{i-t-2}, try random m_{link1} until one of the entry points of the diamond structure are found.



Second Preimage Attack Based on Herding

- Using the herding attack to allow short "patches" to messages $O(2^{m_c-t} + 2^{(m_c+t)/2} + 2^{m_c-l})$ [A+08].
- Generate a diamond structure.
- Try random m_{link2}, until $f(h_\diamond, m_{link2}) = h_i$, for some h_i obtained during the computation of $h(M)$.
- So starting from h_{i-t-2}, try random m_{link1} until one of the entry points of the diamond structure are found.



Outline

1 Introducing Hash Functions
- The Definition War
- Collision Resistance
- (Second) Preimage Security
- Other Security Properties
- Universal One-Way Hash Functions

2 The Merkle-Damgård Construction

3 Why Merkle-Damgård Does Not Offer 2^n Second Preimage Resistance
- Using Fix Points
- Expandable Messages
- Herding Second Preimage Attacks

4 And then Came Prof. Wang

5 New Directions in Cryptographic Hash Functions
- Alternative Modes of Iteration
- New Design Methodologies
- Permutation-Based Hashing
- Proving the Security of the Hash Function
- SHA-3 — the AHS Competition

The MD/SHA-Family

The MD/SHA family is composed of many hash functions with similar design criteria:

- Davies-Meyer transformation of a block cipher into a compression function.
- Merkle-Damgård hash function.
- Simple round functions (with little nonlinearity).
- The nonlinearity is "introduced" bit-by-bit (AND, MAJ operations) and using addition modulo 2^{32}.
- The message expansion (key schedule) is linear (either repetition, or through an LFSR).
- Very software-friendly (not so bad on hardware as well).
- Message block: 512-bit; Digest size: 128-bit (MD4/5), 160-bit (SHA).

History of the World (part I)

- MD4 introduced in 1990 by Rivest. Collision attack — Dobbertin (1996) (attack on the last two steps — den Boer & Bosselaers, 1991).
- MD5 introduced in 1991 by Rivest. Some non-randomness problems by Berson (1992) and a free-start collision by den Boer & Bosselaers (1993).
- SHA-0 introduced in 1995 by NIST. Larger digest size, message is expanded using an LFSR. A collision attack by Chabaud & Joux (1998).
- SHA-1 followed immediately after SHA-0.
- And the land had rest eight years . . .

History of the World (part II)

- Crypto 2004: Near collisions of SHA0 (Biham & Chen).
- Rump session: Wang presents collision attacks against MD4.
- Eurocrypt 2005: Wang et al. publish the MD4 paper, finding collisions in MD4, RIPEMD, MD5. Biham et al. find collisions in SHA-0, reduced round SHA-1.
- Crypto 2005: Wang, Yu, Yin: Better SHA-0 collisions, SHA-1 collision attack.
- NIST 2005: Wang announces better collision attack on SHA-1.
- Asiacrypt 2006: De Canniére & Rechberger, improved collision attack on SHA-1.
- August 2007: Graz people start their SHA-1 BOINC project.
- FSE 2008: Preimage attack on MD4 (Leurent).
- Crypto 2008: Preimage attacks on reduced SHA-0 and SHA-1 (De Canniére & Rechberger).

History of the World (part III)

- MD4/MD5 collisions start to be applied to NMAC/HMAC.
- In the related-key model NMAC-MD4/-MD5 (Contini & Yin 2006, Fouque, Leurent & Nguyen 2007, ...) can be attacked.
- HMAC-MD4 is also broken (Wang, Ohta, & Kunihiro 2008).
- Things start to get complicated...

History of the World (part IV)

- Random collisions can be source of trouble for some file formats (Daum & Lucks 2005, later extended by Gebhardt, Illies, & Schindler 2005).
- Colliding X.509 certificates with same name, different keys (Lenstra & de-Weger 2005).
- Technique was improved to generate colliding X.509 certificates for different names (Stevens, Lenstra & de-Weger 2007).

