Generic Related-key Attacks for HMAC

Thomas Peyrin
(joint work with Yu Sasaki and Lei Wang)

Nanyang Technological University - Singapore

TCCM-CACR 2012

Chengdu, China - November 24, 2012

Outline

Outline

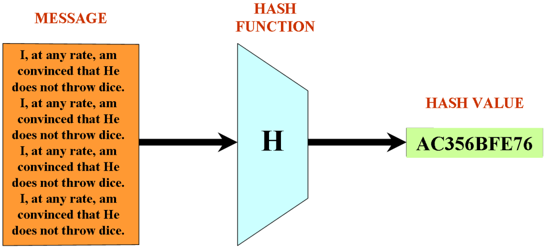
Introduction: hash functions

HMAC: MACing with hash functions
- What is HMAC
- The attack models
- Current state of HMAC

A generic related-key attack on HMAC
- Distinguish-R attack
- Intermediate internal state recovery
- Existential forgery and distinguish-H attack

Patching HMAC and Conclusion

What is a Hash Function ?



- H maps an **arbitrary length input** (the message M) to a **fixed length output** (typically $n = 128$, $n = 160$ or $n = 256$).
- no secret parameter.
- H must be easy to compute.
- examples: MD5 (1992), SHA-1 (1995), SHA-2 (2001), SHA-3 (2012)

The security goals

pre-image resistance:

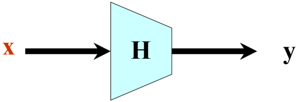
given an output challenge y, the attacker can not find a message x such that $H(x) = y$, in less than $\theta(2^n)$ operations.

2nd pre-image resistance:

given a challenge (x, y) so that $H(x) = y$, the attacker can not find a message $x' \neq x$ such that $H(x') = y$, in less than $\theta(2^n)$ operations.

collision resistance:

the attacker can not find two messages (x, x') such that $H(x) = H(x')$, in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79]).



The security goals

pre-image resistance:

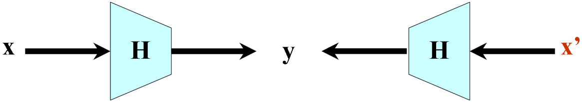
given an output challenge y, the attacker can not find a message x such that $H(x) = y$, in less than $\theta(2^n)$ operations.

2nd pre-image resistance:

given a challenge (x, y) so that $H(x) = y$, the attacker can not find a message $x' \neq x$ such that $H(x') = y$, in less than $\theta(2^n)$ operations.

collision resistance:

the attacker can not find two messages (x, x') such that $H(x) = H(x')$, in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79]).



The security goals

pre-image resistance:

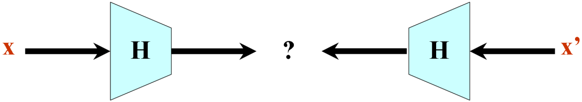
given an output challenge y, the attacker can not find a message x such that $H(x) = y$, in less than $\theta(2^n)$ operations.

2nd pre-image resistance:

given a challenge (x, y) so that $H(x) = y$, the attacker can not find a message $x' \neq x$ such that $H(x') = y$, in less than $\theta(2^n)$ operations.

collision resistance:

the attacker can not find two messages (x, x') such that $H(x) = H(x')$, in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79]).



The security goals

pre-image resistance:

given an output challenge y, the attacker can not find a message x such that $H(x) = y$, in less than $\theta(2^n)$ operations.

2nd pre-image resistance:

given a challenge (x, y) so that $H(x) = y$, the attacker can not find a message $x' \neq x$ such that $H(x') = y$, in less than $\theta(2^n)$ operations.

collision resistance:

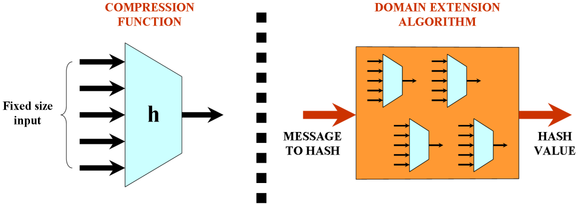
the attacker can not find two messages (x, x') such that $H(x) = H(x')$, in less than $\theta(2^{n/2})$ operations (a generic attack with the birthday paradox exists [Yuval-79]).

And other ones: near collisions, multicollisions, random oracle look-alike, ...

