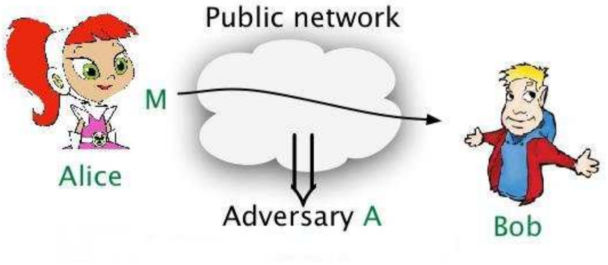
MESSAGE AUTHENTICATION

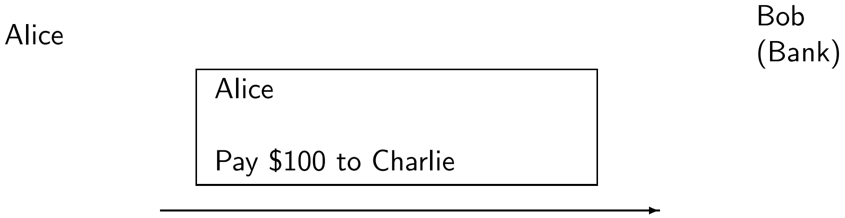
Integrity and authenticity



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



Alice

Alice

Pay $100 to Charlie

Bob
(Bank)

Adversary Eve might

- Modify “Charlie” to “Eve”
- Modify “$100” to “$1000”

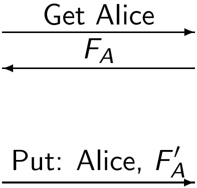
Integrity prevents such attacks.

Medical databases

Doctor

Reads F_A

Modifies F_A to F'_A

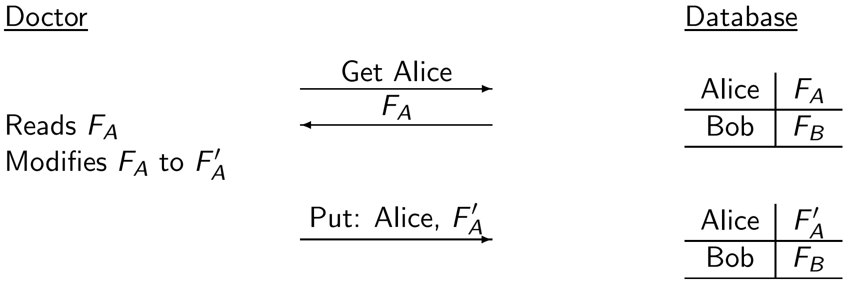


Database

Alice	F_A
Bob	F_B

Alice	F'_A
Bob	F_B

Medical databases



Doctor

Database

Get Alice

F_A

Alice	F_A
Bob	F_B

Reads F_A

Modifies F_A to F'_A

Put: Alice, F'_A

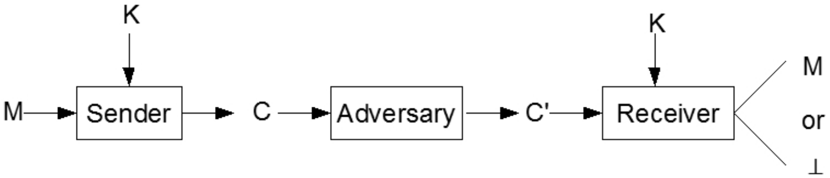
Alice	F'_A
Bob	F_B

Need to ensure

- doctor is authorized to get Alice's file
- F_A, F'_A are not modified in transit
- F_A is really sent by database
- F'_A is really sent by (authorized) doctor

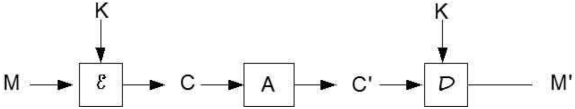
Symmetric Setting

We will study how to authenticate messages in the *symmetric* setting where Sender and Receiver share a random key K not given to the adversary.



Does privacy provide authenticity?

Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a (IND-CPA secure) symmetric encryption scheme.



Say

$$M = \text{“Pay \$100 to Bob”}$$

Adversary wants Receiver to get

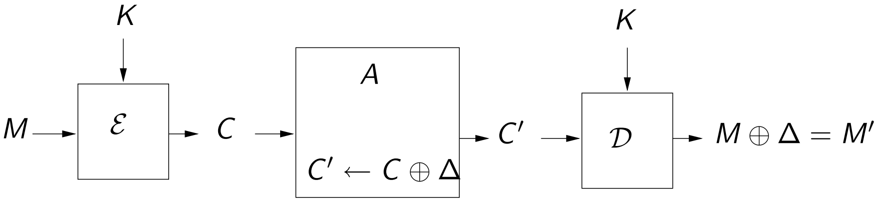
$$M' = \text{“Pay \$1,000 to Bob”}$$

Adversary needs to modify C to C' such that $\mathcal{D}_K(C') = M'$.

Intuition: It is hard to modify C to ensure above, since modifying C will result in $\mathcal{D}_K(C)$ being garbled/random and Receiver will reject.

Counterexample: OTP

Say $\mathcal{E}_K(M) = K \oplus M$ and $\mathcal{D}_K(C) = K \oplus C$. Should assume adversary knows M. Then it can let $\Delta = M \oplus M'$ and



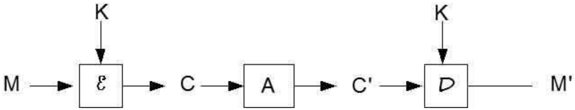
because

$$\mathcal{D}_K(C \oplus \Delta) = K \oplus C \oplus \Delta = M \oplus \Delta$$

Adding redundacy

Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a (IND-CPA secure) symmetric encryption scheme. To send M, sender computes $C \overset{\$}{\leftarrow} \mathcal{E}_K(0^{128}||M)$ and sends C to receiver.

Receiver gets C' and lets $R||M \leftarrow \mathcal{D}_K(C')$. If $R = 0^{128}$ it outputs M else $\perp$.



Intuition: If C is modified to C' then most probably the first 128 bits of $\mathcal{D}_K(C')$ will not all be 0 and Receiver will reject.

However, OTP again provides a counterexample to show that this does not provide integrity.

What went wrong?

Possible reaction: OTP is bad! Use CBC instead.

But CBC has similar problems.

What went wrong?

Possible reaction: OTP is bad! Use CBC instead.

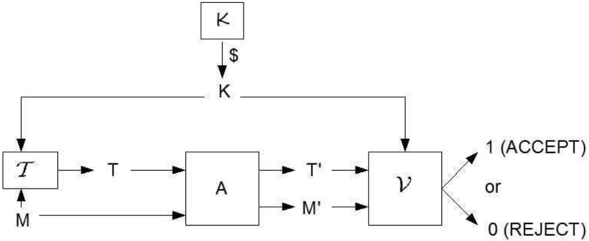
But CBC has similar problems.

The real problem: There is no good reason to think that privacy provides authenticity. Encryption is the wrong tool here.

To call an encryption scheme bad because it does not provide authenticity is like calling a car bad because it does not fly. To fly you need an airplane.

Message authentication schemes

A message authentication (MA) scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ consists of three algorithms:



We refer to T as the MAC or tag.

We let

- $\mathcal{T}_K(\cdot) = \mathcal{T}(K, \cdot)$
- $\mathcal{V}_K(\cdot) = \mathcal{V}(K, \cdot, \cdot)$

Consistency

Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be any MA scheme. We require that for all messages M,

$$\mathcal{V}_K(M, \mathcal{T}_K(M)) = 1$$

with probability one, where the probability is over the choice of K and the coins of $\mathcal{T}$.

That is, unaltered tags are accepted.

