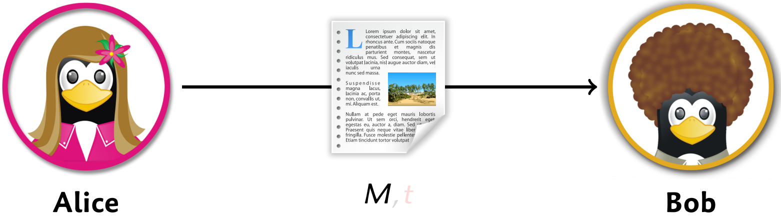
New Generic Attacks on Hash-based MACs

Gaëtan Leurent, Thomas Peyrin, Lei Wang

UCL Crypto Group, Belgium
Nanyang Technological University, Singapore

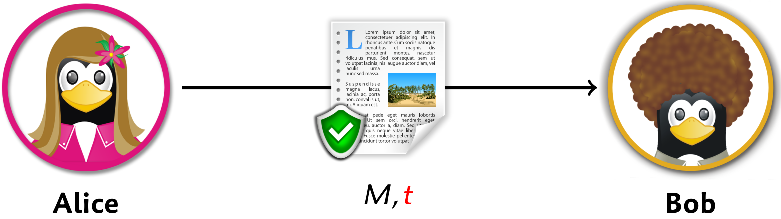
TCCM-CACR 2013

Message Authentication Codes



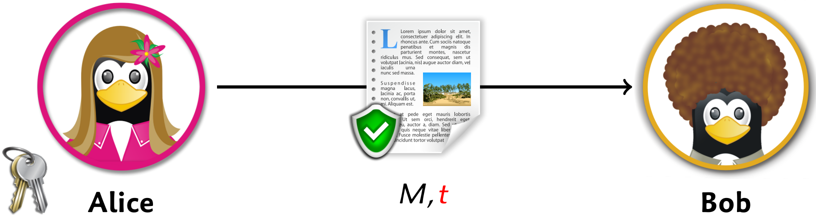
- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice use a key k to compute a tag: $t = \mathrm{MAC}_k(M)$
- Bob verifies the tag with the same key k: $t \overset{?}{=} \mathrm{MAC}_k(M)$
- Symmetric equivalent to digital signatures

Message Authentication Codes



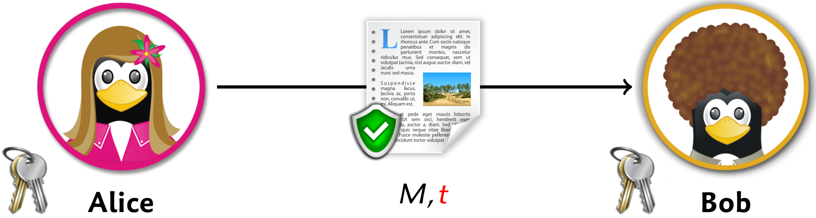
- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice use a key k to compute a tag: $t = \mathrm{MAC}_k(M)$
- Bob verifies the tag with the same key k: $t \overset{?}{=} \mathrm{MAC}_k(M)$
- Symmetric equivalent to digital signatures

Message Authentication Codes



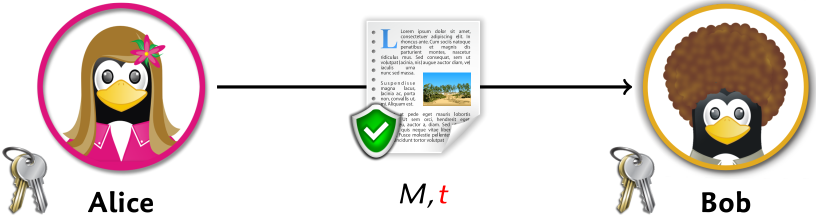
- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice use a key k to compute a tag: $t = \mathrm{MAC}_k(M)$
- Bob verifies the tag with the same key k: $t \overset{?}{=} \mathrm{MAC}_k(M)$
- Symmetric equivalent to digital signatures

