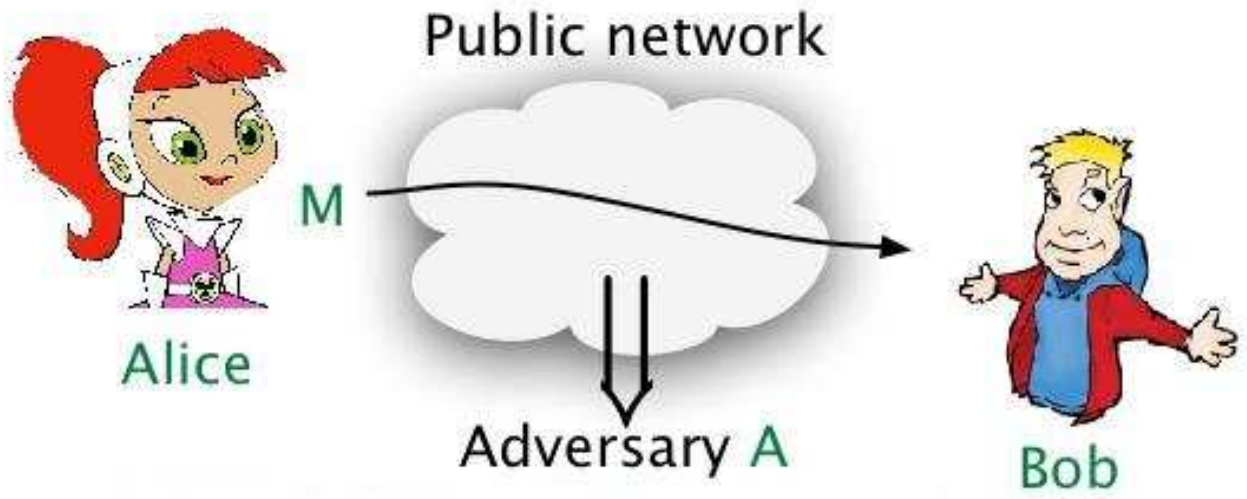
MESSAGE AUTHENTICATION CODES
and
PRF DOMAIN EXTENSION

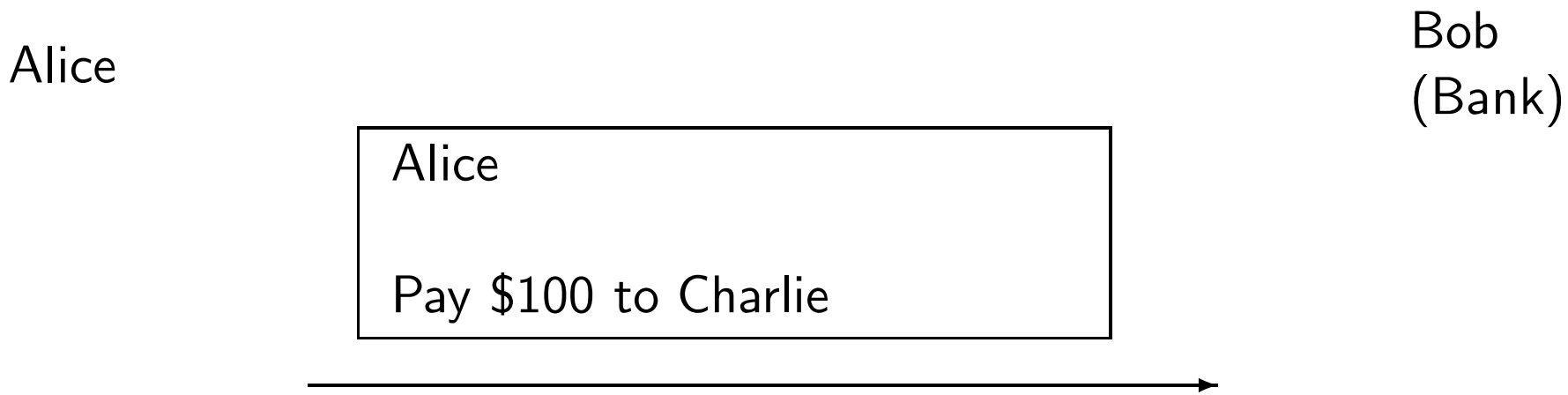
Integrity and authenticity



The goal is to ensure that

- M really originates with Alice and not someone else
- M has not been modified in transit