Example

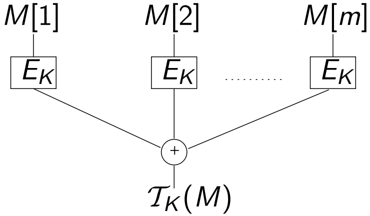
Let $E\colon \{0,1\}^k \times B \to B$ be a block cipher, where $B = \{0,1\}^n$. View a message $M \in B^*$ as a sequence of n-bit blocks,

$$M = M[1] \cdots M[m]$$

```
Alg $\mathcal{K}$
$K \xleftarrow{\$} \{0,1\}^k$
return $K$
```

```
Alg $\mathcal{T}_K(M)$
$T \leftarrow 0^n$
for $i = 1, \ldots, m$ do
    $T \leftarrow T \oplus E_K(M[i])$
return $T$
```

```
Alg $\mathcal{V}_K(M, T)$
if $T = \mathcal{T}_K(M)$ then
    return 1
else return 0
```



Security: What the adversary gets

Certainly it knows the scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$

We should also assume it can see a sequence $(M_1, T_1), \ldots, (M_q, T_q)$ of correctly tagged messages sent by the sender, meaning $T_i \overset{\$}{\leftarrow} \mathcal{T}_K(M_i)$ for $i = 1, \ldots, q$.

Some choices here

- Known message attack: Adversary does not influence choice of $M_1, \ldots, M_q$
- Chosen-message attack: Adversary chooses $M_1, \ldots, M_q$

Security: Key-recovery

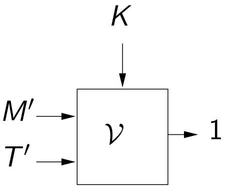
We certainly want to ensure that an adversary cannot recover the key.

But this condition, while necessary for security, is not sufficient.

Security: Forgery

We say that an adversary succeeds in forgery if it produces M', T' such that

- Verifier accepts

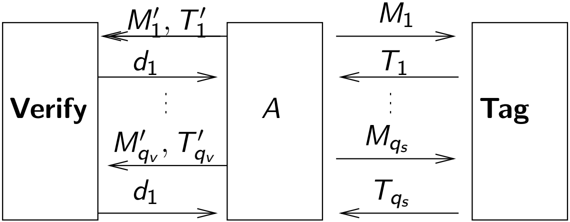


- But sender never sent (tagged) M'

We want to prevent forgery.

uf-cma adversaries

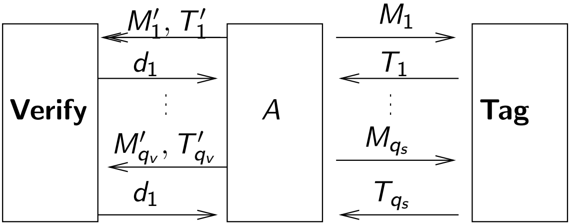
Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a MA scheme. A uf-cma adversary has oracles **Tag**$(\cdot) = \mathcal{T}_K(\cdot)$ and **Verify**$(\cdot,\cdot) = \mathcal{V}_K(\cdot,\cdot)$.



Tag represents the sender and **Verify** represents the receiver.

uf-cma adversaries

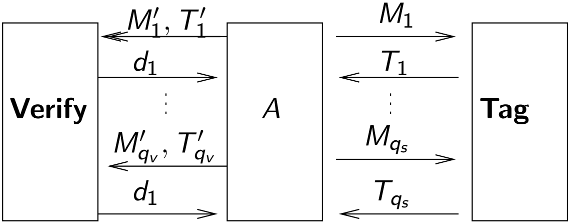
Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a MA scheme. A uf-cma adversary has oracles **Tag**$(\cdot) = \mathcal{T}_K(\cdot)$ and **Verify**$(\cdot,\cdot) = \mathcal{V}_K(\cdot,\cdot)$.



We want to say that A wins if it ever gets **Verify** to accept. But it can do this trivially by sending, say, (M_1, T_1) to **Verify**. This however isn't really a forgery because M_1 is authentic, meaning tagged by the sender.

UF-CMA

Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a MA scheme. A uf-cma adversary has oracles **Tag**$(\cdot) = \mathcal{T}_K(\cdot)$ and **Verify**$(\cdot, \cdot) = \mathcal{V}_K(\cdot, \cdot)$.



Verify

M'_1, T'_1

d_1

$\vdots$

M'_{q_v}, T'_{q_v}

d_1

A

M_1

T_1

$\vdots$

M_{q_s}

T_{q_s}

Tag

We say A wins if $\exists i$ such that

- **Verify**(M'_i, T'_i) returned 1, but
- A did not query M'_i to **Tag** prior to querying M'_i, T'_i to **Verify**

Security means that the adversary can't get the receiver to accept a message that was not already transmitted by the sender.

Definition: UF-CMA

Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme and A a uf-cma adversary.

Game $\mathrm{UFCMA}_{\mathcal{MA}}$

```
procedure Initialize
$K \xleftarrow{\$} \mathcal{K}; S \leftarrow \emptyset$

procedure Verify($M, T$)
$d \leftarrow \mathcal{V}_K(M, T)$
If $(d = 1 \wedge M \notin S)$ then win ← true
return $d$

procedure Tag($M$)
$T \xleftarrow{\$} \mathcal{T}_K(M)$
$S \leftarrow S \cup \{M\}$
return $T$

procedure Finalize
return win
```

The uf-cma advantage of adversary A is

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

The measure of success

Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme and A a uf-cma adversary. Then

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

is a number between 0 and 1.

A "large" (close to 1) advantage means

- A is doing well
- $\mathcal{MA}$ is not secure

A "small" (close to 0) advantage means

- A is doing poorly
- $\mathcal{MA}$ resists the attack A is mounting

MAC security

Adversary advantage depends on its

- Strategy
- Resources: Running time t and numbers q_s, q_v of queries to the **Tag** and **Verify** oracles, respectively.

Security: $\mathcal{MA}$ is a secure MA scheme (UF-CMA) if $\mathbf{Adv}_F^{\mathrm{uf\text{-}cma}}(A)$ is "small" for ALL A that use "practical" amounts of resources.

Insecurity: $\mathcal{MA}$ is insecure (not UF-CMA) if there exists A using "few" resources that achieves "high" advantage.

Tag lengths

Suppose MA scheme $\mathcal{MA}$ has tags of length ℓ. Then one can forge with probability $q/2^\ell$ in q verification attempts:

```
adversary $A$
Let $M$ be any message
For $i = 1, \ldots, q$ do $d \leftarrow \textbf{Verify}(M, \langle i \rangle)$
```

Here $\langle i \rangle$ is the ℓ-bit binary representation of i. The advantage of A is

$$\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{MA}}(A) = \frac{q}{2^\ell} \ .$$

Conclusion: Tags have to be long enough.

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

MACs

Associate to a family of functions $F : \{0,1\}^k \times D \to \{0,1\}^n$ the MA scheme $\mathcal{MA}[F] = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ with

```
Alg $\mathcal{K}$
$K \xleftarrow{\$} \{0,1\}^k$
return $K$
```

```
Alg $\mathcal{T}(K, M)$
$T \leftarrow F_K(M)$
return $T$
```

```
Alg $\mathcal{V}(K, M, T)$
if $T = F_K(M)$ then return 1
else return 0
```

We refer to such a MA scheme as a MAC (message authentication code). Its features are:

- Tag computation is deterministic and stateless.
- Verification is by tag re-computation.

Most MA scheme we will see will be MACs.

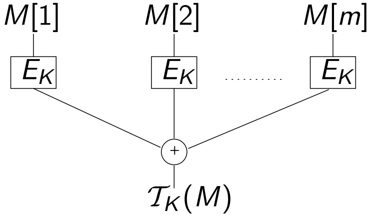
Example 1

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

$$M = M[1] \dots M[m]$$

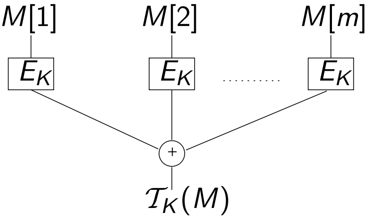
Consider the family of functions $\mathcal{T} : \{0,1\}^k \times B^* \to B$ defined by

$$\mathcal{T}_K(M[1] \dots M[m]) = E_K(M[1]) \oplus \cdots \oplus E_K(M[m]).$$



Is the MAC $\mathcal{MA}[\mathcal{T}]$ secure?

