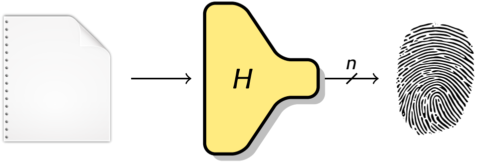
Generic Attacks against MAC Algorithms and Hash Functions

Gaëtan Leurent

Inria, France

IACR-CROSSING School on Combinatorial Techniques in Cryptography

Hash functions



- Hash function: public function $\{0,1\}^* \to \{0,1\}^n$
  - Maps arbitrary-length message to fixed-length hash
- Hash function should behave like a random function
  - Hard to find collisions, preimages
  - Hash can be used as fingerprint, identifier
  - Used to instantiate the Random Oracle Model
- Used in many different contexts
  - Signature: hash-and-sign
  - MAC: hash-and-PRF, HMAC
  - Commitments, proof-of-work, ...

Concrete security goals

Preimage attack

Given F and $\overline{H}$, find M s.t. $F(M) = \overline{H}$.
Ideal security: 2^n.

Second-preimage attack

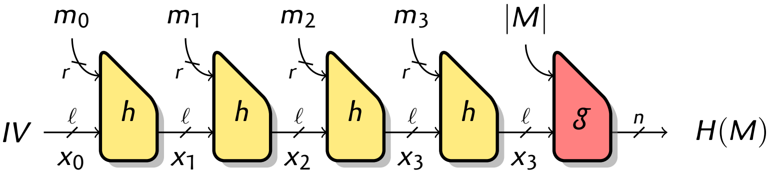
Given F and M_1, find $M_2 \neq M_1$ s.t. $F(M_1) = F(M_2)$.
Ideal security: 2^n.

Collision attack

Given F, find $M_1 \neq M_2$ s.t. $F(M_1) = F(M_2)$.
Ideal security: $2^{n/2}$.

- Defined for a family of hash functions

Iterated hash function

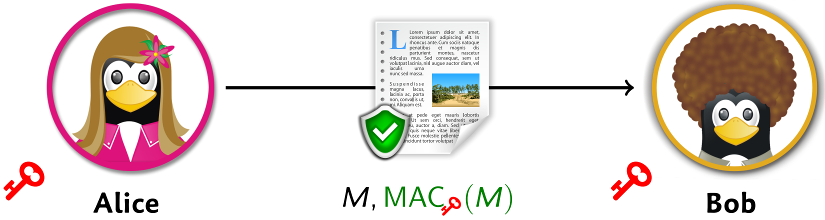


- ℓ-bit state (usually $\ell \geq n$, often $\ell = n$)
- Split M into blocks $M = m_0 \parallel m_1 \parallel \dots$
- Compression function $h : \{0,1\}^\ell \times \{0,1\}^r \to \{0,1\}^\ell$
- Finalization function $g : \{0,1\}^\ell \times \{0,1\}^s \to \{0,1\}^n$
  - Finalization function can use message length (MD strengthening)

Notations

- Compression function h
- Iterated compression function h^*: $h^*(x, m_1 \parallel m_2 \parallel m_3) = h(h(h(x, m_1), m_2), m_3)$
- Hash function H $\qquad H(M) = g(h^*(IV, M), |M|)$

Message Authentication Codes (MAC)



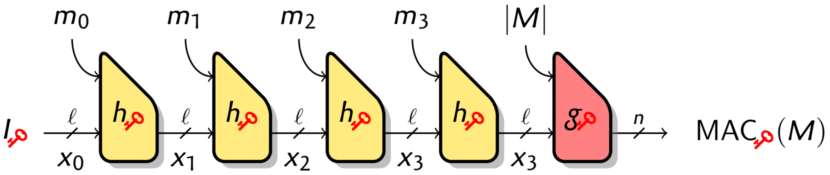
- MAC: keyed function $\{0,1\}^* \to \{0,1\}^n$
  - Maps arbitrary-length message to fixed-length hash

- Authenticating a message with a secret key
  - Symmetric equivalent to digital signatures
- Alice uses a key 🔑 to compute a tag: $t =$ MAC🔑(M)
- Bob verifies the tag with the same key 🔑: $t \stackrel{?}{=}$ MAC🔑(M)

MAC Security Notions

- Key-recovery: given access to a MAC oracle, extract the key
- Forgery: given access to a MAC oracle, forge a valid pair
  - For a message chosen by the adversary: existential forgery
  - For a challenge given to the adversary: universal forgery
- Distinguishing games:
  - Distinguish $\mathrm{MAC}_k^{\mathcal{H}}$ from a PRF: distinguishing-R
    e.g. distinguish HMAC from a PRF
  - Distinguish $\mathrm{MAC}_k^{\mathcal{H}}$ from $\mathrm{MAC}_k^{\mathrm{PRF}}$: distinguishing-H
    e.g. distinguish HMAC-SHA1 from HMAC-PRF

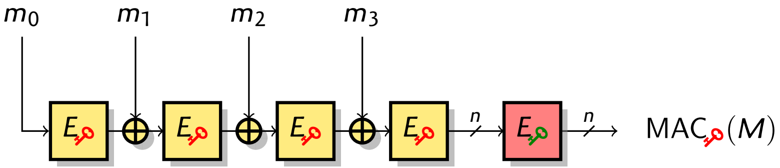
Iterative MACs



- ℓ-bit state
- n-bit MAC
- k-bit key

- Key-dependant initial value I
- Key-dependant state update h
- Key-dependant finalization, with message length g
- Example: ECBC-MAC

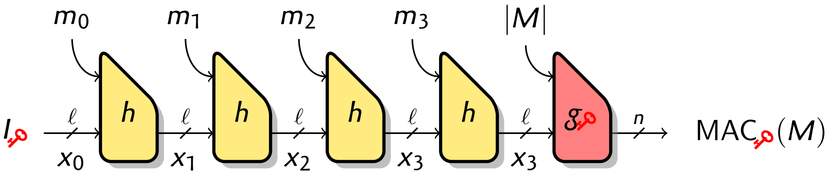
Iterative MACs



- ℓ-bit state
- n-bit MAC
- k-bit key

- Key-dependant initial value I
- Key-dependant state update h
- Key-dependant finalization, with message length g
- Example: ECBC-MAC

Hash-based MACs



- ℓ-bit chaining value
- n-bit MAC
- k-bit key

we focus on $\ell = n = k$

- Key-dependant initial value I
- Unkeyed compression function h
- Key-dependant finalization, with message length g
- Example: HMAC

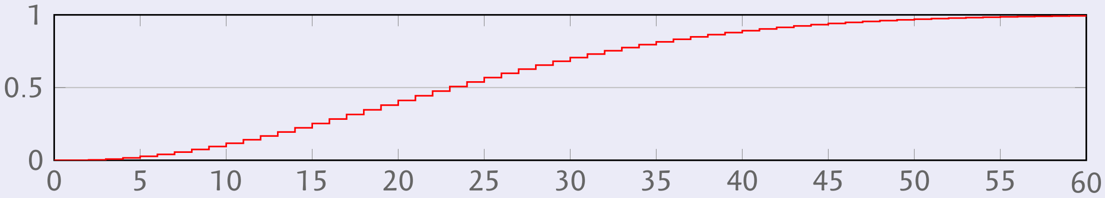
Outline

Birthday Paradox

- Draw r random values from $[0, N-1]$
  - Expected number of collisions is about $r^2/2N$
  - Constant probability of having a collision with $r = \Theta(\sqrt{N})$
- Variant: Let $\mathcal{A}, \mathcal{B}$ be random subsets of $[0, N-1]$
  - Expected number of matches $|\mathcal{A} \cap \mathcal{B}| \approx |\mathcal{A}| \times |\mathcal{B}|/N$
  - In particular, $\mathcal{A} \cap \mathcal{B} \neq \varnothing$ with high probability if $|\mathcal{A}| = |\mathcal{B}| = \sqrt{N}$

The birthday paradox

- In a room with 23 people, 50% chance that two of them share the same birthday.



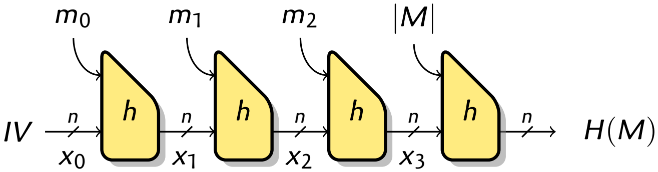
Birthday Paradox

- Draw r random values from $[0, N-1]$
  - Expected number of collisions is about $r^2/2N$
  - Constant probability of having a collision with $r = \Theta(\sqrt{N})$
- Variant: Let $\mathcal{A}, \mathcal{B}$ be random subsets of $[0, N-1]$
  - Expected number of matches $|\mathcal{A} \cap \mathcal{B}| \approx |\mathcal{A}| \times |\mathcal{B}|/N$
  - In particular, $\mathcal{A} \cap \mathcal{B} \neq \varnothing$ with high probability if $|\mathcal{A}| = |\mathcal{B}| = \sqrt{N}$

Collision search in practice

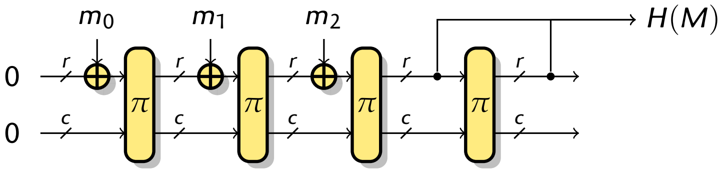
- Sort data to avoid quadratic complexity
- Pollard's rho (memoryless)
- Parallel collision search by van Oorschot and Wiener

The Merkle-Damgård construction (SHA-1, SHA-2)



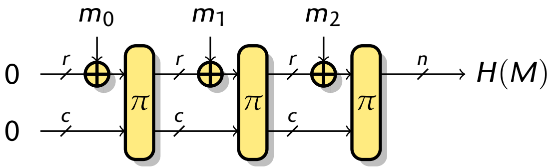
- n-bit state, compression function $h : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^n$
- Finalization with the same function, using message length (MD strengthening)
- Security reduction:
  - Hash collisions imply compression function collision (at least $2^{n/2}$ operations for ideal compression function)
  - Hash preimages imply finalization preimages (at least 2^n operations for ideal compression function)

The Sponge Construction (SHA-3)



- b-bit permutation, $b = c + r$
  - r-bit outer state (rate r)
  - c-bit inner state (capacity c)
- Assume $r \geq n$
- Security with ideal permutation:
  - Collision attack: $\min\{2^{n/2}, 2^{c/2}\}$
  - Preimage attack: $\min\{2^{n}, 2^{c/2}\}$
  - SHA-3: $c = 2n$, n-bit security
  - SHAKE: variable n, $c/2$-bit security

The Sponge Construction (SHA-3)



- b-bit permutation, $b = c + r$
  - r-bit outer state (rate r)
  - c-bit inner state (capacity c)
- Assume $r \geq n$
- Security with ideal permutation:
  - Collision attack: $\min\{2^{n/2}, 2^{c/2}\}$
  - Preimage attack: $\min\{2^{n}, 2^{c/2}\}$
  - SHA-3: $c = 2n$, n-bit security
  - SHAKE: variable n, $c/2$-bit security

Generic attacks

- Generic attacks target the mode without using properties of the inner functions

Merkle-Damgård

- Collision security $2^{n/2}$ (optimal)
- Preimage security 2^n (optimal)
- What about other security notions? (*e.g.* between $2^{n/2}$ and 2^n)

Sponge

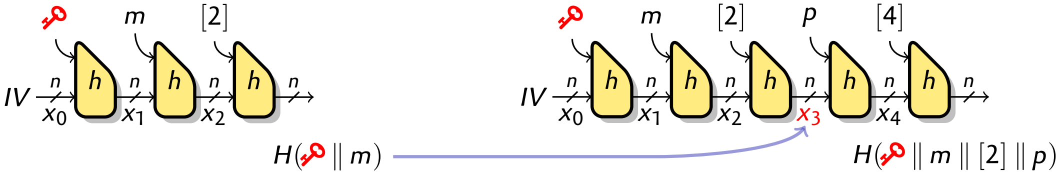
- Collision security $2^{c/2}$
- Preimage security $2^{c/2}$
- Indifferentiable up to $2^{c/2}$

- Generic attacks on Merkle-Damgård
  - Strong combinatorial structure
  - Nice and surprising results 😁

Length extension

- A random oracle can be used to build a MAC with a secret prefix
  - $\text{MAC}_{🔑}(M) = H(🔑 \parallel M)$
  - Secure in the Random Oracle Model

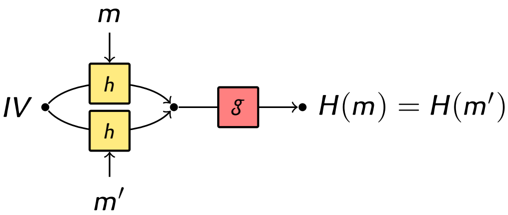
- Insecure with Merkle-Damgård: length-extension attack



  - Compute $H(🔑 \parallel m \parallel [2] \parallel p)$ from $H(🔑 \parallel m)$ without 🔑
    $H(🔑 \parallel m \parallel [2] \parallel p) = h(h(H(🔑 \parallel m), p), [4])$
  - Practical attack against Flickr API [Duong & Rizzo '09]

- Merkle-Damgård with an ideal compression function cannot be used as random oracle
- Fixed with a different finalization function

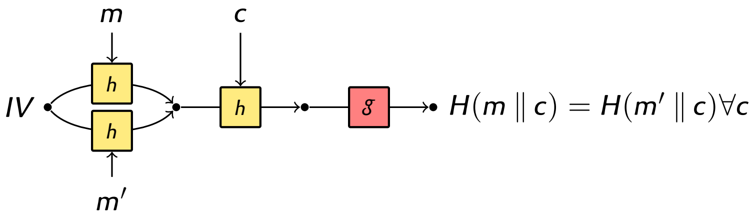
Collision extension



1. Find a collision $h^*(IV, M) = h^*(IV, M')$ $(|M| = |M'|)$
2. For any suffix c $H(m \| c) = H(m' \| c)$

- Distinguish an iterated hash function from a random function ($\mathcal{O}(2^{n/2})$ queries)

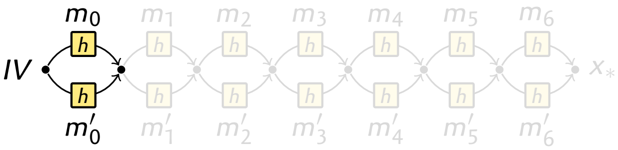
Collision extension



1. Find a collision $h^*(IV, M) = h^*(IV, M')$ $(|M| = |M'|)$
2. For any suffix c $H(m \parallel c) = H(m' \parallel c)$

- Distinguish an iterated hash function from a random function $(\mathcal{O}(2^{n/2})$ queries)

Multicollisions [Joux, Crypto '04]

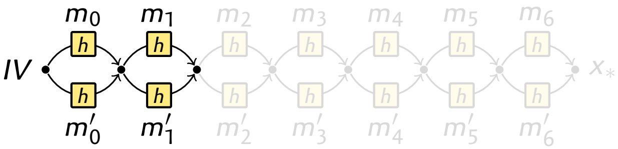


1. Find a collision pair m_0 / m'_0 starting from IV
2. Find a collision pair m_1 / m'_1 starting from $x_1 = h(IV, m_0)$
3. Repeat t times
4. This yields 2^t messages with the same hash:

$$m_0 m_1 m_2 \ldots \quad m'_0 m_1 m_2 \ldots \quad m_0 m'_1 m_2 \ldots \quad m'_0 m'_1 m_2 \ldots$$
$$m_0 m_1 m'_2 \ldots \quad m'_0 m_1 m'_2 \ldots \quad m_0 m'_1 m'_2 \ldots \quad m'_0 m'_1 m'_2 \ldots$$

- Complexity $t \cdot 2^{n/2}$ vs. $\approx 2^{\frac{2^t-1}{2^t} n}$ for a random function

Multicollisions [Joux, Crypto '04]



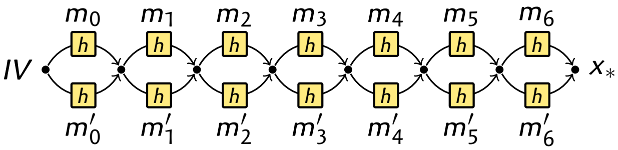
1. Find a collision pair m_0 / m'_0 starting from IV
2. Find a collision pair m_1 / m'_1 starting from $x_1 = h(IV, m_0)$
3. Repeat t times
4. This yields 2^t messages with the same hash:

$m_0 m_1 m_2 \dots$ $m'_0 m_1 m_2 \dots$ $m_0 m'_1 m_2 \dots$ $m'_0 m'_1 m_2 \dots$

$m_0 m_1 m'_2 \dots$ $m'_0 m_1 m'_2 \dots$ $m_0 m'_1 m'_2 \dots$ $m'_0 m'_1 m'_2 \dots$

- Complexity $t \cdot 2^{n/2}$ vs. $\approx 2^{\frac{2^t-1}{2^t} n}$ for a random function

Multicollisions [Joux, Crypto '04]



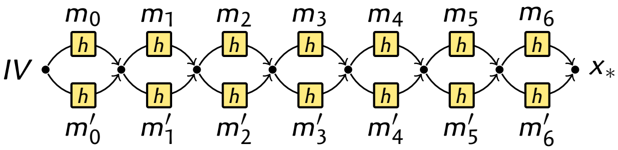
1. Find a collision pair m_0 / m'_0 starting from IV
2. Find a collision pair m_1 / m'_1 starting from $x_1 = h(IV, m_0)$
3. Repeat t times
4. This yields 2^t messages with the same hash:

$m_0 m_1 m_2 \dots$ $m'_0 m_1 m_2 \dots$ $m_0 m'_1 m_2 \dots$ $m'_0 m'_1 m_2 \dots$

$m_0 m_1 m'_2 \dots$ $m'_0 m_1 m'_2 \dots$ $m_0 m'_1 m'_2 \dots$ $m'_0 m'_1 m'_2 \dots$

- Complexity $t \cdot 2^{n/2}$ vs. $\approx 2^{\frac{2^t-1}{2^t} n}$ for a random function

Multicollisions [Joux, Crypto '04]

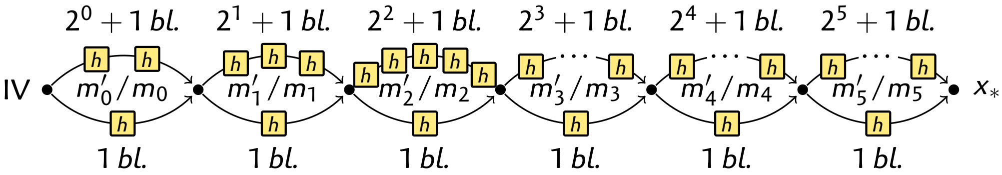


1. Find a collision pair m_0 / m'_0 starting from IV
2. Find a collision pair m_1 / m'_1 starting from $x_1 = h(IV, m_0)$
3. Repeat t times
4. This yields 2^t messages with the same hash:

$$
\begin{array}{llll}
m_0 m_1 m_2 \dots & m'_0 m_1 m_2 \dots & m_0 m'_1 m_2 \dots & m'_0 m'_1 m_2 \dots \\
m_0 m_1 m'_2 \dots & m'_0 m_1 m'_2 \dots & m_0 m'_1 m'_2 \dots & m'_0 m'_1 m'_2 \dots
\end{array}
$$

- Complexity $t \cdot 2^{n/2}$ vs. $\approx 2^{\frac{2^t-1}{2^t} n}$ for a random function

Expandable message [Kelsey & Schneier, Eurocrypt '05]

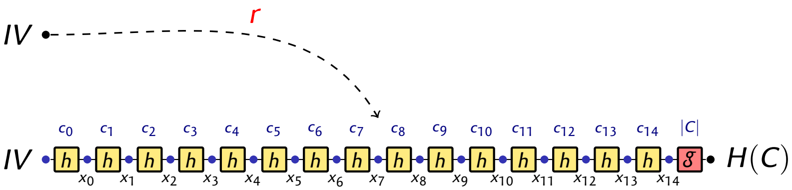


- Multicollision with messages of difference length
  2^t messages of length $t, t+1, \ldots t+2^t-1$ blocks
  - Length 0+6: $m_0 m_1 m_2 m_3 m_4 m_5$
  - Length 1+6: $m'_0 m_1 m_2 m_3 m_4 m_5$
  - Length 2+6: $m_0 m'_1 m_2 m_3 m_4 m_5$
  - Length 3+6: $m'_0 m'_1 m_2 m_3 m_4 m_5$
  - ...
  - Length 63+6: $m'_0 m'_1 m'_2 m'_3 m'_4 m'_5$
- Complexity $t \cdot 2^{n/2}$

Second-preimage for long challenges

- Given a challenge C, find M with $H(M) = H(C)$ — $\text{len}(C) = 2^s$

0. Build expandable message $\mathcal{M}$ of length 2^s (final state w)
1. Compute internal states $\{x_i\}$ for $H(C)$ — (No key: public values)
2. Find r, i with $h(IV, r) = x_i$ — (Complexity 2^{n-s})
3. Preimage is $\mathcal{M}_{i-1} \parallel r \parallel C[i:]$

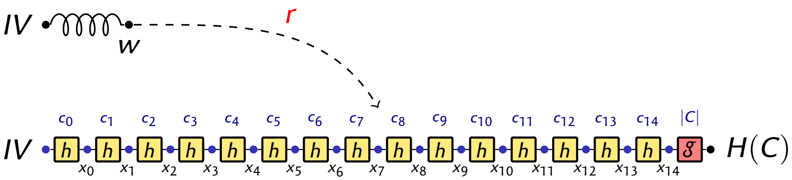


- Complexity $2^s + 2^{n-s}$ — ($2^{n/2}$ for $s = n/2$)

Second-preimage for long challenges

- Given a challenge C, find M with $H(M) = H(C)$ $\quad$ $\text{len}(C) = 2^s$

0. Build expandable message $\mathcal{M}$ of length 2^s (final state w)
1. Compute internal states $\{x_i\}$ for $H(C)$ $\quad$ (No key: public values)
2. Find r, i with $h(w, r) = x_i$ $\quad$ (Complexity 2^{n-s})
3. Preimage is $\mathcal{M}_{i-1} \parallel r \parallel C[i:]$



- Complexity $2^s + 2^{n-s}$ $\quad$ ($2^{n/2}$ for $s = n/2$)

Nostradamus attack / Herding

Simple commitment scheme

- Commit to m: chose random r, send $H(m \| r)$
- Open commitment: send r and m

Diamond structure *[Kelsey & Kohno, EC'06]*

Herd S initial states to a common state

- Try $\approx 2^{n/2}/\sqrt{S}$ msg from each state.
- Whp, the initial states can be paired
- Repeat...