Outline

1 Introducing Hash Functions
- The Definition War
- Collision Resistance
- (Second) Preimage Security
- Other Security Properties
- Universal One-Way Hash Functions

2 The Merkle-Damgård Construction

3 Why Merkle-Damgård Does Not Offer 2^n Second Preimage Resistance
- Using Fix Points
- Expandable Messages
- Herding Second Preimage Attacks

4 And then Came Prof. Wang

5 New Directions in Cryptographic Hash Functions
- Alternative Modes of Iteration
- New Design Methodologies
- Permutation-Based Hashing
- Proving the Security of the Hash Function
- SHA-3 — the AHS Competition

So What's Next?

Open issues:

- Mode of iteration that preserves second preimage resistance.
- Better compression functions.
- Information theoretic approach?
- Proofs! We want proofs!
- The next generation hash function — SHA-3.

Randomized Hashing

- Introduced by Halevi & Krawczyk to solve the issue of a random collision collapsing the entire security of the hash function.
- The main idea: Instead of hashing m, one chooses a random value r, and hashes $h(m \oplus r||r||\dots||r)$ or $h_r(m \oplus r||r||\dots||r)$.
- The security is enhanced Target Collision Resistant (eTCR) which defines the advantage in the game:
  1. The adversary commits to a message M.
  2. The adversary is given a key k (chosen at random).
  3. The adversary has to find M', k' s.t., $h_k(M) = h_{k'}(M')$.

Dithering Sequences

- Suggested by Rivest as a solution to expandable message issues.
- The compression function is called every time with a dither sequence.
- One proposal uses a dither sequence over 4 characters which has very nice properties.
- Practical proposal: take the nice sequence, and embed it into a more efficient sequence. Use 16-bit dither sequence:
  - First bit is 0, but for the last block (1).
  - Next two bits are encoding of the “nice sequence”.
  - Next thirteen bits are a counter. Once the counter overflows, change the character in the “nice sequence”.

Dithering Sequences (cont.)

- While the security of the dithered hash is indeed better than of plain Merkle-Damgård it is not optimal.
- The second preimage attack based on herding is still applicable (even though there is an “added” security of 2^{15}).

Enveloped Merkle-Damgård

- The Enveloped Merkle-Damgård [BR06] is a transformation of a “good” compression function into a hash function which preserves the following three properties:
  1. Collision resistance.
  2. Pseudo-random oracle behavior.
  3. Pseudo-random function behavior.
- The mode is similar to Merkle-Damgård, up to the last block, where in the last block:
  1. The chaining value is fixed to a second IV value.
  2. The previous chaining value (the output of the one before last compression function call) is concatenated to the message block (the last message block is shorter than the previous ones).

ROX

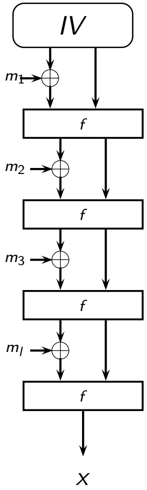
- The ROX transformation [A+07], is a way to preserve the compression function's properties (Coll, (a/e)Sec, (a/e)Pre) in the hash function.
- The proposal follows Shoup's hash (a UOWHF [S01]):
  - Before each compression function call, the chaining value is XORed with a masks $\mu_{\nu(i)}$ when hashing the i'th block, where $\nu(i) = \max_j\{2^j | i\}$.
  - The padding is derived using a random oracle query.
  - The masks are also derived using a random oracle queries.
  - The random oracle queries are "keyed" by a prefix of the message.

Widepipe [L05]

- We know to prove that the (second) preimage resistance is as secure as collision resistance.
- Internal collisions cause many problems.
- Solution: increase the chaining value.
- For example, with chaining value of length twice the digest size.
- If the compression function is good (as well as the last block which compresses the double chaining value), then we have a secure hash function.