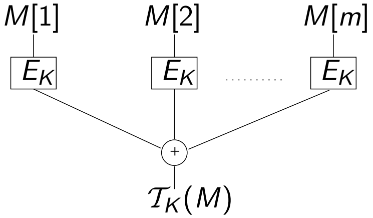
Example 1



Is there a way to produce a message M' and its correct tag T'

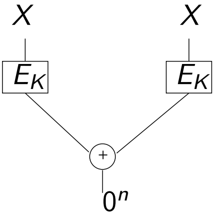
- without knowing K
- possibly knowing a few input-output examples of $\mathcal{T}_K$?

Example 1



Weakness:

$$\mathcal{T}_K(XX) = E_K(X) \oplus E_K(X) = 0^n$$



Example 1

Let $\mathcal{T} : \{0,1\}^k \times B^* \rightarrow B$ be defined by

$$\mathcal{T}_K(M[1]\ldots M[m]) = E_K(M[1]) \oplus \cdots \oplus E_K(M[m])$$

and let $\mathcal{MA}[\mathcal{T}] = (\mathcal{K}, \mathcal{T}, \mathcal{V})$.

adversary A

$M \leftarrow 0^n||0^n$; $T \leftarrow 0^n$; $d \leftarrow$ **Verify**(M, T)

Then

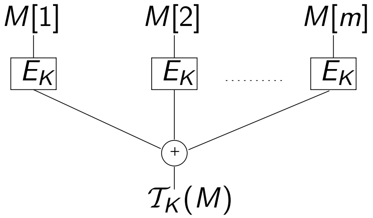
$$\mathcal{T}_K(M) = E_K(0^n) \oplus E_K(0^n) = 0^n = T$$

so

$$\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{MA}[\mathcal{T}]}(A) = 1$$

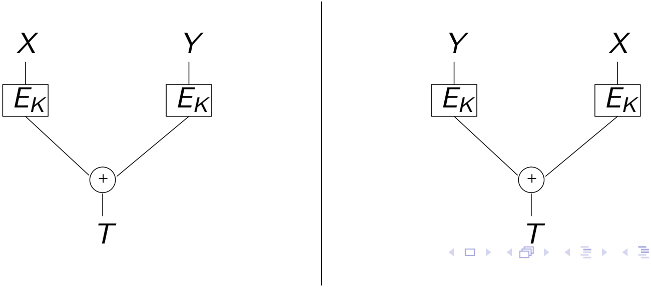
So $\mathcal{MA}[\mathcal{T}]$ is not UF-CMA secure.

Example 1



Another weakness:

$$\mathcal{T}_K(XY) = E_K(X) \oplus E_K(Y) = E_K(Y) \oplus E_K(X) = \mathcal{T}_K(YX)$$



Example 1

Let $\mathcal{T} : \{0,1\}^k \times B^* \to B$ be defined by

$$\mathcal{T}_K(M[1]\dots M[m]) = E_K(M[1]) \oplus \cdots \oplus E_K(M[m])$$

and let $\mathcal{MA}[\mathcal{T}] = (\mathcal{K}, \mathcal{T}, \mathcal{V})$.

adversary A
$T \leftarrow \textbf{Tag}(1^n0^n)\,;\; d \leftarrow \textbf{Verify}(0^n1^n, T)$

Then

$$\begin{aligned} \mathcal{T}_K(1^n0^n) &= E_K(1^n) \oplus E_K(0^n) \\ &= E_K(0^n) \oplus E_K(1^n) \\ &= \mathcal{T}_K(0^n1^n) \end{aligned}$$

so

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}[\mathcal{T}]}(A) = 1$$

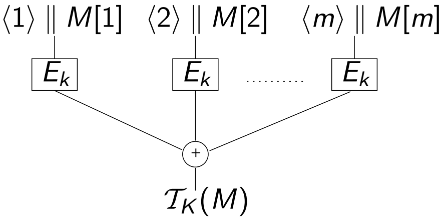
Example 2

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

$$M = M[1] \ldots M[m]$$

where $\ell = n - 32$. Let $\mathcal{T} : \{0,1\}^k \times B^* \to B$ be defined by

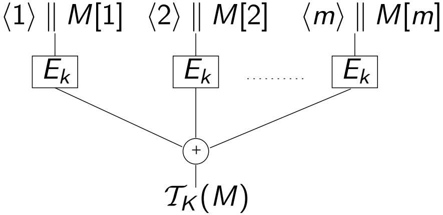
$$\mathcal{T}_K(M[1] \ldots M[m]) = E_K(\langle 1 \rangle || M[1]) \oplus \cdots \oplus E_K(\langle m \rangle || M[m])$$


$\langle 1 \rangle \parallel M[1]$
$\langle 2 \rangle \parallel M[2]$
$\langle m \rangle \parallel M[m]$
E_k
E_k
E_k
+
$\mathcal{T}_K(M)$

Notation:

$\langle i \rangle$ is the 32-bit binary representation of the block index i

Example 2

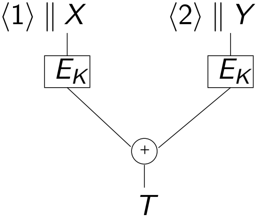

$\langle 1 \rangle \parallel M[1]$
$\langle 2 \rangle \parallel M[2]$
$\langle m \rangle \parallel M[m]$
E_k
E_k
E_k
$+$
$\mathcal{T}_K(M)$

$$\begin{aligned}
\mathcal{T}_K(0^\ell || 0^\ell) &= E_K(\langle 1 \rangle || 0^\ell) \oplus E_K(\langle 2 \rangle || 0^\ell) \\
&\neq 0^n
\end{aligned}$$

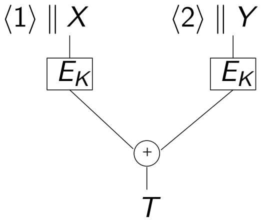
$$\begin{aligned}
\mathcal{T}_K(1^\ell || 0^\ell) &= E_K(\langle 1 \rangle || 1^\ell) \oplus E_K(\langle 2 \rangle || 0^\ell) \\
&\neq E_K(\langle 1 \rangle || 0^\ell) \oplus E_K(\langle 2 \rangle || 1^\ell) \\
&= \mathcal{T}_K(0^\ell || 1^\ell)
\end{aligned}$$

So previous attacks fail.

Example 2


$\langle 1 \rangle \parallel X$
$\langle 2 \rangle \parallel Y$
E_K
E_K
+
T

Example 2


$\langle 1\rangle \parallel X$
$\langle 2\rangle \parallel Y$
E_K
E_K
$+$
T

Weakness: suppose we have

$$\begin{aligned} T_1 &= \mathcal{T}_K(X_1 Y_1) = E_K(\langle 1\rangle \parallel X_1) \oplus E_K(\langle 2\rangle \parallel Y_1) \\ T_2 &= \mathcal{T}_K(X_1 Y_2) = E_K(\langle 1\rangle \parallel X_1) \oplus E_K(\langle 2\rangle \parallel Y_2) \\ T_3 &= \mathcal{T}_K(X_2 Y_1) = E_K(\langle 1\rangle \parallel X_2) \oplus E_K(\langle 2\rangle \parallel Y_1) \end{aligned}$$

Add these and we get

$$T_1 \oplus T_2 \oplus T_3 = E_K(\langle 1\rangle \parallel X_2) \oplus E_K(\langle 2\rangle \parallel Y_2) = \mathcal{T}_K(X_2 Y_2)$$

so we computed the tag of $X_2 \parallel Y_2$.