Message Authentication Codes



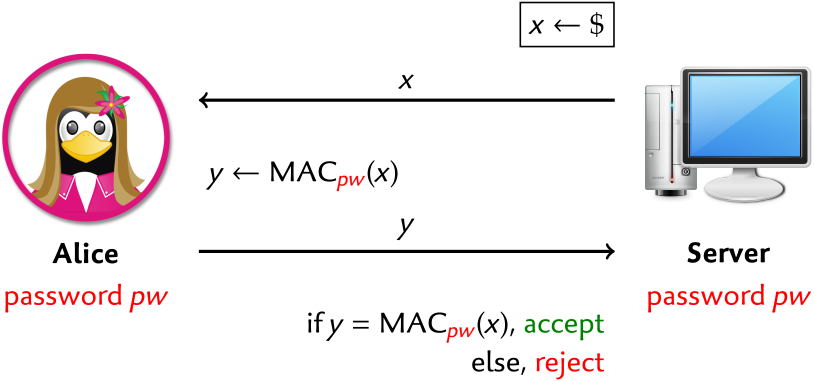
- Alice sends a message to Bob
- Bob wants to authenticate the message.
- Alice use a key k to compute a tag: $t = \mathrm{MAC}_k(M)$
- Bob verifies the tag with the same key k: $t \overset{?}{=} \mathrm{MAC}_k(M)$
- Symmetric equivalent to digital signatures

Message Authentication Codes



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

Example use: challenge-response authentication



$\boxed{x \leftarrow \$}$

x

$y \leftarrow \mathrm{MAC}_{pw}(x)$

y

Alice

password pw

Server

password pw

if $y = \mathrm{MAC}_{pw}(x)$, accept
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

Hash-based MACs (I)

- Secret-prefix MAC: $\mathrm{MAC}_k(M) = H(k \,\|\, M)$
  - Insecure with MD/SHA: length-extension attack
  - Compute $\mathrm{MAC}_k(M \,\|\, P)$ from $\mathrm{MAC}_k(M)$ without the key
- Secret-suffix MAC: $\mathrm{MAC}_k(M) = H(M \,\|\, k)$
  - 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: $H((k \oplus \mathtt{opad}) \,\|\, H((k \oplus \mathtt{ipad}) \,\|\, M))$
  - Security proofs

Hash-based MACs (I)

- Secret-prefix MAC: $\mathrm{MAC}_k(M) = H(k \,\|\, M)$
    - Insecure with MD/SHA: length-extension attack
    - Compute $\mathrm{MAC}_k(M \,\|\, P)$ from $\mathrm{MAC}_k(M)$ without the key
- Secret-suffix MAC: $\mathrm{MAC}_k(M) = H(M \,\|\, k)$
    - 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: $H((k \oplus \mathrm{opad}) \,\|\, H((k \oplus \mathtt{ipad}) \,\|\, M))$
    - Security proofs

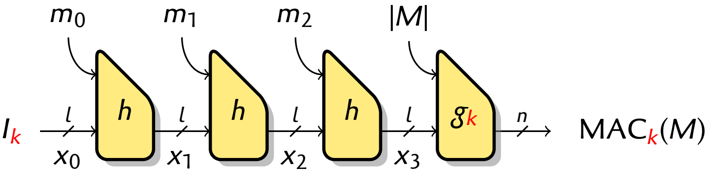
Hash-based MACs (I)

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

- Secret-suffix MAC: $\mathrm{MAC}_k(M) = H(M \,\|\, k)$
  - 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: $H((k \oplus \mathtt{opad}) \,\|\, H((k \oplus \mathtt{ipad}) \,\|\, M))$
  - Security proofs

Hash-based MACs (II)



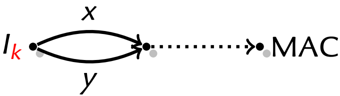
- l-bit chaining value
- n-bit output
- k-bit key

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

Security notions

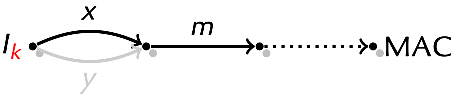
- Key-recovery: given access to a MAC oracle, extract the key
- Forgery: given access to a MAC oracle, forge a valid pair
  - For a message chosen by the adversary: existential forgery
  - For a challenge given to the adversary: universal forgery
- Distinguishing games for hash-based MACs:
  - Distinguish $\mathrm{MAC}_k^{\mathcal{H}}$ 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