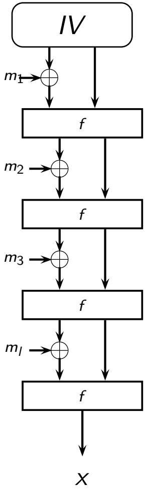
Sponges

- A theoretical framework for constructions like PANAMA.
- The internal state is relatively large (e.g., 59 l-bit words in PANAMA's successor, RadioGATÚN).



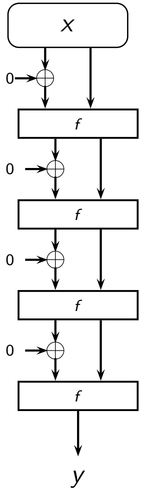
Sponges

- A theoretical framework for constructions like PANAMA.
- The internal state is relatively large (e.g., 59 l-bit words in PANAMA's successor, RadioGATÚN).
- During message processing, each round, a small message block is processed, and the new internal state is computed.



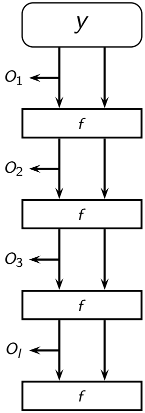
Sponges

- A theoretical framework for constructions like PANAMA.
- The internal state is relatively large (e.g., 59 l-bit words in PANAMA's successor, RadioGATÚN).
- After all the message blocks affect the internal state, some blank rounds are run (i.e., processing an all-zero block).



Sponges

- A theoretical framework for constructions like PANAMA.
- The internal state is relatively large (e.g., 59 l-bit words in PANAMA's successor, RadioGATÚN).
- For output, the sponge is squeezed, each round some of its internal state leaks as an output.



Sponges (cont.)

- If the update function is random (permutation/function) than the sponge is indifferentable from a random oracle [B+08].

Sponges (cont.)

- If the update function is random (permutation/function) than the sponge is indifferentable from a random oracle [B+08].
- This requires a "strong" f which diffuses and confuses the entire (large) internal state.
- Such functions are very resource consuming, and the actual designs have a relatively "light" f.
- PANAMA [DC98] was broken using attacks which uses the slow "diffusion" & "confusion" [R01,DvA07].
- Grindhal [KRT07], was broken using the quick diffusion and the weak confusion [P07].
- Only "surviving" candidate — RadioGATÚN (and to some extent Grindhal 2).

HAsh Iterative FrAmework (HAIFA)

- Major features:
  - Supports salts (defines families of hash functions).
  - Supports variable output size.
  - Offers as good security properties as can be.
  - Strong backward compatibility.
  - All suggested modes can be realized as HAIFA.

(This a joint work with Eli Biham)

The HAIFA Compression Function

- Accepts as inputs:
  - A chaining value (of size m_c)
  - A message block (of size n)
  - A bit counter (of size b)
  - A salt (of size s)

$$f : \{0,1\}^{m_c} \times \{0,1\}^n \times \{0,1\}^b \times \{0,1\}^s \rightarrow \{0,1\}^{m_c}.$$

The HAIFA Initialization

- Let m be the target digest size.
- Let IV be a general initial value.
- $IV_m = C(IV, m, 0, 0)$.

The HAIFA Computation

- Take M, the message, and pad it:
  - Pad a single bit of 1.
  - Pad as many 0 bits as needed such that the length of the padded message (with the 1 bit and the 0's) is congruent modulo n to $(n-(t+r))$.
  - Pad the message length encoded in t bits.
  - **Pad the digest size encoded in r bits.**
- Set $h_0 = IV_m$
- For $i = 1, 2, \ldots, l$ compute $h_i = C(h_{i-1}, M_i, \#bits, salt)$.
- Truncate h_l to m bits.

Permutation-Based Hashing

- Standard compression functions are a transformation of a block cipher into a hash function (following the PGV "approved" list).
- In all of them, there is a need to re-key the block cipher.
- But block ciphers are efficient when the key is fixed.