Attack on Example 2

Let $\mathcal{T} : \{0,1\}^k \times B^* \rightarrow B$ be defined by

$$\mathcal{T}_K(M[1] \ldots M[m]) = E_K(\langle 1 \rangle || M[1]) \oplus \cdots \oplus E_K(\langle m \rangle || M[m])$$

and let $\mathcal{MA}[\mathcal{T}] = (\mathcal{K}, \mathcal{T}, \mathcal{V})$.

```
adversary $A$
Let $x_1, x_2, y_1, y_2$ be distinct $\ell$-bit strings
$T_1 \leftarrow \textbf{Tag}(x_1 \parallel y_1)$    // $T_1 = E_K(\langle 1 \rangle \parallel x_1) \oplus E_K(\langle 2 \rangle \parallel y_1)$
$T_2 \leftarrow \textbf{Tag}(x_1 \parallel y_2)$    // $T_2 = E_K(\langle 1 \rangle \parallel x_1) \oplus E_K(\langle 2 \rangle \parallel y_2)$
$T_3 \leftarrow \textbf{Tag}(x_2 \parallel y_1)$    // $T_3 = E_K(\langle 1 \rangle \parallel x_2) \oplus E_K(\langle 2 \rangle \parallel y_1)$
$T_4 \leftarrow T_1 \oplus T_2 \oplus T_3$
$d \leftarrow \textbf{Verify}(x_2 \parallel y_2, T_4)$
```

So

$$T_4 = E_K(\langle 1 \rangle || x_2) \oplus E_K(\langle 2 \rangle || y_2)$$

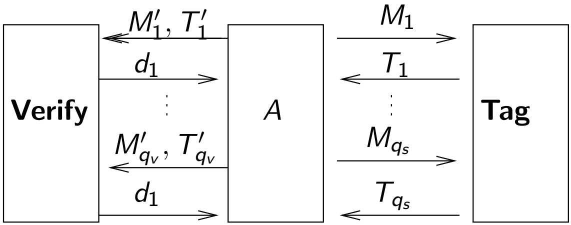
and

$$\textbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}[\mathcal{T}]}(A) = 1$$

UF-CMA

Adversary

- Is allowed a chosen-message attack (CMA)
- Yet should not succeed in existential forgery (UF)



Verify

M'_1, T'_1

d_1

$\vdots$

M'_{q_v}, T'_{q_v}

d_1

A

M_1

T_1

$\vdots$

M_{q_s}

T_{q_s}

Tag

We say A wins if $\exists i$ such that

- **Verify**(M'_i, T'_i) returned 1, but
- A did not query M'_i to **Tag** prior to querying M'_i, T'_i to **Verify**.

Plan

- Replay
- Justifying UF
- Justifying CMA

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 notion of security does not ask for protection against replay.

Question: Why not?

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

Preventing Replay Using Timestamps

Let T_A be the time as per Alice's local clock and T_B the time as per Bob's local clock.

- Alice sends $(M, \mathcal{T}_K(M), T_A)$
- Bob receives (M, tag, T) and accepts iff $\mathcal{V}_K(M, tag) = 1$ and $|T_B - T| \leq \Delta$ where Δ is a small threshold.

Does this work?

Preventing Replay Using Timestamps

Let T_A be the time as per Alice's local clock and T_B the time as per Bob's local clock.

- Alice sends $(M, \mathcal{T}_K(M), T_A)$
- Bob receives (M, tag, T) and accepts iff $\mathcal{V}_K(M, tag) = 1$ and $|T_B - T| \leq \Delta$ where Δ is a small threshold.

Does this work?

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

$$(M, \mathcal{T}_K(M), T_1),\ (M, \mathcal{T}_K(M), T_2),\ \ldots$$

for any times $T_1, T_2, \ldots$ of its choice, and Bob will accept.

Preventing Replay Using Timestamps

Let T_A be the time as per Alice's local clock and T_B the time as per Bob's local clock.

- Alice sends $(M, \mathcal{T}_K(M \| T_A), T_A)$
- Bob receives (M, tag, T) and accepts iff $\mathcal{V}_K(M \| T, tag) = 1$ and $|T_B - T| \leq \Delta$ 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, tag). If $\mathcal{V}_K(M \| ctr_B, tag) = 1$ then Bob accepts and increments ctr_B.

Counters need to stay synchronized.

Types of message authentication schemes

Special purpose: Used in a specific setting, to authenticate data of some known format or distribution. Comes with a

WARNING! only use under conditions X.

General purpose: Used to authenticate in many different settings, where the data format and distribution are not known in advance.

We want general purpose schemes because

- They can be standardized and broadly used.
- Once a scheme is out there, it gets used for everything anyway.
- General purpose schemes are easier to use and less subject to mis-use: it is hard for application designers to know whether condition X is met.

Why UF-CMA?

A possible critique of existential forgery:

- In practice we usually care only that A cannot forge tags for "important" or"meaningful" messages.
- Yet the UF-CMA definition declare A successful even if it forges the tag of a "garbage" message

Why UF-CMA?

A possible critique of existential forgery:

- In practice we usually care only that A cannot forge tags for "important" or"meaningful" messages.
- Yet the UF-CMA definition declare A successful even if it forges the tag of a "garbage" message

Response: We want general purpose schemes!

- We cannot anticipate application contexts and it is dangerous to let security depend on assumptions about message semantics.
- In fact, "random" messages are possible, for example
    - Keys
    - Executable files
    - Scientific data being read by sensors

Why UF-CMA?

Possible critique of CMAs: They cannot be mounted in practice.

Why UF-CMA?

Possible critique of CMAs: They cannot be mounted in practice.

Response:

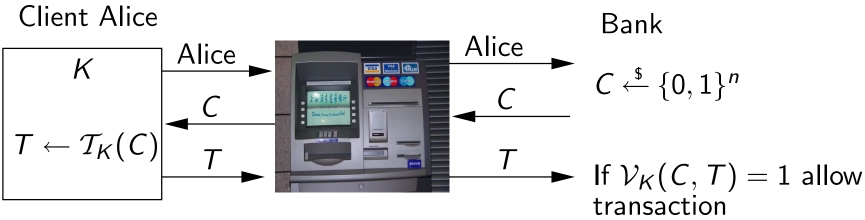
- Actually, they sometime can
- Security against CMA is important for security of some protocols using MA
- Better safe than sorry

CMAs in “real life”

- Message forwarding: Charlie sends M to Alice who authenticates it under a key K she shares with Bob, sending (M, τ) to the latter
- Notary public: Will authenticate any given data

CMAs in Protocols: Example

Alice's smartcard contains a key K also held by Bank.



Client Alice

K

$T \leftarrow \mathcal{T}_K(C)$

Alice

C

T

Bank

Alice

C

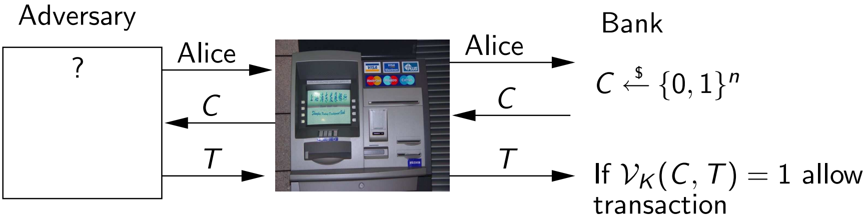
T

$C \overset{\$}{\leftarrow} \{0,1\}^n$

If $\mathcal{V}_K(C,T) = 1$ allow transaction

CMAs in Protocols: Example

Adversary card attemps to get Bank to accept under Alice's name.



Adversary

?

Alice

C

T

Alice

C

T

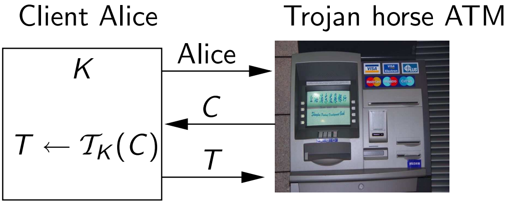
Bank

$C \xleftarrow{\$} \{0,1\}^n$

If $\mathcal{V}_K(C, T) = 1$ allow transaction

CMAs in Protocols: Example

Trojan horse ATM can mount a CMA to try to find key K.



