New Generic Attacks on Hash-based MACs

Gaëtan Leurent, Thomas Peyrin, Lei Wang

UCL Crypto Group, Belgium
Nanyang Technological University, Singapore

TCCM-CACR 2013
Message Authentication Codes

- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice uses a key $k$ to compute a tag:
  \[ t = MAC_k(M) \]
- Bob verifies the tag with the same key $k$:
  \[ t = MAC_k(M) \]
- Symmetric equivalent to digital signatures
Message Authentication Codes

- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice uses a key $k$ to compute a tag:
  \[ t = \text{MAC}_k(M) \]
- Bob verifies the tag with the same key $k$:
  \[ t = \text{MAC}_k(M) \]
- Symmetric equivalent to digital signatures
New Generic Attacks on Hash-based MACs

Introduction

New attacks

Key-recovery Attack on HMAC-GOST

Message Authentication Codes

Alice

Bob

- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice uses a key $k$ to compute a tag:
  - Bob verifies the tag with the same key $k$:
  - Symmetric equivalent to digital signatures

$t = \text{MAC}_k(M)$
$t \neq \text{MAC}_k(M)$
Message Authentication Codes

- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice uses a key $k$ to compute a tag:
  \[ t = \text{MAC}_k(M) \]
- Bob verifies the tag with the same key $k$:
  \[ t = \text{MAC}_k(M) \]
- Symmetric equivalent to digital signatures
Message Authentication Codes

- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice uses a key $k$ to compute a tag: $t = \text{MAC}_k(M)$
- Bob verifies the tag with the same key $k$: $t = \text{MAC}_k(M)$
- Symmetric equivalent to digital signatures
Introduction

### New attacks

**Key-recovery Attack on HMAC-GOST**

**Example use: challenge-response authentication**

- **Alice**
  - password $pw$

- **Server**
  - password $pw$

1. $x \leftarrow \$

2. $y \leftarrow \text{MAC}_{pw}(x)$

3. If $y = \text{MAC}_{pw}(x)$, accept
   
4. Else, reject

- CRAM-MD5 authentication in SASL, POP3, IMAP, SMTP, ...
MAC Constructions

- Dedicated designs
  - Pelican-MAC, SQUASH, SipHash

- From universal hash functions
  - UMAC, VMAC, Poly1305

- From block ciphers
  - CBC-MAC, OMAC, PMAC

- From hash functions
  - HMAC, Sandwich-MAC, Envelope-MAC
MAC Constructions

- Dedicated designs
  - Pelican-MAC, SQUASH, SipHash

- From universal hash functions
  - UMAC, VMAC, Poly1305

- From block ciphers
  - CBC-MAC, OMAC, PMAC

- From hash functions
  - HMAC, Sandwich-MAC, Envelope-MAC
New attacks

Key-recovery Attack on HMAC-GOST

Introduction

New Generic Attacks on Hash-based MACs

Hash-based MACs (I)

- Secret-prefix MAC: $\text{MAC}_k(M) = H(k \| M)$
  - Insecure with MD/SHA: length-extension attack
  - Compute $\text{MAC}_k(M \| P)$ from $\text{MAC}_k(M)$ without the key

- Secret-suffix MAC: $\text{MAC}_k(M) = H(M \| k)$
  - Can be broken using offline collisions

- Use the key at the beginning and at the end
  - Sandwich-MAC: $H(k_1 \| M \| k_2)$
  - NMAC: $H(k_2 \| H(k_1 \| M))$
  - HMAC: $H((k \oplus \text{opad}) \| H((k \oplus \text{ipad}) \| M))$
  - Security proofs
Hash-based MACs (I)

- Secret-prefix MAC: \( \text{MAC}_k(M) = H(k \| M) \)
  - Insecure with MD/SHA: length-extension attack
  - Compute \( \text{MAC}_k(M \| P) \) from \( \text{MAC}_k(M) \) without the key

- Secret-suffix MAC: \( \text{MAC}_k(M) = H(M \| k) \)
  - Can be broken using offline collisions

- Use the key at the beginning and at the end
  - Sandwich-MAC: \( H(k_1 \| M \| k_2) \)
  - NMAC: \( H(k_2 \| H(k_1 \| M)) \)
  - HMAC: \( H((k \oplus \text{opad}) \| H((k \oplus \text{iPad}) \| M)) \)
New Generic Attacks on Hash-based MACs

Hash-based MACs (I)

- Secret-prefix MAC:  \( \text{MAC}_k(M) = H(k \| M) \)
  - Insecure with MD/SHA: length-extension attack
  - Compute \( \text{MAC}_k(M \| P) \) from \( \text{MAC}_k(M) \) without the key