Total $\tilde{O}(\sqrt{S} \cdot 2^{n/2})$

- Open commitment to any value in a set S with complexity $\tilde{O}(\sqrt{S} \cdot 2^{n/2})$
- Arbitrary opening of commitment with complexity $\tilde{O}(2^{2n/3})$ $(S = 2^{n/3})$
  - With long messages, complexity $\tilde{O}(2^{n/2})$

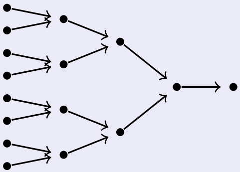
Nostradamus attack / Herding

Simple commitment scheme

- Commit to m: chose random r, send $H(m \parallel r)$
- Open commitment: send r and m

Diamond structure

[Kelsey & Kohno, EC'06]



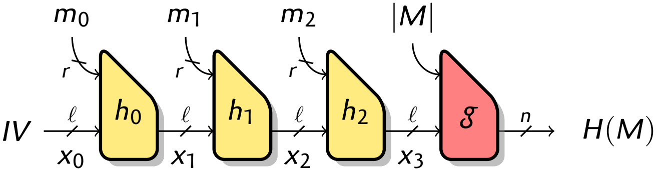
Herd S initial states to a common state

- Try $\approx 2^{n/2}/\sqrt{S}$ msg from each state.
- Whp, the initial states can be paired
- Repeat...

Total $\tilde{\mathcal{O}}(\sqrt{S} \cdot 2^{n/2})$

- Open commitment to any value in a set S with complexity $\tilde{\mathcal{O}}(\sqrt{S} \cdot 2^{n/2})$
- Arbitrary opening of commitment with complexity $\tilde{\mathcal{O}}(2^{2n/3})$ $(S = 2^{n/3})$
  - With long messages, complexity $\tilde{\mathcal{O}}(2^{n/2})$

Tweaking Merkle-Damgård



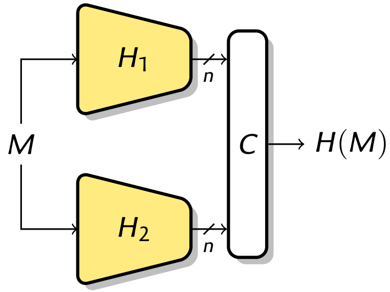
HAIFA (e.g. BLAKE)

- Finalization function
- Block counter in each h
  - Avoids copy-paste attacks (Second-preimage w/ long messages)
- Ideal behaviour up to $2^{\ell/2}$
- After $2^{\ell/2}$: multicollisions, herding, ...

Wide pipe (e.g. SHA-512/256)

- Finalization function
- Larger state: $\ell > n$
- Ideal behaviour with $\ell \geq 2n$ (assuming finalization function)

Combining two hash functions



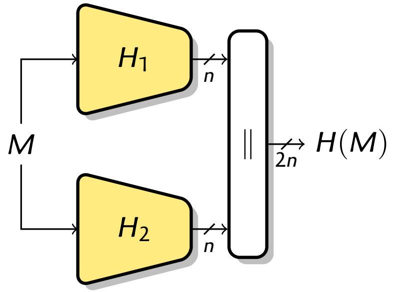
"In order to make the PRF as secure as possible, it uses two hash algorithms in a way which should guarantee its security if either algorithm remains secure."
– RFC 2246 (TLS 1.0)

Classical combiners:

- Concatenation: $H_1(M) \parallel H2(M)$
- Xor: $H_1(M) \oplus H2(M)$

"The whole is greater than the sum of its parts"
– Aristotle

Combining two hash functions



"In order to make the PRF as secure as possible, it uses two hash algorithms in a way which should guarantee its security if either algorithm remains secure."

– RFC 2246 (TLS 1.0)

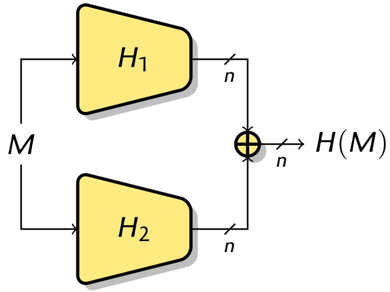
Classical combiners:

- Concatenation: $H_1(M) \| H2(M)$
- Xor: $H_1(M) \oplus H2(M)$

"The whole is greater than the sum of its parts"

– Aristotle

Combining two hash functions



"In order to make the PRF as secure as possible, it uses two hash algorithms in a way which should guarantee its security if either algorithm remains secure."

– RFC 2246 (TLS 1.0)

Classical combiners:

- Concatenation:
  $H_1(M) \| H2(M)$
- Xor:
  $H_1(M) \oplus H2(M)$

"The whole is greater than the sum of its parts"

– Aristotle

Known results: Concatenation combiner

- $H(M) = H_1(M) \parallel H_2(M)$
- $2 \times n$-bit internal state, $2n$-bit output

- *Robust combiner* for collisions
  - A collision in H implies a collision in H_1 and H_2

- $2 \times n$-bit internal state can increase security?
  - NO: Multicollision attack [Joux '04]
    - Collisions in $2^{n/2}$
    - Preimages in 2^n
    - Essentially n-bit security

Known results: Concatenation combiner

- $H(M) = H_1(M) \parallel H_2(M)$
- $2 \times n$-bit internal state, $2n$-bit output

- *Robust combiner* for collisions
  - A collision in H implies a collision in H_1 and H_2

- $2 \times n$-bit internal state can increase security?
  - NO: Multicollision attack [Joux '04]
    - Collisions in $2^{n/2}$
    - Preimages in 2^n
    - Essentially n-bit security

Known results: Concatenation combiner

- $H(M) = H_1(M) \parallel H_2(M)$
- $2 \times n$-bit internal state, $2n$-bit output

- *Robust combiner* for collisions
  - A collision in H implies a collision in H_1 and H_2

- $2 \times n$-bit internal state can increase security?
  - NO: Multicollision attack [Joux '04]
    - Collisions in $2^{n/2}$
    - Preimages in 2^n
    - Essentially n-bit security

Known results: Concatenation combiner

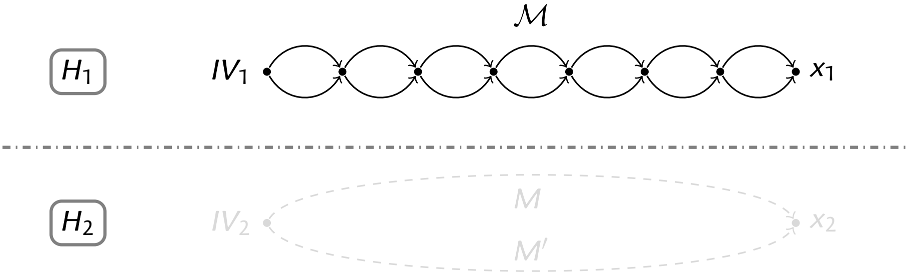
- $H(M) = H_1(M) \parallel H_2(M)$
- $2 \times n$-bit internal state, $2n$-bit output

- *Robust combiner* for collisions
  - A collision in H implies a collision in H_1 and H_2

- $2 \times n$-bit internal state can increase security?

- NO: Multicollision attack [Joux '04]
  - Collisions in $2^{n/2}$
  - Preimages in 2^n
  - Essentially n-bit security

Collision attack for $H_1(M) \parallel H_2(M)$



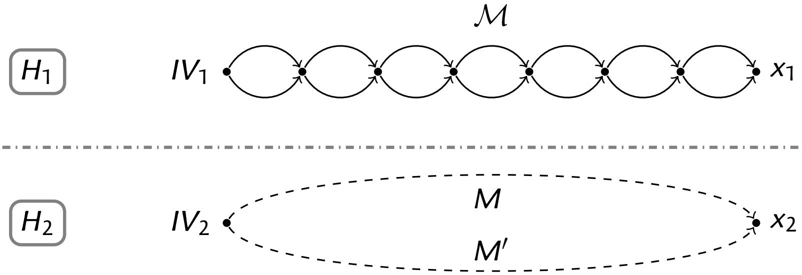
1. Build a $2^{n/2}$-multicollision for H_1

$$\forall M \in \mathcal{M}, H_1(M) = x_1$$

2. Find $M, M' \in \mathcal{M}$ s.t. $H_2(M) = H_2(M')$

- Complexity $n \cdot 2^{n/2}$ vs. 2^n for a $2n$-bit hash function.

Collision attack for $H_1(M) \parallel H_2(M)$



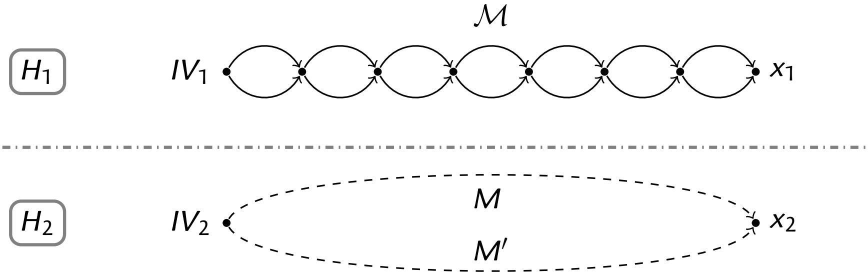
1. Build a $2^{n/2}$-multicollision for H_1

$$\forall M \in \mathcal{M}, H_1(M) = x_1$$

2. Find $M, M' \in \mathcal{M}$ s.t. $H_2(M) = H_2(M')$

- Complexity $n \cdot 2^{n/2}$ vs. 2^n for a $2n$-bit hash function.

Collision attack for $H_1(M) \| H_2(M)$



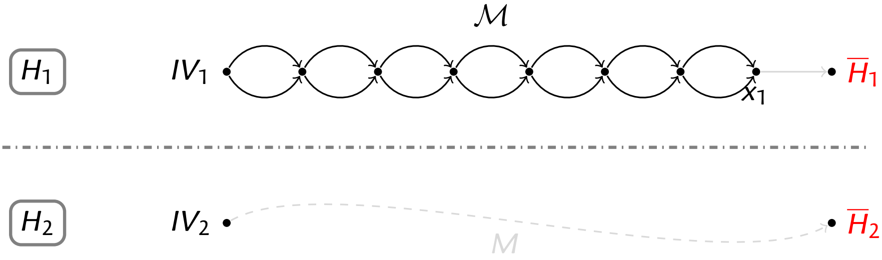
1. Build a $2^{n/2}$-multicollision for H_1

$$\forall M \in \mathcal{M}, H_1(M) = x_1$$

2. Find $M, M' \in \mathcal{M}$ s.t. $H_2(M) = H_2(M')$

- Complexity $n \cdot 2^{n/2}$ vs. 2^n for a $2n$-bit hash function.

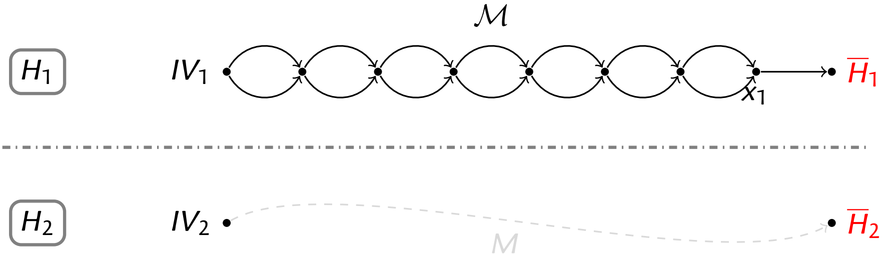
Preimage attack for $H_1(M) \parallel H_2(M)$



1. Build a 2^n-multicollision for H_1 — $\forall M \in \mathcal{M}, h_1^*(M) = x_1$
2. Find a preimage for H_1: $g(h(x_1, r)) = \overline{H}_1$ — $\forall M \in \mathcal{M}, H_1(M) = \overline{H}_1$
3. Find $M \in \mathcal{M}$ s.t. $H_2(M \parallel r) = \overline{H}_2$

- Complexity $\widetilde{\mathcal{O}}(2^n)$ vs. 2^{2n} for a $2n$-bit hash function.

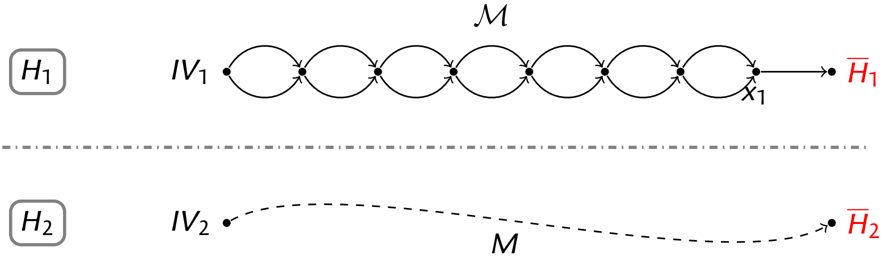
Preimage attack for $H_1(M) \parallel H_2(M)$



1. Build a 2^n-multicollision for H_1 $\qquad \forall M \in \mathcal{M}, h_1^*(M) = x_1$
2. Find a preimage for H_1: $g(h(x_1, r)) = \overline{H}_1$ $\qquad \forall M \in \mathcal{M}, H_1(M) = \overline{H}_1$
3. Find $M \in \mathcal{M}$ s.t. $H_2(M \parallel r) = \overline{H}_2$

- Complexity $\tilde{\mathcal{O}}(2^n)$ vs. 2^{2n} for a $2n$-bit hash function.

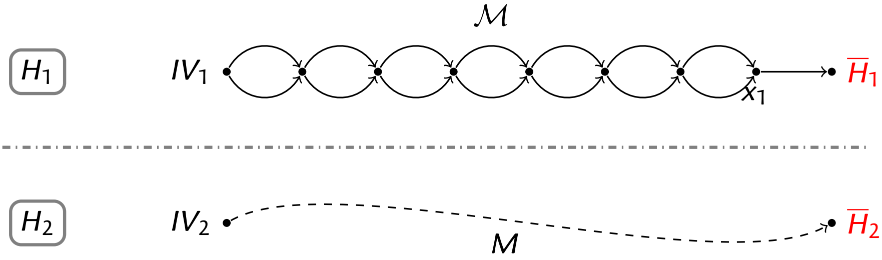
Preimage attack for $H_1(M) \| H_2(M)$



1. Build a 2^n-multicollision for H_1 — $\forall M \in \mathcal{M}, h_1^*(M) = x_1$
2. Find a preimage for H_1: $g(h(x_1, r)) = \overline{H_1}$ — $\forall M \in \mathcal{M}, H_1(M) = \overline{H_1}$
3. Find $M \in \mathcal{M}$ s.t. $H_2(M \| r) = \overline{H_2}$

- Complexity $\tilde{\mathcal{O}}(2^n)$ vs. 2^{2n} for a $2n$-bit hash function.

Preimage attack for $H_1(M) \parallel H_2(M)$



1. Build a 2^n-multicollision for H_1 — $\forall M \in \mathcal{M}, h_1^*(M) = x_1$
2. Find a preimage for H_1: $g(h(x_1, r)) = \overline{H_1}$ — $\forall M \in \mathcal{M}, H_1(M) = \overline{H_1}$
3. Find $M \in \mathcal{M}$ s.t. $H_2(M \parallel r) = \overline{H_2}$

- Complexity $\tilde{\mathcal{O}}(2^n)$ vs. 2^{2n} for a $2n$-bit hash function.

Known results: Xor Combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- $2 \times n$-bit internal state, n-bit output

- *Robust combiner* for PRFs and MACs
- $2 \times n$-bit internal state can increase security?
  - NO: Joux's attacks are applicable

- No short output robust combiners for collision resistance [Boneh & Boyen '06, ...]
  - Doesn't imply a generic attack...

- Secure up to $2^{n/2}$ with weak compression fcts [Hoch & Shamir '08]
  - In particular, no generic collision attack

Known results: Xor Combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- $2 \times n$-bit internal state, n-bit output

- *Robust combiner* for PRFs and MACs
- $2 \times n$-bit internal state can increase security?
  - NO: Joux's attacks are applicable
- No short output robust combiners for collision resistance [Boneh & Boyen '06, ...]
  - Doesn't imply a generic attack...
- Secure up to $2^{n/2}$ with weak compression fcts [Hoch & Shamir '08]
  - In particular, no generic collision attack

Known results: Xor Combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- $2 \times n$-bit internal state, n-bit output

- *Robust combiner* for PRFs and MACs
- $2 \times n$-bit internal state can increase security?
  - NO: Joux's attacks are applicable

- No short output robust combiners for collision resistance [Boneh & Boyen '06, ...]
  - Doesn't imply a generic attack...

- Secure up to $2^{n/2}$ with weak compression fcts [Hoch & Shamir '08]
  - In particular, no generic collision attack

Known results: Xor Combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- $2 \times n$-bit internal state, n-bit output

- *Robust combiner* for PRFs and MACs
- $2 \times n$-bit internal state can increase security?
  - NO: Joux's attacks are applicable

- No short output robust combiners for collision resistance [Boneh & Boyen '06, ...]
  - Doesn't imply a generic attack...

- Secure up to $2^{n/2}$ with weak compression fcts [Hoch & Shamir '08]
  - In particular, no generic collision attack

Known results: Xor Combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- $2 \times n$-bit internal state, n-bit output

- *Robust combiner* for PRFs and MACs
- $2 \times n$-bit internal state can increase security?
  - NO: Joux's attacks are applicable

- No short output robust combiners for collision resistance [Boneh & Boyen '06, ...]
  - Doesn't imply a generic attack...

- Secure up to $2^{n/2}$ with weak compression fcts [Hoch & Shamir '08]
  - In particular, no generic collision attack

Known results: Xor Combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- $2 \times n$-bit internal state, n-bit output

- *Robust combiner* for PRFs and MACs
- $2 \times n$-bit internal state can increase security?
  - NO: Joux's attacks are applicable

- No short output robust combiners for collision resistance [Boneh & Boyen '06, ...]
  - Doesn't imply a generic attack...

- Secure up to $2^{n/2}$ with weak compression fcts [Hoch & Shamir '08]
  - In particular, no generic collision attack

Known results: Xor Combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- $2 \times n$-bit internal state, n-bit output

- *Robust combiner* for PRFs and MACs
- $2 \times n$-bit internal state can increase security?
  - NO: Joux's attacks are applicable

- No short output robust combiners for collision resistance [Boneh & Boyen '06, . . .]
  - Doesn't imply a generic attack...

- Secure up to $2^{n/2}$ with weak compression fcts [Hoch & Shamir '08]
  - In particular, no generic collision attack

Generic attacks against combiniers

Concatenation combiner

- $H(M) = H_1(M) \parallel H_2(M)$
- $2n$-bit output
- Generic attacks:
  - Collisions in $2^{n/2}$
  - Preimages in 2^n
  - Non-ideal after $2^{n/2}$

XOR combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- n-bit output
- Generic attacks:
  - Collisions in $2^{n/2}$
  - Preimages in $\leq 2^n$
  - Non-ideal after $2^{n/2}$

Suprising result

If H_1 and H_2 are good MD hash functions, $H_1 \oplus H_2$ is weak!