Client Alice

Trojan horse ATM

K

$T \leftarrow \mathcal{T}_K(C)$

Alice

C

T

Strong unforgeability

$\mathrm{UF\text{-}CMA}$ asks that adversary be unable to forge a tag for a "new" message. $\mathrm{SUF\text{-}CMA}$ asks that adversary be unable to

- forge a tag for a "new" message
- forge a new tag even for an "old" message

"New message": A message not authenticated by sender

"Old message": A message authenticated by sender

"New tag": Not a tag computed/sent by sender for this message

Definition: SUF-CMA

Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme and A an adversary,

Game $\mathrm{SUFCMA}_{\mathcal{MA}}$

```
procedure Initialize
$K \xleftarrow{\$} \mathcal{K}; S \leftarrow \emptyset$

procedure Verify($M, T$)
$d \leftarrow \mathcal{V}_K(M, T)$
If $(d = 1 \wedge (M, T) \notin S)$ then $\mathrm{win} \leftarrow$ true
return $d$

procedure Tag($M$)
$T \xleftarrow{\$} \mathcal{T}_K(M)$
$S \leftarrow S \cup \{(M, T)\}$
return $T$

procedure Finalize
return win
```

The suf-cma advantage of adversary A is

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

$\mathrm{SUF\text{-}CMA} \Rightarrow \mathrm{UF\text{-}CMA}$

Any MA scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ that is $\mathrm{SUF\text{-}CMA}$ scheme is also $\mathrm{UF\text{-}CMA}$ scheme.

Why? Suppose A's **Tag** queries are $M_1, \ldots, M_q$, resulting in tags

$$T_1 \xleftarrow{\$} \mathcal{T}_K(M_1), \ldots, T_q \xleftarrow{\$} \mathcal{T}_K(M_q)$$

Now suppose A queries **Verify**(M, T). Then

$$M \notin \{M_1, \ldots, M_q\} \Rightarrow (M, T) \notin \{(M_1, T_1), \ldots, (M_q, T_q)\}$$

So if A wins in game $\mathrm{UFCMA}_{\mathcal{MA}}$ it also wins in game $\mathrm{SUFCMA}_{\mathcal{MA}}$.

Theorem: For any A,

$$\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{MA}}(A) \leq \mathbf{Adv}^{\mathrm{suf\text{-}cma}}_{\mathcal{MA}}(A)$$

Any PRF is a MAC

Let $F : \{0,1\}^k \times D \to \{0,1\}^n$ be a family of functions.

Proposition: If F is a secure PRF then $\mathcal{MA}[F]$ is a secure (UF-CMA and SUF-CMA) MAC.

Intuition for why PRFs are good MACs

- Random functions make good MACs
- PRFs are pretty much as good as random functions

Random functions are good MACs

Suppose **Fn** : $D \rightarrow \{0,1\}^n$ is random and consider A who

- Can query **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,

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

Random functions are good MACs

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 $\mathbf{Fn}(x)$.

PRFs are nearly as good MACs as random functions

Suppose $F : \{0,1\}^k \times D \to \{0,1\}^n$ and let $K \overset{\$}{\leftarrow} \{0,1\}^k$. 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.

PRFs are good MACs

Theorem [GGM86,BKR96]: Let $F : \{0,1\}^k \times D \to \{0,1\}^n$ be a family of functions and let $\mathcal{MA}[F] = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be the associated MAC. Let A be a uf-cma adversary making q_s **Tag** queries and $q_v \leq 2^n/2$ **Verify** queries, and having running time t. Then there is a prf-adversary B such that

$$\mathbf{Adv}^{\text{suf-cma}}_{\mathcal{MA}[F]}(A) \leq \mathbf{Adv}^{\text{prf}}_{F}(B) + \frac{2q_v}{2^n},$$

and B makes $q_s + q_v$ **Fn** queries and has running time t plus some overhead.

Games for proof

Game G_0

```
procedure Initialize
$K \xleftarrow{\$} \{0,1\}^k; S \leftarrow \emptyset$

procedure Tag($M$)
if $T[M] = \perp$ then $T[M] \leftarrow F_K(M)$
$S \leftarrow S \cup \{M\}$; return $T[M]$

procedure Verify($M, T'$)
if $T[M] = \perp$ then $T[M] \leftarrow F_K(M)$
if $T' = T[M]$ then $d \leftarrow 1$ else $d \leftarrow 0$
if $(d = 1 \wedge M \notin S)$ then win $\leftarrow$ true
return $d$

procedure Finalize
return win
```

Game G_1

```
procedure Initialize
$S \leftarrow \emptyset$

procedure Tag($M$)
if $T[M] = \perp$ then $T[M] \xleftarrow{\$} \{0,1\}^n$
$S \leftarrow S \cup \{M\}$; return $T[M]$

procedure Verify($M, T'$)
if $T[M] = \perp$ then $T[M] \xleftarrow{\$} \{0,1\}^n$
if $T' = T[M]$ then $d \leftarrow 1$ else $d \leftarrow 0$
if $(d = 1 \wedge M \notin S)$ then win $\leftarrow$ true
return $d$

procedure Finalize
return win
```

Adversary B

```
adversary $B$
$S \leftarrow \emptyset$
Run $A^{\mathrm{TagSim}(\cdot),\mathrm{VerifySim}(\cdot,\cdot)}$
if win then return 1
else return 0
```

```
subroutine $\mathrm{TagSim}(M)$
if $T[M] = \perp$ then $T[M] \leftarrow \mathbf{Fn}(M)$
$S \leftarrow S \cup \{M\}$; return $T[M]$

subroutine $\mathrm{VerifySim}(M, T')$
if $T[M] = \perp$ then $T[M] \leftarrow \mathbf{Fn}(M)$
if $T' = T[M]$ then $d \leftarrow 1$ else $d \leftarrow 0$
if $(d = 1 \wedge M \notin S)$ then win $\leftarrow$ true
return $d$
```

If **Fn** $= F_K$ then B is providing A the environment of game G_0 so

$$\Pr[\mathrm{Real}_F^B \Rightarrow 1] = \Pr[G_0^A \Rightarrow \text{true}]$$

If **Fn** is random then B is providing A the environment of game G_1 so

$$\Pr[\mathrm{Rand}_F^B \Rightarrow 1] = \Pr[G_1^A \Rightarrow \text{true}]$$

Analysis

$$\begin{aligned}\mathbf{Adv}_F^{\mathrm{prf}}(B) &= \Pr\left[\mathrm{Real}_F^B \Rightarrow 1\right] - \Pr\left[\mathrm{Rand}_F^B \Rightarrow 1\right] \\ &= \Pr[G_0^A \Rightarrow \mathsf{true}] - \Pr[G_1^A \Rightarrow \mathsf{true}]\end{aligned}$$

Claim 1:

$$\Pr[G_0^A \Rightarrow \mathsf{true}] = \mathbf{Adv}_{\mathcal{MA}[F]}^{\mathrm{suf\text{-}cma}}(A)$$

Claim 2:

$$\Pr[G_1^A \Rightarrow \mathsf{true}] \leq \frac{2q_v}{2^n}$$

Proof of Claim 1

Game G_0

```
procedure Initialize
$K \xleftarrow{\$} \{0,1\}^k$; $S \leftarrow \emptyset$

procedure Tag($M$)
if $T[M] = \perp$ then $T[M] \leftarrow F_K(M)$
$S \leftarrow S \cup \{M\}$; return $T[M]$

procedure Verify($M, T'$)
if $T[M] = \perp$ then $T[M] \leftarrow F_K(M)$
if $T' = T[M]$ then $d \leftarrow 1$ else $d \leftarrow 0$
if $(d = 1 \wedge M \notin S)$ then win ← true
return $d$

procedure Finalize
return win
```

Game $\mathrm{SUFCMA}_{\mathcal{MA}[F]}$

```
procedure Initialize
$K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$

procedure Tag($M$)
$T \leftarrow F_K(M)$
$S \leftarrow S \cup \{M\}$; return $T$

procedure Verify($M, T'$)
if $(T' = F_K(M) \wedge M \notin S)$ then
    win ← true
return $d$

procedure Finalize
return win
```

Claim 1: $\Pr[G_0^A \Rightarrow \text{true}] = \mathbf{Adv}^{\text{suf-cma}}_{\mathcal{MA}[F]}(()A)$

Proof: The above games are equivalent.