- Secret-suffix MAC:  \( \text{MAC}_k(M) = H(M \| k) \)
  - Can be broken using offline collisions

- Use the key at the beginning and at the end
  - Sandwich-MAC:  \( H(k_1 \| M \| k_2) \)
  - NMAC:  \( H(k_2 \| H(k_1 \| M)) \)
  - HMAC:  \( H((k \oplus \text{opad}) \| H((k \oplus \text{ipad}) \| M)) \)
  - Security proofs
Introduction

New attacks

Key-recovery Attack on HMAC-GOST

Hash-based MACs (II)

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

- Key-dependant initial value $I_k$
- Unkeyed compression function $h$
- Key-dependant finalization, with message length $g_k$
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 for hash-based MACs**:
  - Distinguish $\text{MAC}_k^H$ from a PRF: **distinguishing-R**
    - e.g. distinguish HMAC from a PRF
  - Distinguish $\text{MAC}_k^H$ from $\text{MAC}_k^{\text{PRF}}$: **distinguishing-H**
    - e.g. distinguish HMAC-SHA1 from HMAC-PRF
Generic Attack on Hash-based MACs

1. Find internal collisions
   - Query $2^{l/2}$ 1-block messages
   - 1 internal collision expected, detected in the output

2. Query $t = \text{MAC}(x \parallel m)$

3. $(y \parallel m, t)$ is a forgery
Generic Attack on Hash-based MACs

1. Find internal collisions
   - Query $2^{l/2}$ 1-block messages
   - 1 internal collision expected, detected in the output

2. Query $t = \text{MAC}(x \parallel m)$

3. $(y \parallel m, t)$ is a forgery
**Generic Attack on Hash-based MACs**

1. **Find internal collisions**
   - Query $2^{l/2}$ 1-block messages
   - 1 internal collision expected, detected in the output

2. **Query** $t = \text{MAC}(x || m)$

3. $(y || m, t)$ is a forgery
**Generic Attack on Hash-based MACs**

1. **Find internal collisions**
   - Query $2^{l/2}$ 1-block messages
   - 1 internal collision expected, detected in the output

2. **Query** $t = \text{MAC}(x \| m)$ and $t' = \text{MAC}(y \| m)$

3. **If** $t = t'$ the oracle is a hash-based MAC: distinguishing-R
Security of hash-based MACS

With $n = l = k$:

- Existential forgery
- Distinguishing-$R$
- Universal forgery
- Distinguishing-$H$
- Key recovery

Security proof
Outline

Introduction
  MACs
  Generic Attacks

New attacks
  Cycle detection
  Distinguishing-H attack
  State recovery attack

Key-recovery Attack on HMAC-GOST
  GOST
  HMAC-GOST
Outline

Introduction

MACs
Generic Attacks

New attacks

Cycle detection
Distinguishing-H attack
State recovery attack

Key-recovery Attack on HMAC-GOST

GOST
HMAC-GOST
Distinguishing-H attack

- Security notion from PRF
- Distinguish HMAC-SHA-1 from HMAC with a PRF
Distinguishing-H attack

- Collision-based attack does not work:
  - Any compression function has collisions
  - Secret key prevents pre-computed collision

- Common assumption: distinguishing-H attack should require $2^l$

"If we can recognize the hash function inside HMAC, it's a bad hash function"
Main Idea

- Using a **fixed message block**, we iterate a fixed function
- Starting point and ending point unknown because of the key
- **Can we still 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 $x \rightarrow f(x)$
- Iterate $f$: $x_i = f(x_{i-1})$
- Collision after $\approx 2^{n/2}$ iterations
  - **Cycles**
  - **Trees** rooted in the cycle
  - Several components
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
Introduction

New attacks

Key-recovery Attack on HMAC-GOST

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: $\sqrt{\pi N/8}$
- Rho length: $\sqrt{\pi N/2}$
- Largest tree: $0.48N$
- Largest component: $0.76N$
**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: $\sqrt{\pi N}/8$
- Rho length: $\sqrt{\pi N}/2$
- 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 $t = \text{MAC}(r \| [0]^{2^{l/2}})$ and $t' = \text{MAC}(r \| [0]^{2^{l/2}+L})$

Success if

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

Randomize starting point
Using the cycle length

1. **Offline**: find the cycle length $L$ of the main component of $h_0$
2. **Online**: query $t = \text{MAC}(r \parallel [0]^{2^{l/2}})$ and $t' = \text{MAC}(r \parallel [0]^{2^{l/2}+L})$