General construction

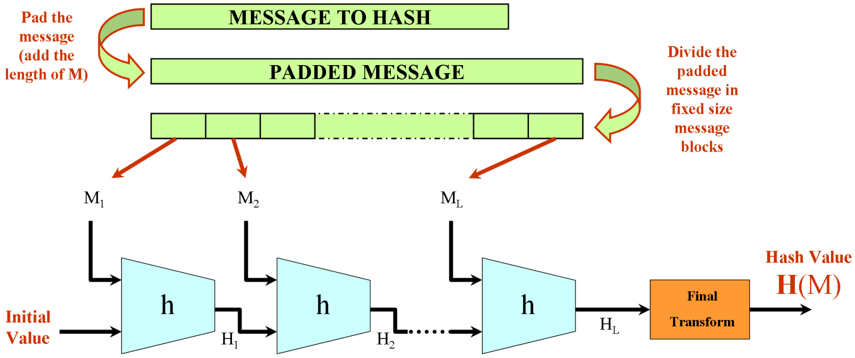
For historical reasons, most hash functions are composed of two elements:

- **a compression function** h**:** a function for which the input and output size is fixed.
- **a domain extension algorithm:** an iterative process that uses the compression function h so that the hash function H can handle inputs of arbitrary length.



The Merkle-Damgård domain extension algorithm

The most famous domain extension algorithm used is called the **Merkle-Damgård** [Merkle Damgård-89] iterative algorithm.



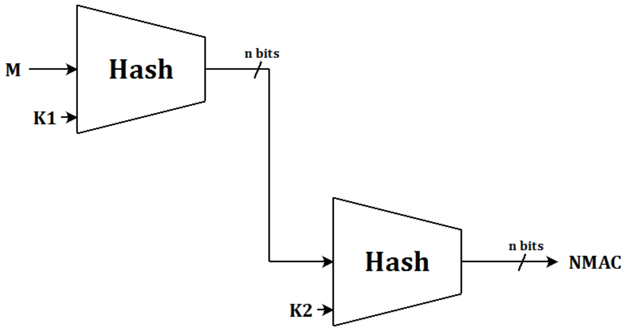
Outline

Outline

HMAC and NMAC (Bellare *et al.* - 1996)

A MAC outputs an n-bit value from a k-bit key K and an arbitrary long message M.

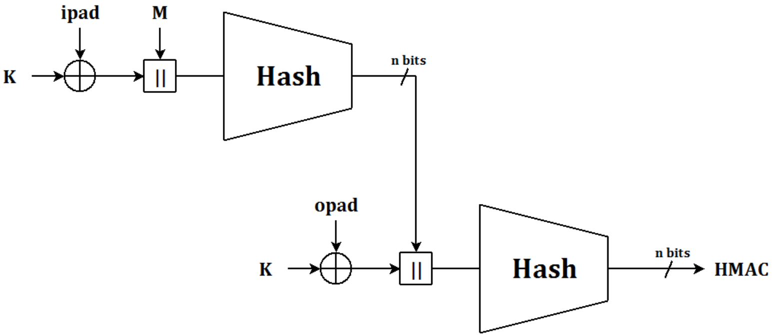
$$\mathtt{NMAC}(K_1, K_2, M) = H(K_2, H(K_1, M))$$



HMAC and NMAC (Bellare *et al.* - 1996)

A MAC outputs an n-bit value from a k-bit key K and an arbitrary long message M.

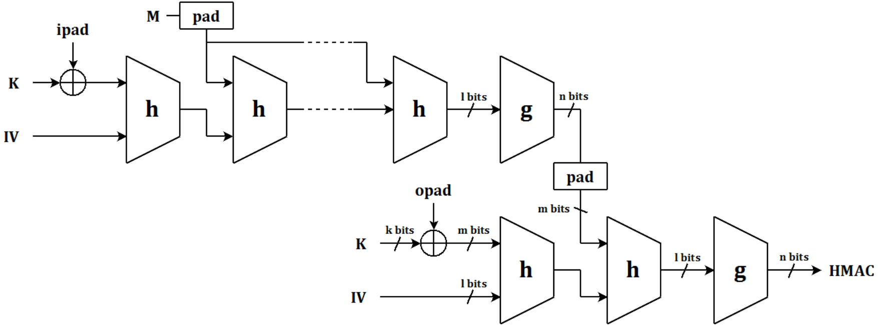
$$\texttt{HMAC}(K, M) = H(K \oplus \texttt{opad} \,||\, H(K \oplus \texttt{ipad} \,||\, M))$$



HMAC and NMAC (Bellare *et al.* - 1996)

A MAC outputs an n-bit value from a k-bit key K and an arbitrary long message M.

$$\texttt{HMAC}(K, M) = H(K \oplus \texttt{opad} \,||\, H(K \oplus \texttt{ipad} \,||\, M))$$