Generic Attack on Hash-based MACs



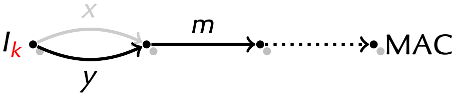
1. Find internal collisions
    - Query $2^{l/2}$ 1-block messages
    - 1 internal collision expected, detected in the output
2. Query $t = \mathrm{MAC}(x \,\|\, m)$
3. $\left(y \,\|\, m, t\right)$ is a forgery

Generic Attack on Hash-based MACs



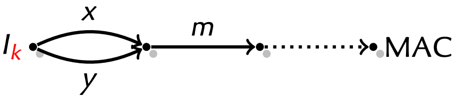
1. Find internal collisions
    - Query $2^{l/2}$ 1-block messages
    - 1 internal collision expected, detected in the output
2. Query $t = \text{MAC}(x \parallel m)$
3. $\left(y \parallel m, t\right)$ is a forgery

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 $t = \mathrm{MAC}(x \,\|\, m)$
3. $\left(y \,\|\, m, t\right)$ is a forgery

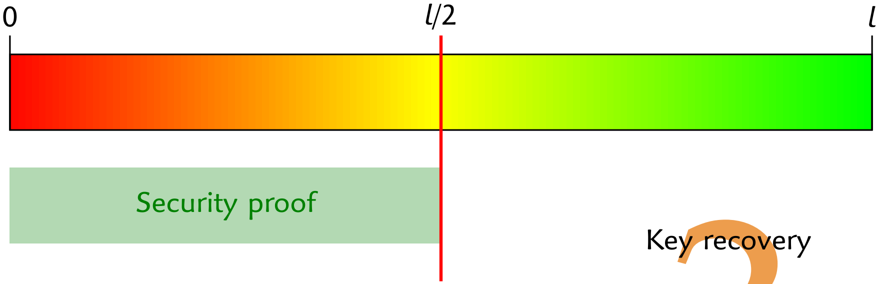
Generic Attack on Hash-based MACs



1. Find internal collisions
    - Query $2^{l/2}$ 1-block messages
    - 1 internal collision expected, detected in the output
2. Query $t = \mathrm{MAC}(x \| m)$ and $t' = \mathrm{MAC}(y \| m)$
3. If $t = t'$ the oracle is a hash-based MAC: distinguishing-R

Security of hash-based MACS

With $n = l = k$:



Security proof

Key recovery

Existential forgery

Universal forgery

Distinguishing-R

Distinguishing-H

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

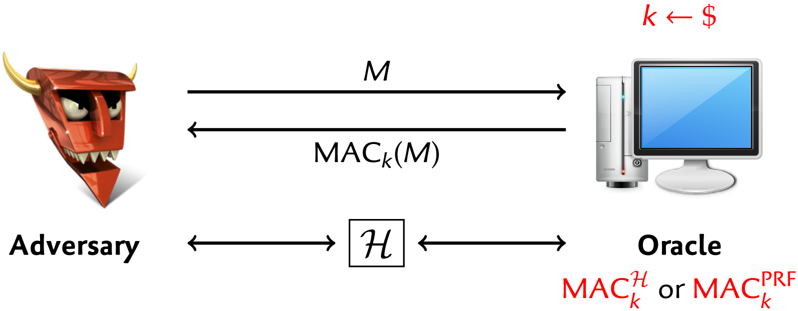
New attacks

- Cycle detection
- Distinguishing-H attack
- State recovery attack

Key-recovery Attack on HMAC-GOST

- GOST
- HMAC-GOST

Distinguishing-H attack



$k \leftarrow \$$

M

$\mathrm{MAC}_k(M)$

Adversary

$\mathcal{H}$

Oracle

$\mathrm{MAC}_k^{\mathcal{H}}$ or $\mathrm{MAC}_k^{\mathrm{PRF}}$

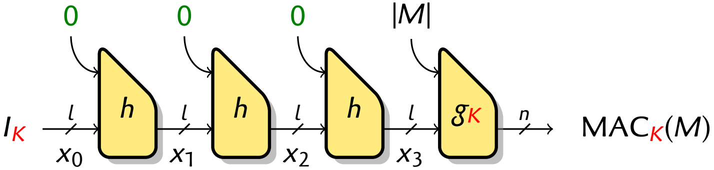
- Security notion from PRF
- Distinguish HMAC-SHA-1 from HMAC with a PRF