Success if

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

**Problem:** most MACs use the message length.

\[ \text{MAC}_k(M) = \text{g}_k(h_{0 \times 0}^{l \times 0} h_{0 \times 1}^{l \times 1} h_{0 \times 2}^{l \times 2} h_{0 \times 3}^{l \times 3} M) \]
Dealing with the message length

Solution: reach the cycle twice

\[ M = r || [0]^{2^{l/2}} || [1] || [0]^{2^{l/2}} \]
**Solution:** reach the cycle twice

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

\[
M_2 = r \parallel [0]^{2^{l/2}} \parallel [1] \parallel [0]^{2^{l/2}+L}
\]
**Distinguishing-H attack**

1. **Offline**: find the cycle length $L$ of the main component of $h_0$

2. **Online**: query

   \[ t = \text{MAC}(r || [0]^{2^{l/2}} || [1] || [0]^{2^{l/2}+L}) \]

   \[ t' = \text{MAC}(r || [0]^{2^{l/2}+L} || [1] || [0]^{2^{l/2}}) \]

3. If $t = t'$, then $h$ is the compression function in the oracle

**Analysis**

- **Complexity**: $2^{l/2+3}$ compression function calls
- **Success probability**: $p \approx 0.14$
  - Both starting point are in the main component
  - Both cycles are reached with less than $2^{l/2}$ iterations

\[ p = 0.76^2 \]
\[ p \geq 0.5^2 \]
State recovery attack

- With high pr., first cyclic point is the root of the giant tree
- Binary search for first cyclic point

1. Query with several $x$:
   \[ t = \text{MAC}(r \parallel [0]^\alpha \parallel [1] \parallel [0]^{2^l/2+L}) \]
   \[ t' = \text{MAC}(r \parallel [0]^\alpha+L \parallel [1] \parallel [0]^{2^l/2}) \]

2. If $t = t'$ the cycle is reached with less than $\alpha$ steps

- Collision detection probabilistic: repeat with $\beta \log(l)$ messages
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: $\sqrt{\pi N/8}$
- Rho length: $\sqrt{\pi N/2}$
- Largest tree: $0.48N$
- Largest component: $0.76N$
State recovery attack

- With high pr., first cyclic point is the root of the giant tree
- Binary search for first cyclic point

1. Query with several $x$:
   
   $t = \text{MAC}(r \| [0]^{\alpha} \| [1] \| [0]^{2l/2 + L})$
   
   $t' = \text{MAC}(r \| [0]^{\alpha + L} \| [1] \| [0]^{2l/2})$

2. If $t = t'$ the cycle is reached with less than $\alpha$ steps

- Collision detection probabilistic: repeat with $\beta \log(l)$ messages
Introduction

New attacks

Key-recovery Attack on HMAC-GOST

State recovery attack

- With high pr., first cyclic point is the root of the giant tree
- Binary search for first cyclic point

1. Query with several $x$:

   \[
   t = \text{MAC}(r || [0]^\alpha || [1] || [0]^{2^{l/2}+L})
   \]
   \[
   t' = \text{MAC}(r || [0]^\alpha+L || [1] || [0]^{2^{l/2}})
   \]

2. If $t = t'$ the cycle is reached with less than $\alpha$ steps

- Collision detection probabilistic: repeat with $\beta \log(l)$ messages
State recovery attack

- With high pr., first cyclic point is the root of the giant tree
- Binary search for first cyclic point

1. Query with several $x$:
   
   $t = \text{MAC}(r \parallel [0]^\alpha \parallel [x] \parallel [0]^{2^{l/2} + L})$
   
   $t' = \text{MAC}(r \parallel [0]^\alpha + L \parallel [x] \parallel [0]^{2^{l/2}})$

2. If $t = t'$ the cycle is reached with less than $\alpha$ steps

- Collision detection probabilistic: repeat with $\beta \log(l)$ messages
New attacks

Key-recovery attack on HMAC-GOST

Compare with collision finding algorithms

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

Messages of length $2^{l/2}$ are not very practical...