Outline

Universal and existential forgery

The game played:

The attacker can query an oracle, $\texttt{HMAC}_K$, and tries to generate a valid MAC with the key K for a message that he didn't query yet

When the message is chosen by the **challenger**:
it is a **universal forgery**

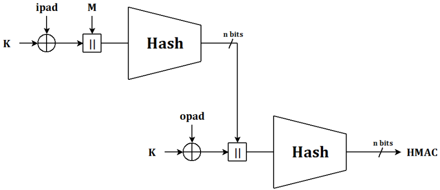
When the message is chosen by the **attacker**:
it is an **existential forgery**

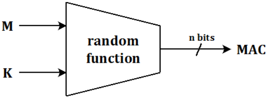
Distinguishing-R

The game played:

The attacker can query an oracle, F_K, that is instantiated either with $\texttt{HMAC}_K$, or with a random function R_K. He must obtain non-negligible advantage in distinguishing the two cases:

$$Adv(\mathcal{A}) = |\Pr[\mathcal{A}(\texttt{HMAC}_K) = 1] - \Pr[\mathcal{A}(R_K) = 1]| \, .$$



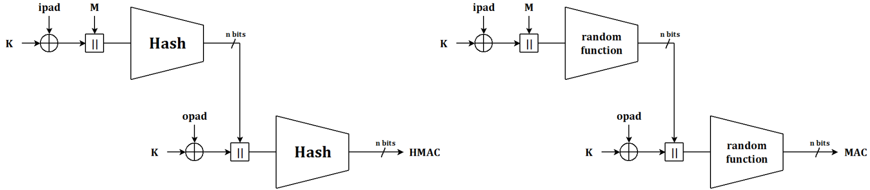


Distinguishing-H

The game played:

The attacker can query an oracle, $\texttt{HMAC}_K$, that is instantiated either with $\texttt{HMAC}_K^{H(h)}$ or with $\texttt{HMAC}_K^{H(r)}$, where H is a known dedicated hash function, h a known dedicated compression function, and r a randomly chosen function. He must obtain non-negligible advantage in distinguishing the two cases:

$$Adv(\mathcal{A}) = \left|\Pr[\mathcal{A}(\texttt{HMAC}_K^{H(h)}) = 1] - \Pr[\mathcal{A}(\texttt{HMAC}_K^{H(r)}) = 1]\right|.$$



Outline

Known dedicated attacks on HMAC

Attack	Key Setting	Target	Size	#Rounds	Comp.	Ref.
Dist.-H	Single key	MD4	128	Full	$2^{121.5}$	[KBPH06]
Dist.-H	Single key	MD5	128	33/64	$2^{126.1}$	[KBPH06]
Dist.-H	Single Key	MD5	128	Full	2^{97}	[WYWZZ09]
Dist.-H	Single key	3p HAVAL	256	Full	$2^{228.6}$	[KBPH06]
Dist.-H	Single key	4p HAVAL	256	102/128	$2^{253.9}$	[KBPH06]
Dist.-H	Single key	SHA0	160	Full	2^{109}	[KBPH06]
Dist.-H	Single key	SHA1	160	43/80	$2^{154.9}$	[KBPH06]
Dist.-H	Single key	SHA1	160	50/80	$2^{153.5}$	[RR08]
Dist.-H	Related Key	SHA1	160	58/80	$2^{158.74}$	[RR08]
Inner key rec.	Single Key	MD4	128	Full	2^{63}	[CY06]
Inner key rec.	Single Key	SHA0	160	Full	2^{84}	[CY06]
Inner key rec.	Single Key	SHA1	64	34/80	2^{32}	[RR08]
Inner key rec.	Single Key	3p HAVAL	256	Full	2^{122}	[LCKSH08]
Full key rec.	Single Key	MD4	128	Full	2^{95}	[FLN07]
Full key rec.	Single Key	MD4	128	Full	2^{77}	[WOK08]

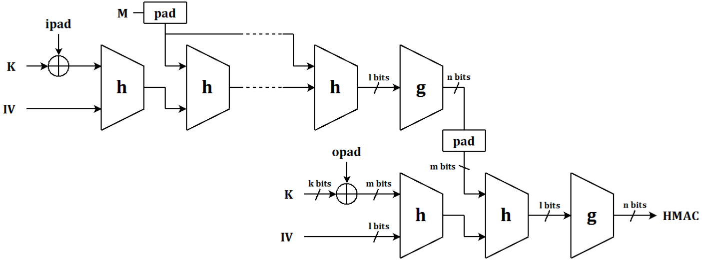
Known generic attacks on HMAC

The setting

We try to find generic attacks on HMAC with a k-bit when instantiated with an n-bit hash function using a l-bit internal state (with $l \leq 2n$ and k sufficiently big to avoid brute force key recovery)