Integrity and authenticity example



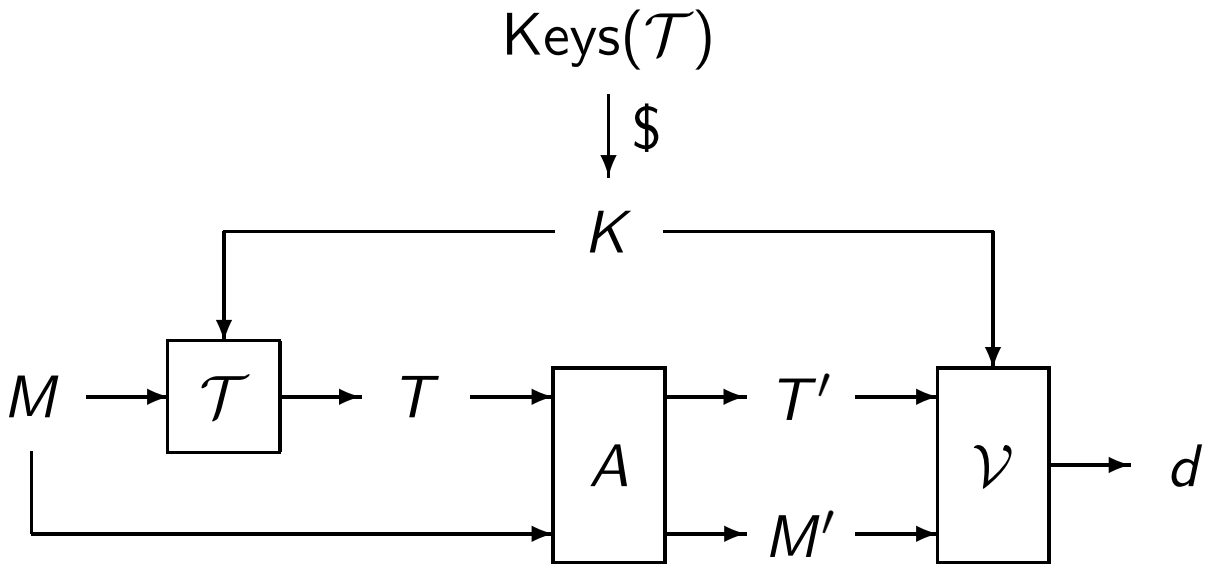
Adversary Eve might

- Modify "Charlie" to "Eve"
- Modify "$100" to "$1000"

Integrity prevents such attacks.

Message authentication codes

A message authentication code $\mathcal{T} : \mathrm{Keys}(\mathcal{T}) \times \mathrm{Dom}(\mathcal{T}) \to \mathrm{Range}(\mathcal{T})$ is a family of functions. The envisaged usage is shown below, where A is the adversary:



We refer to T as the MAC or tag. Here $\mathcal{V}_K(M', T') = 1$ iff $\mathcal{T}_K(M') = T'$.

MAC usage

Sender and receiver share key K.

To authenticate M, sender transmits (M, T) where $T = \mathcal{T}_K(M)$.

Upon receiving (M', T'), the receiver accepts M' as authentic iff $\mathcal{T}_K(M') = T'$.

UF-CMA

Let $\mathcal{T}$: $\mathsf{Keys}(\mathcal{T}) \times \mathsf{Dom}(\mathcal{T}) \to \mathsf{Range}(\mathcal{T})$ be a message authentication code. Let A be an adversary.

Game $\mathrm{UFCMA}_{\mathcal{T}}$	
procedure Initialize $K \xleftarrow{\$} \mathsf{Keys}(\mathcal{T})$; $S \leftarrow \emptyset$ **procedure Tag**(M) $T \leftarrow \mathcal{T}_K(M)$; $S \leftarrow S \cup \{M\}$ return T	**procedure Finalize**(M, T) If $M \in S$ then return false If $M \notin \mathsf{Dom}(\mathcal{T})$ then return false Return $(T = \mathcal{T}_K(M))$

The uf-cma advantage of adversary A is

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{T}}(A) = \Pr\left[\mathrm{UFCMA}^{A}_{\mathcal{T}} \Rightarrow \mathsf{true}\right]$$

UF-CMA: Explanations

Adversary A does not get the key K.

It can call **Tag** with any message M of its choice to get back the correct tag $T = \mathcal{T}_K(M)$.

To win, the adversary A must output a message $M \in \mathrm{Dom}(\mathcal{T})$ and a tag T that are

- Correct: $T = \mathcal{T}_K(M)$
- New: $M \notin S$, meaning M was not a query to **Tag**

Interpretation: Tag represents the sender and **Finalize** represents the receiver. Security means that the adversary can't get the receiver to accept a message that is not authentic, meaning was not already transmitted by the sender.

Tag lengths

For any message authentication code $\mathcal{T}$: $\mathsf{Keys}(\mathcal{T}) \times \mathsf{Dom}(\mathcal{T}) \to \{0,1\}^{\ell}$, we can provide an efficient adversary A achieving $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{T}}(A) = 2^{-\ell}$.

The proof is an exercise.

Conclusion: Tags have to be long enough.

For 80 bit security, tags have to be at least 80 bits.