Proof of Claim 2

Game G_1

```
procedure Initialize
$S \leftarrow \emptyset$

procedure Tag($M$)
if $T[M] = \perp$ then
    $T[M] \xleftarrow{\$} \{0,1\}^n$
$S \leftarrow S \cup \{M\}$; return $T[M]$
```

```
procedure Verify($M, T'$)
if $T[M] = \perp$ then $T[M] \xleftarrow{\$} \{0,1\}^n$
if $T' = T[M]$ then $d \leftarrow 1$ else $d \leftarrow 0$
if $(d = 1 \wedge M \notin S)$ then win $\leftarrow$ true
return $d$

procedure Finalize
return win
```

Claim 2: $\Pr\left[G_1^A \Rightarrow \text{true}\right] \leq 2q_v/2^n$

Proof: For a call **Verify**(M, T') to set win it must be that $T' = T[M]$ and $M \notin S$. Assuming the latter,

$$\Pr\left[T' = T[M]\right] = ?$$

Proof of Claim 2

```
procedure Verify($M, T'$)
if $T[M] = \perp$ then $T[M] \xleftarrow{\$} \{0,1\}^n$
if $T' = T[M]$ then $d \leftarrow 1$ else $d \leftarrow 0$
if $(d = 1 \wedge M \notin S)$ then win $\leftarrow$ true
return $d$
```

The probability that $T' = T[M]$ with $M \notin S$ is 2^{-n} for the first verify call, but what about later? Best strategy for A is to pick some $M \notin S$ and then query

$$\mathbf{Verify}(M, T_1), \mathbf{Verify}(M, T_2), \ldots$$

where $T_1, T_2, \ldots$ are distinct. The probability that the i-th call sets win is

$$\frac{1}{2^n - (i-1)}$$

Proof of Claim 2

Regardless of A's strategy, the probability that the i-th **Verify**(M, T') call with $M \notin S$ sets win is at most

$$\frac{1}{2^n - (i-1)}$$

$$\Pr[G_1^A \Rightarrow \text{true}] \leq \sum_{i=1}^{q_v} \frac{1}{2^n - (i-1)} \leq \sum_{i=1}^{q_v} \frac{1}{2^n - (q_v - 1)} \leq \frac{q_v}{2^n - q_v}$$

But $q_v \leq 2^n/2$ means $2^n - q_v \geq 2^n/2$, so

$$\Pr[G_1^A \Rightarrow \text{true}] \leq \frac{2q_v}{2^n}$$

PRFs are good MACs

Theorem [GGM86,BKR96]: Let $F : \{0,1\}^k \times D \to \{0,1\}^n$ be a family of functions and let $\mathcal{MA}[F] = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be the associated MAC. Let A be a uf-cma adversary making q_s **Tag** queries and $q_v \leq 2^n/2$ **Verify** queries, and having running time t. Then there is a prf-adversary B such that

$$\mathbf{Adv}^{\text{suf-cma}}_{\mathcal{MA}[F]}(A) \leq \mathbf{Adv}^{\text{prf}}_{F}(B) + \frac{2q_v}{2^n},$$

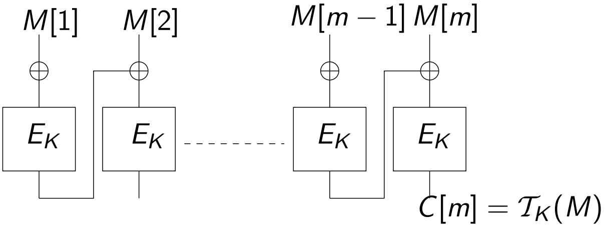
and B makes $q_s + q_v$ **Fn** queries and has running time t plus some overhead.

Basic CBC MAC

Let $E : \{0,1\}^k \times B \to B$ be a block cipher, 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 $\mathcal{MA}[\mathcal{T}]$ defines $\mathcal{T} : \{0,1\}^k \times B^* \to B$ 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 $C[m]$
```

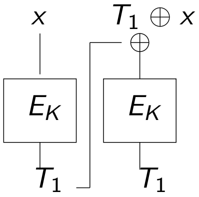


Splicing attack on basic CBC MAC

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

Then,



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

Preventing the splicing attack

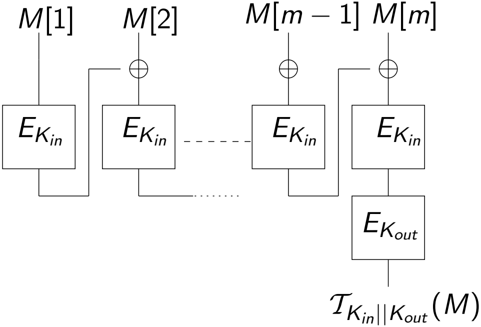
If all authenticated messages have the same number m of blocks then the splicing attack does not apply, so in such a setting we could continue to consider the basic CBC MAC.

But in many uses, we need to authenticate messages of varying lengths. One popular solution has been the ECBC (encrypted CBC) 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 $\mathcal{MA}[\mathcal{T}]$ is obtained by defining $\mathcal{T} : \{0,1\}^{2k} \times B^* \to B$ by

```
**Alg** $\mathcal{T}_{K_{in}||K_{out}}(M)$
$C[0] \leftarrow 0^n$
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 $T$
```



MAC security

The splicing attack fails against the m-restricted basic CBC MAC and the ECBC MAC.

But are there other attacks? Or are these MACs secure?

What's the best attack, and can we prove it is so?

Birthday attacks on MACs

There is a large class of MACs, including

- The *m*-restricted basic CBC MAC
- ECBC MAC, CMAC, HMAC, ...

which are subject to a birthday attack that succeeds in forgery with about $q \approx 2^{n/2}$ **Tag** queries and a few verification 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 block cipher or compression function.

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
Basic DES-CBC-MAC	64	1024	2^{22}
DES-ECBC-MAC	64	1024	2^{22}
Basic AES-CBC-MAC	128	1024	2^{54}
AES-ECBC-MAC	128	1024	2^{54}
Basic AES-CBC-MAC	128	10^6	2^{44}
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 $n = 128$.

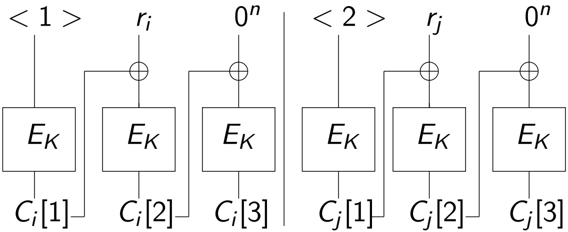
The birthday attack

We now illustrate how the birthday attack works in a simple case, namely the 3-restricted basic CBC MAC.

Here all messages in the adversary's queries, both to the **Tag** oracle and to the **Verify** oracle, must be exactly 3 blocks long.

Internal collisions

Let $M_i = \langle 1 \rangle || r_i || 0^n$ and $M_j = \langle 2 \rangle || r_j || 0^n$.

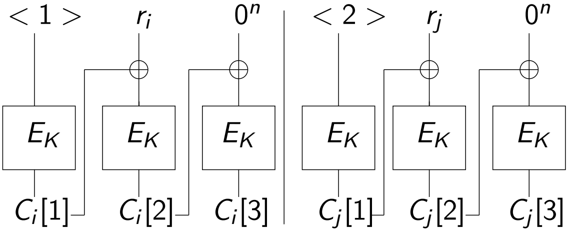


Internal Collision: $C_i[2] = C_j[2]$
Internal collisions can be detected by examining the MAC output, because

$$C_i[2] = C_j[2] \iff C_i[3] = C_j[3]$$

Exploiting internal collisions to forge

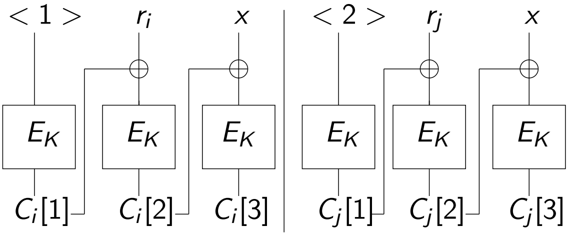
Suppose adversary A has the tags $C_i[3] = C_j[3]$ of messages $\langle 1 \rangle \parallel r_i \parallel 0^n$, $\langle 2 \rangle \parallel r_j \parallel 0^n$ that have an internal collision, namely $C_i[2] = C_j[2]$.



Then if 0^n is changed to some other value x, the tags will continue to be the same.

Exploiting internal collisions to forge

Suppose adversary A has the tags $C_i[3] = C_j[3]$ of messages $\langle 1 \rangle \parallel r_i \parallel 0^n$, $\langle 2 \rangle \parallel r_j \parallel 0^n$ that have an internal collision, namely $C_i[2] = C_j[2]$.



Then for any x we must have $C_i'[3] = C_j'[3]$ meaning $C_i'[3]$ is the correct tag for both messages $\langle 1 \rangle \parallel r_i \parallel x$ and $\langle 2 \rangle \parallel r_j \parallel x$. Thus A can forge by picking some $x \neq 0^n$ and

- Requesting tag of $\langle 1 \rangle \parallel r_i \parallel x$ to get $C_i'[3]$
- Calling **Verify** on $\langle 2 \rangle \parallel r_j \parallel x$ and $C_i'[3]$

Finding internal collisions

Query q 3-block messages

$$\langle 1\rangle||r_1||0^n, \quad \langle 2\rangle||r_2||0^n, \quad \ldots, \quad \langle q\rangle||r_q||0^n,$$

to get back tags

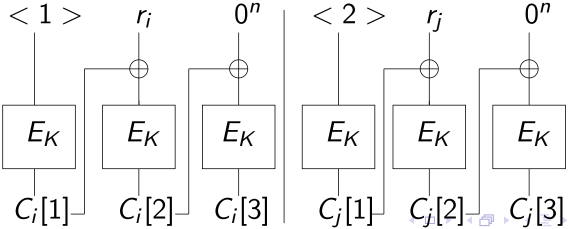
$$C_1[3], C_2[3], \ldots, C_q[3]$$

Hope to find i, j with $1 \leq i < j \leq q$ and

$$C_i[3] = C_j[3].$$

It follows that

$$C_i[2] = C_j[2].$$



Birthday attack on 3-restricted basic CBC MAC