Distinguishing-H attack costs 2^l computations (ideal)

Universal forgery attack costs 2^n computations (ideal)

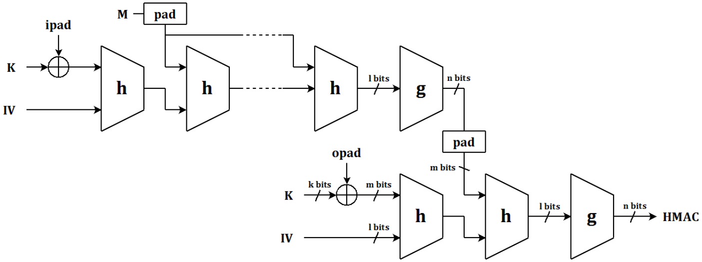


Known generic attacks on HMAC

Distinguishing-R attack costs $2^{l/2}$ computations (not ideal)

The procedure

- **step 1:** query $2^{l/2}$ messages and gather all pairs (M, M') that collides on the output
- **step 2:** for all colliding pairs, append an extra random message block M_1 and check if this new message pair $(M||M_1, M'||M_1)$ collides as well
- **step 3:** if it does, the oracle implements HMAC, otherwise it is a random function

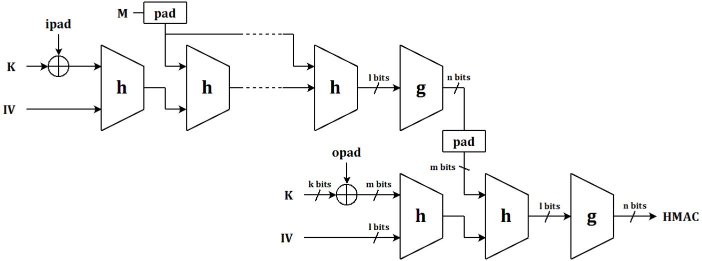


Known generic attacks on HMAC

Existential forgery attack costs $2^{l/2}$ computations (not ideal)

The procedure

- **step 1:** query $2^{l/2}$ messages and gather all pairs (M, M') that collides on the output
- **step 2:** for all colliding pairs, append an extra random message block M_1 and check if this new message pair $(M||M_1, M'||M_1)$ collides as well. Pick one such pair.
- **step 3:** append another extra random message block M_2 and query the MAC for message $M||M_2$. Then it is equal to the MAC for message $(M'||M_2)$



Known generic attacks on HMAC

Attack	Key Setting	Generic Complexity
Universal forgery	Single Key	2^n
Existential forgery	Single Key	$2^{l/2}$
Dist.-R	Single Key	$2^{l/2}$
Dist.-H	Single Key	2^l

Known generic attacks on HMAC

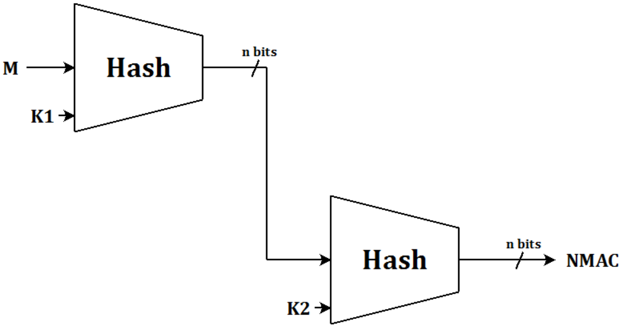
Attack	Key Setting	Generic Complexity
Universal forgery	Related Key	2^n ?
Existential forgery	Related Key	$2^{l/2}$?
Dist.-R	Related Key	$2^{l/2}$?
Dist.-H	Related Key	2^l ?

Outline

Outline

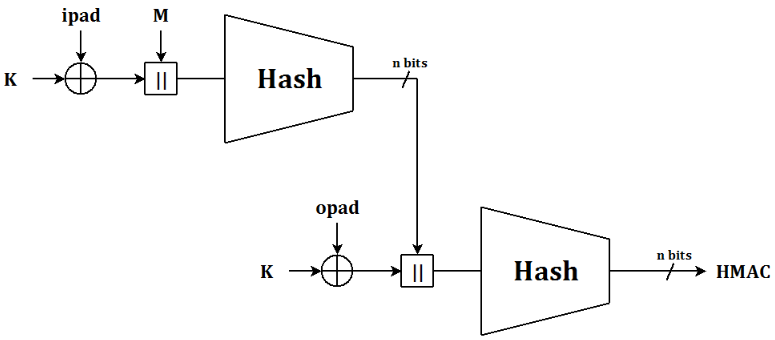
What weakness to attack ?