Example: Basic CBC MAC

Let $E : \{0,1\}^k \times B \to B$ be a blockcipher, where $B = \{0,1\}^n$. View a message $M \in B^*$ as a sequence of n-bit blocks, $M = M[1] \dots M[m]$.

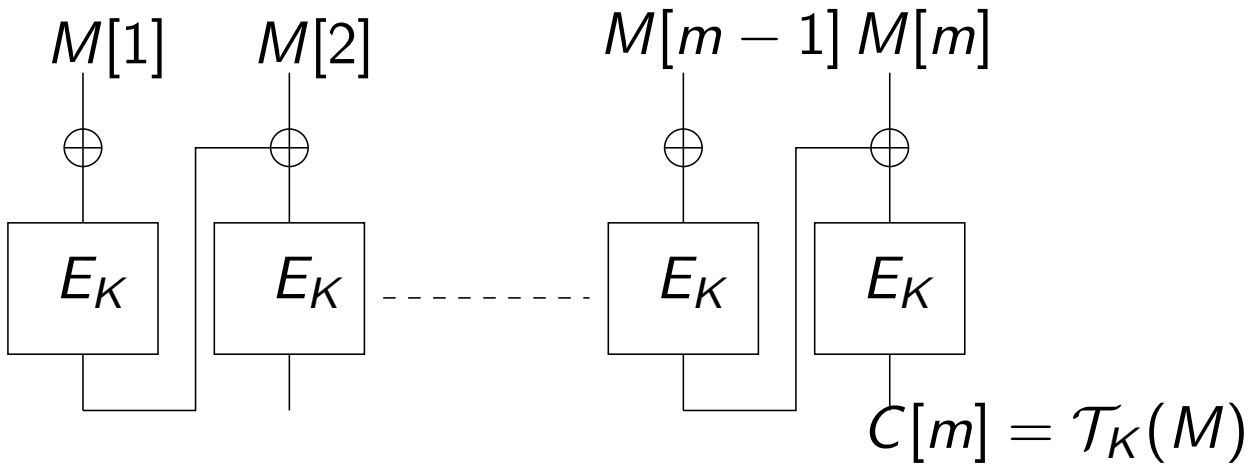
The basic CBC MAC $\mathcal{T} : \{0,1\}^k \times B^* \to B$ is defined by

Alg $\mathcal{T}_K(M)$

$C[0] \leftarrow 0^n$

for $i = 1, \dots, m$ do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$

return $C[m]$



Splicing attack on basic CBC MAC

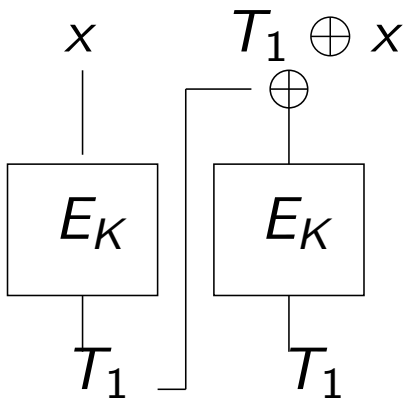
Alg $\mathcal{T}_K(M)$

$C[0] \leftarrow 0^n$
for $i = 1, \ldots, m$ do
    $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$
return $C[m]$

adversary A

Let $x \in \{0,1\}^n$
$T_1 \leftarrow \mathbf{Tag}(x)$
$M \leftarrow x || T_1 \oplus x$
Return M, T_1

Then,



$$\begin{aligned} \mathcal{T}_K(M) &= E_K(E_K(x) \oplus T_1 \oplus x) \\ &= E_K(T_1 \oplus T_1 \oplus x) \\ &= E_K(x) \\ &= T_1 \end{aligned}$$

Insecurity of basic CBC MAC

Alg $\mathcal{T}_K(M)$

$C[0] \leftarrow 0^n$

for $i = 1, \ldots, m$ do

$\quad C[i] \leftarrow E_K(C[i-1] \oplus M[i])$

return $C[m]$

adversary A

Let $x \in \{0,1\}^n$

$T_1 \leftarrow \mathbf{Tag}(x)$

$M \leftarrow x || T_1 \oplus x$

Return M, T_1

Then $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{T}}(A) = 1$ and A is efficient, so the basic CBC MAC is not UF-CMA secure.

Exercise

Let $E\colon\ \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. Let

$$D \ = \ \{\ M \in \{0,1\}^* \ :\ 0 < |M| < n2^n \text{ and } |M| \bmod n = 0\ \}\ .$$

Let $\mathcal{T}\colon\ \{0,1\}^k \times D \to \{0,1\}^n$ be defined as follows:

Alg $\mathcal{T}_K(M)$

$M[1]\ldots M[m] \leftarrow M$; $M[m+1] \leftarrow \langle m \rangle$; $C[0] \leftarrow 0^n$
For $i = 1, \ldots, m+1$ do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$
$T \leftarrow C[m+1]$; return T

Above, $\langle m \rangle$ denotes the n-bit binary representation of the integer m.