Generic attacks against combiniers

Concatenation combiner

- $H(M) = H_1(M) \parallel H_2(M)$
- $2n$-bit output
- Generic attacks:
  - Collisions in $2^{n/2}$
  - Preimages in 2^n
  - Non-ideal after $2^{n/2}$

XOR combiner

- $H(M) = H_1(M) \oplus H_2(M)$
- n-bit output
- Generic attacks:
  - Collisions in $2^{n/2}$
  - Preimages in $\leq 2^{11n/18}$
  - Non-ideal after $2^{n/2}$

Suprising result

If H_1 and H_2 are good MD hash functions, $H_1 \oplus H_2$ is weak!

Outline

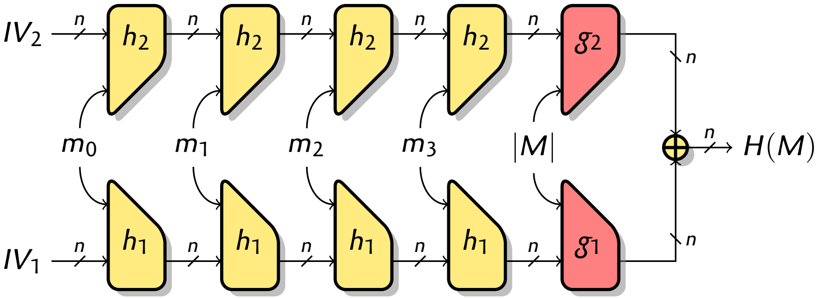
Bibliography

G. Leurent, L. Wang
The Sum Can Be Weaker Than Each Part
EUROCRYPT 2015

Z. Bao, L. Wang, J. Guo, D. Gu
Functional Graph Revisited: Updates on (Second) Preimage Attacks on Hash Combiners.
CRYPTO 2017

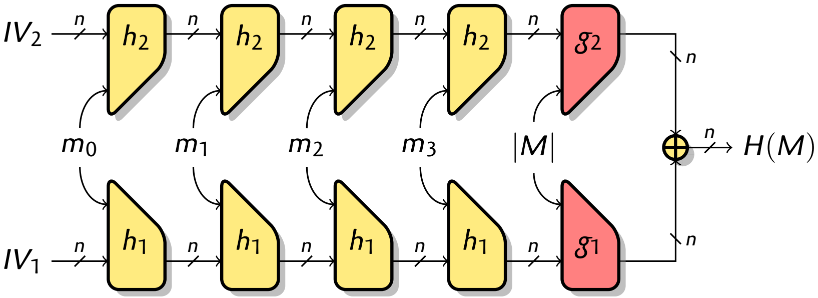
Z. Bao, I. Dinur, J. Guo, G. Leurent, L. Wang
Generic Attacks on Hash Combiners
Journal of Cryptology 2020

Our target: $H(M) = H_1(M) \oplus H_2(M)$



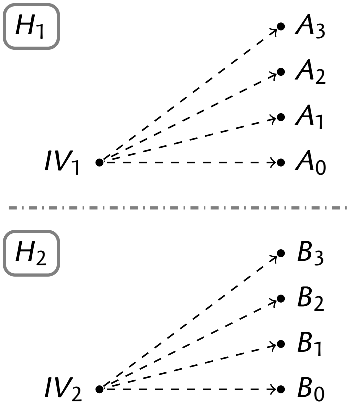
- $2 \times n$-bit state, 2 compression functions
- Can we use a birthday-type attack on the final XOR?

Our target: $H(M) = H_1(M) \oplus H_2(M)$



- $2 \times n$-bit state, 2 compression functions
- Can we use a birthday-type attack on the final XOR?

Strategy



- Build a structure $\{\mathbf{M}_{jk}\}$ to control H_1 and H_2:

$$(IV_1, IV_2) \overset{\mathbf{M}_{jk}}{\rightsquigarrow} (A_j, B_k)$$

- Horizontal lines: common message M

$$(a^i_j, b^i_k) \xrightarrow{M_i} (a^{i+1}_j, b^{i+1}_k)$$

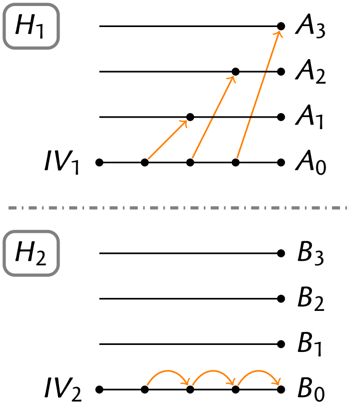
- Orange lines: alternative messages

$$(a^i_{j_0}, b^i_{k_0}) \xrightarrow{M'_i} (a^{i+1}_{j_0}, b^{i+1}_{k_1})$$

- Messages in the structure use a few alternative chunks:

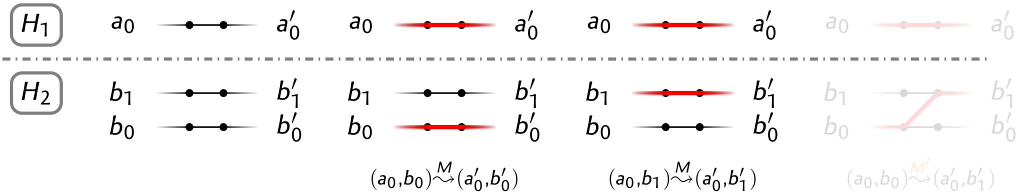
$$\mathbf{M}_{jk} = M_0, M_1, \ldots, M'_{ij}, \ldots$$

Strategy



- Build a structure $\{\mathbf{M}_{jk}\}$ to control H_1 and H_2:
$$(IV_1, IV_2) \overset{\mathbf{M}_{jk}}{\rightsquigarrow} (A_j, B_k)$$
- Horizontal lines: common message M
$$(a^i_j, b^i_k) \xrightarrow{M_i} (a^{i+1}_j, b^{i+1}_k)$$
- Orange lines: alternative messages
$$(a^i_{j_0}, b^i_{k_0}) \xrightarrow{M'_i} (a^{i+1}_{j_0}, b^{i+1}_{k_1})$$
- Messages in the structure use a few alternative chunks:
$$\mathbf{M}_{jk} = M_0, M_1, \dots, M'_{ij}, \dots$$

Switch structure

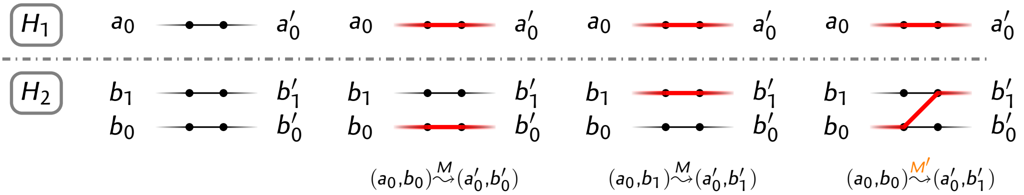


- Simple case: one H_1-chain, and two H_2-chains

- Input: a_0, b_0, b_1
- Output: M, M', a'_0, b'_0, b'_1 s.t.

$$a'_0 = h_1^*(a_0, M) = h_1^*(a_0, M')$$
$$b'_1 = h_2^*(b_1, M) = h_2^*(b_0, M')$$
$$b'_0 = h_2^*(b_0, M) \neq b'_1$$

Switch structure

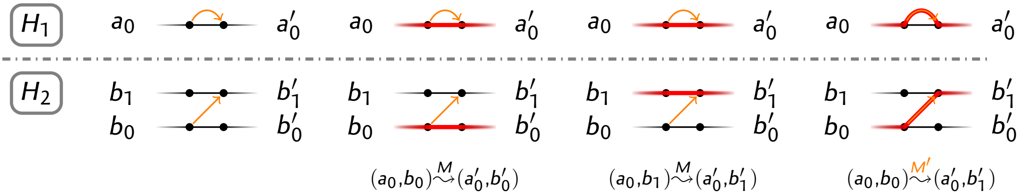


- Simple case: one H_1-chain, and two H_2-chains

- Input: a_0, b_0, b_1
- Output: M, M', a'_0, b'_0, b'_1 s.t.

$$
\begin{aligned}
a'_0 &= h_1^*(a_0, M) = h_1^*(a_0, M') \\
b'_1 &= h_2^*(b_1, M) = h_2^*(b_0, M') \\
b'_0 &= h_2^*(b_0, M) \neq b'_1
\end{aligned}
$$

Switch structure

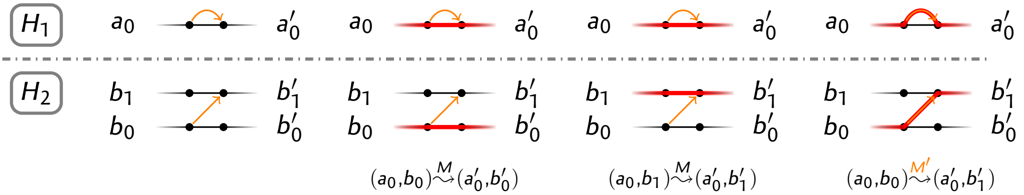


- Simple case: one H_1-chain, and two H_2-chains

- Input: a_0, b_0, b_1
- Output: M, M', a'_0, b'_0, b'_1 s.t.

$$a'_0 = h_1^*(a_0, M) = h_1^*(a_0, M')$$
$$b'_1 = h_2^*(b_1, M) = h_2^*(b_0, M')$$
$$b'_0 = h_2^*(b_0, M) \neq b'_1$$

Switch structure

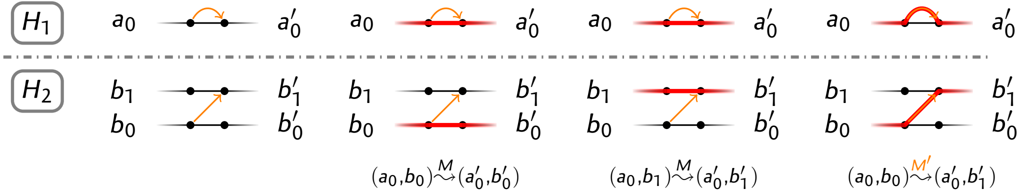


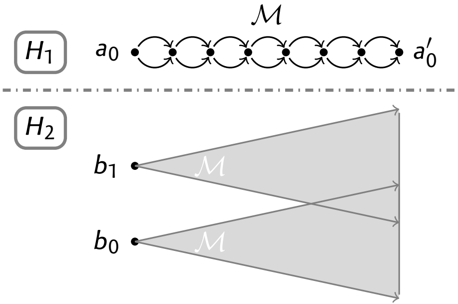
- Simple case: one H_1-chain, and two H_2-chains

- Input: a_0, b_0, b_1
- Output: M, M', a'_0, b'_0, b'_1 s.t.

$$\begin{aligned} a'_0 &= h_1^*(a_0, M) = h_1^*(a_0, M') \\ b'_1 &= h_2^*(b_1, M) = h_2^*(b_0, M') \\ b'_0 &= h_2^*(b_0, M) \neq b'_1 \end{aligned}$$

Switch structure

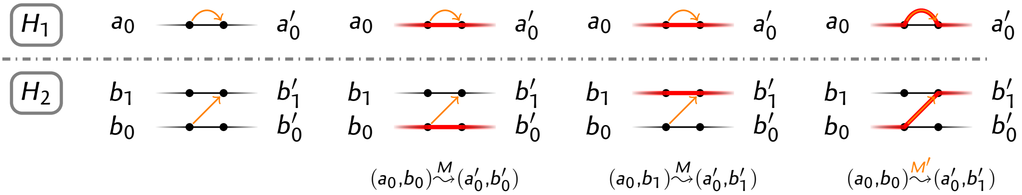


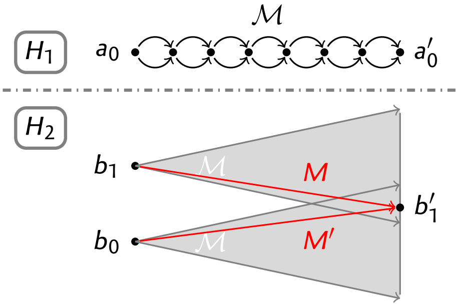


1. Build multicollision $\mathcal{M}$ for H_1
2. Select $M, M' \in \mathcal{M}$ s.t. $h_2^*(b_1, M) = h_2^*(b_0, M') \triangleq b'_1$
3. Set $a'_0 \triangleq h_1^*(a_0, M)$, $b'_0 \triangleq h_2^*(b'_k, M)$

- Complexity $\approx n \cdot 2^{n/2}$

Switch structure

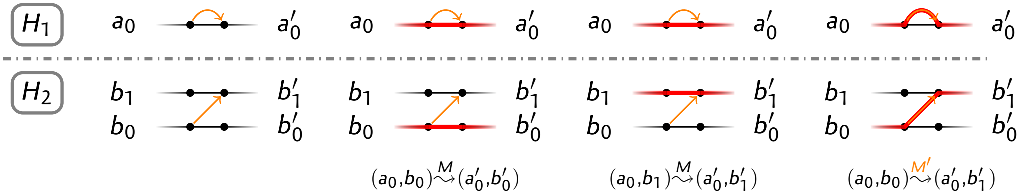


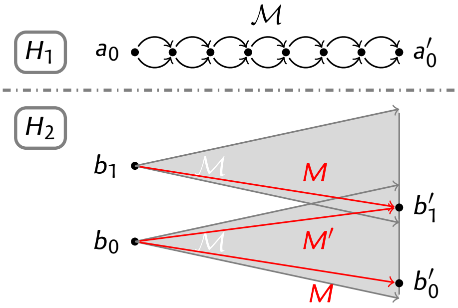


1. Build multicollision $\mathcal{M}$ for H_1
2. Select $M, M' \in \mathcal{M}$ s.t. $h_2^*(b_1, M) = h_2^*(b_0, M') \triangleq b'_1$
3. Set $a'_0 \triangleq h_1^*(a_0, M)$, $b'_0 \triangleq h_2^*(b'_k, M)$

- Complexity $\approx n \cdot 2^{n/2}$

Switch structure





1. Build multicollision $\mathcal{M}$ for H_1
2. Select $M, M' \in \mathcal{M}$ s.t. $h_2^*(b_1, M) = h_2^*(b_0, M') \triangleq b'_1$
3. Set $a'_0 \triangleq h_1^*(a_0, M)$, $b'_0 \triangleq h_2^*(b^i_k, M)$

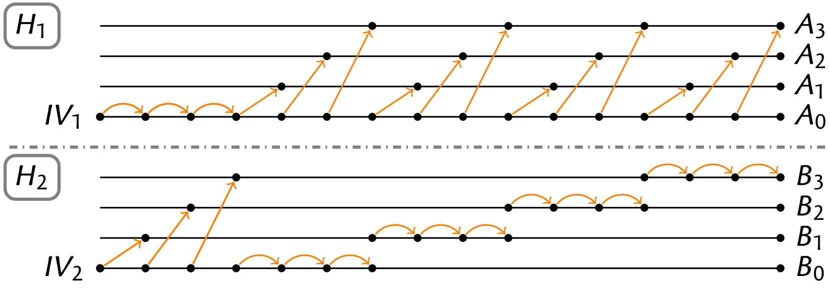
- Complexity $\approx n \cdot 2^{n/2}$

Switch stucture

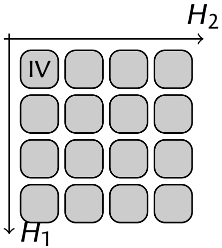
- We call this structure a switch. It can be used with more chains:
  - Update inactive chains with common message M
  - Jump from (a_j, b_k) to $(a_{j'}, b_k)$ or to $(a_j, b_{k'})$
  - Alternate message to be used only from (a_j, b_k)!
- Reach all chain combinations by combining several switches:
  - Interchange structure:

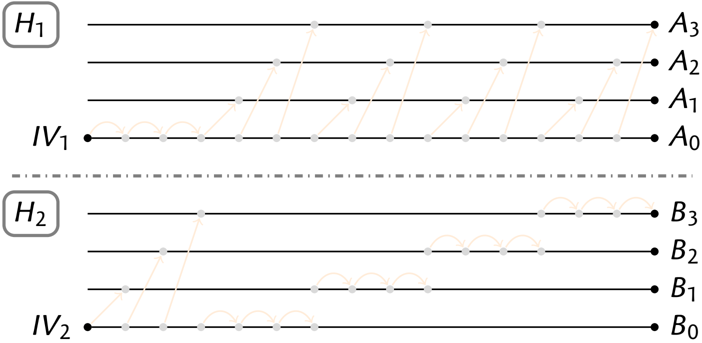
Switch stucture

- We call this structure a switch. It can be used with more chains:
  - Update inactive chains with common message M
  - Jump from (a_j, b_k) to $(a_{j'}, b_k)$ or to $(a_j, b_{k'})$
  - Alternate message to be used only from (a_j, b_k)!
- Reach all chain combinations by combining several switches:
  - Interchange structure:



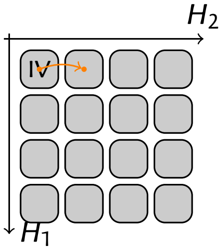
Interchange structure

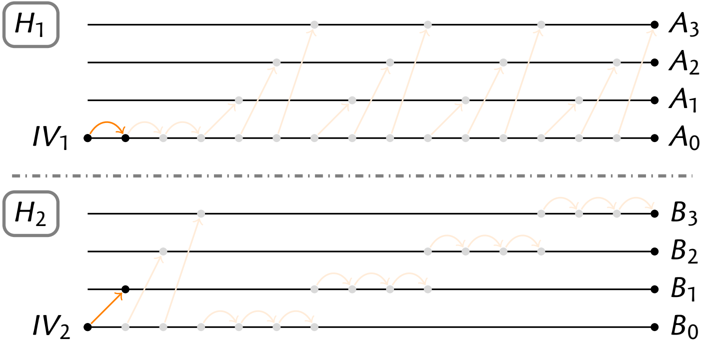




- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

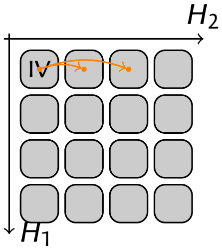
Interchange structure

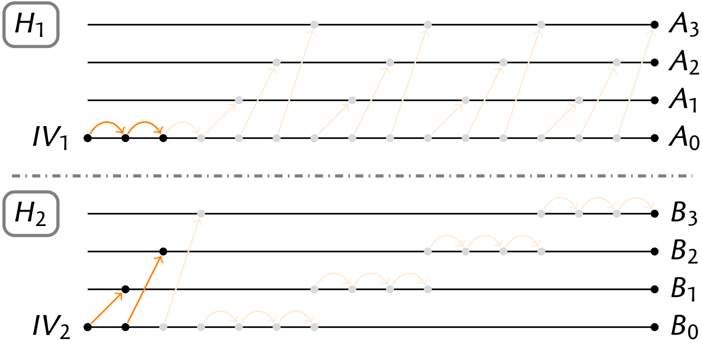




- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

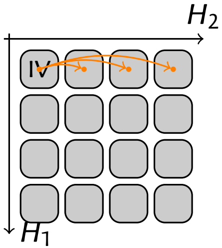
Interchange structure

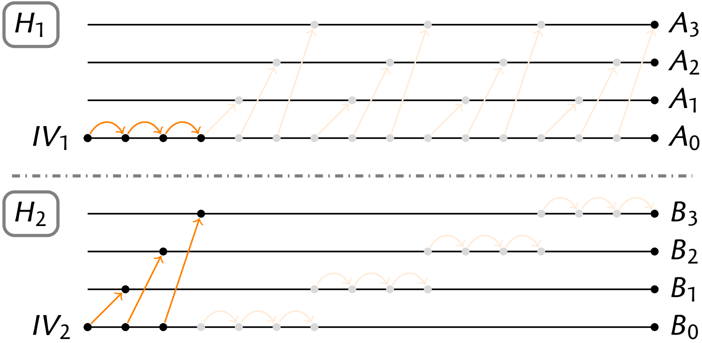




- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

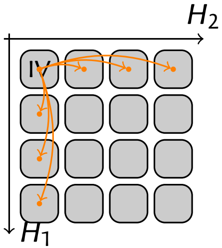
Interchange structure

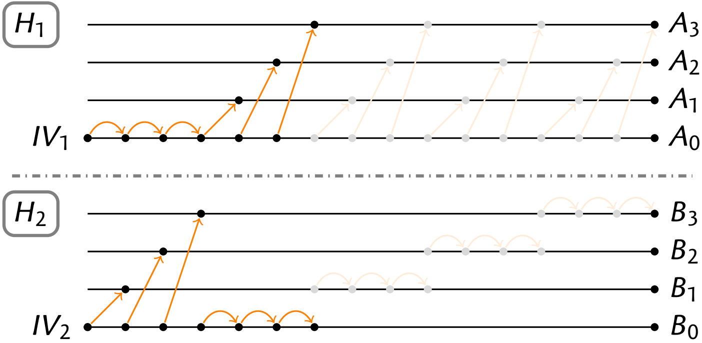




- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

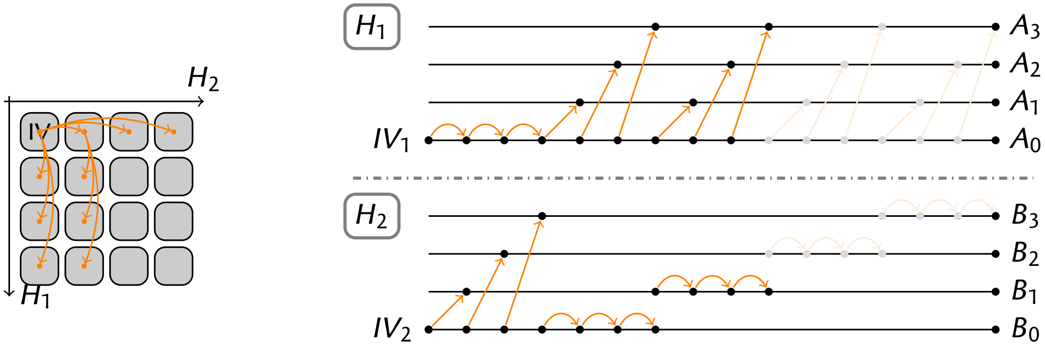
Interchange structure





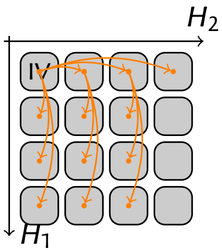
- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