NMAC



What weakness to attack ?

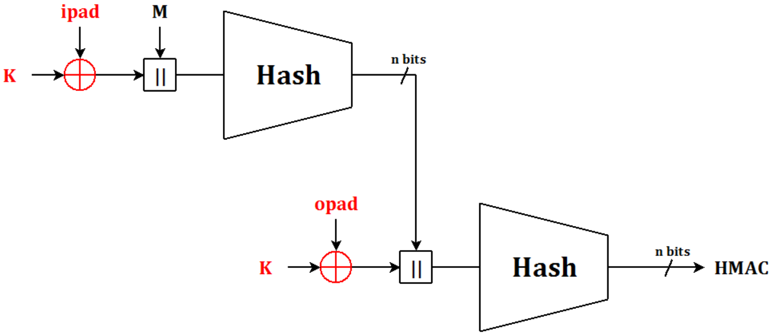
HMAC



What weakness to attack ?

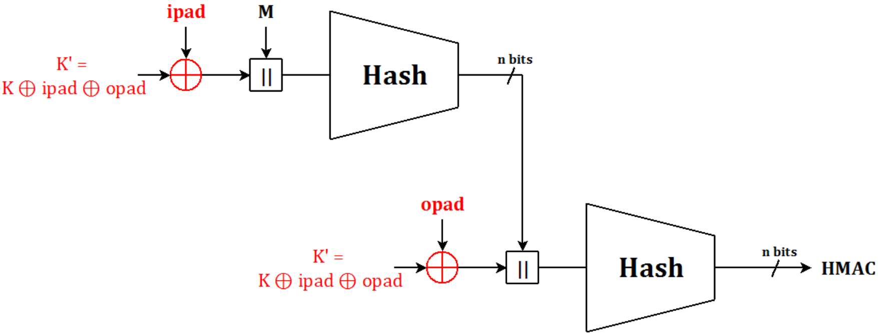
HMAC

(with key K)



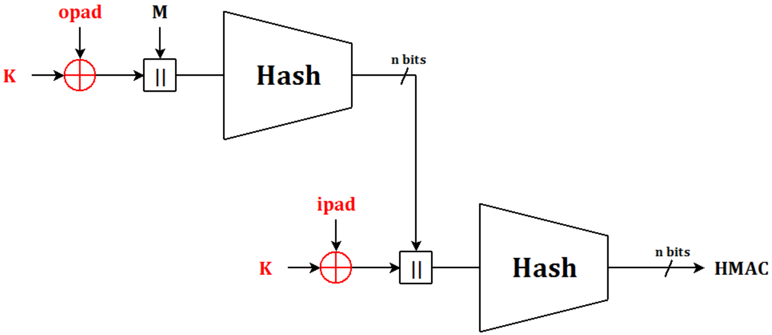
What weakness to attack ?

HMAC
(with key $K' = K \oplus \texttt{ipad} \oplus \texttt{opad}$)



What weakness to attack ?

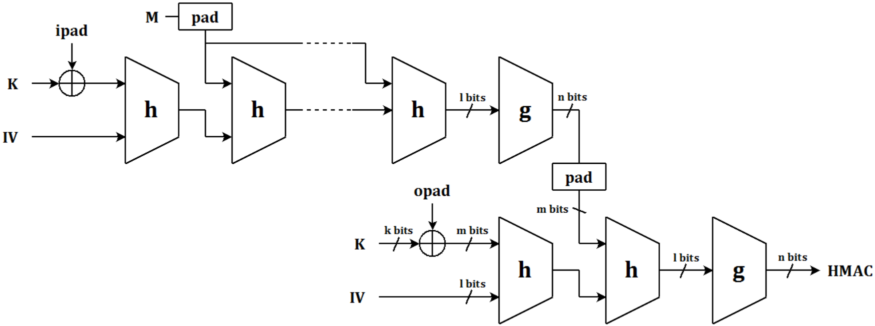
HMAC
(with key $K' = K \oplus$ `ipad` $\oplus$ `opad`)



What to detect ?