Show that $\mathcal{T}$ is not UF-CMA-secure by presenting a practical adversary A making at most 4 **Tag** queries and achieving $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{T}}(A) = 1$.

Exercise

Let $E\colon\ \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. Let

$$D \ = \ \{\ M \in \{0,1\}^* \ :\ 0 < |M| < n2^n \text{ and } |M| \bmod n = 0\ \}\ .$$

Let $\mathcal{T}\colon\ \{0,1\}^k \times D \to \{0,1\}^n$ be defined as follows:

Alg $\mathcal{T}_K(M)$

$M[1] \ldots M[m] \leftarrow M$; $C[0] \leftarrow E_K(\langle m \rangle)$
For $i = 1, \ldots, m$ do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$
$T \leftarrow C[m]$; return T

Above, $\langle m \rangle$ denotes the n-bit binary representation of the integer m.

Is $\mathcal{T}$ UF-CMA-secure? If you say NO, present a practical adversary A achieving $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{T}}(A) \geq 1/2$. If you say YES, prove this correct assuming E is PRF-secure.

Replay

Suppose Alice transmits (M_1, T_1) to Bank where M_1 = "Pay \$100 to Bob". Adversary

- Captures (M_1, T_1)
- Keeps re-transmitting it to bank

Result: Bob gets \$100, \$200, \$300,...

Our UF-CMA notion of security does not ask for protection against replay, because A will not win if it outputs M, T with $M \in S$, even if $T = \mathcal{T}_K(M)$ is the correct tag.

Question: Why not?

Answer: Replay is best addressed as an add-on to standard message authentication.

Preventing Replay Using Timestamps

Let $Time_A$ be the time as per Alice's local clock and $Time_B$ the time as per Bob's local clock.

- Alice sends $(M, \mathcal{T}_K(M), Time_A)$
- Bob receives $(M, T, Time)$ and accepts iff $T = \mathcal{T}_K(M)$ and $|Time_B - Time| \leq \Delta$ where Δ is a small threshold.

Does this work?

Preventing Replay Using Timestamps

Let $Time_A$ be the time as per Alice's local clock and $Time_B$ the time as per Bob's local clock.

- Alice sends $(M, \mathcal{T}_K(M), Time_A)$
- Bob receives $(M, T, Time)$ and accepts iff $T = \mathcal{T}_K(M)$ and $|Time_B - Time| \leq \Delta$ where Δ is a small threshold.

Does this work?

Obviously forgery is possible within a Δ interval. But the main problem is that $Time_A$ is not authenticated, so adversary can transmit

$$(M, \mathcal{T}_K(M), Time_1),\ (M, \mathcal{T}_K(M), Time_2),\ \ldots$$

for any times $Time_1, Time_2, \ldots$ of its choice, and Bob will accept.

Preventing Replay Using Timestamps

Let $Time_A$ be the time as per Alice's local clock and $Time_B$ the time as per Bob's local clock.

- Alice sends $(M, \mathcal{T}_K(M \| Time_A), Time_A)$
- Bob receives $(M, T, Time)$ and accepts iff $\mathcal{T}_K(M \| Time) = T$ and $|Time_B - Time| \leq \Delta$ where Δ is a small threshold.

Preventing Replay Using Counters

Alice maintains a counter ctr_A and Bob maintains a counter ctr_B. Initially both are zero.

- Alice sends $(M, \mathcal{T}_K(M \| ctr_A))$ and then increments ctr_A
- Bob receives (M, T). If $\mathcal{T}_K(M \| ctr_B) = T$ then Bob accepts and increments ctr_B.

Counters need to stay synchronized.

Any PRF is a MAC

If F is PRF-secure then it is also UF-CMA-secure:

Theorem [GGM86,BKR96]: Let $F : \{0,1\}^k \times D \to \{0,1\}^n$ be a family of functions. Let A be a uf-cma adversary making q **Tag** queries and having running time t. Then there is a prf-adversary B such that

$$\mathbf{Adv}^{\text{uf-cma}}_F(A) \leq \mathbf{Adv}^{\text{prf}}_F(B) + \frac{2}{2^n} \ .$$

Adversary B makes $q+1$ queries to its **Fn** oracle and has running time t plus some overhead.

Exercise: Prove this theorem.

We now give some intuition.