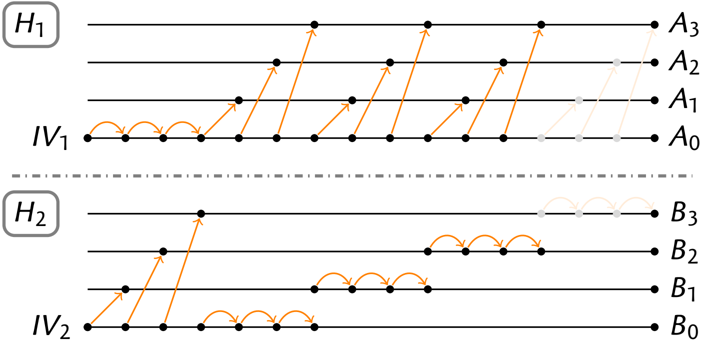
Interchange structure



- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

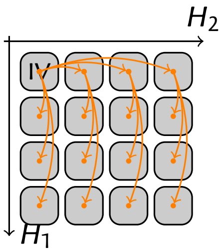
Interchange structure

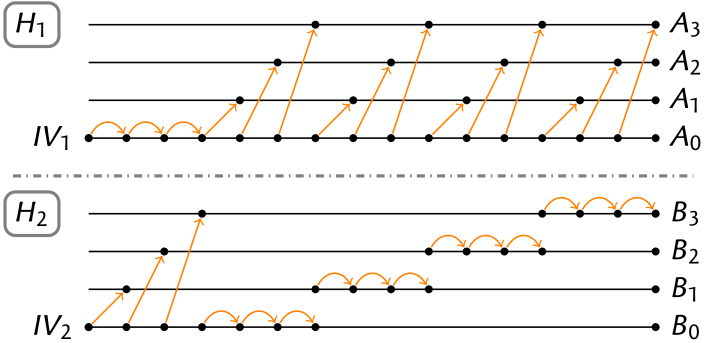




- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

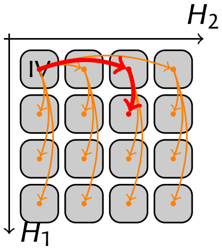
Interchange structure

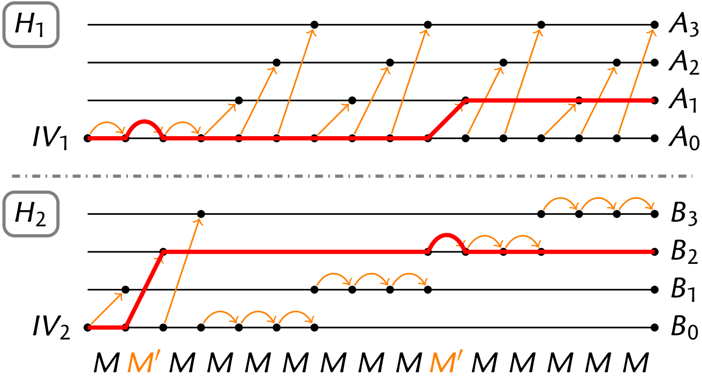




- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

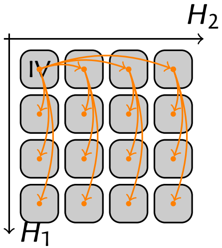
Interchange structure

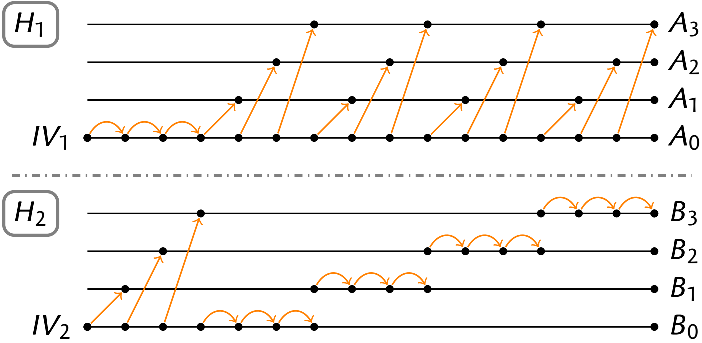




- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

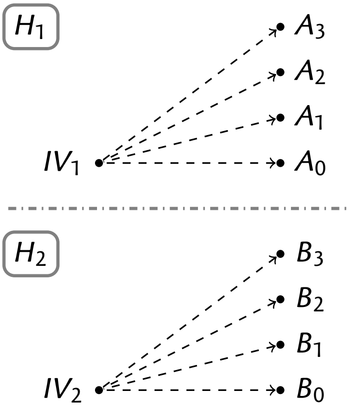
Interchange structure





- Control 2^t H_1-chains and 2^t H_2-chains using 2^{2t} switches
- Message length: $n/2 \cdot 2^{2t}$, memory: $n \cdot 2^{2t}$
- Time complexity: $n/2 \cdot 2^{n/2} \cdot 2^{2t}$

Preimage Attack



1. Build a 2^t-interchange structure $\{\mathbf{M}_{jk}\}$:

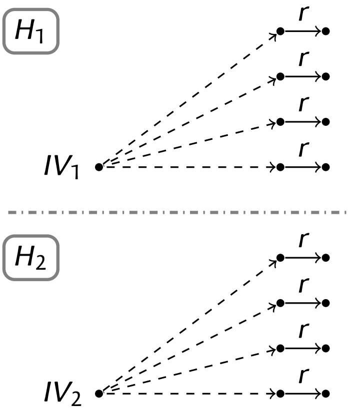
$$(IV_1, IV_2) \overset{\mathbf{M}_{jk}}{\leadsto} (A_j, B_k)$$

   - Complexity: $\tilde{O}(2^{2t} \cdot 2^{n/2})$

2. Preimage search for $\overline{H}$:
   - For random blocks r, match $\{g_1(h_1(A_j, r))\}$ and $\{g_2(h_2(B_k, r)) \oplus \overline{H}\}$
   - If there is a match (j, k): Get $\mathbf{M}_{jk}$, preimage is $M = \mathbf{M}_{jk} \| r$
   - Complexity: $\tilde{O}(2^t \times 2^{n-2t})$

3. Optimal complexity: $\mathcal{O}(n \cdot 2^{5n/6})$
   - $t = n/6$

Preimage Attack



1. Build a 2^t-interchange structure $\{\mathbf{M}_{jk}\}$:

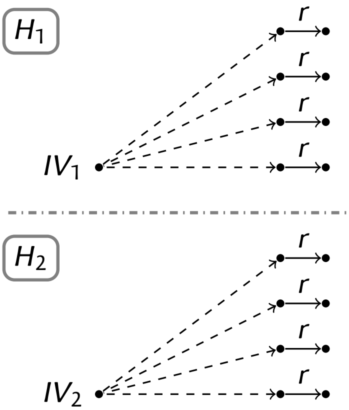
$$(IV_1, IV_2) \overset{\mathbf{M}_{jk}}{\rightsquigarrow} (A_j, B_k)$$

   - Complexity: $\tilde{\mathcal{O}}(2^{2t} \cdot 2^{n/2})$

2. Preimage search for $\overline{H}$:
   - For random blocks r, match $\{g_1(h_1(A_j, r))\}$ and $\{g_2(h_2(B_k, r)) \oplus \overline{H}\}$
   - If there is a match (j, k): Get $\mathbf{M}_{jk}$, preimage is $M = \mathbf{M}_{jk} \| r$
   - Complexity: $\tilde{\mathcal{O}}(2^t \times 2^{n-2t})$

3. Optimal complexity: $\mathcal{O}(n \cdot 2^{5n/6})$
   - $t = n/6$

Preimage Attack



1. Build a 2^t-interchange structure $\{\mathbf{M}_{jk}\}$:

$$(IV_1, IV_2) \overset{\mathbf{M}_{jk}}{\rightsquigarrow} (A_j, B_k)$$

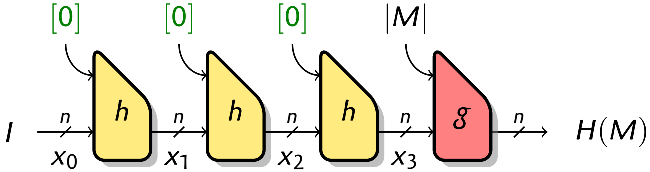
   - Complexity: $\tilde{\mathcal{O}}(2^{2t} \cdot 2^{n/2})$

2. Preimage search for $\overline{H}$:
   - For random blocks r, match $\{g_1(h_1(A_j, r))\}$ and $\{g_2(h_2(B_k, r)) \oplus \overline{H}\}$
   - If there is a match (j, k): Get $\mathbf{M}_{jk}$, preimage is $M = \mathbf{M}_{jk} \parallel r$
   - Complexity: $\tilde{\mathcal{O}}(2^t \times 2^{n-2t})$

3. Optimal complexity: $\mathcal{O}(n \cdot 2^{5n/6})$
   - $t = n/6$

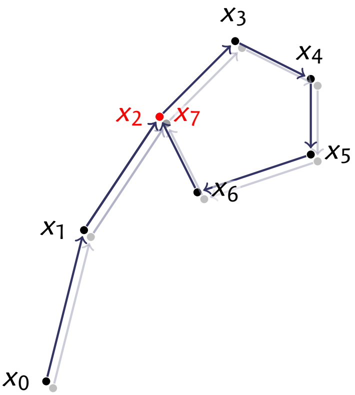
Outline

Alternative attack using cycles



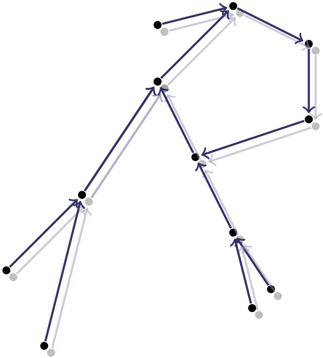
- Consider long messages $M = [0]^\lambda$
- The compression function with fixed message block, is fixed function
- Study properties of the iteration $x_{i+1} = h(x_i, [0])$
  - $h_{[0]} : x \mapsto h(x, [0])$

Random Mappings



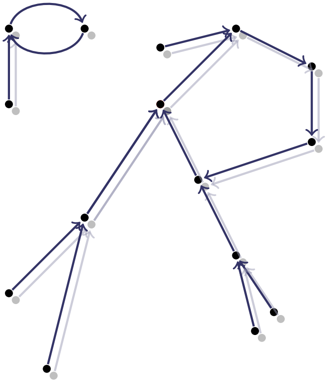
- Functional graph of a random mapping $x \to f(x)$
- Iterate f: $x_i = f(x_{i-1})$
- Collision after $\approx 2^{n/2}$ iterations
  - Cycles
- Trees rooted in the cycle
- Several components

Random Mappings



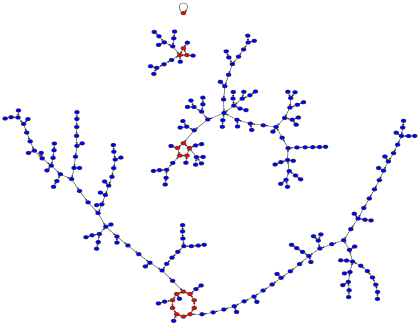
- Functional graph of a random mapping $x \to f(x)$
- Iterate f: $x_i = f(x_{i-1})$
- Collision after $\approx 2^{n/2}$ iterations
  - Cycles
- Trees rooted in the cycle
- Several components

Random Mappings



- Functional graph of a random mapping $x \rightarrow f(x)$
- Iterate f: $x_i = f(x_{i-1})$
- Collision after $\approx 2^{n/2}$ iterations
  - Cycles
- Trees rooted in the cycle
- Several components

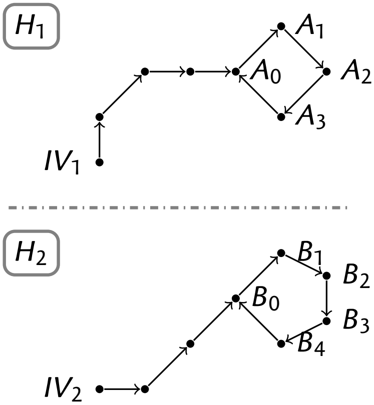
Cycle structure



Expected properties of a random mapping over N points:

- # Components: $\frac{1}{2}\log N$
- # Cyclic nodes: $\sqrt{\pi N/2}$
- Tail length: $\sqrt{\pi N/8}$
- Cycle length: $\sqrt{\pi N/8}$
- Largest tree: $0.48N$
- Largest component: $0.76N$

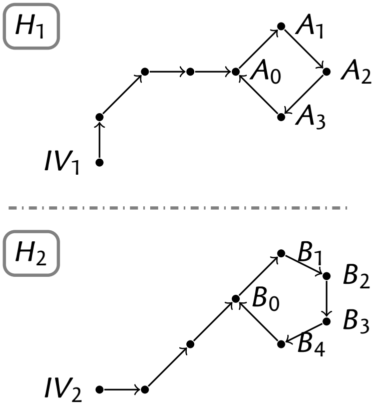
Cycle-based attack



- Long messages with fixed block $M = [0]^\lambda$
- Use cyclic nodes as end-point:
  - H_1 cycle: $\{A_0, A_1, \dots\}$, H_2 cycle: $\{B_0, B_1, \dots\}$
- Reach arbitrary (A_j, B_k) by varying length λ
  - Chinese Remainder Theorem
- Example: (A_0, B_0)

$$\begin{cases} \lambda \bmod 4 = 0 \\ \lambda \bmod 5 = 3 \end{cases} \iff \lambda \bmod 20 = 8$$

Cycle-based attack

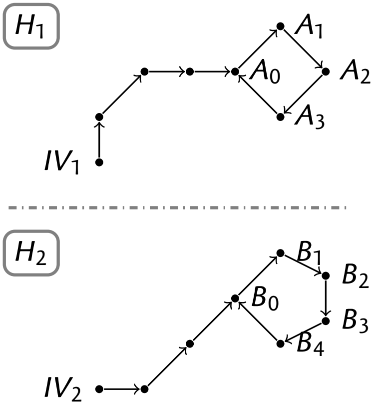


- Long messages with fixed block $M = [0]^\lambda$
- Use cyclic nodes as end-point:
  - H_1 cycle: $\{A_0, A_1, \dots\}$, H_2 cycle: $\{B_0, B_1, \dots\}$
- Reach arbitrary (A_j, B_k) by varying length λ
  - Chinese Remainder Theorem

Preimage search

- For random blocks r, match $\{h_1(A_j, r)\}$ and $\{h_2(B_k, r) \oplus \overline{H}\}$
- If there is a match (j, k), find λ reaching (A_j, B_k). Preimage is $M = [0]^\lambda \parallel r$
- Problem: finalization depend on message length (MD strengthening: $H_1(M) = g(h_1(A_j, r), \lambda)$)

Cycle-based attack



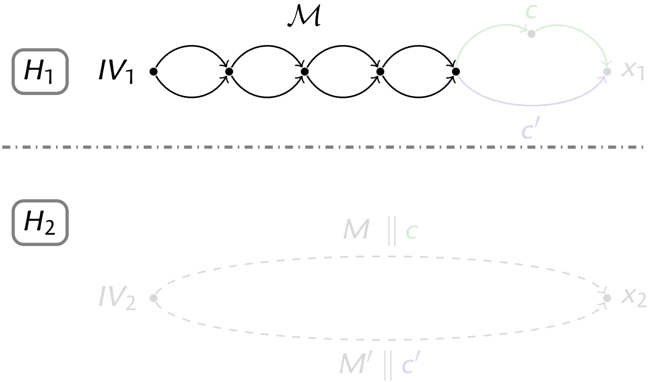
- Long messages with fixed block $M = [0]^\lambda$
- Use cyclic nodes as end-point:
  - H_1 cycle: $\{A_0, A_1, \ldots\}$, H_2 cycle: $\{B_0, B_1, \ldots\}$
- Reach arbitrary (A_j, B_k) by varying length λ
  - Chinese Remainder Theorem

Preimage search

- For random blocks r, match $\{h_1(A_j, r)\}$ and $\{h_2(B_k, r) \oplus \overline{H}\}$
- If there is a match (j, k), find λ reaching (A_j, B_k). Preimage is $M = [0]^\lambda \parallel r$
- Problem: finalization depend on message length (MD strengthening: $H_1(M) = g(h_1(A_j, r), \lambda)$)

Simultaneous expandable message

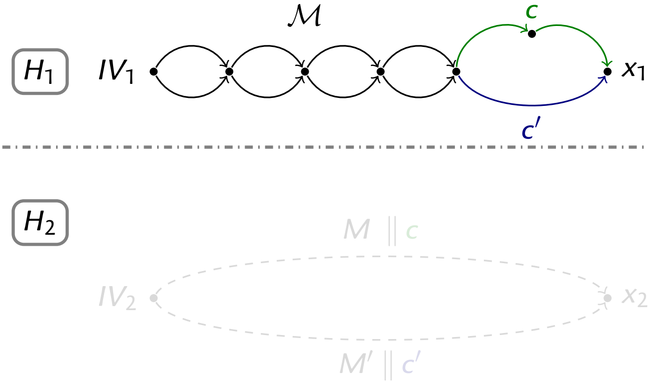
- Same problem as the preimage attack on long messages
- Same solution: expandable message. But we have two hash functions...



1. Build a $2^{n/2}$-multicollision for H_1.
2. Find a collision pair c/c' with different lengths. $\forall M \in \mathcal{M}$, $H_1(M \parallel c) = H_1(M \parallel c') = h_1$
3. Find $M, M' \in \mathcal{M}$ s.t. $H_2(M \parallel c) = H_2(M' \parallel c')$

Simultaneous expandable message

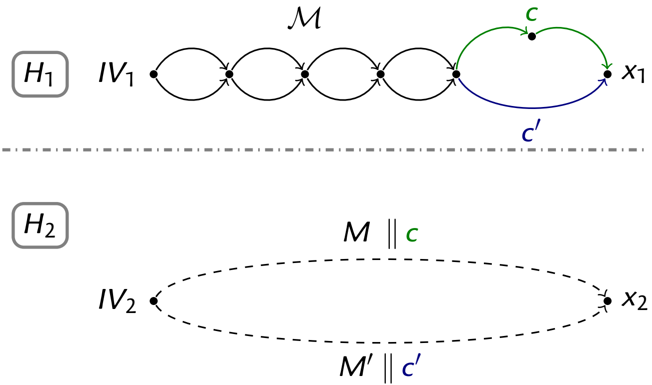
- Same problem as the preimage attack on long messages
- Same solution: expandable message. But we have two hash functions...



