Message integrity: Confidentiality alone not sufficient, also need <u>message integrity</u>. Otherwise adversary can tamper with the message (e.g., "Send \$100 to Bob" $\to$ "Send \$100 to Eve")

In some cases (e.g., software patches), integrity more important than confidentiality

Idea: Append a "tag" (also called a "signature") to the message to prove integrity (property we want is tags should be hard to forge)

this tolerates a <u>single</u> error (better error-correcting codes can do much better)

Observation: The tag should be computed using a keyed-function

↳ Example of keyless integrity check: CRC (cyclic redundancy check) [simple example is to set tag to be the parity]

↳ this was used in SSH v1 (1995) for data integrity! Fixed in SSH v2 (1996)

↳ also used in WEP (802.11b) protocol for integrity — also broken!

Problem: If there is no key, <u>anyone</u> can compute it! Adversary can tamper with message and compute the new tag.

Definition. A message authentication code (MAC) with key-space $\mathcal{K}$, message space $\mathcal{M}$ and tag space $\mathcal{T}$ is a tuple of algorithms $\Pi_{MAC} = (\text{Sign}, \text{Verify})$:

$$\left.\begin{array}{l}\text{Sign}: \mathcal{K} \times \mathcal{M} \to \mathcal{T} \\ \text{Verify}: \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{0,1\}\end{array}\right\} \text{Must be efficiently-computable}$$

Correctness: $\forall k \in \mathcal{K}, \forall m \in \mathcal{M}$:

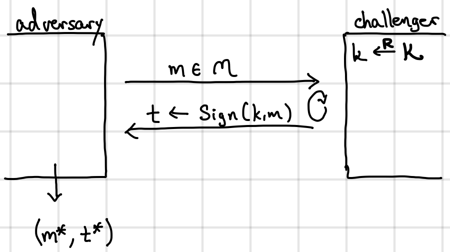
$$\Pr[\text{Verify}(k, m, \text{Sign}(k,m)) = 1] = 1$$

Sign can be a <u>randomized</u> algorithm

Defining security: Intuitively, adversary should not be able to compute a tag on any message without knowledge of the key

↳ Moreover, since adversary might be able to see tags on existing messages (e.g., signed software updates), it should not help towards creating a new MAC

Definition. A MAC $\Pi_{MAC} = (\text{Sign}, \text{Verify})$ satisfies existential unforgeability against $\underbrace{\text{chosen message}}_{\text{adversary gets to choose messages to be signed}}$ attacks (EUF-CMA) if for all efficient adversaries $\mathcal{A}$, $\text{MACAdv}[\mathcal{A}, \Pi_{MAC}] = \Pr[W=1] = \text{negl}(\lambda)$, where W is the output of the following security game:



As usual, λ denotes the length of the MAC secret key (e.g., $\log|\mathcal{K}| = \text{poly}(\lambda)$)

Note: the key can also be sampled by a special KeyGen algorithm (for simplicity, we just define it to be uniformly random)

Let $m_1, \ldots, m_Q$ be the signing queries the adversary submits to the challenger, and let $t_i \leftarrow \text{Sign}(k, m_i)$ be the challenger's responses. Then, $W = 1$ if and only if:

$$\text{Verify}(k, m^*, t^*) = 1 \quad \text{and} \quad (m^*, t^*) \notin \{(m_1, t_1), \ldots, (m_Q, t_Q)\}$$

MAC security notion says that adversary cannot produce a <u>new</u> tag on <u>any</u> message even if it gets to obtain tags on messages of its choosing.

First, we show that we can directly construct a MAC from any PRF.

