CSE107: Intro to Modern Cryptography

https://cseweb.ucsd.edu/classes/sp22/cse107-a/

Emmanuel Thomé

Apr 19, 2022

Lecture 6b

Hash functions (we lagged behind a little bit)

A new set of hash functions

Plan

A new set of hash functions

SHA3

National Institute for Standards and Technology (NIST) held a world-wide competition to develop a new hash function standard.

Contest webpage:
`http://csrc.nist.gov/groups/ST/hash/index.html`

Requested parameters:

- Design: Family of functions with 224, 256, 384, 512 bit output sizes
- Security: CR, one-wayness, near-collision resistance, others...
- Efficiency: as fast or faster than SHA2-256

SHA3

Submissions: 64

Round 1: 51

Round 2: 14: BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grostl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, Skein.

Finalists: 5: BLAKE, Grostl, JH, Keccak, Skein.

SHA3: 1: Keccak

SHA-3 is in the standards

FIPS PUB 202

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions

CATEGORY: COMPUTER SECURITY SUBCATEGORY: CRYPTOGRAPHY

SHA-3 can be used in software

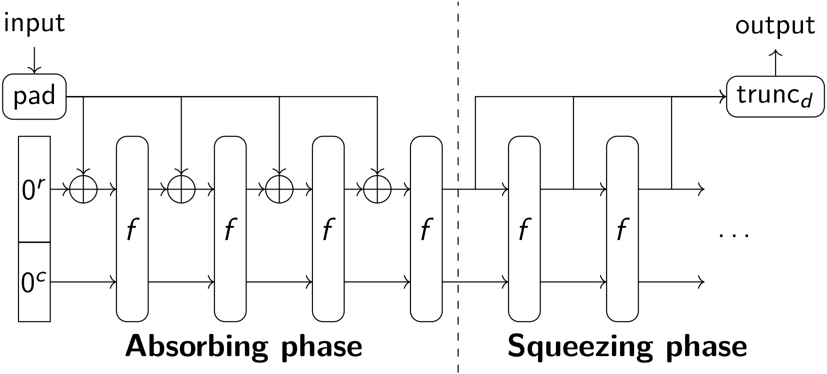
```
Python 3.9.12 (main, Mar 24 2022, 13:02:21)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import hashlib
In [2]: hashlib.sha3_256
Out[2]: <function _hashlib.openssl_sha3_256(string=b'', *, usedforsecurity=
In [3]: hashlib.sha3_256(b"Hello, world").hexdigest()
Out[3]: '3550aba97492de38af3066f0157fc532db6791b37d53262ce7688dcc5d461856'
```

- If you have up-to-date software, you almost certainly have access to SHA-3.
- But as of 2022, it's not quite ubiquitous yet.

SHA3/Keccak: The Sponge construction

SHA3 does not use the MD paradigm used by the MD and SHA2 series.



- c = capacity; r = rate; $b = r + c$ = width; d = digest length;
- $f\colon \{0,1\}^{r+c} \to \{0,1\}^{r+c}$ is a (public, invertible!) permutation.
- d is the number of output bits, and $c = 2d$.

SHA3/Keccak: more than just hash functions

The SHA-3 family consists of four cryptographic hash functions, called SHA3-224, SHA3-256, SHA3-384, and SHA3-512, and two extendable-output functions (XOFs), called SHAKE128 and SHAKE256.

SHAKE $- d$: returns any desired number of bits d.

Keccak operation on an input M, aiming for d-bit output

- Width of the permutation f is $b = 1600$. Capacity is $c = 2d$.
- Domain-separation padding:
  - If computing SHA3-d, pad M to $M^* = M \parallel 01$.
  - If computing SHAKE-d, pad M to $M^* = M \parallel 1111$.
- Pad to a multiple of the rate r: $j \leftarrow (-|M^*| - 2) \bmod r$
  $M^\dagger \leftarrow M^* \parallel 1 \parallel 0^j \parallel 1.$
- Run the sponge scheme and output d bits (r at a time).