1. Build a $2^{n/2}$-multicollision for H_1.
2. Find a collision pair c/c' with different lengths.
   $\forall M \in \mathcal{M}$,
   $H_1(M \parallel c) = H_1(M \parallel c') = h_1$
3. Find $M, M' \in \mathcal{M}$ s.t.
   $H_2(M \parallel c) = H_2(M' \parallel c')$

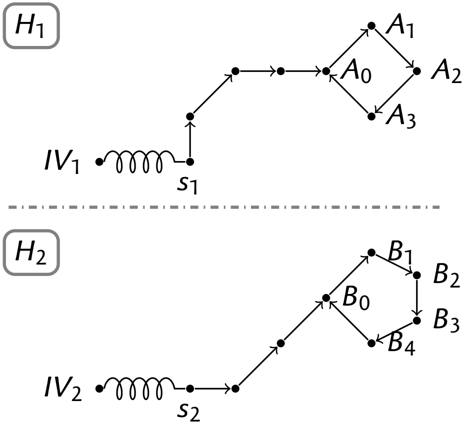
Simultaneous expandable message

- Same problem as the preimage attack on long messages
- Same solution: expandable message. But we have two hash functions...



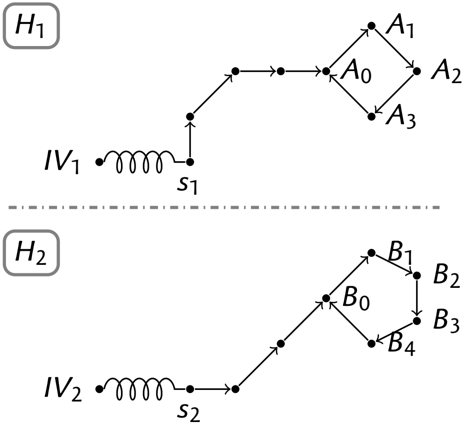
1. Build a $2^{n/2}$-multicollision for H_1.
2. Find a collision pair c/c' with different lengths.
   $\forall M \in \mathcal{M}$,
   $H_1(M \parallel c) = H_1(M \parallel c') = h_1$
3. Find $M, M' \in \mathcal{M}$ s.t.
   $H_2(M \parallel c) = H_2(M' \parallel c')$

Preimage attack using cycles



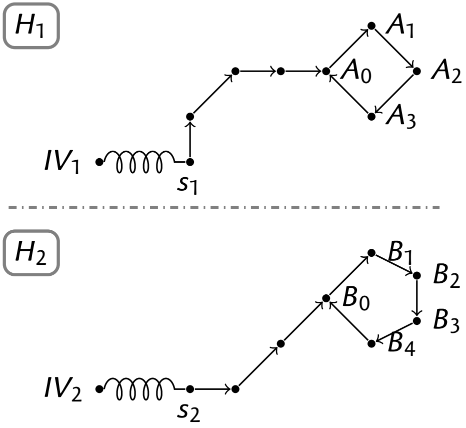
0. Fix length 2^t in advance: g is known
1. Build expandable message $\{\mathbf{M}_i\}$ (length 2^t)
   - Complexity: $\tilde{O}(2^{n/2} + 2^t)$
2. Preimage search for $\overline{H}$:
   - For random blocks r, match $\{g_1(h_1(A_j, r))\}$ and $\{g_2(h_2(B_k, r)) \oplus \overline{H}\}$
   - If there is a match (j, k): Find length λ from (s_1, s_2) to (A_j, B_k).
   - If $\lambda < 2^t$, select $\mathbf{M}_{2^t-\lambda}$ in expandable msg. Preimage is $M = \mathbf{M}_{2^t-\lambda} \parallel [0]^\lambda \parallel r$
   - Complexity: $\tilde{O}(2^{n/2} \times 2^{n-t})$
3. Optimal complexity: $\tilde{O}(2^{3n/4})$
   - $t = 3n/4$
   - Improvement to $2^{11n/9}$

Preimage attack using cycles



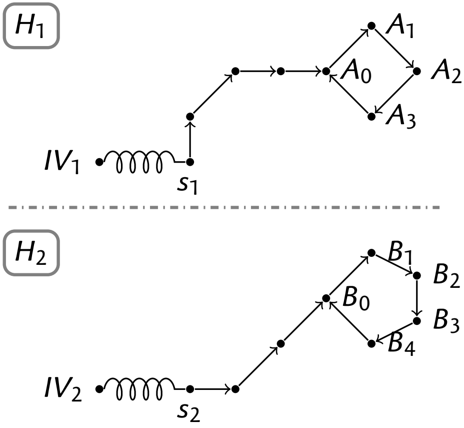
0. Fix length 2^t in advance: g is known
1. Build expandable message $\{\mathbf{M}_i\}$ (length 2^t)
   - Complexity: $\tilde{O}(2^{n/2} + 2^t)$
2. Preimage search for $\overline{H}$:
   - For random blocks r, match $\{g_1(h_1(A_j, r))\}$ and $\{g_2(h_2(B_k, r)) \oplus \overline{H}\}$
   - If there is a match (j, k): Find length λ from (s_1, s_2) to (A_j, B_k).
   - If $\lambda < 2^t$, select $\mathbf{M}_{2^t-\lambda}$ in expandable msg. Preimage is $M = \mathbf{M}_{2^t-\lambda} \parallel [0]^\lambda \parallel r$
   - Complexity: $\tilde{O}(2^{n/2} \times 2^{n-t})$
3. Optimal complexity: $\tilde{O}(2^{3n/4})$
   - $t = 3n/4$
   - Improvement to $2^{11n/9}$

Preimage attack using cycles



0. Fix length 2^t in advance: g is known
1. Build expandable message $\{\mathbf{M}_i\}$ (length 2^t)
   - Complexity: $\tilde{\mathcal{O}}(2^{n/2} + 2^t)$
2. Preimage search for $\overline{H}$:
   - For random blocks r, match $\{g_1(h_1(A_j, r))\}$ and $\{g_2(h_2(B_k, r)) \oplus \overline{H}\}$
   - If there is a match (j, k): Find length λ from (s_1, s_2) to (A_j, B_k).
   - If $\lambda < 2^t$, select $\mathbf{M}_{2^t-\lambda}$ in expandable msg. Preimage is $M = \mathbf{M}_{2^t-\lambda} \parallel [0]^\lambda \parallel r$
   - Complexity: $\tilde{\mathcal{O}}(2^{n/2} \times 2^{n-t})$
3. Optimal complexity: $\tilde{\mathcal{O}}(2^{3n/4})$
   - $t = 3n/4$
   - Improvement to $2^{11n/9}$

Preimage attack using cycles



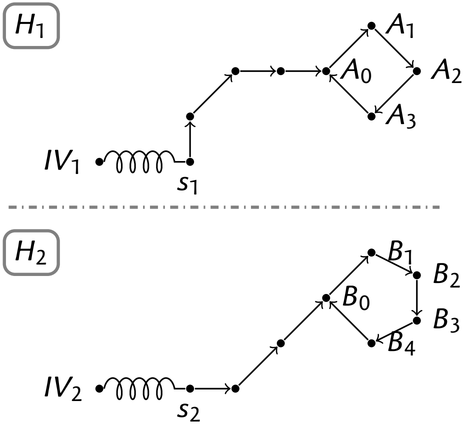
0. Fix length 2^t in advance: g is known

1. Build expandable message $\{\mathbf{M}_i\}$ (length 2^t)
   - Complexity: $\tilde{\mathcal{O}}(2^{n/2} + 2^t)$

2. Preimage search for $\overline{H}$:
   - For random blocks r, match $\{g_1(h_1(A_j, r))\}$ and $\{g_2(h_2(B_k, r)) \oplus \overline{H}\}$
   - If there is a match (j, k): Find length λ from (s_1, s_2) to (A_j, B_k).
   - If $\lambda < 2^t$, select $\mathbf{M}_{2^t-\lambda}$ in expandable msg. Preimage is $M = \mathbf{M}_{2^t-\lambda} \parallel [0]^\lambda \parallel r$
   - Complexity: $\tilde{\mathcal{O}}(2^{n/2} \times 2^{n-t})$

3. Optimal complexity: $\tilde{\mathcal{O}}(2^{3n/4})$
   - $t = 3n/4$
   - Improvement to $2^{11n/9}$

Preimage attack using cycles



0 Fix length 2^t in advance: g is known

1 Build expandable message $\{\mathbf{M}_i\}$ (length 2^t)
- Complexity: $\tilde{O}(2^{n/2} + 2^t)$

2 Preimage search for $\overline{H}$:
- For random blocks r, match $\{g_1(h_1(A_j, r))\}$ and $\{g_2(h_2(B_k, r)) \oplus \overline{H}\}$
- If there is a match (j, k): Find length λ from (s_1, s_2) to (A_j, B_k).
- If $\lambda < 2^t$, select $\mathbf{M}_{2^t-\lambda}$ in expandable msg. Preimage is $M = \mathbf{M}_{2^t-\lambda} \parallel [0]^\lambda \parallel r$
- Complexity: $\tilde{O}(2^{n/2} \times 2^{n-t})$

3 Optimal complexity: $\tilde{O}(2^{3n/4})$
- $t = 3n/4$
- Improvement to $2^{11n/9}$

Summary: Preimage attack for $H_1(M) \oplus H_2(M)$

Interchange structure

- Complexity $\tilde{\mathcal{O}}(2^{5n/6})$
- Works for the HAIFA mode
  - Finalization function, block counter at each round
- Short messages: length $\tilde{\mathcal{O}}(2^{n/3})$
- Can be extended to the sum of three or more (k) hash functions
  - Complexity $\mathcal{O}(n^{k-1} \cdot 2^{5n/6})$

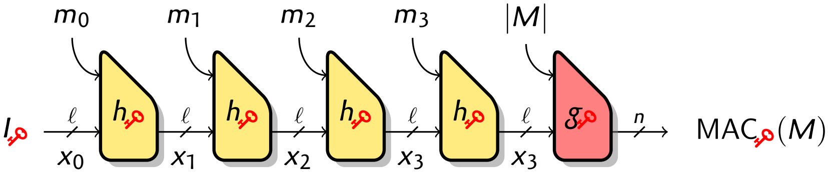
Cycles

- Complexity $\tilde{\mathcal{O}}(2^{11n/9})$
- Works for Merkle-Damgård mode
  - Finalization function, same function at each step
- Long messages: length $\tilde{\mathcal{O}}(2^{11n/9})$

- Works with $H_1(M) \boxplus H_2(M)$
  - Or any easy to invert operation
- Works with internal checksum (GOST)
  - Using pairs of blocks with constant sum

Outline

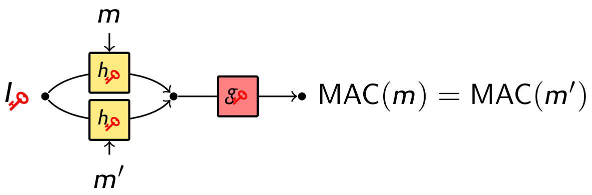
Iterative MACs



- ℓ-bit state
- n-bit MAC
- k-bit key

- Key-dependant initial value I
- Key-dependant state update h
- Key-dependant finalization, with message length g
- Example: ECBC-MAC

Generic Attack against Iterated Deterministic MACs

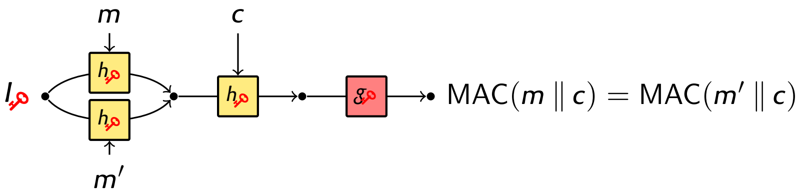


1. Find internal collisions [Preneel & van Oorschot '95]
   - Query $2^{n/2}$ random short messages
   - 1 internal collision expected, detected in the output
2. Query $t = \mathrm{MAC}(m \parallel c)$
3. $(m' \parallel c, t)$ is a forgery

MAC security

- Good MAC have birthday bound security proof
- Generic forgery attack with birthday complexity
- But different security loss after the birthday bound!

Generic Attack against Iterated Deterministic MACs

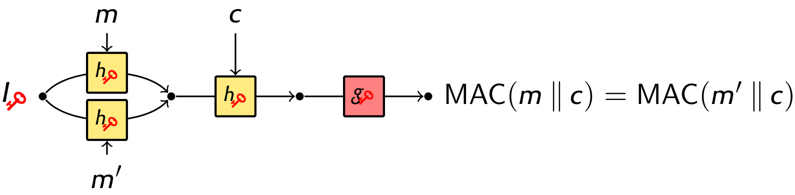


1. Find internal collisions
   [Preneel & van Oorschot '95]
   - Query $2^{n/2}$ random short messages
   - 1 internal collision expected, detected in the output
2. Query $t = \mathrm{MAC}(m \parallel c)$
3. $(m' \parallel c, t)$ is a forgery

MAC security

- Good MAC have birthday bound security proof
- Generic forgery attack with birthday complexity
- But different security loss after the birthday bound!

Generic Attack against Iterated Deterministic MACs



1. Find internal collisions

   [Preneel & van Oorschot '95]

   - Query $2^{n/2}$ random short messages
   - 1 internal collision expected, detected in the output

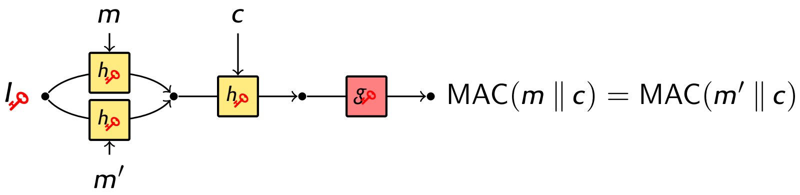
2. Query $t = \mathrm{MAC}(m \parallel c)$

3. $(m' \parallel c, t)$ is a forgery

MAC security

- Good MAC have birthday bound security proof
- Generic forgery attack with birthday complexity
- But different security loss after the birthday bound!

Generic Attack against Iterated Deterministic MACs



1. Find internal collisions

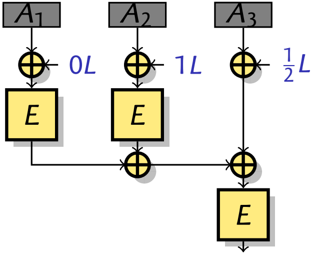
   [Preneel & van Oorschot '95]

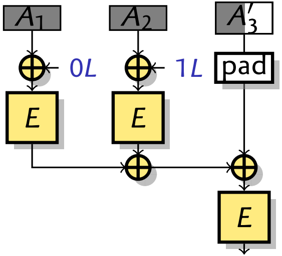
   - Query $2^{n/2}$ random short messages
   - 1 internal collision expected, detected in the output
2. Query $t = \mathrm{MAC}(m \parallel c)$
3. $(m' \parallel c, t)$ is a forgery

MAC security

- Good MAC have birthday bound security proof
- Generic forgery attack with birthday complexity
- But different security loss after the birthday bound!

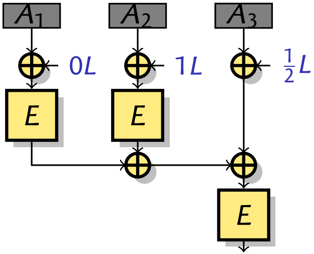
PMAC

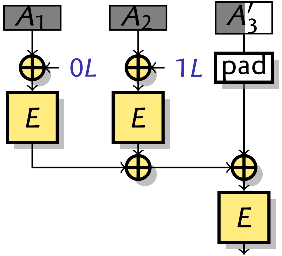




- PMAC: parallelisable block-cipher based MAC [Black & Rogaway '02]
  - Uses secret offsets to the block cipher input: $L = E_k(0)$

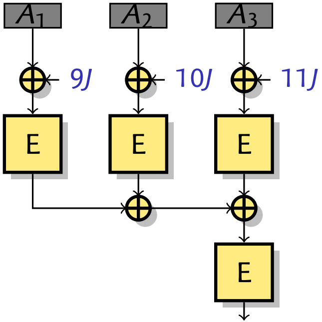
PMAC

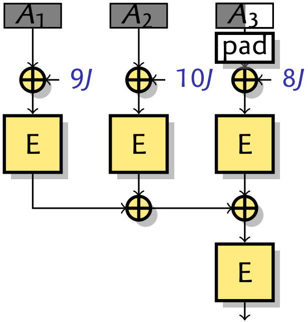




- Collision attack: two sets of messages [Lee & al '06]
- $A_x = [x]$, $|x| = 128$
  - Full block
  - $\mathrm{MAC}(A_x) = E([x] \oplus \frac{1}{2}L)$
- $B_y = [y]$, $|y| < 128$
  - Partial block
  - $\mathrm{MAC}(B_y) = E(\mathrm{pad}([y]))$
- Collision (A_x, B_y)?
  - The MAC collide iff $[x] \oplus \frac{1}{2}L = \mathrm{pad}([y])$
  - Deduce L
  - Universal forgery attack

AEZ





- AEZ uses a variant of PMAC [Hoang, Krovetz & Rogaway '15]
- A collision gives J: $[x] \oplus 9J = \text{pad}([y]) \oplus 8J$
- Key derivation (AEZ v2) $J = E_0(k)$
  - Instead of $L = E_k(0)$ in PMAC
- Collisions reveal the master key! [FLS, AC'15]

Security of block cipher based MACs

Proofs

CBC-MAC, PMAC, and AEZ have security proofs up to the birthday bound

Attacks

Effect of collision attacks with $2^{n/2}$ queries

- CBC-MAC: almost universal forgeries [Jia & al '09]
- PMAC: universal forgeries
- AEZ: key recovery

It is important to study generic attacks to understand the security of MACs

Building MACs from Hash functions

- Hash function should behave like a random function
  - Hard to find collisions, preimages
  - Hash can be used as a fingerprint
- Ideal hash function: Random Oracle

Hash-based MACs

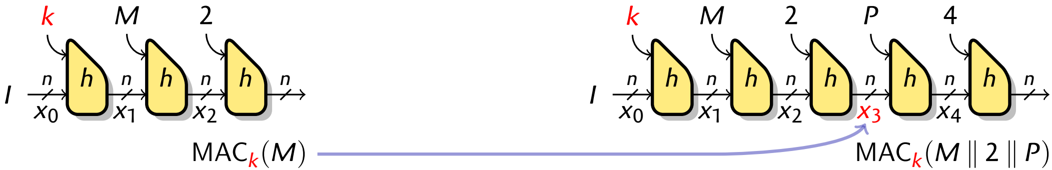
- Good hash functions (families) are indistinguishable from a random oracle up to $2^{n/2}$ queries
- Hashing message and key with a random oracle is a secure MAC
- Internal state size n larger than block ciphers
- Secret-prefix MAC: $\text{MAC}_k(M) = H(k \parallel M)$
- Secret-suffix MAC: $\text{MAC}_k(M) = H(M \parallel k)$

Secret-prefix MAC

Definition (Secret-prefix MAC)

$\mathrm{MAC}_k(M) = H(k \parallel M)$

- **Insecure with MD/SHA:** length-extension attack



  - Can compute $\mathrm{MAC}_k(M \parallel 2 \parallel P)$ from $\mathrm{MAC}_k(M)$ without k
  - Practical attack against Flickr API [Duong & Rizzo '09]

- **Secure** with modern hash functions (with finalization)
  - Recommend with sponges (Keccak)
  - Skein-MAC is essentially Secret-prefix MAC

Secret-suffix MAC (I)

Definition (Secret-suffix MAC)

$\mathrm{MAC}_k(M) = H(M \parallel k)$

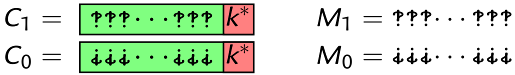
- Can be broken using offline collisions
  - Find a collision $H(M_1) = H(M_2)$ (with full blocks)
  - Since hash function are iterative, $H(M_1 \parallel k) = H(M_2 \parallel k)$
  - Existential forgery
- Finding a collision offline requires $2^{n/2}$ time
  - Almost practical for 128-bit hash functions (*e.g.* RIPEMD-128)
  - Cryptanalytic shortcuts (*e.g.* MD5)
- Finding a collision online require $2^{n/2}$ queries
  - Far from practical, easy to detect the attack

Secret-suffix MAC (II)

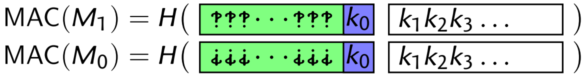
Definition (Secret-suffix MAC)

$\mathrm{MAC}_k(M) = H(M \parallel k)$

- Birthday key-recovery attack [Preneel & van Oorschot '96]
  1. Guess the first key byte as k^*
  2. Find a one-block hash collision (C_0, C_1) with $C_i = M_i \parallel k^*$



  3. Query $\mathrm{MAC}(M_1)$ and $\mathrm{MAC}(M_2)$



  4. If the MACs are equal, the guess was correct

- Practical attack when using MD5 (*e.g.* APOP) [L '07, Sasaki & al '08]
  - Using cryptanalytic shortcuts

Envelope MAC and Sandwich MAC

To avoid problems, use the key at the beginning and at the end

Definition (Envelope MAC)

$\mathrm{MAC}_k(M) = H(k \parallel M \parallel k)$

- Secure up to the birthday bound [Bellare, Canetti & Krawczyk '96]
- Key-recovery attack with complexity $2^{n/2}$ [Preneel & van Oorschot '96]

Definition (Sandwich MAC)

$\mathrm{MAC}_k(M) = H(\mathrm{pad}(k) \parallel \mathrm{pad}(M) \parallel k)$

- Secure up to the birthday bound [Yasuda '07]
- Key-recovery attack does no apply

The proof does not capture this important difference!