```
adversary $A$
for $i = 1, \ldots, q$ do
   $r_i \xleftarrow{\$} \{0,1\}^n$; $C_i[3] \leftarrow \textbf{Tag}(\langle i \rangle || r_i || 0^n)$
$S \leftarrow \{(i,j) : 1 \leq i < j \leq q \text{ and } C_i[3] = C_j[3]\}$
if $S \neq \emptyset$ then
   $(i,j) \xleftarrow{\$} S$
   $C_i'[3] \leftarrow \textbf{Tag}(\langle i \rangle || r_i || 1^n)$
   $d \leftarrow \textbf{Verify}(\langle j \rangle || r_j || 1^n, C_i'[3])$
```

Previous discussion shows that if $S \neq \emptyset$ then A succeeds, so

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}[\mathcal{T}]}(A) = \Pr[S \neq \emptyset].$$

A birthday analysis can be used to show that

$$\Pr[S \neq \emptyset] = C(2^n, q) \geq 0.3 \frac{q(q-1)}{2^n}$$

Truncation

The effectiveness of the birthday attack can be reduced by truncating the MAC output to $t \leq n$ bits.

For example for $n = 128$ one might use $t = 80$.

The reason it helps is that internal collisions can no longer be unambiguiously identified. (A MAC output collision does not necessarily mean there was an internal collision.)

To be effective, truncation must be combined with "throttling," which restricts the attack to a small number of verification queries.

Truncation is an option with many standardized MACs.

A rigorous and tight quantitative analysis of the security of truncation is lacking.

Security of basic CBC MAC

Question: Are there better-than-birthday attacks when authenticating same-length messages?

Answer: NO
And we can prove the answer is correct.

Basic CBC MAC is a PRF (and hence a SUF-CMA MAC) if all messages authenticated have the same length.

Security of basic CBC MAC

Theorem [BKR96]: Let $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a family of functions and $m \geq 1$ an integer. Let $E^m : \{0,1\}^k \times \{0,1\}^{nm} \rightarrow \{0,1\}^n$ be the family of functions defined by

```
**Alg** $E^m_K(M)$
$C[0] \leftarrow 0^n$
for $i = 1, ..., m$ do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$
return $C[m]$
```

Let A be a prf-adversary against E^m that makes q oracle queries and has running time t. Then there is a prf-adversary B against E such that

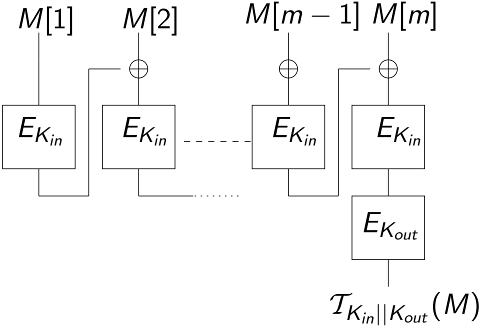
$$\mathbf{Adv}^{\mathrm{prf}}_{E^m}(A) \leq \mathbf{Adv}^{\mathrm{prf}}_{E}(B) + \frac{q^2 m^2}{2^n}$$

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

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 $\mathcal{MA}[\mathcal{T}]$ is obtained by defining $\mathcal{T} : \{0,1\}^{2k} \times B^* \to B$ 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 $T$
```



Security of ECBC

- No splicing attack
- But birthday attack applies

Birthday attack turns out to be best possible: can securely authenticate messages of varying lengths as long as total number of blocks is at most $2^{n/2}$

Security of ECBC

Theorem: Let $E : \{0,1\}^k \times B \to B$ be a block cipher 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 $T$
```

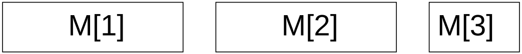
Let A be a prf-adversary against F that makes at most q oracle queries, these totalling at most σ blocks, and has running time t. Then there is a prf-adversary B against E such that

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

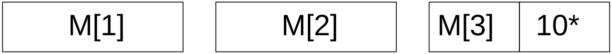
and B makes at most σ oracle queries and has running time about t.

Non-full messages

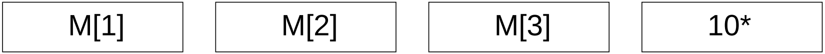
So far we assumed messages have length a multiple of the block-length of the block cipher. Call such messages *full*. How do we deal with non-full messages?


M[1] M[2] M[3]

The obvious approach is padding.


M[1] M[2] M[3] 10*

This works, but if *M* was full, an extra block is needed


M[1] M[2] M[3] 10*

leading to an extra block cipher operation.

Costs

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

Also ECBC uses two block cipher 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 block cipher key

Security:

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

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

CMAC Components and Setup

- $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a block cipher, 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

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

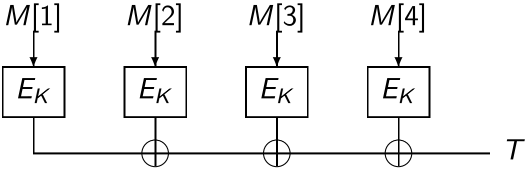
where $x \ll 1$ means x left shifted by 1 bit, so that the msb vanishes and the lsb becomes 0. These bit operations reflect simple finite-field operations.

CMAC Algorithm

```
**Alg** $\mathrm{CMAC}_K(M)$
$M[1]\ldots M[m-1]M[m] \leftarrow M$    // $|M[m]| \leq n$
$\ell \leftarrow |M[m]|$    // $\ell \leq n$
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 $T$
```

Parallelizable MACs?

The following MAC has the nice feature that the block cipher computations can be done in parallel.



But we saw earlier that this is not secure!

Can we fix it?

PMAC [BlRo]

Features:

- Minimal overhead
- A single block cipher key
- Handles variable-length and non-full messages
- Parallelizable

Security:

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

Tweakable Block Ciphers [LRW]

A *tweakable block cipher* is a map

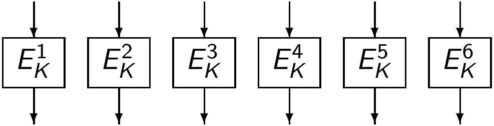
$$E\colon \ \{0,1\}^k \times \mathrm{TwSp} \times \{0,1\}^n \to \{0,1\}^n$$

such that

$$E_K^T\colon \ \{0,1\}^n \to \{0,1\}^n$$

is a permutation for every K, T, where $E_K^T(X) = E(K, T, X)$.

With a single key one thus implicitly has a large number of maps



These appear to be independent random permutations to an adversary who does not know the key K, even if it can choose the tweaks and inputs.

Tweakable Block Cipher Security, Formally

Let E: $\{0,1\}^k \times \mathrm{TwSp} \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a tweakable block cipher