SHA-3 security claims

From FIPS-202:

Function	Output Size	Security Strengths in Bits		
		Collision	Preimage	2nd Preimage
SHA-1	160	< 80	160	$160 - L(M)$
SHA-224	224	112	224	$\min(224, 256 - L(M))$
SHA-512/224	224	112	224	224
SHA-256	256	128	256	$256 - L(M)$
SHA-512/256	256	128	256	256
SHA-384	384	192	384	384
SHA-512	512	256	512	$512 - L(M)$
SHA3-224	224	112	224	224
SHA3-256	256	128	256	256
SHA3-384	384	192	384	384
SHA3-512	512	256	512	512
SHAKE128	d	$\min(d/2, 128)$	$\geq \min(d, 128)$	$\min(d, 128)$
SHAKE256	d	$\min(d/2, 256)$	$\geq \min(d, 256)$	$\min(d, 256)$

Table 4: Security strengths of the SHA-1, SHA-2, and SHA-3 functions

CSE107: Intro to Modern Cryptography

https://cseweb.ucsd.edu/classes/sp22/cse107-a/

Emmanuel Thomé

Apr 19, 2022

Lecture 7a

Message Authentication Codes

Do we need MACs?

PRFs and MACs

MACs from block ciphers

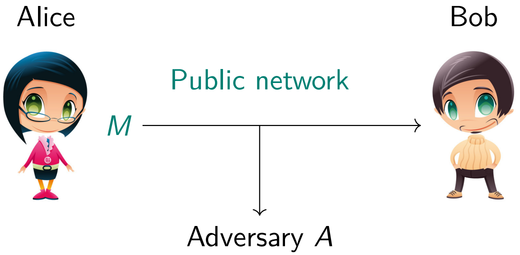
Plan

Do we need MACs?

PRFs and MACs

MACs from block ciphers

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

Bob
(bank)

Alice

Alice:
Pay \$100 to Charlie

Adversary A (Eve)

Adversary Eve might

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

Integrity prevents such attacks.

Does encryption provide integrity?

Suppose that Alice and her Bank share a secret key K.

- Alice sends messages such as

  `PAY TO ACCOUNT 012345`
  `000010.00`

  `5041590a544f204143434f554e542030`
  `31323334350a3030303031302e30300a`

- Alice encrypts with AES128-CTR\$ and sends to Bank.

  `393be75f153bf3b65ce9e7531a90db9b`
  `46e736e087e736677e67bd71065bdbb6`
  `d55f7ee1a62d789ab76b54171a74a96c`

- Bank decrypts and proceeds with the transfer request.

Eve does not know the key, but can nevertheless do:

```
C[2]=xor_strings(C[2],'\x00'*6 + '\x09\x09' + '\x00'*8)
```

Message then decrypts to:

`PAY TO ACCOUNT 012345`
`990010.00`

Encryption alone does **NOT** provide integrity, especially so when any ciphertext can decrypt to something.

Is a hash function a good cryptographic integrity check?

Alice

Bob

Public network

$M, T \longrightarrow M', T'$

Adversary A

$T = H(M)$

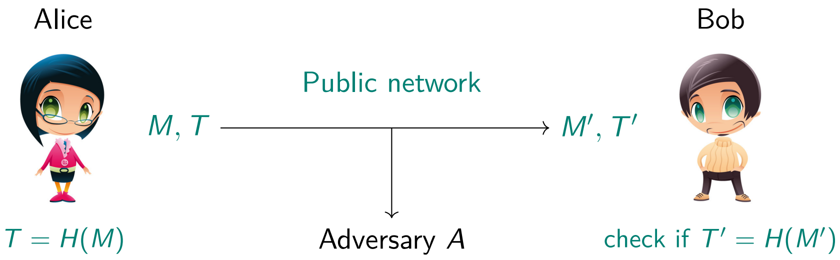
check if $T' = H(M')$

Proposal:

- Alice sends $(M, T = H(M))$ using a collision-resistant hash function like SHA3-256.
- Bob receives (M', T') and checks that $T' = H(M')$.

Assume the adversary A can read and modify messages in transit.
Does this ensure the integrity of M?

Is a hash function a good cryptographic integrity check?



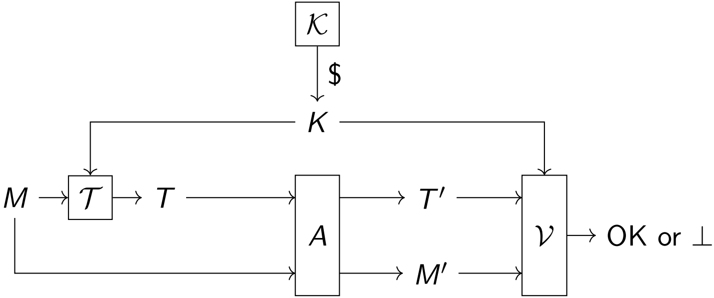
Proposal:

- Alice sends $(M, T = H(M))$ using a collision-resistant hash function like SHA3-256.
- Bob receives (M', T') and checks that $T' = H(M')$.