Envelope MAC and Sandwich MAC

To avoid problems, use the key at the beginning and at the end

Definition (Envelope MAC)

$$\text{MAC}_k(M) = H(k \parallel M \parallel k)$$

- Secure up to the birthday bound [Bellare, Canetti & Krawczyk '96]
- Key-recovery attack with complexity $2^{n/2}$ [Preneel & van Oorschot '96]

Definition (Sandwich MAC)

$$\text{MAC}_k(M) = H(\text{pad}(k) \parallel \text{pad}(M) \parallel k)$$

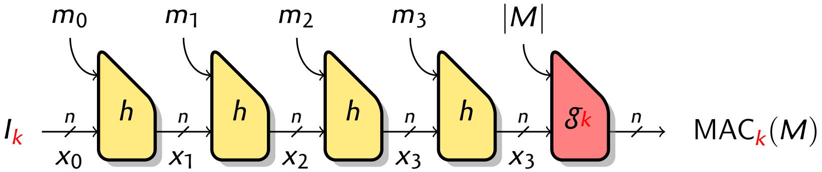
- Secure up to the birthday bound [Yasuda '07]
- Key-recovery attack does no apply

The proof does not capture this important difference!

HMAC

- HMAC has been designed by Bellare, Canetti, and Krawczyk in 1996
- Standardized by ANSI, IETF, ISO, NIST.
- Used in many applications:
  - To provide authentication:
    - SSL, IPSEC, ...
  - To provide identification:
    - Challenge-response protocols
    - CRAM-MD5 authentication in SASL, POP3, IMAP, SMTP, ...
  - For key-derivation:
    - HMAC as a PRF in IPsec
    - HMAC-based PRF in TLS

Hash-based MACs



- ℓ-bit chaining value
- n-bit output
- k-bit key

we focus on $\ell = n = k$

- Key-dependant initial value I_k
- Unkeyed compression function h
- Key-dependant finalization, with message length g_k

Security of hash-based MACS

- Security proofs up to the birthday bound
- Generic attacks based on collisions
  - Proof is tight for some security notions
    - Existential forgery
    - Distinguishing-R
- What is the remaining security above the birthday bound?
  - Generic distinguishing-H attack?
  - Generic state-recovery attack?
  - Generic universal forgery attack?
  - Generic key-recovery attack?

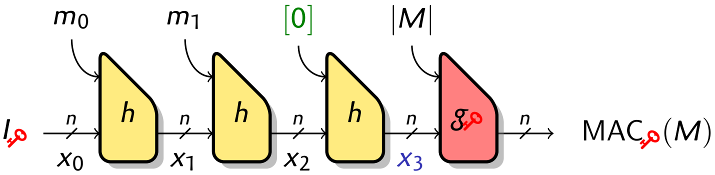
Outline

Bibliography

- Y. Naito, Y. Sasaki, L. Wang, K. Yasuda
  Generic State-Recovery and Forgery Attacks on ChopMD-MAC and on NMAC/HMAC
  IWSEC 2013
- G. Leurent, T. Peyrin, L. Wang
  New Generic Attacks against Hash-Based MACs
  Asiacrypt 2013
- I. Dinur, G. Leurent
  Improved Generic Attacks against Hash-Based MACs and HAIFA
  Crypto 2014

Multi-collision based attack

[Naito, Sasaki, Wang & Yasuda '13]



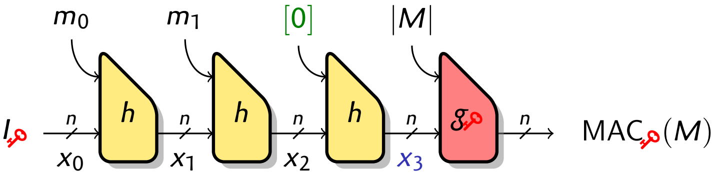
- Using a fixed message block, we apply a fixed function
- Starting point and ending point unknown because of the key

Can we detect properties of the function $h_{[0]} : x \mapsto h(x, 0)$?

- Use bias in the output of the compression function
  - Some outputs are more likely than others
  - With $2^{n-\varepsilon}$ work, find a value x_* with $\approx n$ preimages (offline)
- How to detect when this state is reached?

Multi-collision based attack

[Naito, Sasaki, Wang & Yasuda '13]



- Using a fixed message block, we apply a fixed function
- Starting point and ending point unknown because of the key

Can we detect properties of the function $h_{[0]} : x \mapsto h(x, 0)$?

- Use bias in the output of the compression function
  - Some outputs are more likely than others
  - With $2^{n-\varepsilon}$ work, find a value x_* with $\approx n$ preimages (offline)
- How to detect when this state is reached?

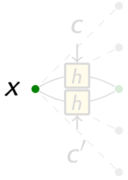
Building filters

Filters to compare online and online states

Test whether the state reached after processing M is equal to x

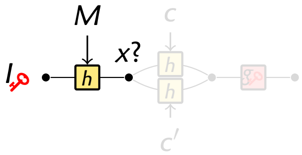
- Collisions are preserved by the finalization (for same-length messages)

1 Find a collision: $h(x,c) = h(x,c')$



Offline Structure

2 $\mathrm{MAC}(M \parallel c) \stackrel{?}{=} \mathrm{MAC}(M \parallel c')$



Online Structure

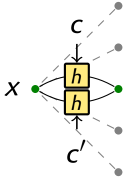
Building filters

Filters to compare online and online states

Test whether the state reached after processing M is equal to x

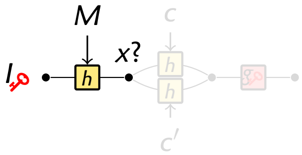
- Collisions are preserved by the finalization (for same-length messages)

1 Find a collision: $h(x,c) = h(x,c')$



Offline Structure

2 $\mathrm{MAC}(M \| c) \stackrel{?}{=} \mathrm{MAC}(M \| c')$



Online Structure

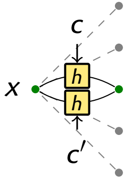
Building filters

Filters to compare online and online states

Test whether the state reached after processing M is equal to x

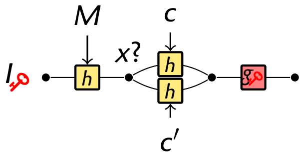
- Collisions are preserved by the finalization (for same-length messages)

1 Find a collision:
$h(x, c) = h(x, c')$



Offline Structure

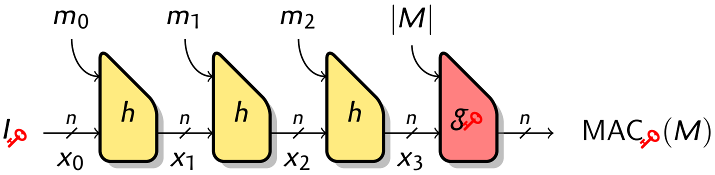
2 $\mathrm{MAC}(M \| c) \stackrel{?}{=} \mathrm{MAC}(M \| c')$



Online Structure

First state-recovery attack

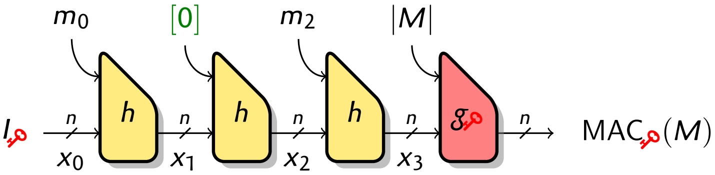
[Naito, Sasaki, Wang & Yasuda '13]



1. Fix a message block $m_1 = [0]$.
   With $2^{n-\varepsilon}$ work, find a value x_* with n preimages for $x \mapsto h(x, [0])$
2. Find a collision $h(x_*, c) = h(x_*, c')$
3. For random m_0, compare $\mathrm{MAC}(m_0 \| [0] \| c)$ and $\mathrm{MAC}(m_0 \| [0] \| c')$
   If they are equal, $x_2 = x_*$

First state-recovery attack

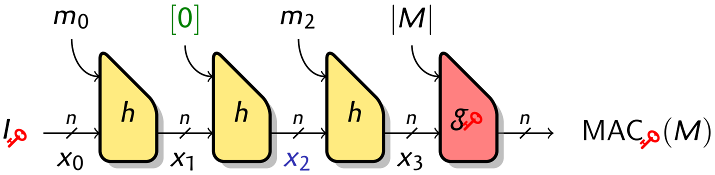
[Naito, Sasaki, Wang & Yasuda '13]



1. Fix a message block $m_1 = [0]$.
   With $2^{n-\varepsilon}$ work, find a value x_* with n preimages for $x \mapsto h(x, [0])$
2. Find a collision $h(x_*, c) = h(x_*, c')$
3. For random m_0, compare $\text{MAC}(m_0 \parallel [0] \parallel c)$ and $\text{MAC}(m_0 \parallel [0] \parallel c')$
   If they are equal, $x_2 = x_*$

First state-recovery attack

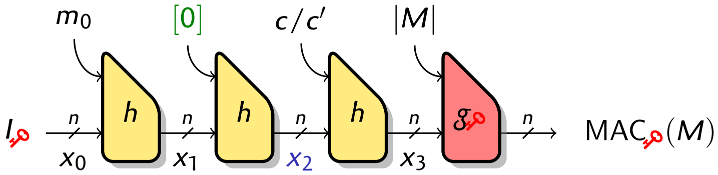
[Naito, Sasaki, Wang & Yasuda '13]



1. Fix a message block $m_1 = [0]$.
   With $2^{n-\varepsilon}$ work, find a value x_* with n preimages for $x \mapsto h(x, [0])$
2. Find a collision $h(x_*, c) = h(x_*, c')$
3. For random m_0, compare $\mathrm{MAC}(m_0 \parallel [0] \parallel c)$ and $\mathrm{MAC}(m_0 \parallel [0] \parallel c')$
   If they are equal, $x_2 = x_*$

First state-recovery attack

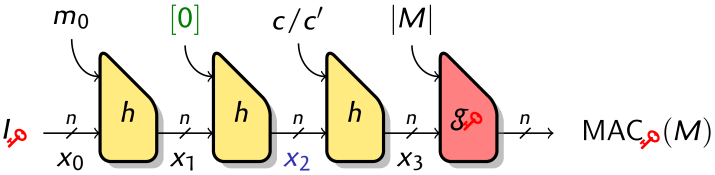
[Naito, Sasaki, Wang & Yasuda '13]



1. Fix a message block $m_1 = [0]$.
   With $2^{n-\varepsilon}$ work, find a value x_* with n preimages for $x \mapsto h(x, [0])$
2. Find a collision $h(x_*, c) = h(x_*, c')$
3. For random m_0, compare $\mathrm{MAC}(m_0 \,\|\, [0] \,\|\, c)$ and $\mathrm{MAC}(m_0 \,\|\, [0] \,\|\, c')$
   If they are equal, $x_2 = x_*$

First state-recovery attack

[Naito, Sasaki, Wang & Yasuda '13]



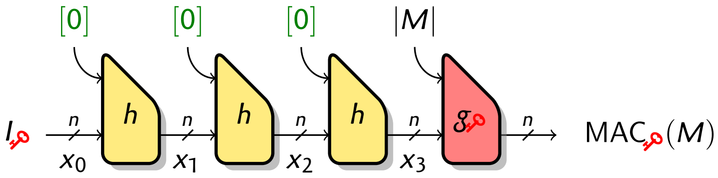
1. Fix a message block $m_1 = [0]$.
   With $2^{n-\varepsilon}$ work, find a value x_* with n preimages for $x \mapsto h(x, [0])$
2. Find a collision $h(x_*, c) = h(x_*, c')$
3. For random m_0, compare $\mathrm{MAC}(m_0 \| [0] \| c)$ and $\mathrm{MAC}(m_0 \| [0] \| c')$
   If they are equal, $x_2 = x_*$

Structure of state-recovery attacks

1. Identify special states easier to reach
2. Build filter for special states
3. Build messages to reach special states
   Test if special state reached using filters

- In this attack, steps 1 & 2 offline, step 3 online.

Cycle based attack

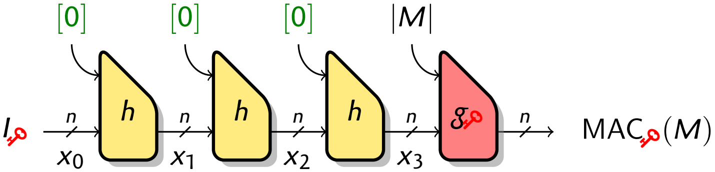


- Using a fixed message block, we iterate a fixed function
- Starting point and ending point unknown because of the key

Can we detect properties of the function $h_{[0]} : x \mapsto h(x, 0)$?

- Study the cycle structure of random mappings
- Used to attack HMAC in related-key setting [Peyrin, Sasaki & Wang, Asiacrypt 12]

Cycle based attack

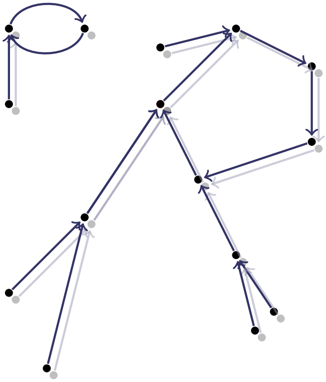


- Using a fixed message block, we iterate a fixed function
- Starting point and ending point unknown because of the key

Can we detect properties of the function $h_{[0]} : x \mapsto h(x, 0)$?

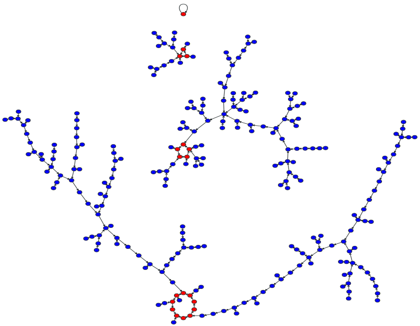
- Study the cycle structure of random mappings
- Used to attack HMAC in related-key setting [Peyrin, Sasaki & Wang, Asiacrypt 12]

Random Mappings



- Functional graph of a random mapping $x \to f(x)$
- Iterate f: $x_i = f(x_{i-1})$
- Collision after $\approx 2^{n/2}$ iterations
  - Cycles
- Trees rooted in the cycle
- Several components

Cycle structure

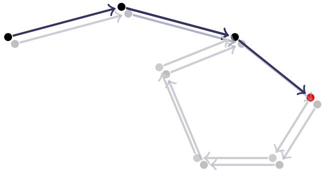


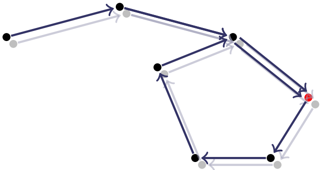
Expected properties of a random mapping over N points:

- # Components: $\frac{1}{2}\log N$
- # Cyclic nodes: $\sqrt{\pi N/2}$
- Tail length: $\sqrt{\pi N/8}$
- Cycle length: $\sqrt{\pi N/8}$
- Largest tree: $0.48N$
- Largest component: $0.76N$

Using the cycle length

1. Offline: find the cycle length L of the main component of $h_{[0]}$
2. Online: query $t = \mathrm{MAC}(r \parallel [0]^{2^{n/2}})$ and $t' = \mathrm{MAC}(r \parallel [0]^{2^{n/2}+L})$



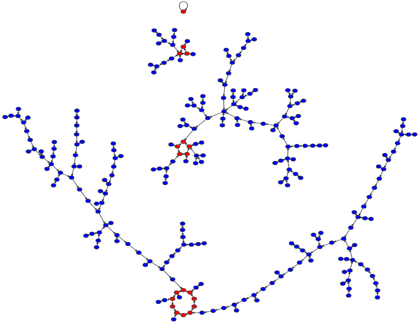


Success if

- The starting point is in the main component $p = 0.76$
- The cycle is reached with less than $2^{n/2}$ iterations $p \geq 0.5$

Randomize starting point

Cycle structure



Expected properties of a random mapping over N points:

- # Components: $\frac{1}{2}\log N$
- # Cyclic nodes: $\sqrt{\pi N/2}$
- Tail length: $\sqrt{\pi N/8}$
- Cycle length: $\sqrt{\pi N/8}$
- Largest tree: $0.48N$
- Largest component: $0.76N$

Using the cycle length

1. Offline: find the cycle length L of the main component of $h_{[0]}$
2. Online: query $t = \mathrm{MAC}(r \,\|\, [0]^{2^{n/2}})$ and $t' = \mathrm{MAC}(r \,\|\, [0]^{2^{n/2}+L})$

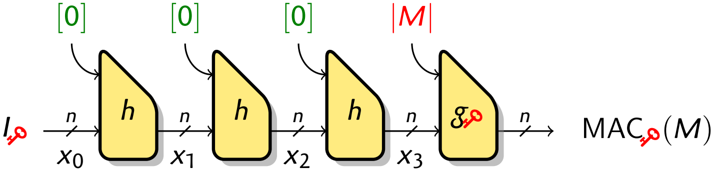
Success if

- The starting point is in the main component $p = 0.76$
- The cycle is reached with less than $2^{n/2}$ iterations $p \geq 0.5$

Randomize starting point

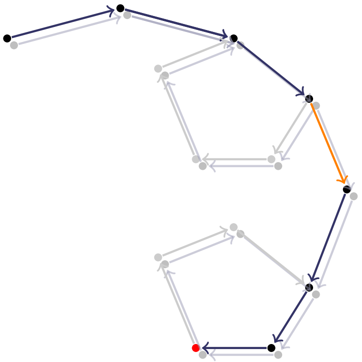
Dealing with the message length

Problem: most MACs use the message length.



Dealing with the message length

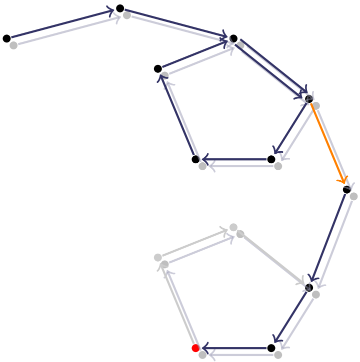
Solution: reach the cycle twice



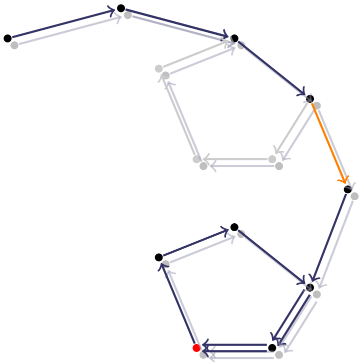
$$M = r \parallel [0]^{2^{n/2}} \parallel [1] \parallel [0]^{2^{n/2}}$$

Dealing with the message length

Solution: reach the cycle twice



$M_1 = r \parallel [0]^{2^{n/2}+L} \parallel [1] \parallel [0]^{2^{n/2}}$



$M_2 = r \parallel [0]^{2^{n/2}} \parallel [1] \parallel [0]^{2^{n/2}+L}$

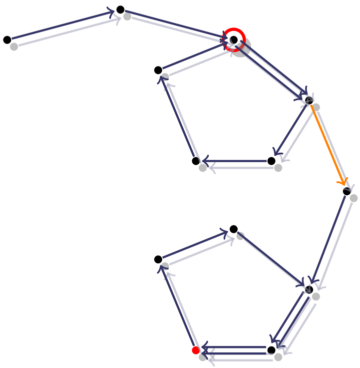
Distinguishing-H attack

1. Offline: find the cycle length L of the main component of $h_{[0]}$
2. Online: query
$$t = \mathrm{MAC}(r \parallel [0]^{2^{n/2}} \parallel [1] \parallel [0]^{2^{n/2}+L})$$
$$t' = \mathrm{MAC}(r \parallel [0]^{2^{n/2}+L} \parallel [1] \parallel [0]^{2^{n/2}})$$
3. If $t = t'$, then h is the compression function in the oracle

Analysis

- Complexity: $2^{n/2}$ compression function calls
- Success probability: $p \simeq 0.14$
  - Both starting point are in the main component $p = 0.76^2$
  - Both cycles are reached with less than $2^{n/2}$ iterations $p \geq 0.5^2$

State recovery attack



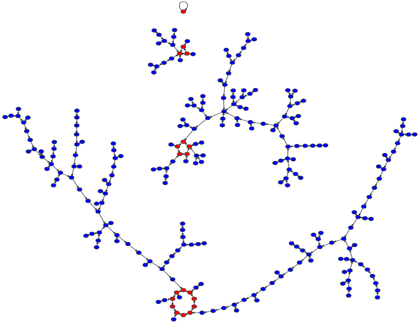
- Consider the first cyclic point
- With high pr., root of the giant tree

1. Offline: find cycle length L, and root of giant tree α
2. Online: Binary search for smallest z with collisions:
   $\mathrm{MAC}(r \parallel [0]^{z} \parallel [x] \parallel [0]^{2^{n/2}+L})$,
   $\mathrm{MAC}(r \parallel [0]^{z+L} \parallel [x] \parallel [0]^{2^{n/2}})$
3. State after $r \parallel [0]^{z}$ is α (with high pr.)