HMAC

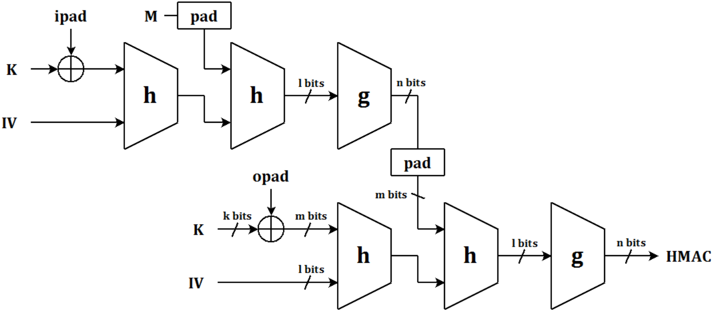
(with key *K* and arbitrary message)



What to detect ?

HMAC

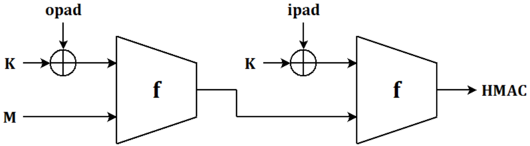
(with key K and n-bit message)



What to detect ?

HMAC

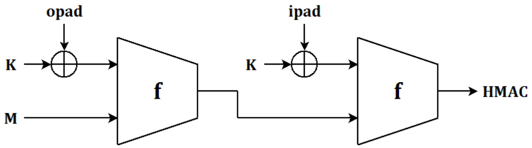
(with key K and n-bit message)

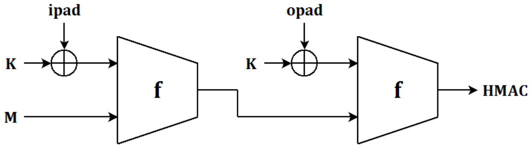


What to detect ?

HMAC

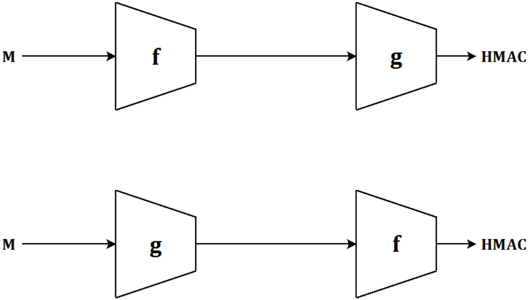
(with K and $K' = K \oplus$ ipad $\oplus$ opad and n-bit message)





What to detect ?

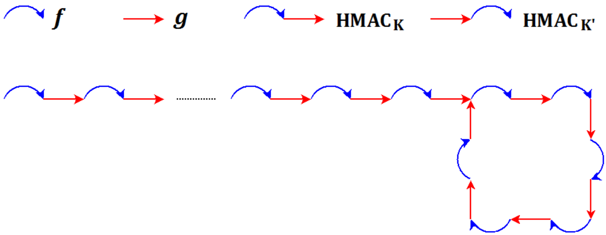
HMAC
(with K and $K' = K \oplus \texttt{ipad} \oplus \texttt{opad}$ and n-bit message)



What to detect ?

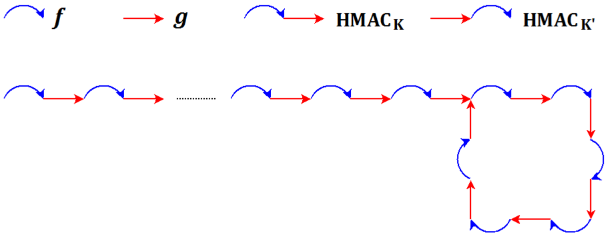
Functions $f(g(x))$ and $g(f(x))$ have a particular cycle structure:

there is a 1-to-1 correspondence between cycles of $f(g(x))$ and $g(f(x))$



How to detect the cycle structure ?

$\Longrightarrow$ by measuring cycles length



The game played (distinguishing-R in the related-key model):

The attacker can query two oracles, F_K and $F_{K'}$, that are instantiated either with $\texttt{HMAC}_K$ and $\texttt{HMAC}_{K'}$, or with two independent random functions R_K and $R_{K'}$. He must obtain non-negligible advantage in distinguishing the two cases:

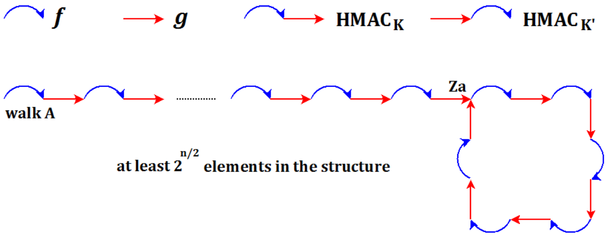
$$Adv(\mathcal{A}) = |\Pr[\mathcal{A}(\texttt{HMAC}_K, \texttt{HMAC}_{K'}) = 1] - \Pr[\mathcal{A}(R_K, R_{K'}) = 1]|$$

The attack

First step (walk A)

Start from an n-bit random input message, query F_K, and keep querying as new message the MAC just received. Continue so for about $2^{n/2} + 2^{n/2-1}$ queries until getting a collision among the MACs received.