Permutation-Based Hashing

- A compression function from mn bits to rn bits using k calls to permutations of n-bit to n-bit, has a maximal information theoretic security of

  $$2^{n[1-(m-0.5r)/k]} \quad / \quad 2^{n[1-(m-r)/k]}$$

  queries for collision resistance/preimage resistance [BR08].
- Note that this results discuss the number of **queries** to the permutation.
- This means that if the compression function uses 8-bit S-boxes and compresses 768 bits to 256 bits, it has security of $2^{8(1-80/k)}$ or $2^{8(1-64/k)}$ queries.
- Finding the actual collisions/preimages are very time consuming.

Proving the Security of the Compression Function

- Very Smooth Hash [CLS06] is a provable secure hash function.

Proving the Security of the Compression Function

- Very Smooth Hash [CLS06] is a provable secure hash function.
- Provable collision resistance that is.

Proving the Security of the Compression Function

- Very Smooth Hash [CLS06] is a provable secure hash function.
- Provable collision resistance that is.
- Finding a collision means a factorization of a large number (following prior works [D87]).
- The construction:
  1. Let n be a large number (whose factorization is unknown).
  2. Let p_i be the ith prime number, and let k be the maximal for which $\sum_{i=1}^{k} p_i < n$.
  3. To compress a message block (of length k) x_i, and a chaining value h_i, compute

$$h_{i+1} = h_i^2 \times \prod_{j=1}^{k} p_j^{x_{i,j}}$$

Some More on VHS

- VHS is very slow (even though it is way faster than previous similar constructions) — about 8.8 Mbit/sec on 1 GHz machine (about 910 cpb).

Some More on VHS

- VHS is very slow (even though it is way faster than previous similar constructions) — about 8.8 Mbit/sec on 1 GHz machine (about 910 cpb).
- Also VHS is not a hash function.

Some More on VHS

- VHS is very slow (even though it is way faster than previous similar constructions) — about 8.8 Mbit/sec on 1 GHz machine (about 910 cpb).
- Also VHS is not a hash function.
- Knowing the factorization of *n* enables preimage attacks.

Some More on VHS

- VHS is very slow (even though it is way faster than previous similar constructions) — about 8.8 Mbit/sec on 1 GHz machine (about 910 cpb).
- Also VHS is not a hash function.
- Knowing the factorization of n enables preimage attacks.
- And it has multiplicative problems. Let x, y, z be three strings such that $z = 0$, and $x \wedge y = z$, then

$$H(z)H(x \vee y) = H(x)H(y) \bmod n$$

- And when the output is truncated, collisions are easier to find [S06].

Other Provable Compression Functions

- Compression function proposals were also suggested based on syndrome-decoding of a random code ([AFS05]).
- Due to speed, it was suggested in [FGS07] to change the matrix of the code to a quasi-cyclic, leading a more efficient hashing.

Other Provable Compression Functions

- Compression function proposals were also suggested based on syndrome-decoding of a random code ([AFS05]).
- Due to speed, it was suggested in [FGS07] to change the matrix of the code to a quasi-cyclic, leading a more efficient hashing.
- The change led to an attack ([FL08]).

Other Provable Compression Functions

- Compression function proposals were also suggested based on syndrome-decoding of a random code ([AFS05]).
- Due to speed, it was suggested in [FGS07] to change the matrix of the code to a quasi-cyclic, leading a more efficient hashing.
- The change led to an attack ([FL08]).
- Lattices were also suggested as a building block [GGH96].
- Due to attack algorithms on lattices, it requires large parameters.
- In LASH, the construction was tweaked a bit to allow much faster implementations [B+06].

Other Provable Compression Functions

- Compression function proposals were also suggested based on syndrome-decoding of a random code ([AFS05]).
- Due to speed, it was suggested in [FGS07] to change the matrix of the code to a quasi-cyclic, leading a more efficient hashing.
- The change led to an attack ([FL08]).
- Lattices were also suggested as a building block [GGH96].
- Due to attack algorithms on lattices, it requires large parameters.
- In LASH, the construction was tweaked a bit to allow much faster implementations [B+06].
- Of course, this led to attacks (collision and preimage) [S+08].