Intuition for why PRFs are UF-CMA-secure

1. Random functions make good (UF-CMA) MACs
2. PRFs are pretty much as good as random functions

For (1), suppose $\mathbf{Fn} : D \to \{0,1\}^n$ is random and consider A who

- Can query $\mathbf{Fn}$ at any points $x_1, \ldots, x_q \in D$ it likes
- To win, must output x, T such that $x \notin \{x_1, \ldots, x_q\}$ but $T = \mathbf{Fn}(x)$

Then,

$$\Pr[A \text{ wins}] =$$

Intuition for why PRFs are UF-CMA-secure

1. Random functions make good (UF-CMA) MACs
2. PRFs are pretty much as good as random functions

For (1), suppose $\mathbf{Fn} : D \to \{0,1\}^n$ is random and consider A who

- Can query $\mathbf{Fn}$ at any points $x_1, \ldots, x_q \in D$ it likes
- To win, must output x, T such that $x \notin \{x_1, \ldots, x_q\}$ but $T = \mathbf{Fn}(x)$

Then,

$$\Pr[A \text{ wins}] = \frac{1}{2^n}$$

because A did not query $\mathbf{Fn}(x)$.

Intuition for why PRFs are UF-CMA-secure

1. Random functions make good (UF-CMA) MACs
2. PRFs are pretty much as good as random functions

For (2), consider A who

- Can query F_K at any points $x_1, \ldots, x_q \in D$ it likes
- To win, must output x, T such that $x \notin \{x_1, \ldots, x_q\}$ but $T = F_K(x)$

If $\Pr[A \text{ wins}]$ is significantly more then 2^{-n} then we are detecting a difference between F_K and a random function.

Exercise: PRF $\neq$ UF-CMA

In this exercise you will show that UF-CMA-security does not imply PRF-security, meaning there exist function families that are UF-CMA-secure but not PRF-secure.

You are given F': $\{0,1\}^k \times D \to \{0,1\}^n$ which is UF-CMA-secure. Construct F: $\{0,1\}^k \times D \to \{0,1\}^\ell$ such that

(1) F is UF-CMA-secure, but

(2) F is not PRF-secure

Specify F, including specifying ℓ. Then prove (1) by reduction based on the assumed UF-CMA-security of F'. Finally prove (2) by giving pseudocode for a prf-adversary that has prf-advantage $\geq 1/2$ against F.

PRF domain extension

A family of functions F: $\text{Keys}(F) \times D \to R$ is

- FIL (Fixed-input-length) if $D = \{0,1\}^\ell$ for some ℓ
- VIL (Variable-input-length) if D is a "large" set like $D = \{0,1\}^*$ or

$$D = \{ M \in \{0,1\}^* : 0 < |M| < n2^n \text{ and } |M| \bmod n = 0 \} .$$

for some $n \geq 1$ or ...

We have families we are willing to assume are PRFs, namely blockciphers and compression functions, but they are FIL.

PRF Domain Extension Problem: Given a FIL PRF, construct a VIL PRF.

PRF domain extension

PRF Domain Extension Problem: Given a FIL PRF, construct a VIL PRF.

The basic CBC MAC is a candidate construction but we saw above that it fails to be UF-CMA and thus also fails to be a PRF. The exercises explored other solutions.

We will see solutions that work including

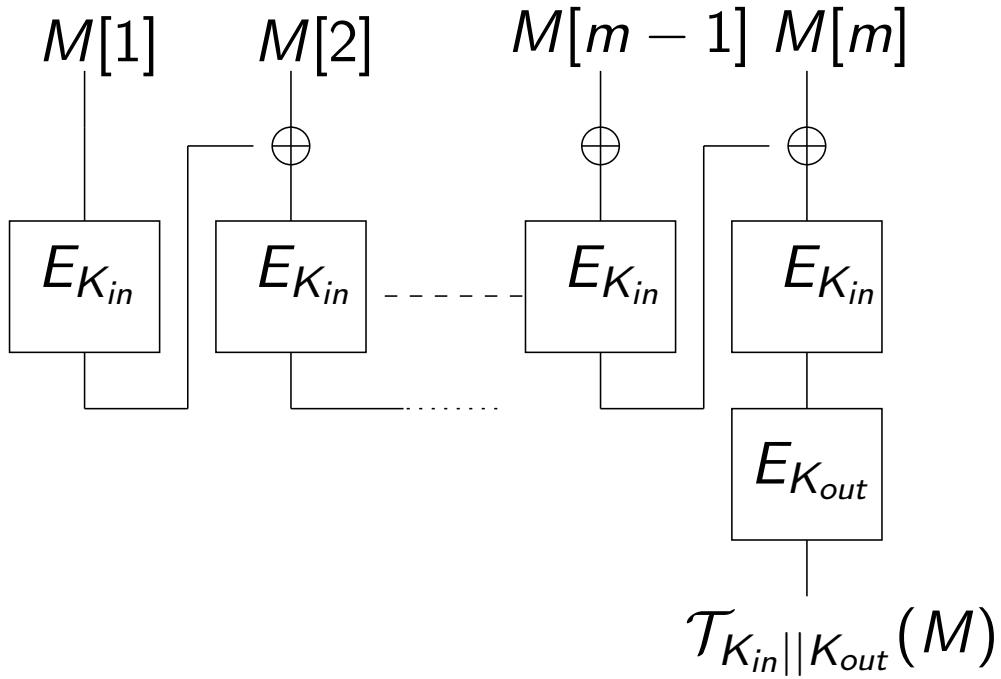
- ECBC: The encrypted CBC-MAC
- CMAC: A NIST standard
- HMAC: A highly standardized and used hash-function based MAC