Analysis

- Complexity $2^{n/2} \times n \times \log(n)$

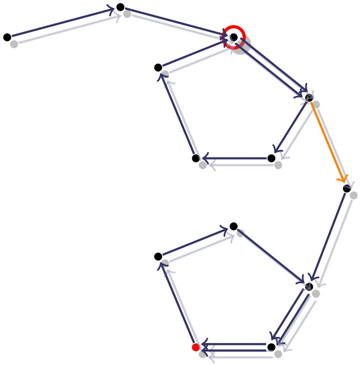
Cycle structure



Expected properties of a random mapping over N points:

- # Components: $\frac{1}{2}\log N$
- # Cyclic nodes: $\sqrt{\pi N/2}$
- Tail length: $\sqrt{\pi N/8}$
- Cycle length: $\sqrt{\pi N/8}$
- Largest tree: $0.48N$
- Largest component: $0.76N$

State recovery attack



- Consider the first cyclic point
- With high pr., root of the giant tree

1. Offline: find cycle length L, and root of giant tree α
2. Online: Binary search for smallest z with collisions:
   $\mathrm{MAC}(r \parallel [0]^{z} \parallel [x] \parallel [0]^{2^{n/2}+L})$,
   $\mathrm{MAC}(r \parallel [0]^{z+L} \parallel [x] \parallel [0]^{2^{n/2}})$
3. State after $r \parallel [0]^{z}$ is α (with high pr.)

Analysis

- Complexity $2^{n/2} \times n \times \log(n)$

Outline

Short message attacks

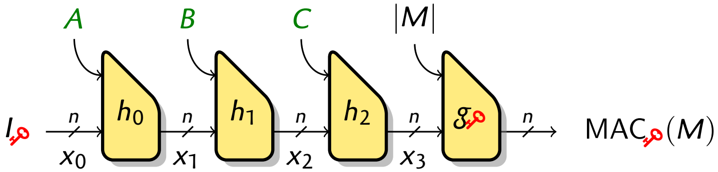
Limitations of cycle-based attacks

- Messages of length $2^{n/2}$ are not very practical...
  - SHA-1 and HAVAL limit the message length to 2^{64} bits
- Cycle detection inefficient with messages shorter than $L \approx 2^{n/2}$
  - Shorter cycles have a small component
  - Variant attack with higher complexity
- Not applicable to HAIFA hash functions

Compare with collision finding algorithms

- Pollard's rho algorithm use cycle detection
- Parallel collision search for van Oorschot and Wiener uses shorter chains

Chain-based attack

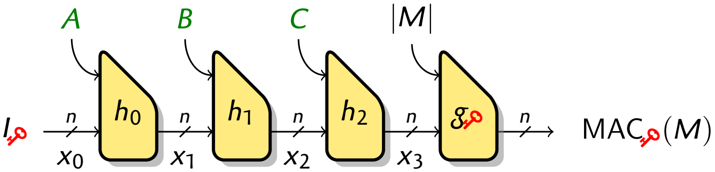


- Using a fixed message, we iterate a fixed sequence of function
- Starting point and ending point unknown because of the key

Can we detect properties of the iteration of fixed functions?

- Study the entropy loss

Chain-based attack

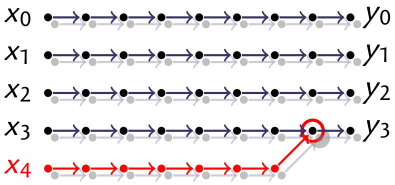


- Using a fixed message, we iterate a fixed sequence of function
- Starting point and ending point unknown because of the key

Can we detect properties of the iteration of fixed functions?

- Study the entropy loss

Collision finding with short chains



1. Compute chains $x \rightsquigarrow y$
   Stop when y distinguished
2. If $y \in \{y_i\}$, collision found

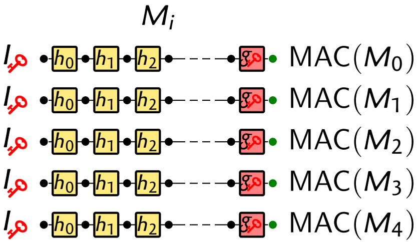
Theorem (Entropy loss)

Let $f_1, f_2, \ldots, f_{2^s}$ be a fixed sequence of random functions; the image of $g_{2^s} \triangleq f_{2^s} \circ \ldots \circ f_2 \circ f_1$ contains about 2^{n-s} points.

- Use these state as special states (instead of cycle entry point)

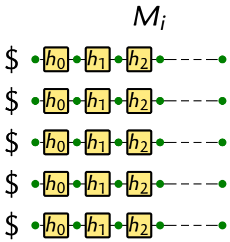
State-recovery attacks

- Send messages to the oracle



Online Structure

- Do some computations offline with the compression function

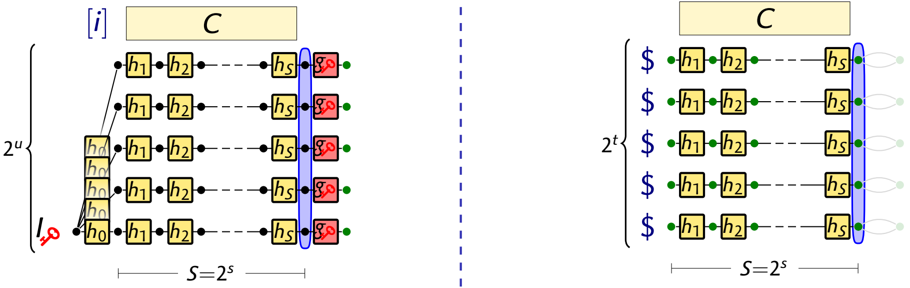


Offline Structure

- Match the sets of points?
  - How to test equality? Online chaining values unknown
  - How many equality test do we need?

First attempt

- Chains of length 2^s, with a fixed message C



Online Structure

Offline Structure

1. Evaluate 2^t chains offline
   Build filters for endpoints

$s + t + u = n$

2. Query 2^u message $M_i = [i] \parallel C$
   Test endpoints with filters

Cplx: 2^{s+t+u}

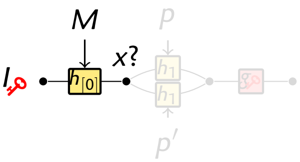
Building filters

Filters to compare online and online states

Test whether the state reached after processing M is equal to x

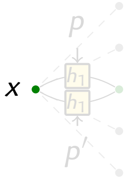
- Collisions are preserved by the finalization (for same-length messages)

2 $\mathrm{MAC}(M \| p) \stackrel{?}{=} \mathrm{MAC}(M \| p')$



Online Structure

1 Find a collision: $h(x,p) = h(x,p')$



Offline Structure

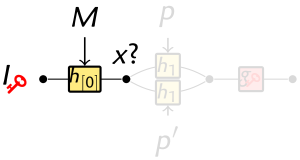
Building filters

Filters to compare online and online states

Test whether the state reached after processing M is equal to x

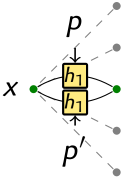
- Collisions are preserved by the finalization (for same-length messages)

2 $\mathrm{MAC}(M \| p) \stackrel{?}{=} \mathrm{MAC}(M \| p')$



Online Structure

1 Find a collision:
$h(x, p) = h(x, p')$



Offline Structure

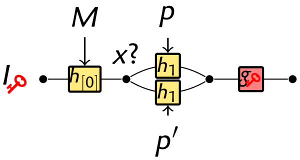
Building filters

Filters to compare online and online states

Test whether the state reached after processing M is equal to x

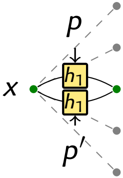
- Collisions are preserved by the finalization (for same-length messages)

2 $\mathrm{MAC}(M\|p) \stackrel{?}{=} \mathrm{MAC}(M\|p')$



Online Structure

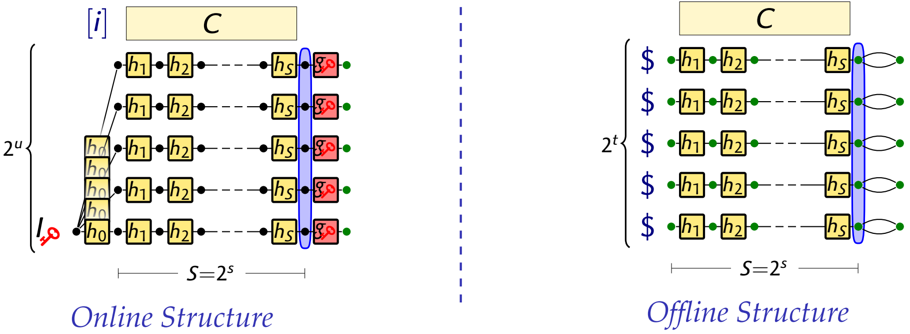
1 Find a collision:
$h(x,p) = h(x,p')$



Offline Structure

First attempt

- Chains of length 2^s, with a fixed message C



1. Evaluate 2^t chains offline
   Build filters for endpoints
2. Query 2^u message $M_i = [i] \| C$
   Test endpoints with filters

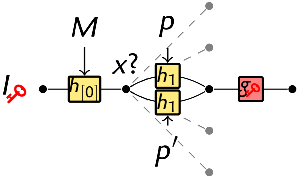
$s + t + u = n$

Cplx: 2^{s+t+u}

Online filters

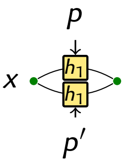
- Using the filters is too expensive.
- If we build filters online, using them is cheap.

1 Find p, p' s.t. $\mathrm{MAC}(M||p) = \mathrm{MAC}(M||p')$



Online Structure

2 $h(x, m) \stackrel{?}{=} h(x, m')$

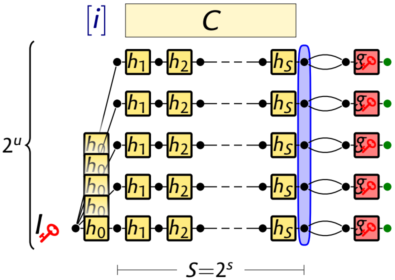


Offline Structure

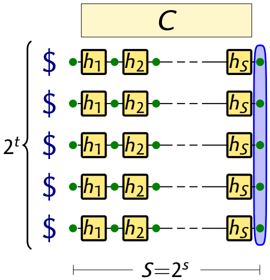
Cost	Build	Test
Offline filter	$2^{n/2}$	2^s
Online filter	$2^{n/2+s}$	1

First attack on HMAC-HAIFA

- Chains of length 2^s, with a fixed message C



Online Structure



Offline Structure

1. Query 2^u message $M_i = [i] \parallel C$
   Build filters for M_i
2. Evaluate 2^t chains offline
   Test endpoints with filters

$s + t + u = n$

Cplx: $2^{s+u+n/2}$

Cplx: 2^{t+s}

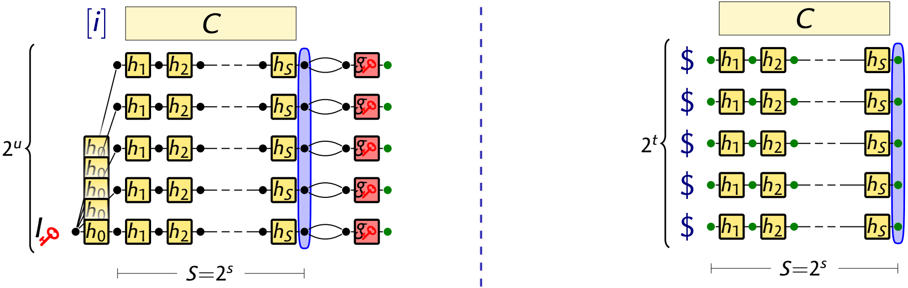
Cplx: 2^{t+u}

Optimal complexity

2^{n-s}, for $s \leq n/6$
(using $u = s$)
Minimum: $2^{5n/6}$

First attack on HMAC-HAIFA

- Chains of length 2^s, with a fixed message C



Online Structure *Offline Structure*

1. Query 2^u message $M_i = [i] \parallel C$
   Build filters for M_i
2. Evaluate 2^t chains offline
   Test endpoints with filters

$s + t + u = n$

Cplx: $2^{s+u+n/2}$

Cplx: 2^{t+s}

Cplx: 2^{t+u}

Optimal complexity

2^{n-s}, for $s \leq n/6$
(using $u = s$)
Minimum: $2^{5n/6}$

Diamond filters

- Building filers is a bottleneck.
- Can we amortize the cost of building many filters?

Diamond structure [Kelsey & Kohno, EC'06]

Herd N initial states to a common state

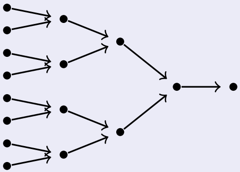
- Try $\approx 2^{n/2}/\sqrt{N}$ msg from each state.
- Whp, the initial states can be paired
- Repeat...

Total $\approx \sqrt{N} \cdot 2^{n/2}$

Diamond filters

- Building filers is a bottleneck.
- Can we amortize the cost of building many filters?

Diamond structure [Kelsey & Kohno, EC'06]



Herd N initial states to a common state

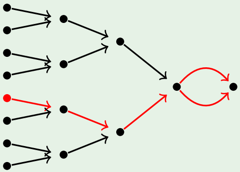
- Try $\approx 2^{n/2}/\sqrt{N}$ msg from each state.
- Whp, the initial states can be paired
- Repeat...

Total $\approx \sqrt{N} \cdot 2^{n/2}$

Diamond filters

- Building filers is a bottleneck.
- Can we amortize the cost of building many filters?

Diamond filter



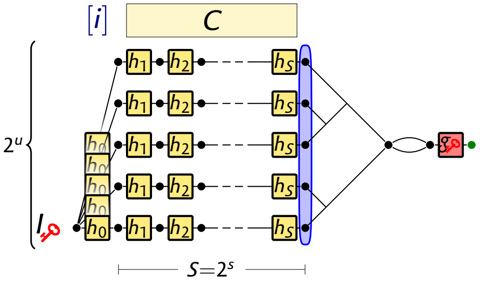
1. Build a diamond structure
2. Build a collision filter for the final state

- Can also be built online

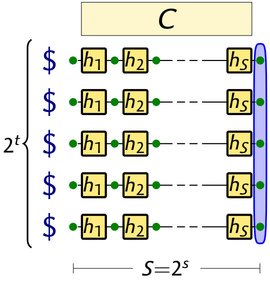
- Building N offline filters: $\sqrt{N} \cdot 2^{n/2}$ rather than $N \cdot 2^{n/2}$
- Building N online filters: $\sqrt{N} \cdot 2^{n/2+s}$ rather than $N \cdot 2^{n/2+s}$

Improved attack on HMAC-HAIFA

- Chains of length 2^s, with a fixed message C



Online Structure



Offline Structure

1. Query 2^u message $M_i = [i] \parallel C$
   Build diamond filter for M_i
2. Evaluate 2^t chains offline
   Test endpoints with filters

$s + t + u = n$

Cplx: $2^{s+u/2+n/2}$

Cplx: 2^{t+s}

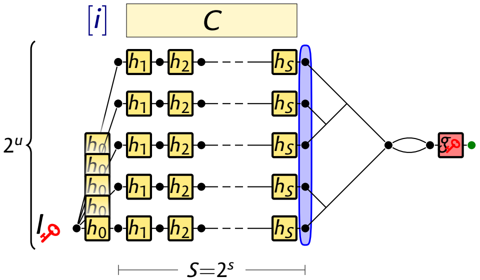
Cplx: 2^{t+u}

Optimal complexity

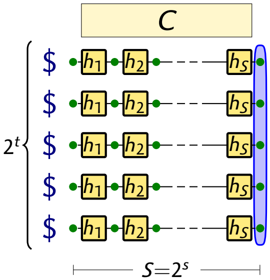
2^{n-s}, for $s \leq n/5$
(using $u = s$)
Minimum: $2^{4n/5}$

Improved attack on HMAC-HAIFA

- Chains of length 2^s, with a fixed message C



Online Structure



Offline Structure

1. Query 2^u message $M_i = [i] \parallel C$
   Build diamond filter for M_i
2. Evaluate 2^t chains offline
   Test endpoints with filters

$s + t + u = n$

Cplx: $2^{s+u/2+n/2}$

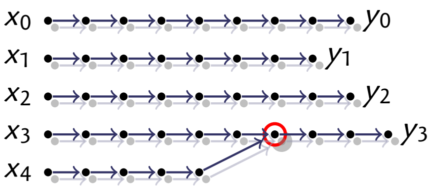
Cplx: 2^{t+s}

Cplx: 2^{t+u}

Optimal complexity

2^{n-s}, for $s \leq n/5$
(using $u = s$)
Minimum: $2^{4n/5}$

Improvement using collisions (fixed function)



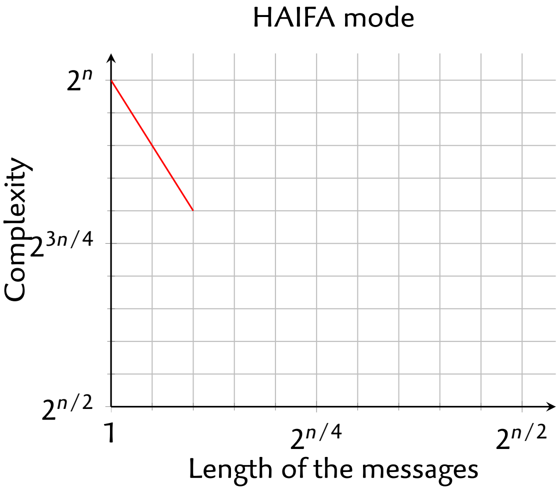
1. Compute chains $x \rightsquigarrow y$
   Stop when y distinguished
2. If $y \in \{y_i\}$, collision found

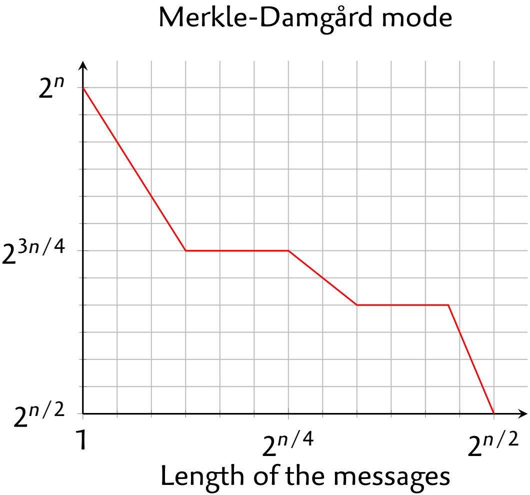
Theorem (Entropy loss for collisions)

Let $\hat{x}$ and $\hat{y}$ be two collisions found using chains of length 2^s, with a fixed n-bit random function f. Then $\Pr[\hat{x} = \hat{y}] = \Theta(2^{2s-n})$.