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”

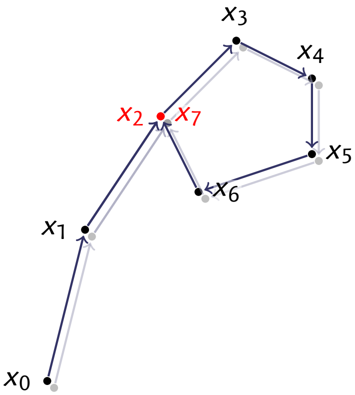
Main Idea



- 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

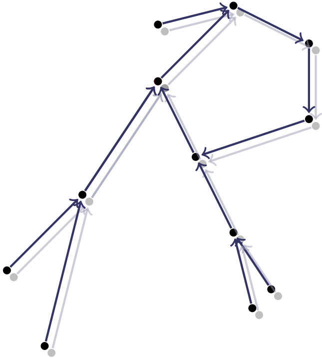
[Peyrin, Sasaki & Wang, Asiacrypt 12]

Random Mappings



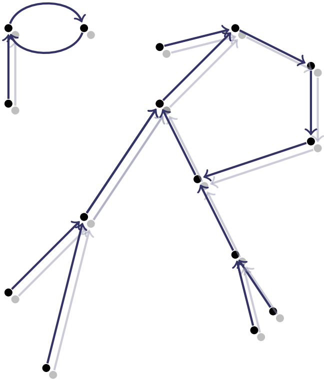
- Functional graph of a random mapping $x \longrightarrow f(x)$
- Iterate f: $x_i = f(x_{i-1})$
- Collision after $\approx 2^{n/2}$ iterations
  - Cycles
- Trees rooted in the cycle
- Several components

Random Mappings



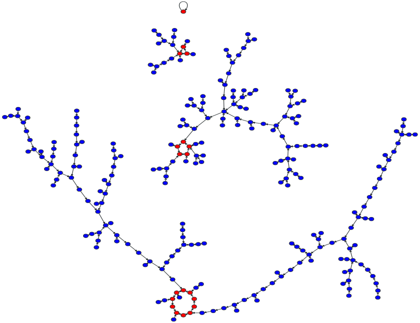
- Functional graph of a random mapping $x \longrightarrow f(x)$
- Iterate f: $x_i = f(x_{i-1})$
- Collision after $\approx 2^{n/2}$ iterations
  - Cycles
- Trees rooted in the cycle
- Several components

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

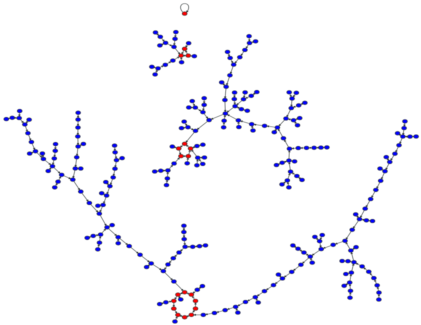
Cycle structure



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

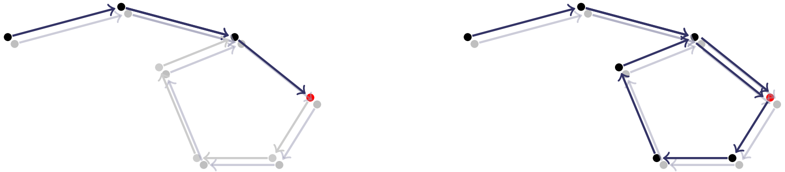


Expected properties of a random mapping over N points:

- # 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 $t = \mathrm{MAC}(r \parallel [0]^{2^{\ell/2}})$ and $t' = \mathrm{MAC}(r \parallel [0]^{2^{\ell/2}+L})$