ECBC MAC

Let $E : \{0,1\}^k \times B \to B$ be a block cipher, where $B = \{0,1\}^n$. The encrypted CBC (ECBC) MAC $\mathcal{T} : \{0,1\}^{2k} \times B^* \to B$ is defind by

Alg $\mathcal{T}_{K_{in}||K_{out}}(M)$

$C[0] \leftarrow 0^n$
for $i = 1, ..., m$ do
  $C[i] \leftarrow E_{K_{in}}(C[i-1] \oplus M[i])$
$T \leftarrow E_{K_{out}}(C[m])$
return T



Birthday attacks on MACs

There is a large class of MACs, including ECBC MAC, CMAC, HMAC, ... which are subject to a birthday attack that violates UF-CMA using about $q \approx 2^{n/2}$ **Tag** queries, where n is the tag (output) length of the MAC.

Furthermore, we can typically show this is best possible, so the birthday bound is the "true" indication of security.

The class of MACs in question are called iterated-MACs and work by iterating some lower level primitive such as a blockcipher or compression function.

Exercise

Let $E\colon\ \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher and $\mathcal{T}$ the corresponding ECBC MAC. Let $q \leq 2^{n/2}$.

Give an adversary A that, via a birthday attack, achieves

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{T}}(A) = \Omega\left(\frac{q^2}{2^n}\right)$$

using q **Tag** queries and running time $\mathcal{O}(nq \cdot \log(nq))$.

Security of ECBC

Birthday attack is best possible:

Theorem: Let $E : \{0,1\}^k \times B \to B$ be a family of functions, where $B = \{0,1\}^n$. Define $F : \{0,1\}^{2k} \times B^* \to \{0,1\}^n$ by

Alg $F_{K_{in}||K_{out}}(M)$

$C[0] \leftarrow 0^n$
for $i = 1, \ldots, m$ do $C[i] \leftarrow E_{K_{in}}(C[i-1] \oplus M[i])$
$T \leftarrow E_{K_{out}}(C[m])$; return T

Let A be a prf-adversary against F that makes at most q oracle queries, these totalling at most σ blocks, and has running time t. Then there is a prf-adversary B against E such that

$$\mathbf{Adv}^{\mathrm{prf}}_F(A) \leq \mathbf{Adv}^{\mathrm{prf}}_E(B) + \frac{\sigma^2}{2^n}$$

and B makes at most σ oracle queries and has running time about t.

Security of iterated MACs

The number q of m-block messages that can be safely authenticated is about $2^{n/2}/m$, where n is the block-length of the blockcipher, or the length of the chaining input of the compression function.

MAC	n	m	q
DES-ECBC-MAC	64	1024	2^{22}
AES-ECBC-MAC	128	1024	2^{54}
AES-ECBC-MAC	128	10^6	2^{44}
HMAC-SHA1	160	10^6	2^{60}
HMAC-SHA256	256	10^6	2^{108}

$m = 10^6$ means message length 16Mbytes when $n = 128$.

Non-full messages

So far we assumed messages have length a multiple of the block-length of the blockcipher. Call such messages *full*. How do we deal with non-full messages?

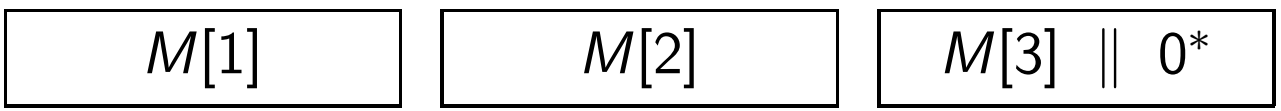
The obvious approach is padding. But how we pad matters.

Non-full messages

So far we assumed messages have length a multiple of the block-length of the blockcipher. Call such messages *full*. How do we deal with non-full messages?

The obvious approach is padding. But how we pad matters.

Padding with 0^*:

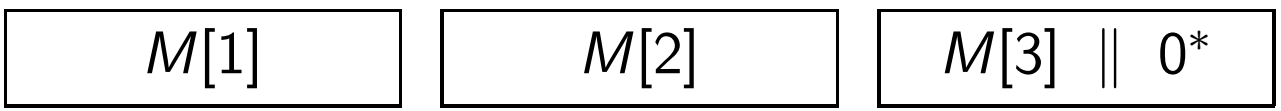


Non-full messages

So far we assumed messages have length a multiple of the block-length of the blockcipher. Call such messages *full*. How do we deal with non-full messages?