MACs from PRFs: Let $F: \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ be a PRF. We construct a MAC Π_{MAC} over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ as follows:

$\text{Sign}(k, m)$: output $t \leftarrow F(k, m)$

$\text{Verify}(k, m, t)$: output 1 if $t = F(k, m)$ and 0 otherwise

Theorem. If F is a secure PRF with a sufficiently large range, then Π_{MAC} defined above is a secure MAC. Specifically, for every efficient MAC adversary $\mathcal{A}$, there exists an efficient PRF adversary $\mathcal{B}$ such that

$$\text{MACAdv}[\mathcal{A}, \Pi_{MAC}] \leq \text{PRFAdv}[\mathcal{B}, F] + \frac{1}{|\mathcal{T}|}.$$

Intuition for proof:
1. Output of PRF is computationally indistinguishable from that of a truly random function.
2. If we replace the PRF with a truly random function, adversary wins the MAC game only if it correctly predicts the random function at a new point. Success probability is then exactly $1/|\mathcal{T}|$.

Formalize using a "hybrid argument" [see Boneh-Shoup or ask in OH]

Implication: Any PRF with large output space can be used as a MAC.

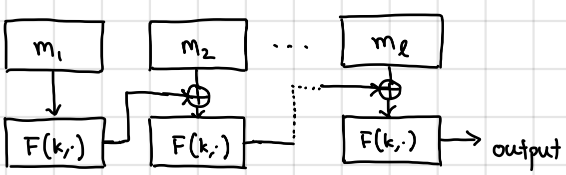
↳ AES has 128-bit output space, so can be used as a MAC

Drawback: Domain of AES is 128-bits, so can only sign 128-bit (16-byte) messages

How do we sign longer messages? We will look at two types of constructions:

1. Constructing a large-domain PRF from a small-domain PRF (i.e., AES)
2. Hash-based constructions

Approach 1: use CBC (without IV)



Not encrypting messages so no need for IV (or intermediate blocks)

↳ Mode often called "raw-CBC"

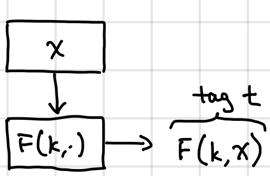
Raw-CBC is a way to build a large-domain PRF from a small-domain one

↳ Can show security for "prefix-free" messages [more precisely, raw-CBC is a prefix-free PRF: pseudorandom as long as PRF never evaluated on two values where one is a prefix of other]

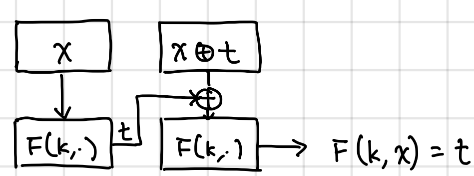
↳ includes fixed-length messages as a special case

But not secure for variable-length messages: "Extension attack"

1. Query for MAC on arbitrary block x:



2. Output forgery on message $(x, x \oplus t)$ and tag t



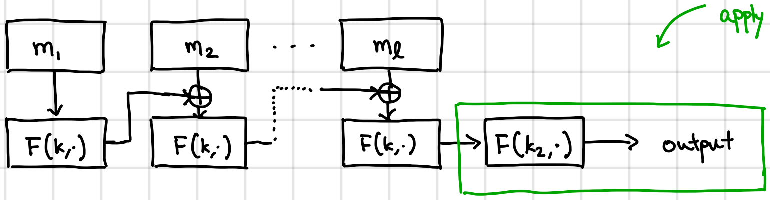
$\Rightarrow t$ is a valid tag on extended message $(x, t \oplus x)$

↳ Adversary succeed with advantage 1

raw CBC can be used to build a MAC on fixed-length messages, but not variable-length messages
(more generally, prefix-free)

For variable-length messages, we use "encrypted CBC" (ECBC):
↳ variant used in ANSI X9.9, ANSI X19.9 standards ($\overbrace{\text{ANSI X9.9, ANSI X19.9}}^{\text{standards for banking / financial services}}$)

apply another PRF with a <u>different</u> key to the output of raw CBC
critical for security (using the same key <u>not</u> secure)



To use encrypted CBC-MAC, we need to assume message length is even multiple of block size (similar to CBC encryption)
↳ to sign messages that are not a multiple of the block size, we need to first <u>pad</u> the message
↳ as was the case with encryption, padding must be injective
  ↳ in the case of encryption, injectivity needed for correctness
  ↳ in the case of integrity, injectivity needed for <u>security</u> [if $\text{pad}(m_0) = \text{pad}(m_1)$, m_0 and m_1 will have the same tag]