Other Provable Compression Functions

- An interesting approach is the DAKOTA construction [DKT08] also inspired by [D87].
- Let $f : \{0,1\}^m \longrightarrow QR(n)$, where n is a number whose factorization is unknown.
- To compress the input (m_i, h_i):

$$h_{i+1} = f(m_i) \cdot h_i^2,$$

  where $h_0 \in QR(n)$.
- This is secure as long as finding $(b, y), (b', y')$ s.t. $f(b) f^{-1}(b') = y' y^{-1} \bmod n$ is hard.

Other Provable Compression Functions

- An interesting approach is the DAKOTA construction [DKT08] also inspired by [D87].
- Let $f : \{0,1\}^m \longrightarrow QR(n)$, where n is a number whose factorization is unknown.
- To compress the input (m_i, h_i):

$$h_{i+1} = f(m_i) \cdot h_i^2,$$

  where $h_0 \in QR(n)$.
- This is secure as long as finding $(b, y), (b', y')$ s.t. $f(b)f^{-1}(b') = y'y^{-1} \bmod n$ is hard.
- It is also possible to use a random $f(\cdot)$:

$$h_{i+1} = (f(m_i) \cdot h_i)^2.$$

Other Provable Compression Functions

- An interesting approach is the DAKOTA construction [DKT08] also inspired by [D87].
- Let $f : \{0,1\}^m \longrightarrow QR(n)$, where n is a number whose factorization is unknown.
- To compress the input (m_i, h_i):

  $$h_{i+1} = f(m_i) \cdot h_i^2,$$

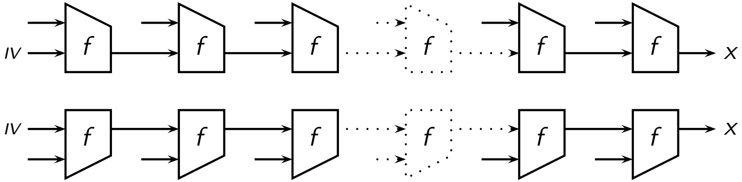
  where $h_0 \in QR(n)$.
- This is secure as long as finding $(b, y), (b', y')$ s.t. $f(b)f^{-1}(b') = y'y^{-1} \bmod n$ is hard.
- It is also possible to use a random $f(\cdot)$:

  $$h_{i+1} = (f(m_i) \cdot h_i)^2.$$

- If the assumption holds, then the security proof holds.

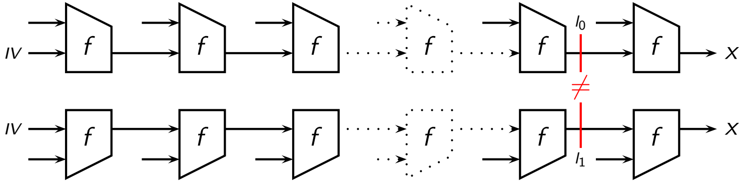
Collision Resistance of Merkle-Damgård

- Assume that the compression function is optimal.
- Let assume that there is an adversary A which can find collisions in $MD^f(\cdot)$ efficiently, and we transform it into A' which finds collisions in $f(\cdot)$.
- Examine the collision produced by A. If the messages are not of the same length, then, necessarily there is a pair of inputs $(h, m) \neq (h', m')$ s.t. $f(h, m) = f(h', m')$.



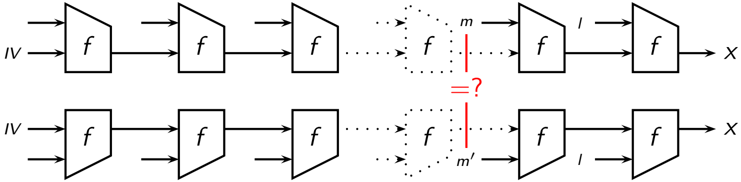
Collision Resistance of Merkle-Damgård

- Assume that the compression function is optimal.
- Let assume that there is an adversary A which can find collisions in $MD^f(\cdot)$ efficiently, and we transform it into A' which finds collisions in $f(\cdot)$.
- Examine the collision produced by A. If the messages are not of the same length, then, necessarily there is a pair of inputs $(h, m) \neq (h', m')$ s.t. $f(h, m) = f(h', m')$.