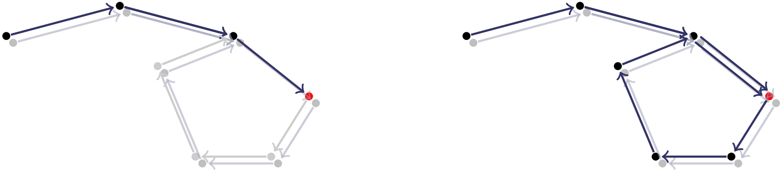
Success if

- The starting point is in the main component $p = 0.76$
- The cycle is reached with less than $2^{\ell/2}$ 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 $t = \mathrm{MAC}(r \,\|\, [0]^{2^{\ell/2}})$ and $t' = \mathrm{MAC}(r \,\|\, [0]^{2^{\ell/2}+L})$



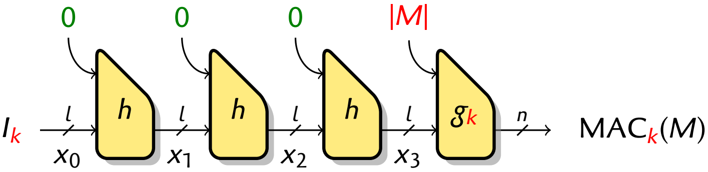
Success if

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

Randomize starting point

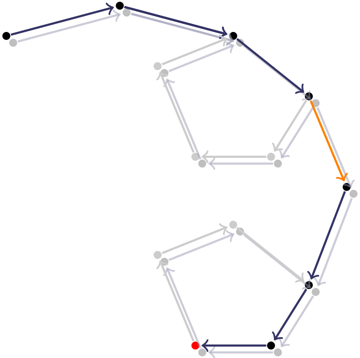
Dealing with the message length

Problem: most MACs use the message length.



Dealing with the message length

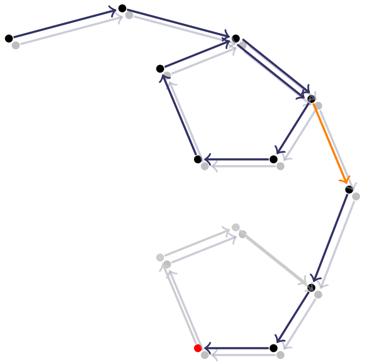
Solution: reach the cycle twice



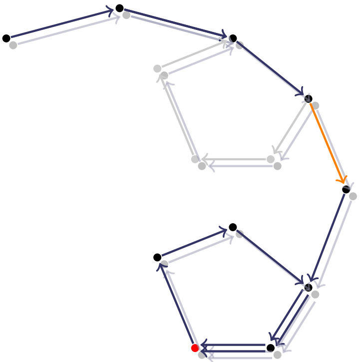
$$M = r \parallel [0]^{2^{\ell/2}} \parallel [1] \parallel [0]^{2^{\ell/2}}$$

Dealing with the message length

Solution: reach the cycle twice



$M_1 = r \,\|\, [0]^{2^{\ell/2}+L} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}}$



$M_2 = r \,\|\, [0]^{2^{\ell/2}} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}+L}$

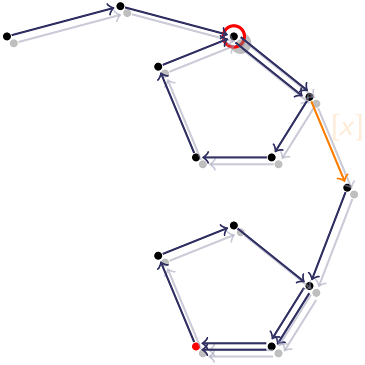
Distinguishing-H attack

1. Offline: find the cycle length L of the main component of h_0
2. Online: query
$$t = \mathrm{MAC}(r \,\|\, [0]^{2^{\ell/2}} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}+L})$$
$$t' = \mathrm{MAC}(r \,\|\, [0]^{2^{\ell/2}+L} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}})$$
3. If $t = t'$, then h is the compression function in the oracle

Analysis

- Complexity: $2^{\ell/2+3}$ 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 $2^{\ell/2}$ iterations $p \geq 0.5^2$