Standard approach to pad: append $1000\cdots0$ to fill up block [ANSI X9.9 and ANSI X9.19 standards]
- <u>Note</u>: if message is an even multiple of the block length, need to introduce a dummy block
  ↳ Necessary for any injective function: $|\{0,1\}^{\leq n}| > |\{0,1\}^n|$
- This is a <u>bit-padding</u> scheme [PKCS #7 that we discuss previously in the context of CBC encryption is a <u>byte-padding</u> scheme]

Encrypted CBC-MAC drawbacks: always need at least 2 PRF evaluations (using <u>different</u> keys)
messages must be padded to block size
} especially bad for authenticating short (e.g., single-byte) messages

Better approach: raw CBC-MAC secure for prefix-free messages
↳ Can we apply a "prefix-free" encoding to the message?
- <u>Option 1</u>: <u>Prepend</u> the message length to the message
  - equal-length messages cannot have one be prefix of other
  - different-length messages differ in first block

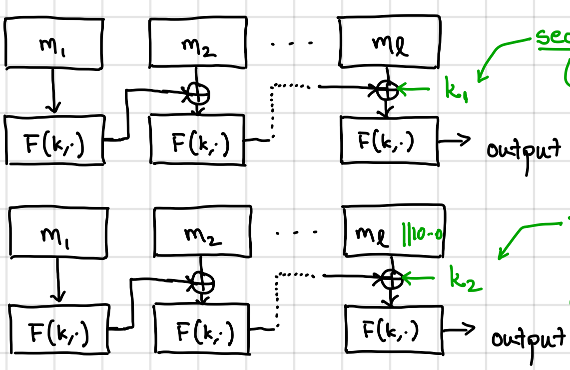
  Problematic if we do not know message length at the beginning (e.g., in a streaming setting)
  Still requires padding message to multiple of block size)
- <u>Option 2</u>: Apply a random secret shift to the last block of the message

$$(x_1, x_2, \ldots, x_\ell) \mapsto (x_1, x_2, \ldots, x_\ell \oplus k) \quad \text{where } k \xleftarrow{R} X$$

  Adversary that does not know k cannot construct two messages that are prefixes except with probability $1/|X|$ (by guessing k)

Cipher-based MAC (CMAC): variant of CBC-MAC standardized by NIST in 2005
→ randomized prefix-free encoding
→ clever technique to avoid extra padding block

better than encrypted CBC (should be preferred over ANSI standards)



<u>secret</u> random shift (part of the MAC key)

<u>different</u> keys needed to avoid collision between unpadded message and padded message (ending in $100\cdots0$)

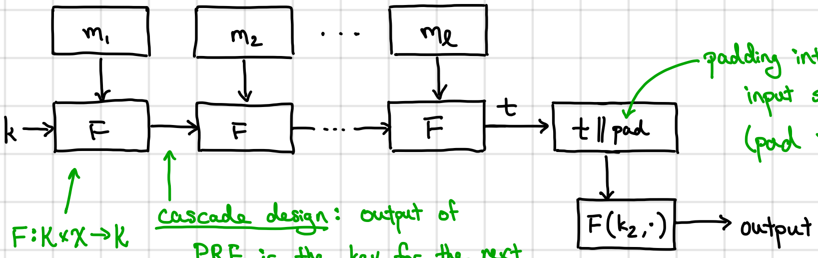
if message is not a multiple of block length, then pad (ANSI) and xor with <u>different</u> secret key k_2

never needs to introduce an additional block!

key: (k, k_1, k_2) — CMAC standard uses a specific key-derivation function to derive these keys from <u>one</u> key

Another approach based on a "cascade" design [Nested MAC (NMAC)]

- Variant of this is HMAC (IETF standard - widely used MAC protocol on the web — will discuss later)



$F: \mathcal{K} \times \mathcal{X} \to \mathcal{K}$

cascade design: output of PRF is the key for the next PRF

padding introduced if input size is larger than key size

(pad is a hard-coded constant in the standard)