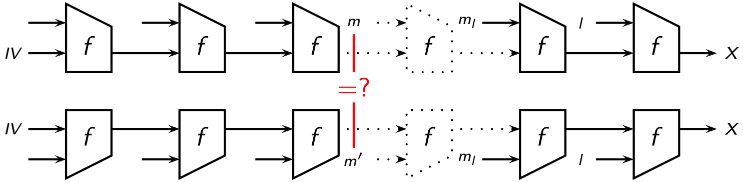
Collision Resistance of Merkle-Damgård

- Assume that the compression function is optimal.
- Let assume that there is an adversary A which can find collisions in $MD^f(\cdot)$ efficiently, and we transform it into A' which finds collisions in $f(\cdot)$.
- If the messages are of the same length, start from the last block and go backwards, until you find the block which differs. And voilá — a collision in $f(\cdot)$.



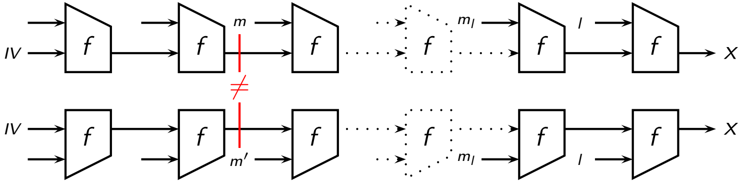
Collision Resistance of Merkle-Damgård

- Assume that the compression function is optimal.
- Let assume that there is an adversary A which can find collisions in $MD^f(\cdot)$ efficiently, and we transform it into A' which finds collisions in $f(\cdot)$.
- If the messages are of the same length, start from the last block and go backwards, until you find the block which differs. And voilá — a collision in $f(\cdot)$.



Collision Resistance of Merkle-Damgård

- Assume that the compression function is optimal.
- Let assume that there is an adversary A which can find collisions in $MD^f(\cdot)$ efficiently, and we transform it into A' which finds collisions in $f(\cdot)$.
- If the messages are of the same length, start from the last block and go backwards, until you find the block which differs. And voilá — a collision in $f(\cdot)$.



Second Preimage Resistance of Merkle-Damgård

- Let A be a second preimage adversary for $MD^f(\cdot)$.
- A accepts M and returns M' s.t. $MD^f(M) = MD^f(M')$.
- There is no known method to transform it into a second preimage adversary for $f(\cdot)\ldots$.

The eSec Game

- An adversary $A^{eSec}_{h_f(\cdot)}$ picks a message M, gets a key K, and outputs M' s.t., $h_K(M) = h_K(M')$.
- Our adversary $A^{eSec}_{f(\cdot)}$ has to pick a message block m_i as input to $f(\cdot)$.
- Hence, it is required to embed the short challenge (the key $A^{eSec}_{f(\cdot)}$ gets) in the long challenge.
- The best known way to do so (when it is known), is to guess where the $A^{eSec}_{h_f(\cdot)}$ is going to generate a second preimage.

The eSec Game

- An adversary $A^{eSec}_{h_f(\cdot)}$ picks a message M, gets a key K, and outputs M' s.t., $h_K(M) = h_K(M')$.
- Our adversary $A^{eSec}_{f(\cdot)}$ has to pick a message block m_i as input to $f(\cdot)$.
- Hence, it is required to embed the short challenge (the key $A^{eSec}_{f(\cdot)}$ gets) in the long challenge.
- The best known way to do so (when it is known), is to guess where the $A^{eSec}_{h_f(\cdot)}$ is going to generate a second preimage.
- This means, that if the second preimage resistance of $f(\cdot)$ is at most 2^n, the (provable) second preimage resistance of $h_f(\cdot)$ is at most $2^n/l$ for an l-block messages.