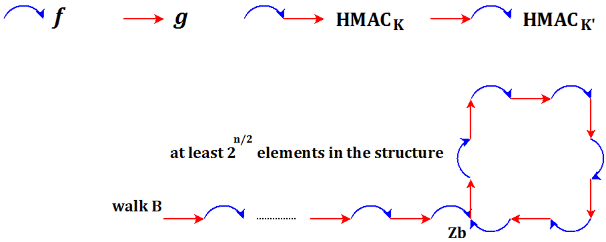
If no collision is found, or if the collision occurred in the $2^{n/2}$ first queries, the attacker outputs 0.



The attack

Second step (walk B)

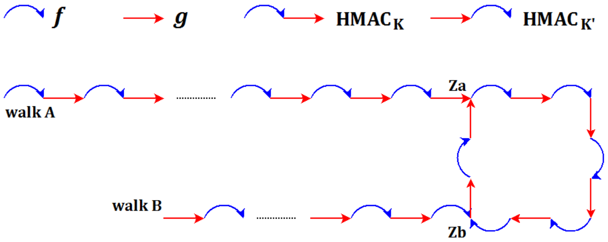
Do the same for oracle $F_{K'}$.



The attack

Third step (colliding walk A and walk B)

If the cycle of walk A has the same length as the one from walk B, then output 1. Otherwise output 0.



Results - distinguishing-R for HMAC with wide-pipe

The advantage of the attacker is non-negligible and **the complexity of the distinguisher** is about $2^{n/2} + 2^{n/2-1}$ computations for each of the first and second phase, thus **about** $2^{n/2+1}$ **computations in total**.

We implemented and verified the distinguisher. With SHA−2 truncated to 32 bits, we found two walks A and B that have the same cycle length of 79146 elements with 2^{17} computations. The best previously known attack for HMAC instantiated with SHA−2 truncated to 32 bits required 2^{128} computations.

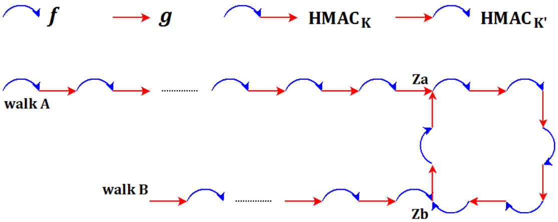
Attack	Key Setting	Target	Old Generic Complexity	New Generic Complexity
Dist.-R	Related Key	Wide-pipe	$2^{l/2}$	$2^{n/2+1}$

Outline

How to recover the intermediate internal state ?

We would like to know some of the intermediate internal state of $\texttt{HMAC}_K$ **and** $\texttt{HMAC}_{K'}$

Inside a colliding cycle for $\texttt{HMAC}_K$ and $\texttt{HMAC}_{K'}$, the input or output queries to $\texttt{HMAC}_K$ are intermediate internal state of $\texttt{HMAC}_{K'}$ (and vice-versa) ... but we don't know which one it is, so **we need to synchronize the cycles**

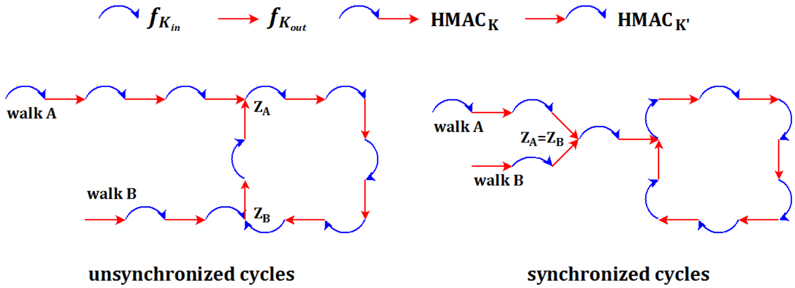


Synchronized and Unsynchronized cycles

There are two cases for a collision between walk A and walk B:

- collision in the tail
- collision in the cycle

If the collision happens in the tail, then the cycles are directly synchronized

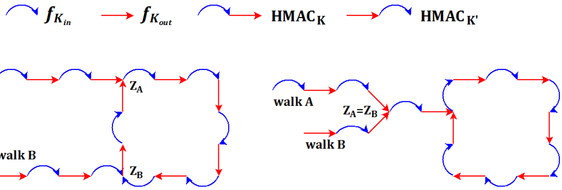


Synchronized and Unsynchronized cycles

We just build walk A and walk B with a tail long enough, such that the collision is likely to happen in the tail.