↳ can just be string of 0s [not a padding function since input/output sizes are fixed]

key for NMAC is (k, k_2)

CBC-MAC, CMAC, and NMAC are PRF-based MACs (both approaches implicitly construct a variable-length PRF)

- All are in fact streaming MACs (message blocks can be streamed - no need to know a priori bound)
- All constructions are sequential

<u>Theorem.</u> Let $F: \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ be a secure PRF. (for NMAC, $F: \mathcal{K} \times \mathcal{X} \to \mathcal{K}$) Let Π_{ECBC} be the encrypted CBC MAC formed by F and let Π_{NMAC} be the NMAC formed by F. Then, for all MAC adversaries $\mathcal{A}$, there exists a PRF adversary $\mathcal{B}$ where

$$\mathrm{MACAdv}[\mathcal{A}, \Pi_{ECBC}] \leq 2 \cdot \mathrm{PRFAdv}[\mathcal{B}, F] + \frac{Q^2(\ell+1)^2}{|\mathcal{X}|}$$

$$\mathrm{MACAdv}[\mathcal{A}, \Pi_{NMAC}] \leq [Q(\ell+1)+1]\,\mathrm{PRFAdv}[\mathcal{B}, F] + \frac{Q^2}{2|\mathcal{K}|}$$

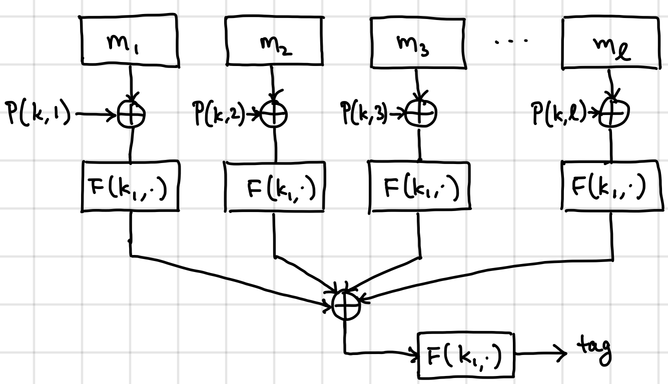
quadratic dependence on Q arises for similar reason as in analyzing CPA security (argue that all inputs to PRF are unique)

<u>Proof</u>. See Boneh-Shoup, Chapter 6.

<u>Implication</u>: Block size of PRF is important!

- 3DES: $|\mathcal{X}| = 2^{64}$; need to update key after $< 2^{32}$ signing queries
- AES: $|\mathcal{X}| = 2^{128}$; can use key to sign many more messages ($\sim 2^{64}$ messages)

A parallelizable MAC (PMAC) — general idea:



$P(k, \cdot)$ are important — otherwise, adversary can permute the blocks

derived as $F(k_1, 0^n)$ — so key is just k_1

↳ "mask" term is of the form $\gamma_i \cdot k$ where multiplication is done over $GF(2^n)$ where n is the block size (constants γ_i carefully chosen for efficient evaluation)

Can use similar ideas as CMAC (randomized prefix-free encoding) to support messages that is not constant multiple of block size

Parallel structure of PMAC makes it easily updateable (assuming F is a PRP)

↳ suppose we change block i from $m[i]$ to $m'[i]$:

$$\text{compute } F^{-1}(k_1, \text{tag}) \oplus \underbrace{F(k_1, m[i] \oplus P(k,i))}_{\text{old value}} \oplus \underbrace{F(k_1, m'[i] \oplus P(k,i))}_{\text{new value}}$$

PMAC is "incremental": can make local updates without full recomputation

In terms of performance:

- On sequential machine, PMAC comparable to ECBC, NMAC, CMAC
- On parallel machine, PMAC much better

} Best MAC we've seen so far, but not used...
Reason: patents :([not patented anymore!]

Summary: Many techniques to build a large-domain PRF from a small-domain one (domain extension for PRF)

↳ Each method (ECBC, NMAC, CMAC, PMAC) gives a MAC on variable-length messages

↳ Many of these designs (or their variants) are standardized