Achieving the Best

- To overcome the lose of $1/l$, it is needed to be able to embed the query in several places.
- But the adversary $A^{eSec}_{h_f(\cdot)}$ can easily notice that we are embedding the same query in several places and refuse answering.
- So for the proof to work the adversary $A^{eSec}_{f(\cdot)}$ has to embed its query in several places.
- Very very tricky . . .

Achieving the Best (cont.)

- The problem is in the proof technique.
- That means that you can still have second preimage resistance of 2^n, even though you will not be able to prove it.

Achieving the Best (cont.)

- The problem is in the proof technique.
- That means that you can still have second preimage resistance of 2^n, even though you will not be able to prove it.
- The second preimage attacks work because each invocation of the compression function is the "same".

Achieving the Best (cont.)

- The problem is in the proof technique.
- That means that you can still have second preimage resistance of 2^n, even though you will not be able to prove it.
- The second preimage attacks work because each invocation of the compression function is the "same".
- In HAIFA, for example, there is very strong reasons to believe that it has second preimage resistance of 2^n.

SHA-3 — The Next (Next) Generation

- A response of NIST to all the advances in the cryptanalysis of SHA-1.
- The Advanced Hash Standard (AHS) competition is all about finding a secure replacement for the SHA-2 family.

SHA-3 — The Next (Next) Generation

- A response of NIST to all the advances in the cryptanalysis of SHA-1.
- The Advanced Hash Standard (AHS) competition is all about finding a secure replacement for the SHA-2 family.
- But SHA-2 family has not been broken (yet)!

SHA-3 — The Next (Next) Generation

- A response of NIST to all the advances in the cryptanalysis of SHA-1.
- The Advanced Hash Standard (AHS) competition is all about finding a secure replacement for the SHA-2 family.
- But SHA-2 family has not been broken (yet)!
- SHA-2 family has some security issues due to the Merkle-Damgård construction (second preimage attacks).
- SHA-256/-224 is much slower than SHA-1 (29 cpb vs. 10 cpb on a 32-bit machine).

SHA-3 — The Next (Next) Generation (cont.)

- NIST expects many candidates to be submitted.

SHA-3 — The Next (Next) Generation (cont.)

- NIST expects many candidates to be submitted.
- So does everybody else.

SHA-3 — The Next (Next) Generation (cont.)

- NIST expects many candidates to be submitted.
- So does everybody else.
- Open issues:
  1. Mode of iteration (Merkle-Damgård vs. HAIFA vs. widepipe vs. tree hashes vs. provable modes vs. weird constructions).
  2. Good security.
  3. Good performance:
     - On a 32-bit platform? 64-bit platform? 8-bit machines?
     - ASIC/FPGA? Other hardware models?
     - Single core? Multiple cores? Multiple CPUs?
  4. Side channel resistance?

SHA-3 — My Guesses

Things which will label this entire thing as a waste of resources:

- Selecting something which offers less security than "optimal".
- Selecting something much slower than SHA.
- If performance requirements much larger than SHA.

SHA-3 — My Guesses (Mode of Iteration)

- Merkle-Damgård— Not the best security achievable.
- Sponges — too new, not such a good track-record.
- Widepipe or HAIFA — probably the winning mode.
- Other provable modes — not so likely.

SHA-3 — My Guesses (Compression Functions)

- Performance on 32-bit machine up to 35–40 cpb (33% slowdown with respect to SHA256).
- Performance on 64-bit machine up to 25–30 cpb for 256-bit digests. Up to 20 cpb for 512-bit digests.
- Implementable on 8-bit platforms.
- ASIC speeds that can reach 5 Gbps.
- Possible to implement with “restricted” memory.
- RFID **will not** play any role.
- Good differential and linear properties.
- Known and well-understood components to be preferred over new and/or not fully understood (e.g., XOR vs. addition).

Questions?

Thank you for your attention!

and Smakelijk!