The obvious approach is padding. But how we pad matters.

Padding with 0^*:

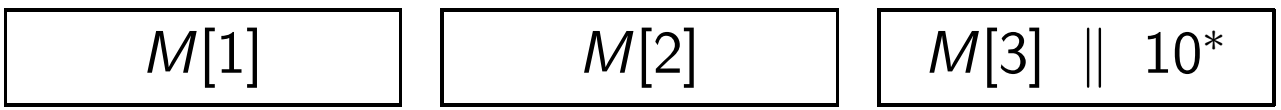


adversary A
$T \leftarrow \mathbf{Tag}(1^n 1^n 0)$; $\mathbf{Verify}(1^n 1^n 00, T)$

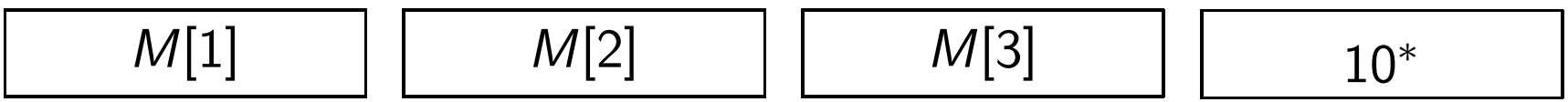
This adversary has uf-cma advantage 1.

Non-full messages

Padding with 10^*: For a non-full message



For a full message



This works, but if M was full, an extra block is needed leading to an extra blockcipher operation.

Costs

Handling length-variablity and non-full messages leads to two extra blockcipher invocations in ECBC MAC as compared to basic CBC MAC.

Also ECBC uses two blockcipher keys and needs to rekey, which is expensive.

Can we do better?

CMAC

Standards: NIST SP 800-38B, RFCs 4493, 4494, 4615

Features: Handles variable-length and non-full messages with

- Minimal overhead
- A single blockcipher key

Security:

- Subject to a birthday attack
- Security proof shows there is no better attack

History: XCBC[BIRo], OMAC/OMAC1[IW]

CMAC Components and Setup

- $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a blockcipher, in practice AES.
- $\mathrm{CBC}_K(M)$ is the basic CBC MAC of a full message M under key $K \in \{0,1\}^n$ and using E.
- $J \in \{0,1\}^n$ is a particular fixed constant.

CMAC uses its key $K \in \{0,1\}^n$ to derive subkeys K_1, K_2 via

Alg $\mathrm{CMAC\text{-}KEYGEN}(K)$

$K_0 \leftarrow E_K(0)$
if $\mathrm{msb}(K_0) = 0$ then $K_1 \leftarrow (K_0 \ll 1)$ else $K_1 \leftarrow (K_0 \ll 1) \oplus J$
if $\mathrm{msb}(K_1) = 0$ then $K_2 \leftarrow (K_1 \ll 1)$ else $K_2 \leftarrow (K_1 \ll 1) \oplus J$
Return (K_1, K_2)

where $x \ll 1$ means x left shifted by 1 bit, so that the msb vanishes and the lsb becomes 0. These bit operations use simple finite-field operations.

CMAC Algorithm

Alg $\mathrm{CMAC}_K(M)$

$(K_1, K_2) \leftarrow \mathrm{CMAC\text{-}KEYGEN}(K)$
$M[1] \ldots M[m-1]M[m] \leftarrow M \quad // \ |M[m]| \leq n$
$\ell \leftarrow |M[m]| \quad // \ \ell \leq n$
if $\ell = n$ then $M[m] \leftarrow K_1 \oplus M[m]$
else $M[m] \leftarrow K_2 \oplus (M[m] \| 10^{n-\ell-1})$
$M \leftarrow M[1] \ldots M[m-1]M[m]$
$T \leftarrow \mathrm{CBC}_K(M)$
return T

In an implementation, $\mathrm{CMAC\text{-}KEYGEN}(K)$ is run once, meaning K_1, K_2 are pre-computed, stored and re-used. Performance is then optimal.

MACing with hash functions

The software speed of hash functions (MD5, SHA1) lead people in 1990s to ask whether they could be used to MAC.

But such cryptographic hash functions are keyless.

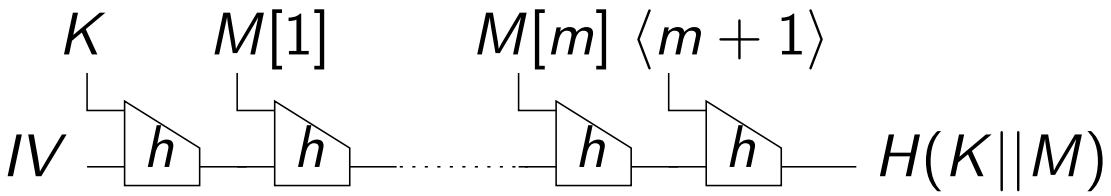
Question: How do we key hash functions to get MACs?