State recovery attack



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

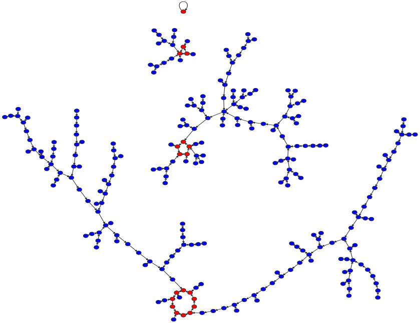
1. Query with several x:

$$t = \mathrm{MAC}(r \,\|\, [0]^{\alpha} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}+L})$$
$$t' = \mathrm{MAC}(r \,\|\, [0]^{\alpha+L} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}})$$

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

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

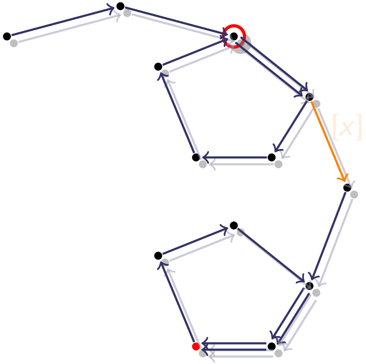
Cycle structure



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:

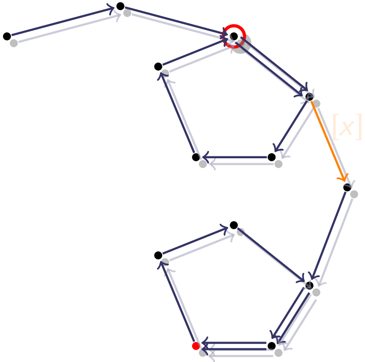
$$t = \mathrm{MAC}(r \,\|\, [0]^{\alpha} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}+L})$$

$$t' = \mathrm{MAC}(r \,\|\, [0]^{\alpha+L} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}})$$

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

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

State recovery attack

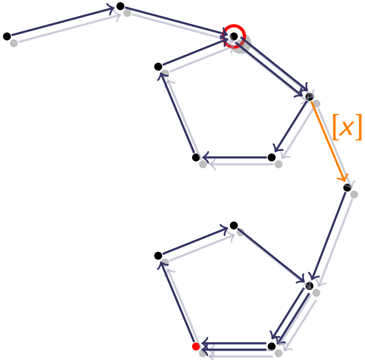


- With high pr., first cyclic point is the root of the giant tree
- Binary search for first cyclic point

1. Query with several x:
   $$t = \mathrm{MAC}(r \,\|\, [0]^{\alpha} \quad \|\, [1] \,\|\, [0]^{2^{\ell/2}+L})$$
   $$t' = \mathrm{MAC}(r \,\|\, [0]^{\alpha+L} \,\|\, [1] \,\|\, [0]^{2^{\ell/2}} \quad)$$
2. If $t = t'$ the cycle is reached with less than α steps

- Collision detection probabilistic: repeat with $\beta \log(\ell)$ 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:
$$t = \mathrm{MAC}(r \,\|\, [0]^{\alpha} \,\|\, [x] \,\|\, [0]^{2^{l/2}+L})$$
$$t' = \mathrm{MAC}(r \,\|\, [0]^{\alpha+L} \,\|\, [x] \,\|\, [0]^{2^{l/2}})$$
2. If $t = t'$ the cycle is reached with less than α steps