The procedure

- **step 1 (build walk A):** same as before, but just ensure that tail in walk A has size at least $2^{n/2-2}$
- **step 2 (build walk B):** same as step 1, but with queries to $K' = K \oplus \texttt{ipad} \oplus \texttt{opad}$
- **step 3:** check if the cycle have the same length, and if so, there is a good chance that it happened in the tail. Then you can recover the intermediate internal states.



unsynchronized cycles

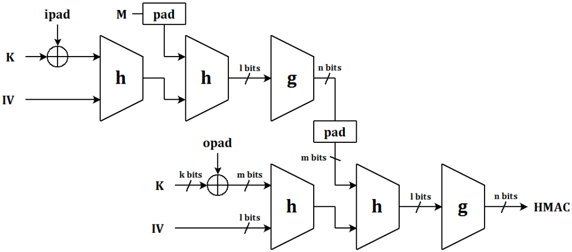
synchronized cycles

Internal state recovery for wide-pipe

For a wide-pipe hash, the attack is not over, because we have to revert the output truncation function from the intermediate internal state and recover all l bits.

The procedure

- **step 1:** obtain an intermediate internal state
- **step 2:** find a collision by doing query with one extra block of random data
- **step 3:** go through all the 2^{l-n} candidates and check offline which one would have give you this collision



Results - internal state recovery for HMAC

The complexity of the internal state recovery is **about** $2^{n/2+2}$ **queries and** 2^{l-n+1} **computations in total**.

Attack	**Key Setting**	**Target**	**Old Generic Complexity**	**New Generic Complexity**
Dist.-R	Related Key	Wide-pipe	$2^{l/2}$	$2^{n/2+1}$
Inner state rec.	Related Key	Narrow or Wide	2^n	$2^{n/2+2} + 2^{l-n+1}$

Outline

Existential forgery and distinguish-H attack

- **once we have recovered an internal state, forging a valid MAC is easy**
- **if we can recover an internal state, then distinguish-H is easy**

The **complexity to forge a valid MAC or distinguish-H** is the complexity of the internal state recovery ($2^{n/2+2} + 2^{l-n+1}$ **computations**)

Attack	**Key Setting**	**Target**	**Old Generic Complexity**	**New Generic Complexity**
Dist.-R	Related Key	Wide-pipe	$2^{l/2}$	$2^{n/2+1}$
Inner state rec.	Related Key	Narrow or Wide	2^n	$2^{n/2+2} + 2^{l-n+1}$
Ex. forgery	Related Key	Wide-pipe	$2^{l/2}$	$2^{n/2+2} + 2^{l-n+1}$
Dist.-H	Related Key	Narrow or Wide	2^l	$2^{n/2+2} + 2^{l-n+1}$

Outline

Our results

Our attacks on HMAC work when the key has length m, or $m - 1$ because ipad = 0x3636 $\cdots$ 36 and opad = 0x5C5C $\cdots$ 5C

$\Longrightarrow$ **The choice of ipad and opad was in fact important**

Attack	**Key Setting**	**Target**	**Old Generic Complexity**	**New Generic Complexity**
Dist.-R	Related Key	Wide-pipe	$2^{l/2}$	$2^{n/2+1}$
Inner state rec.	Related Key	Narrow or Wide	2^{n}	$2^{n/2+2} + 2^{l-n+1}$
Ex. forgery	Related Key	Wide-pipe	$2^{l/2}$	$2^{n/2+2} + 2^{l-n+1}$
Dist.-H	Related Key	Narrow or Wide	2^{l}	$2^{n/2+2} + 2^{l-n+1}$

Patching HMAC

1^{st} try:

We use a different IV for the hash function in the inner and outer call ...
... but that would require to change the *H* definition and implementations

2^{nd} try:

We truncate the HMAC output ...
... but having a smaller output reduces the expected security

Our solution:

Just **prepend a "0" bit to the message** *M*:

- no more possible for the attacker to synchronize the computation chains: the inner and outer function are made distinct
- no need to change the specification of *H*, even better: can be done on top of HMAC implementations
- almost zero performance drop

Thank you for your attention !