```
Game $\mathrm{Real}_E$
procedure Initialize
$K \xleftarrow{\$} \{0,1\}^k$
procedure Fn$(T, x)$
Return $E_K^T(x)$
```

```
Game $\mathrm{Rand}_{\{0,1\}^n}$
procedure Fn$(T, x)$
$Y \xleftarrow{\$} \{0,1\}^n$
Return $Y$
```

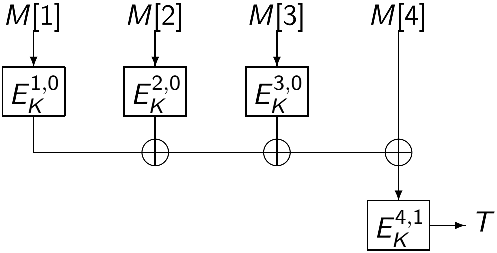
Associated to E, A are the probabilities

$$\Pr\left[\mathrm{Real}_E^A \Rightarrow 1\right] \qquad \Big| \qquad \Pr\left[\mathrm{Rand}_{\{0,1\}^n}^A \Rightarrow 1\right]$$

that A outputs 1 in each world. The advantage of A is

$$\mathbf{Adv}_E^{\mathrm{prf}}(A) = \Pr\left[\mathrm{Real}_E^A \Rightarrow 1\right] - \Pr\left[\mathrm{Rand}_{\{0,1\}^n}^A \Rightarrow 1\right]$$

PMAC Algorithm

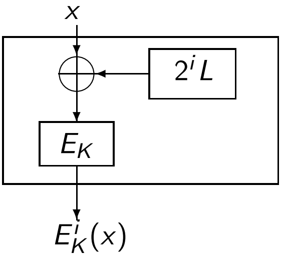


Illustrated for a full message of 4 blocks.

Building a Tweakable Block Cipher

We want to tweak block ciper $E : \{0,1\}^k \times \mathrm{TwSp} \times \{0,1\}^n \to \{0,1\}^n$ with $\mathrm{TwSp} = \{1, \ldots, 2^{64}\}$.

$L \leftarrow E_K(0)$
$E_K^i(x) = \mathsf{AES}_K(x \oplus 2^i L)$



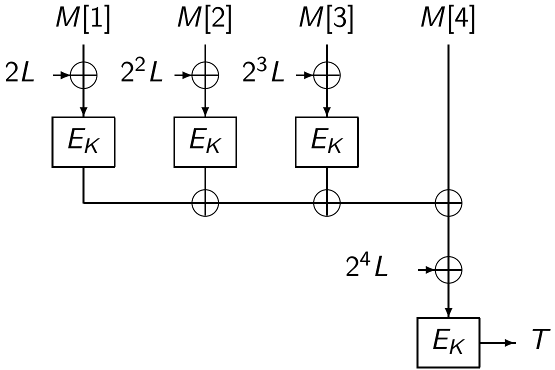
$L \to 2L \to 4L \to \cdots$

$$2\Delta = \begin{cases} (\Delta \ll 1) & \text{if } \mathrm{msb}(\Delta) = 0 \\ (\Delta \ll 1) \oplus 87_{16} & \text{otherwise} \end{cases}$$

Doubling is cheap: 0.3–0.8 cpb

Intuition: Hard for adversary to find distinct $(x_1, i_1), (x_2, i_2)$ such that $x_1 \oplus 2^{i_1} L = x_2 \oplus 2^{i_2} L$

PMAC Instantiated



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 hash functions are keyless.

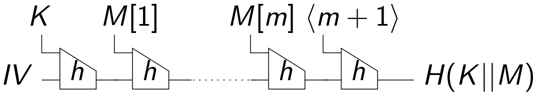
Question: How do we key hash functions to get MACs?

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

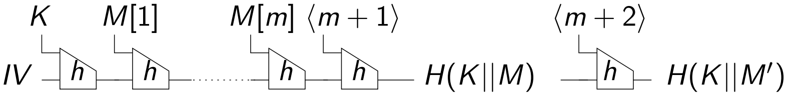
$$\mathcal{T}_K(M) = H(K||M)$$

Is this 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'.

HMAC [BCK96]

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

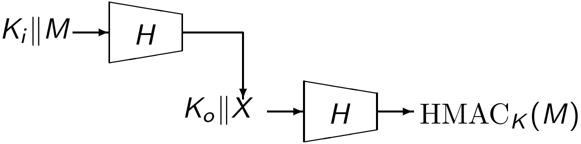
$$K_o = \mathsf{opad} \oplus K||0^{352} \text{ and } K_i = \mathsf{ipad} \oplus K||0^{352}$$

where

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

in HEX. Then

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



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

Paradigms for MACing

- Block cipher based: CBC-MAC, ECBC-MAC, CMAC, PMAC, XCBC, OMAC, XOR-MAC, RMAC, . . .
- Hash function based: HMAC
- Carter-Wegman (CW) MACs: UMAC, Poly127-AES, Poly1305-AES, . . .

CW MACs can be very fast.

AU Families

A family of functions $H : \text{Keys}(H) \times D \to \{0,1\}^l$ is ϵ-AU if for all distinct $M_1, M_2, \in D$ we have

$$\Pr\left[H_K(M_1) = H_K(M_2)\right] \leq \epsilon$$

where the probability is over $K \xleftarrow{\$} \text{Keys}(H)$.

This is a weak form of collision resistance in which the attacker must select its collision M_1, M_2 without seeing the key K.

One can design fast, non-cryptographic ϵ-AU-families: NH [BHKKR], Poly127 [Ber], Poly1305[Ber], . . .

NH [BHKKR]

$w = 16,\ 32,\ \text{or } 64$ // word size
$M = M[1] \cdots M[m]$ // $M[i] \in \{0, \ldots, 2^w - 1\}$
$K = K[1] \cdots K[m]$ // $K[i] \in \{0, \ldots, 2^w - 1\}$

```
Alg $NH_K(M)$
for $i = 1, \ldots, m/2$ do
    $a[i] \leftarrow (M[2i-1] + K[2i-1]) \bmod 2^w$
    $b[i] \leftarrow (M[2i] + K[2i]) \bmod 2^w$
$S \leftarrow (a[1]b[1] + \cdots + a[m/2]b[m/2]) \bmod 2^{2w}$
return $S$
```

This is ϵ-AU for $\epsilon = 2^{-w}$

Care or assembly code required to get $2w$-bit product of w-bit operands.

From AU to MAC

H : Keys(H) $\times D \rightarrow \{0,1\}^l$ an ϵ-AU family
F : Keys(F) $\times \{0,1\}^l \rightarrow \{0,1\}^n$ a PRF (e.g. AES)
N : nonce, different for each message

```
**Alg** $\mathrm{MAC}(K_1 K_2, N, M)$
$\mathrm{return}\ (N, F(K_1, N) \oplus H(K_2, M))$
```

This is a UF-CMA-secure (nonce-based) MAC, assuming F is a PRF and H is AU.

NH + HMAC-SHA1 → UMAC
Poly127 + AES → Poly127-AES
Poly1305 + AES → Poly1305-AES

Performance

Table shows Pentium-4 machine-cycles per byte for processing various byte-length messages. UMAC here has a 96-bit tag while Poly127-AES has a 128-bit tag.

	44	**64**	**256**	**552**	**1024**	**1500**
UMAC	22	15	4.5	2.7	1.9	2.2
Poly127-AES	23	17	7.5	5.8	5.1	4.8
SHA1		76	34.5		23.6	

This data is from the UMAC webpage. SHA1 speeds via OpenSSL.