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

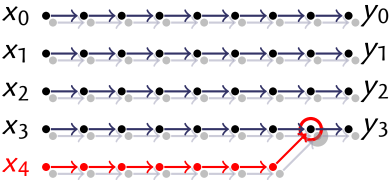
Variant with small messages

- 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 $L \approx 2^{l/2}$

Compare with collision finding algorithms

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

Collision finding with small chains



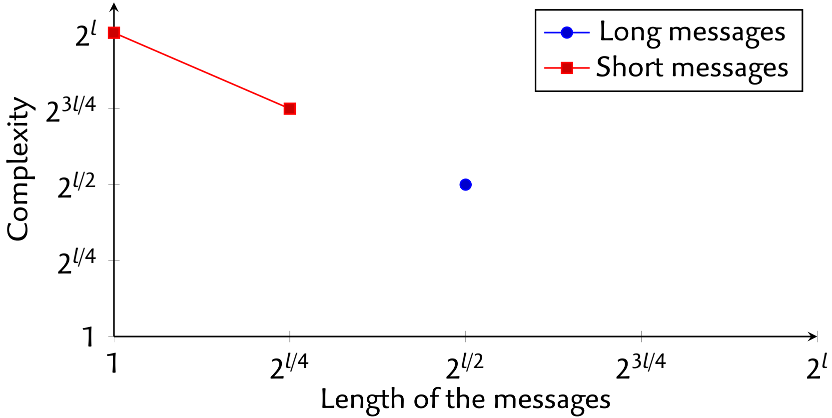
1. Compute chains $x \rightsquigarrow y$
   Stop when y distinguished
2. If $y \in \{y_i\}$, collision found

Using collisions for state recovery

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

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

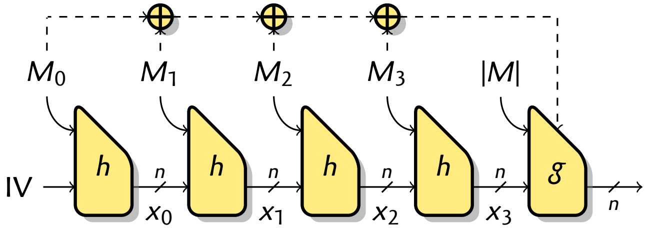
New attacks

- Cycle detection
- Distinguishing-H attack
- State recovery attack

Key-recovery Attack on HMAC-GOST

- GOST
- HMAC-GOST

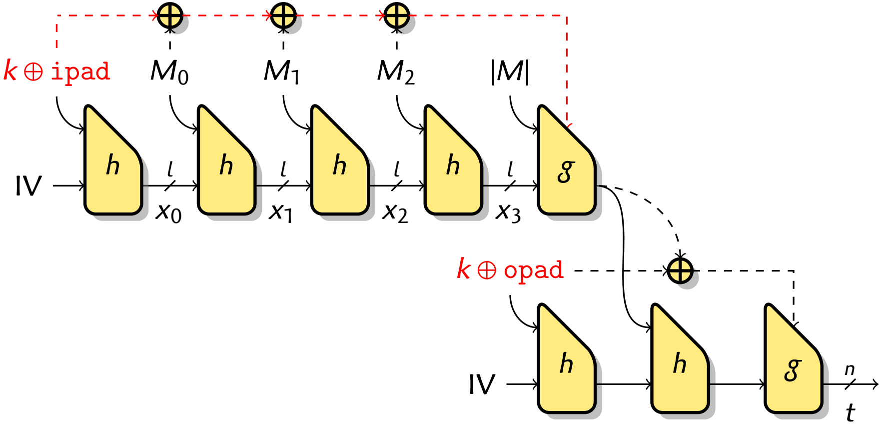
GOST



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

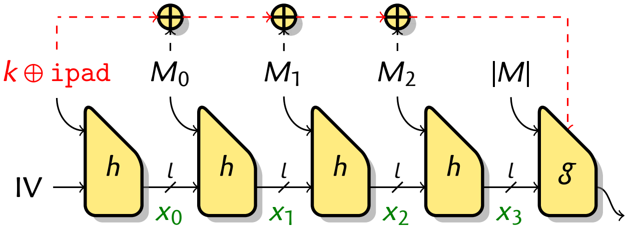
- Checksum (dashed lines)
  - Larger state should increase the security

HMAC-GOST



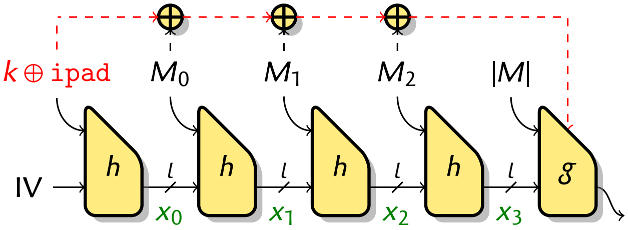
- In HMAC, key-dependant value used after the message
  - Related-key attacks on the last block