Proposal: Let $H : D \to \{0,1\}^n$ represent the hash function and set

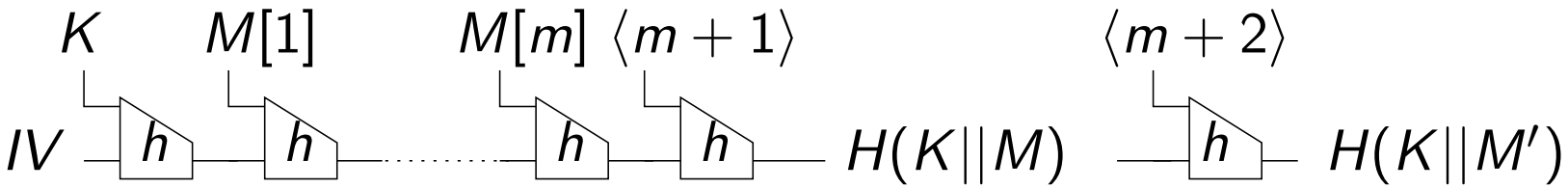
$$\mathcal{T}_K(M) = H(K||M)$$

Is this UF-CMA / PRF secure?

Extension attack



Extension attack



Let $M' = M||\langle m+1 \rangle$. Then

$$H(K||M') = h(\langle m+2 \rangle || H(K||M))$$

so given the MAC $H(K||M)$ of M we can easily forge the MAC of M'.

Exercise: Specify in pseudocode an adversary mounting the above attack to achieve uf-cma advantage 1 using 1 **Tag** query.

HMAC [BCK96]

Suppose $H : D \to \{0,1\}^{160}$ is the hash function. HMAC has a 160-bit key K. Let

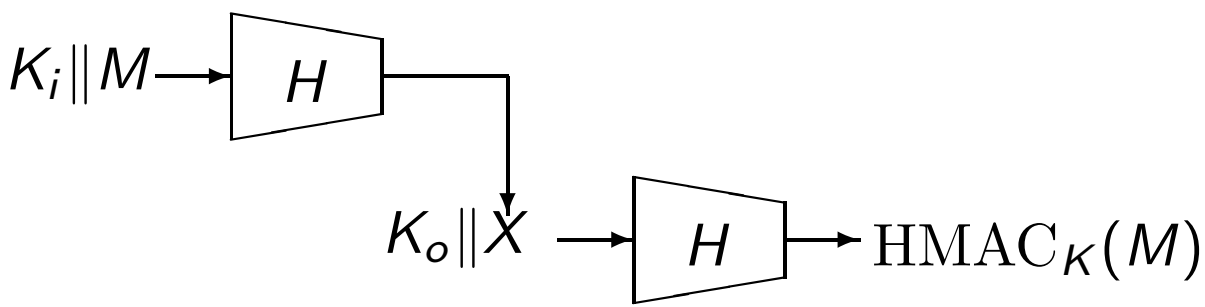
$$K_o = \text{opad} \oplus K||0^{352} \text{ and } K_i = \text{ipad} \oplus K||0^{352}$$

where

$$\text{opad} = 5D \text{ and ipad} = 36$$

in HEX. Then

$$\text{HMAC}_K(M) = H(K_o||H(K_i||M))$$



HMAC

Features:

- Blackbox use of the hash function, easy to implement
- Fast in software

Usage:

- As a MAC for message authentication
- As a PRF for key derivation

Security:

- Subject to a birthday attack
- Security proof shows there is no better attack [BCK96,Be06]

Adoption and Deployment: HMAC is one of the most widely standardized and used cryptographic constructs: SSL/TLS, SSH, IPSec, FIPS 198, IEEE 802.11, IEEE 802.11b, ...

HMAC Security

Theorem: [BCK96] HMAC is a secure PRF assuming

- The compression function is a PRF
- The hash function is collision-resistant (CR)

But recent attacks show MD5 is not CR and SHA1 may not be either.

So are HMAC-MD5 and HMAC-SHA1 secure?

- No attacks so far, but
- Proof becomes vacuous!

Theorem: [Be06] HMAC is a secure PRF assuming only

- The compression function is a PRF

Current attacks do not contradict this assumption. This new result may explain why HMAC-MD5 is standing even though MD5 is broken with regard to collision resistance.

HMAC Recommendations

- Don't use HMAC-MD5
- No immediate need to remove HMAC-SHA1
- Use HMAC-SHA256 for new applications