- SHA-1 and HAVAL limit the message length to $2^{64}$ bits.
- Cycle detection impossible with messages shorter than $L \approx 2^{L/2}$.
Collision finding with small chains

Using collisions for state recovery

- Collision points are not random
- Longer chains give more biased distribution
- Precompute collisions offline, and test online

1. Compute chains $x \leadsto y$
   Stop when $y$ distinguished

2. If $y \in \{y_i\}$, collision found
Generic attacks on hash-based MACs

- Distinguishing-H and state recovery attacks
- Complexity $2^{l-s}$ with messages of length $2^s$
Outline

Introduction

MACs
Generic Attacks

New attacks

Cycle detection
Distinguishing-H attack
State recovery attack

Key-recovery Attack on HMAC-GOST

GOST
HMAC-GOST
\section*{Introduction}

\section*{New attacks}

\subsection*{Key-recovery Attack on HMAC-GOST}

\section*{GOST}

- Russian standard from 1994
- GOST and HMAC-GOST standardized by IETF
  \[ n = l = m = 256 \]

- Checksum (dashed lines)
  - Larger state should increase the security
New attacks

Key-recovery Attack on HMAC-GOST

HMAC-GOST

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

1. Recover the state
2. Build a multicollision: $2^{3l/4}$ messages with the same $x_3$
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{l/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x', x)$ for $2^{l/2}$ collisions
5. Detect match $M \oplus M' = x \oplus x'$. With high probability $M \oplus k = x$
Key recovery attack

1. Recover the state
2. Build a multicollision: $2^{3l/4}$ messages with the same $x_3$
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{l/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x', x)$ for $2^{l/2}$ collisions
5. Detect match $M \oplus M' = x \oplus x'$. With high probability $M \oplus k = x$
**Key recovery attack**

1. Recover the state
2. Build a multicollision: $2^{3l/4}$ messages with the same $x_3$
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{l/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x', x)$ for $2^{l/2}$ collisions
5. Detect match $M \oplus M' = x \oplus x'$. With high probability $M \oplus k = x$
Key recovery attack

1. Recover the state
2. Build a multicollision: $2^{3l/4}$ messages with the same $x_3$
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{l/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x', x)$ for $2^{l/2}$ collisions
5. Detect match $M \oplus M' = x \oplus x'$. With high probability $M \oplus k = x$
**Key recovery attack**

1. Recover the state
2. Build a multicollision: $2^{3l/4}$ messages with the same $x_3$
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{l/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x', x)$ for $2^{l/2}$ collisions
5. Detect match $M \oplus M' = x \oplus x'$. With high probability $M \oplus k = x$
Introduction

New attacks

Key-recovery Attack on HMAC-GOST

Conclusion

New generic attacks against hash-based MACs (single-key):

1. **Distinguishing-H attack in** $2^{l/2}$
   - State-recovery attack in $2^{l/2} \times l$
   - Not harder than distinguishing-R.

2. **Key-recovery attack on HMAC-GOST in** $2^{3l/4}$
   - Generic attack against hash functions with a checksum
   - The checksum weakens the design!
Questions?

With the support of ERC project CRASH
## Comparison

<table>
<thead>
<tr>
<th>Function</th>
<th>Attack</th>
<th>Complexity</th>
<th>M. len</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMAC-MD5</td>
<td>dist-H, st. rec.</td>
<td>$2^{97}$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>HMAC-SHA-0</td>
<td>dist-H</td>
<td>$2^{100}$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>HMAC-HAVAL (3-pass)</td>
<td>dist-H</td>
<td>$2^{228}$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>HMAC-SHA-1 62 mid. steps</td>
<td>dist-H</td>
<td>$2^{157}$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Generic</td>
<td>dist-H, st. rec.</td>
<td>$\tilde{O}(2^{l/2})$</td>
<td>$2^{l/2}$</td>
<td></td>
</tr>
<tr>
<td>Generic: checksum</td>
<td>key recovery</td>
<td>$O(2^{3l/4})$</td>
<td>$2^{l/4}$</td>
<td>$s \leq l/4$</td>
</tr>
<tr>
<td>HMAC-MD5*</td>
<td>dist-H, st. rec.</td>
<td>$2^{66}$, $2^{78}$</td>
<td>$2^{64}$</td>
<td></td>
</tr>
<tr>
<td>HMAC-HAVAL$^\dagger$ (any)</td>
<td>dist-H, st. rec.</td>
<td>$O(2^{202})$</td>
<td>$2^{54}$</td>
<td></td>
</tr>
<tr>
<td>HMAC-SHA-1$^\dagger$</td>
<td>dist-H, st. rec.</td>
<td>$O(2^{120})$</td>
<td>$2^{40}$</td>
<td></td>
</tr>
<tr>
<td>HMAC-GOST*</td>
<td>key-recovery</td>
<td>$2^{200}$</td>
<td>$2^{64}$</td>
<td></td>
</tr>
</tbody>
</table>

* MD5, GOST: arbitrary-length; $^\dagger$ SHA-1, HAVAL: limited message length.