Key recovery attack



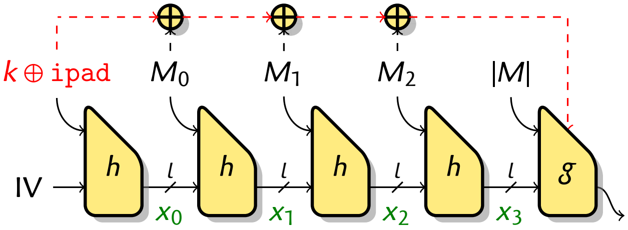
1. Recover the state
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_3
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x',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

Key recovery attack



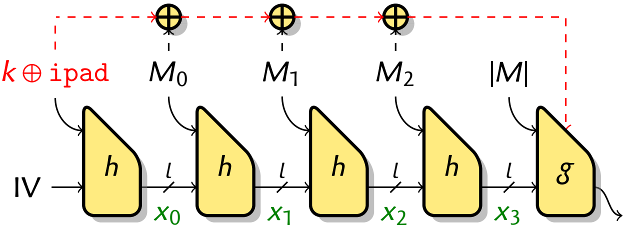
1. Recover the state
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_3
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x',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

Key recovery attack



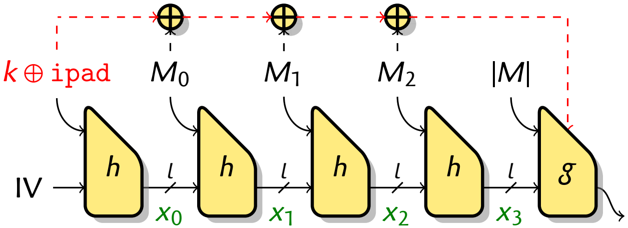
1. Recover the state
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_3
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x',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

Key recovery attack



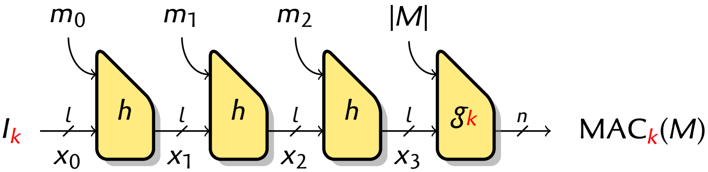
1. Recover the state
2. Build a multicollision: $2^{3\ell/4}$ messages with the same x_3
3. Query messages, detect collisions $g(x_3, k \oplus M) = g(x_3, k \oplus M')$
   Store $(M \oplus M', M)$ for $2^{\ell/2}$ collisions
4. Find collisions $g(x_3, x) = g(x_3, x')$ offline
   Store $(x \oplus x',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

Conclusion



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

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

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

Thanks

Questions?

With the support of ERC project CRASH

European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

Comparison

Function	Attack	Complexity	M. len	Notes
`HMAC-MD5`	dist-H, st. rec.	2^{97}	2	
`HMAC-SHA-0`	dist-H	2^{100}	2	
`HMAC-HAVAL` (3-pass)	dist-H	2^{228}	2	
`HMAC-SHA-1` 62 mid. steps	dist-H	2^{157}	2	
Generic	dist-H, st. rec.	$\tilde{O}(2^{l/2})$	$2^{l/2}$	
	dist-H, st. rec.	$O(2^{l-s})$	2^s	$s \leq l/4$
Generic: checksum	key recovery	$O(2^{3l/4})$	$2^{l/4}$	
`HMAC-MD5`*	dist-H, st. rec.	2^{66}, 2^{78}	2^{64}	
		$O(2^{96})$	2^{32}	
`HMAC-HAVAL`§ (any)	dist-H, st. rec.	$\mathrm{O}(2^{202})$	2^{54}	
`HMAC-SHA-1`§	dist-H, st. rec.	$\mathrm{O}(2^{120})$	2^{40}	
`HMAC-GOST`*	key-recovery	2^{200}	2^{64}	

* `MD5`, `GOST`: arbitrary-length; § `SHA-1`, `HAVAL`: limited message length.