- Use the collisions as special states (instead of cycle entry point)

Trade-offs for state-recovery attacks





Outline

Bibliography

T. Peyrin, L. Wang
Generic Universal Forgery Attack on Iterative Hash-Based MACs
EUROCRYPT 2014

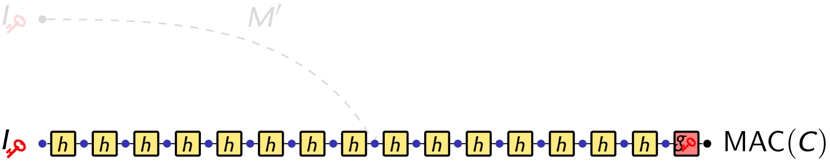
J. Guo, T. Peyrin, Y. Sasaki, L. Wang
Updates on Generic Attacks against HMAC and NMAC
CRYPTO 2014

I. Dinur, G. Leurent
Improved Generic Attacks against Hash-Based MACs and HAIFA
CRYPTO 2014

Universal forgery attack

- Given a challenge message C, compute $\mathrm{MAC}(C)$
  - $\mathrm{len}(C) = 2^s$
  - Oracle access to the MAC, can't ask $\mathrm{MAC}(C)$
- Study internal states for the computation of $\mathrm{MAC}(C)$
  - Unknown because of initial key and final key

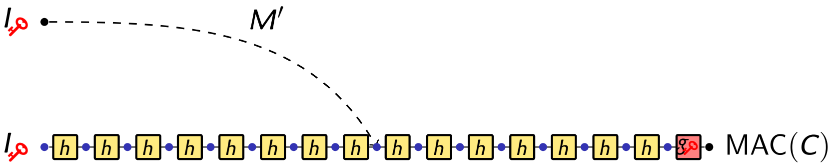
1. Build a different message reaching same states
2. Query $\mathrm{MAC}(M')$, use as forgery



Universal forgery attack

- Given a challenge message C, compute $\mathrm{MAC}(C)$
  - $\mathrm{len}(C) = 2^s$
  - Oracle access to the MAC, can't ask $\mathrm{MAC}(C)$
- Study internal states for the computation of $\mathrm{MAC}(C)$
  - Unknown because of initial key and final key

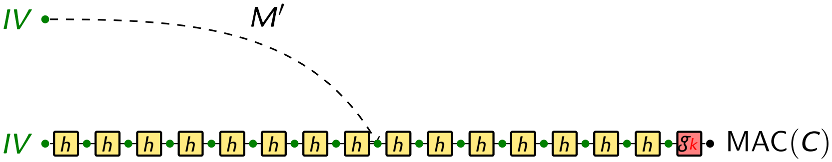
1. Build a different message reaching same states
2. Query $\mathrm{MAC}(M')$, use as forgery



UF against secret-suffix MAC

- Secret-suffix has no key at the beginning
  - All internal states for challenge message are known!
- Long-message second-preimage attack [Kelsey & Schneier '05]
  - $H(M) = H(C) \implies MAC(M) = H(M \parallel k) = H(C \parallel k) = MAC(C)$

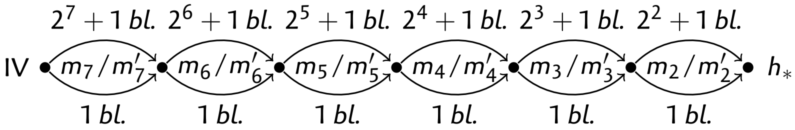
1. Build a expandable message — Cplx: $2^{n/2}$
2. Find a connexion from the IV to the target states — Cplx: 2^{n-s}
3. Select expandable message

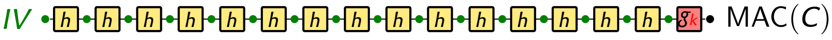


UF against secret-suffix MAC

- Secret-suffix has no key at the beginning
  - All internal states for challenge message are known!
- Long-message second-preimage attack [Kelsey & Schneier '05]
  - $H(M) = H(C) \implies MAC(M) = H(M \,\|\, k) = H(C \,\|\, k) = MAC(C)$

1. Build a expandable message Cplx: $2^{n/2}$

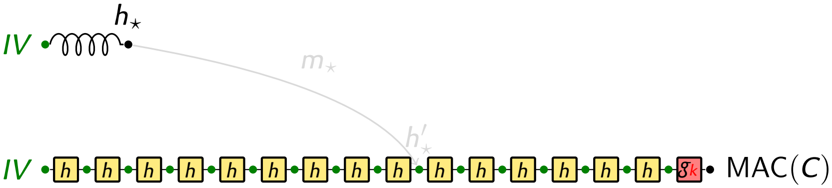




UF against secret-suffix MAC

- Secret-suffix has no key at the beginning
  - All internal states for challenge message are known!
- Long-message second-preimage attack [Kelsey & Schneier '05]
  - $H(M) = H(C) \implies MAC(M) = H(M \| k) = H(C \| k) = MAC(C)$

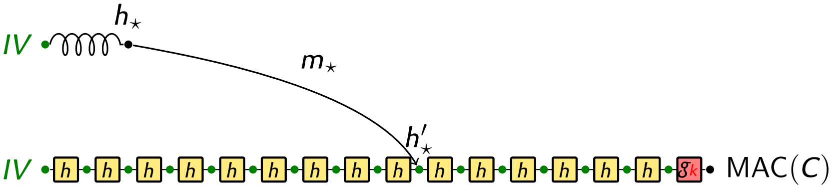
1. Build a expandable message Cplx: $2^{n/2}$
2. Find a connexion from $x_\star$ to the target states Cplx: 2^{n-s}
3. Select expandable message



UF against secret-suffix MAC

- Secret-suffix has no key at the beginning
  - All internal states for challenge message are known!
- Long-message second-preimage attack [Kelsey & Schneier '05]
  - $H(M) = H(C) \implies MAC(M) = H(M \| k) = H(C \| k) = MAC(C)$

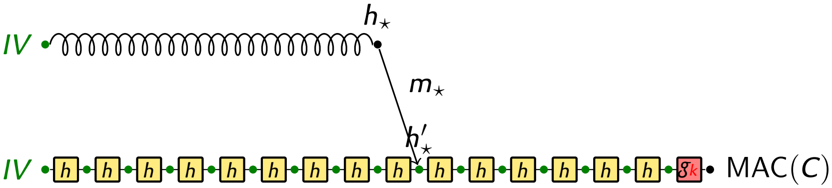
1. Build a expandable message — Cplx: $2^{n/2}$
2. Find a connexion from $x_\star$ to the target states — Cplx: 2^{n-s}
3. Select expandable message



UF against secret-suffix MAC

- Secret-suffix has no key at the beginning
  - All internal states for challenge message are known!
- Long-message second-preimage attack [Kelsey & Schneier '05]
  - $H(M) = H(C) \implies MAC(M) = H(M \parallel k) = H(C \parallel k) = MAC(C)$

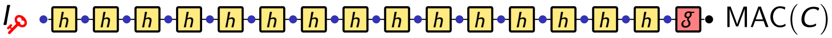
1. Build a expandable message Cplx: $2^{n/2}$
2. Find a connexion from $x_\star$ to the target states Cplx: 2^{n-s}
3. Select expandable message



UF against secret-prefix MAC

- Secret-suffix has no key at the end
  - Finalization function is known!

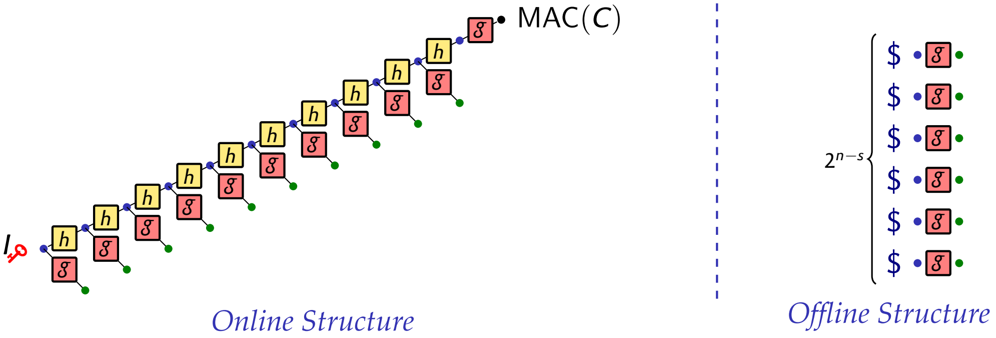
1. Query the MAC of $C[i:]$ (truncated to i blocks) Cplx: $2^{2 \cdot s}$
2. Evaluate the finalization function on 2^{n-s} states Cplx: 2^{n-s}
3. Find a match, compute MAC



UF against secret-prefix MAC

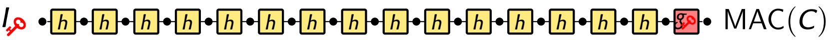
- Secret-suffix has no key at the end
  - Finalization function is known!

1. Query the MAC of $C[i:]$ (truncated to i blocks) — Cplx: $2^{2\cdot s}$
2. Evaluate the finalization function on 2^{n-s} states — Cplx: 2^{n-s}
3. Find a match, compute MAC



UF attack against hash-based MAC

- Combine both techniques
  1. Recover an internal state of the challenge
  2. Use second-preimage attack with known state
- Hard part is to recover an internal state
- Extract information about the challenge state through g
  - Compute distance to cycle
  - Use entropy loss of iterations

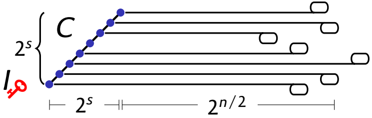


Outline

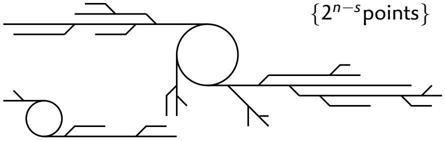
Using cycles

Main idea

- Measure the distance from challenge point to cycle in $h_{[0]}$
  - Add zero blocks after the challenge
- Match with offline points with known distance



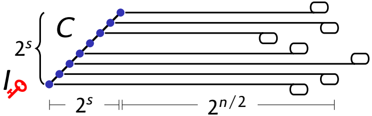
Online Structure



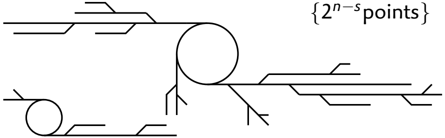
Offline Structure

Using cycles

1. (online) For each challenge state, use binary search to find distance
   $\mathrm{MAC}(C_{|i} \parallel 0^{d+L} \parallel 1 \parallel 0^{2^{n/2}}) \stackrel{?}{=} \mathrm{MAC}(C_{|i} \parallel 0^{d} \parallel 1 \parallel 0^{2^{n/2+L}})$
2. (offline) Build a structure with 2^{n-s} points with known distance.
3. (offline) Match the challenge states and the offline structure
4. (online) Test candidates at the right distance.



Online Structure

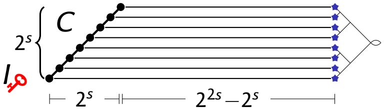


Offline Structure

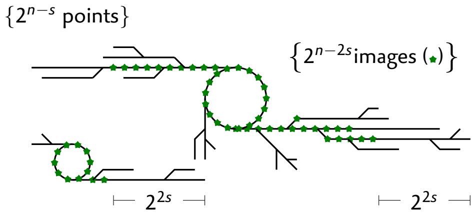
Using chains

Main idea

- Add a sequence of fixed message blocks to reduce image space
- Match in the reduced space



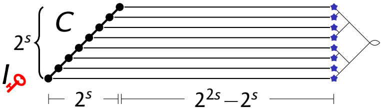
Online Structure



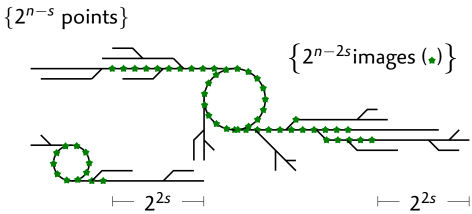
Offline Structure

Using chains

1. (online) Query messages $M_i = C_{|i} \parallel [0]^{2^{2s}-i}$.
   Build diamond filter for endpoints Y
2. (offline) Build a structure with 2^{n-s} points.
   Consider 2^{2s}-images X. $|X| \leq 2^{n-2s}$
3. (offline) Match X and Y.
4. (offline) For each match, find preimages as candidates.



Online Structure



Offline Structure

Universal forgery attacks: summary

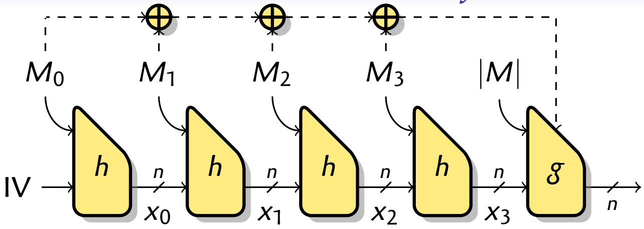
Universal forgery attacks

- It is possible to perform a generic universal forgery attack
- Best attack so far: 2^{n-s}, with $s \leq n/4$ ($2^{3n/4}$ with $s = n/4$)

- Using distance to the cycle: query length $2^{n/2}$
  - Complexity $2^{n-s}, s \leq n/6$ [Peyrin & Wang, EC '14]
    Optimal: $2^{5n/6}$, with $s = 2^{n/6}$
  - Complexity $2^{n-s}, s \leq n/4$ [Guo, Peyrin, Sasaki & Wang, CR '14]
    Optimal: $2^{3n/4}$, with $s = 2^{n/4}$
- Later attack using chains: shorter query length 2^t
  - Complexity 2^{n-s} $, s \leq$ $n/7, t = 2s$ [Dinur & L, CR '14]
    Optimal: $2^{6n/7}$, with $s = 2^{n/7}, t = 2n/7$
  - Complexity $2^{n-s/2}, s \leq 2n/5, t = s$ [Dinur & L, CR '14]
    Optimal: $2^{4n/5}$, with $s = 2^{2n/5}, t = 2n/5$

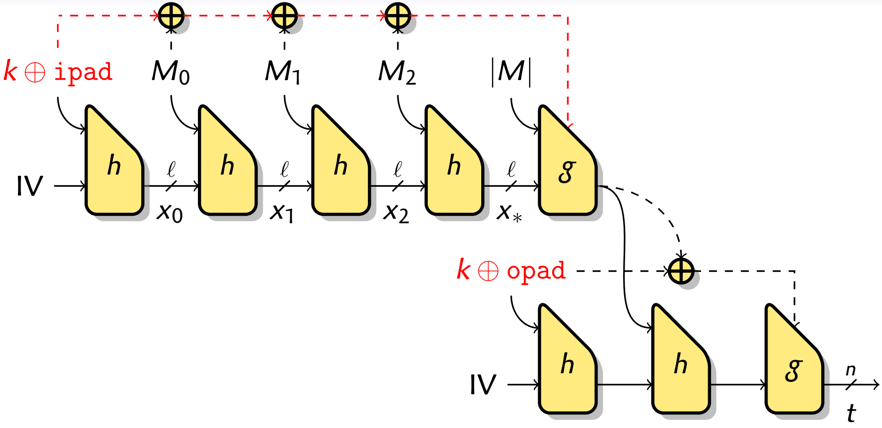
Outline

GOST hash functions



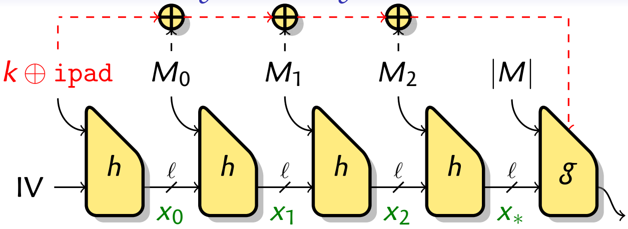
- Family of Russian standards
  - GOST-1994: $n = \ell = 256$
  - GOST-2012: $n \leq \ell = 512$, HAIFA mode (aka Streebog)
- GOST and HMAC-GOST standardized by IETF
- Checksum (dashed lines)
  - Larger state should increase the security

HMAC-GOST



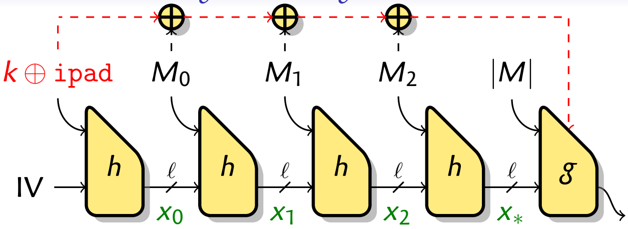
- In HMAC, key-dependant value used after the message
  - Related-key attacks on the last block

Key recovery attack on HMAC-GOST



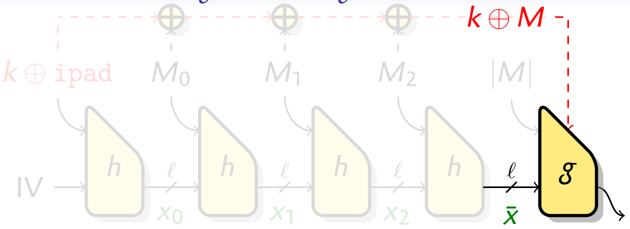
1. Recover the state of a short message
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_*
3. Query messages, detect collisions $g(\bar{x}, k \oplus M) = g(\bar{x}, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(\bar{x}, y) = g(\bar{x}, y')$ offline
   Store $(x \oplus y', y)$ for $2^{\ell/2}$ collisions
5. Detect match $M \oplus M' = y \oplus y'$. With high probability $M \oplus k = y$

Key recovery attack on HMAC-GOST



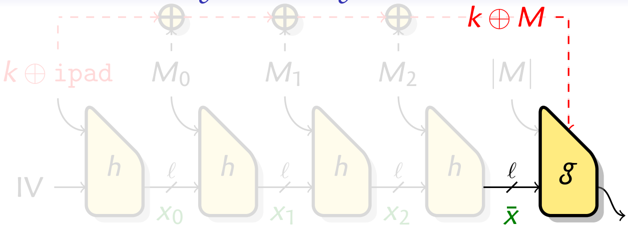
1. Recover the state of a short message
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_*
3. Query messages, detect collisions $g(\bar{x}, k \oplus M) = g(\bar{x}, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(\bar{x}, y) = g(\bar{x}, y')$ offline
   Store $(x \oplus y', y)$ for $2^{\ell/2}$ collisions
5. Detect match $M \oplus M' = y \oplus y'$. With high probability $M \oplus k = y$

Key recovery attack on HMAC-GOST



1. Recover the state of a short message
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_*
3. Query messages, detect collisions $g(\bar{x}, k \oplus M) = g(\bar{x}, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(\bar{x}, y) = g(\bar{x}, y')$ offline
   Store $(x \oplus y', y)$ for $2^{\ell/2}$ collisions
5. Detect match $M \oplus M' = y \oplus y'$. With high probability $M \oplus k = y$

Key recovery attack on HMAC-GOST

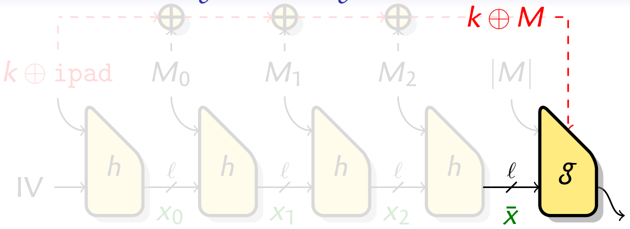


1. Recover the state of a short message
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_*
3. Query messages, detect collisions $g(\bar{x}, k \oplus M) = g(\bar{x}, k \oplus M')$

   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(\bar{x}, y) = g(\bar{x}, y')$ offline

   Store $(x \oplus y', y)$ for $2^{\ell/2}$ collisions
5. Detect match $M \oplus M' = y \oplus y'$. With high probability $M \oplus k = y$

Key recovery attack on HMAC-GOST



1. Recover the state of a short message
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_*
3. Query messages, detect collisions $g(\bar{x}, k \oplus M) = g(\bar{x}, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(\bar{x}, y) = g(\bar{x}, y')$ offline
   Store $(x \oplus y', y)$ for $2^{\ell/2}$ collisions
5. Detect match $M \oplus M' = y \oplus y'$. With high probability $M \oplus k = y$

Complexity

Surprising result

The checksum actually make the hash function weaker!

- HMAC-GOST-1994 is weaker than HMAC-SHA256
- HMAC-GOST-2012 is weaker than HMAC-SHA512

It is important to recover the state of a short message

- For GOST-1994, we can recover the state of a short message from a longer one using padding tricks — Total complexity $2^{3\ell/4}$
- For GOST-2012, we use an advanced attack with message length $2^{\ell/10}$ — Total complexity $2^{4\ell/5}$

Attack complexity

Function	Mode	ℓ	s	St. rec.	Univ. F	K. rec.
SHA-1	MD	160	2^{55}	2^{107}	2^{132}	
SHA-224	MD	256	2^{55}	2^{192}		
SHA-256	MD	256	2^{55}	2^{192}	2^{228}	
SHA-512	MD	512	2^{118}	2^{384}	2^{453}	
HAVAL	MD	256	2^{54}	2^{192}	2^{229}	
WHIRLPOOL	MD	512	2^{247}	2^{283}	2^{446}	
BLAKE-256	HAIFA	256	2^{55}	2^{213}		
BLAKE-512	HAIFA	512	2^{118}	2^{419}		
Skein-512	HAIFA	512	2^{90}	2^{419}		
GOST-94	MD+σ	256	∞	2^{128}	2^{192}	2^{192}
Streebog	HAIFA+σ	512	∞	2^{419}	2^{419}	2^{419}

Conclusion

Surprising results with generic attacks

- The sum of two hash functions is weaker than each individual function
- The use of a checksum makes HMAC-GOST weaker
- Constructions with similar proof can have different security beyond the bound of the proof
  - Generic key-recovery for envelope-MAC, AEZ, HMAC-GOST
- Don't assume too much after the security bound of the proof

Gaps between proofs and attacks!

- Better generic attacks?
- Better proofs?

Thanks

Questions?