Assume the adversary A can read and modify messages in transit.
Does this ensure the integrity of M? **No.**
Keyless integrity checks cannot work!

Message Authentication Codes (MAC) ; not

A Message Authentication Code (MAC) $\mathcal{T}$: Keys $\times D \rightarrow R$ 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. We have defined

Alg $\mathcal{V}_K(M', T')$
If $\mathcal{T}_K(M') = T'$ then return 1 (OK, VALID) else return 0 ($\perp$, INVALID)

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 $\mathcal{V}_K(M', T') = 1$, or, equivalently, iff $\mathcal{T}_K(M') = T'$.

The security notion that we want for MACs is called UF-CMA.

Vocabulary: UF-CMA

UF-CMA = Unforgeability against chosen-message attacks

UF-CMA

Let $\mathcal{T}$: Keys $\times D \to R$ be a message authentication code. Let A be an adversary.

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

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

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 $M \notin D$ then return false
Return $(T = \mathcal{T}_K(M))$

Definition: uf-cma advantage

The uf-cma advantage of adversary A is

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

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 $M \in D$ 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.

UF-CMA properties

If $\mathbf{Adv}_{\mathcal{T}}^{\text{uf-cma}}(A)$ is small for any adversary A, it implies that:

- it is hard to do selective forgery: forge a tag on a specific message that the adversary chooses;
- it is hard to find the key K that $\mathcal{T}$ uses;
- the tag must be long enough! If the tag is only 16 bits, then it is easy to find an adversary with advantage 2^{-16}.

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. This can be done using timestamps or synchronized counters.

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 $(M, \mathcal{T}_K(M \parallel Time_A), Time_A)$
- Bob receives $(M, T, Time)$ and accepts iff $\mathcal{T}_K(M \parallel Time) = T$ 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 $(M, \mathcal{T}_K(M \parallel ctr_A))$ and then increments ctr_A
- Bob receives (M, T). If $\mathcal{T}_K(M \parallel ctr_B) = T$ then Bob accepts and increments ctr_B.

Counters need to stay synchronized.

Plan

Do we need MACs?

PRFs and MACs

MACs from block ciphers

Plan

PRFs and MACs

- Any PRF is a MAC
- Is it hard to go from FIL to VIL?

Any PRF is a MAC

Theorem [GGM86,BKR96]: F is PRF-secure $\Rightarrow$ F is UF-CMA-secure

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

$$\mathbf{Adv}_F^{\mathrm{uf\text{-}cma}}(A) \leq \mathbf{Adv}_F^{\mathrm{prf}}(B) + \frac{1}{2^n}\ .$$

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

We do not prove this here, but we give a little 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 than 2^{-n} then we are detecting a difference between F_K and a random function.

PRF domain extension

Definition: Fixed/Variable Input Length

A family of functions F: Keys $\times D \to R$ 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 block ciphers 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.

- Simple examples which don't work.
- More advanced algorithms with block ciphers.

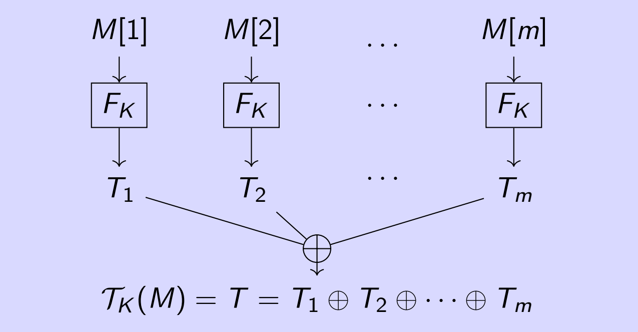
Plan

PRFs and MACs

- Any PRF is a MAC
- Is it hard to go from FIL to VIL?

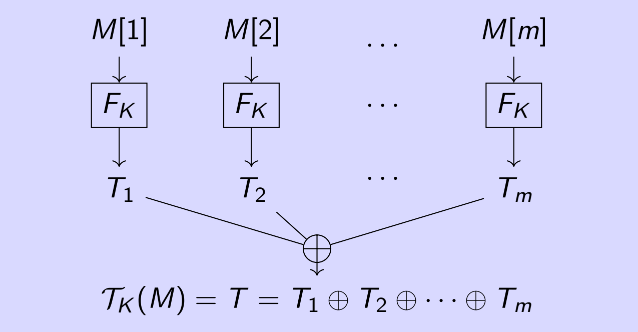
Example: combine per-block outputs

Let F : Keys $\times\ D \to R$ be any FIL PRF. Define $\mathcal{T}$: Keys $\times\ D^* \to R$ as:



Example: combine per-block outputs

Let F : Keys $\times D \to R$ be any FIL PRF. Define $\mathcal{T}$: Keys $\times D^* \to R$ as:

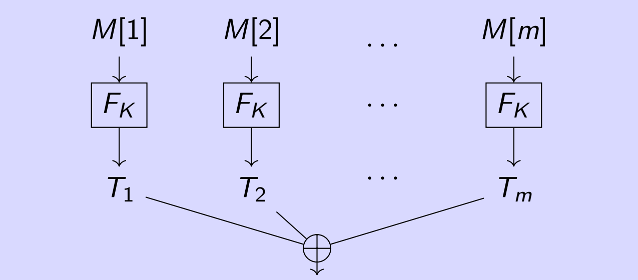


This is awfully bad! Adversary with $\mathbf{Adv}_{\mathcal{T}}^{\mathrm{uf\text{-}cma}}(A) = 1$:

- Query the oracle for the MAC $T = T_1 \oplus T_2$ of a message $M = M[1]M[2]$.
- Swap two blocks: T is a valid MAC of $M' = M[2]M[1]$.

Example: combine per-block outputs

Let F : Keys $\times D \to R$ be any FIL PRF. Define $\mathcal{T}$: Keys $\times D^* \to R$ as:



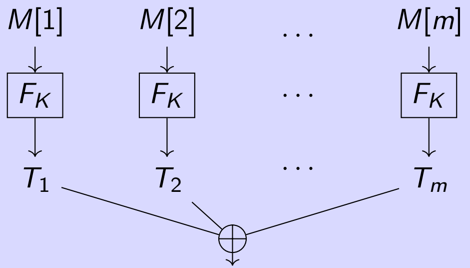
$$\mathcal{T}_K(M) = T = T_1 \oplus T_2 \oplus \cdots \oplus T_m$$

This is awfully bad! Adversary with $\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{T}}(A) = 1$:

Other example: 0^n is a valid MAC for any repeated message $M \parallel M$.

Example: combine per-block outputs

Let F : Keys $\times D \to R$ be any FIL PRF. Define $\mathcal{T}$: Keys $\times D^* \to R$ as:



$$\mathcal{T}_K(M) = T = T_1 \oplus T_2 \oplus \cdots \oplus T_m$$

This is awfully bad! Adversary with $\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{T}}(A) = 1$:

Other example: 0^n is a valid MAC for the empty message ε.

More failing examples

Many variants of the previous example also fail miserably:

- Concatenate all the per-block outputs.
- Many sorts of simple combinations of the per-block outputs.

We need something better.

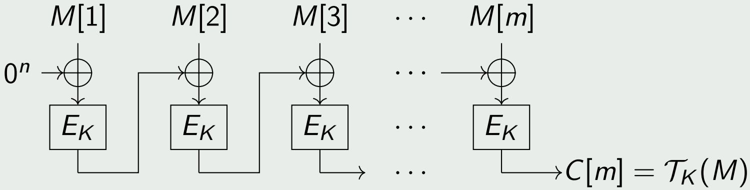
Example: Basic CBC MAC

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

Definition: Basic CBC-MAC

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 $C[m]$

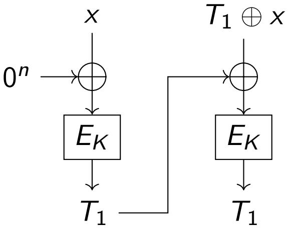


Splicing attack on basic CBC MAC

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

PRF domain extension

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.

We will see solutions that work, including

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

Plan

Do we need MACs?

PRFs and MACs

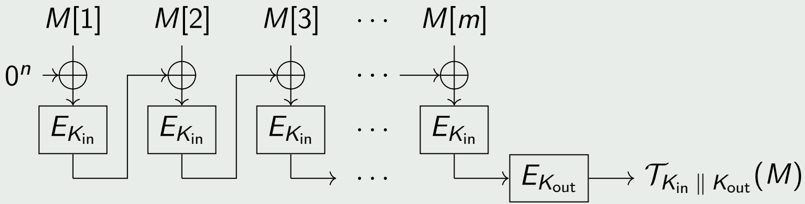
MACs from block ciphers

ECBC MAC

Definition: ECBC-MAC

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

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



Birthday attacks on MACs

There is a large class of MACs, including ECBC MAC, 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 block cipher or compression function.

Security of ECBC

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

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

Theorem: Birthday attack is best possible

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 D against E such that

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

and D 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 block cipher, 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 $n = 128$.