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

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 tolerates a single error (better error-correcting codes can do much better)

↳ 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, anyone 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 $\Pi_{MAC} = (\text{Sign}, \text{Verify})$:

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

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

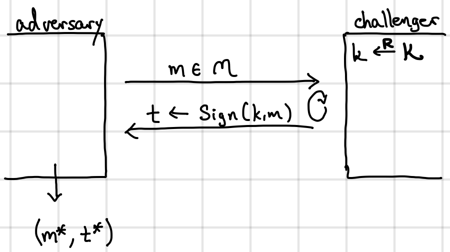
$$\Pr[\text{Verify}(k, m, \text{Sign}(k,m)) = 1] = 1$$

Sign can be a randomized 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 $\underbrace{\text{chosen message attacks}}_{\text{adversary gets to choose messages to be signed}}$ (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 $m_1, \dots, m_Q$ 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:

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

MAC security notion says that adversary cannot produce a new tag on any 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:

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

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

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

__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 $1/|\mathcal{T}|$.

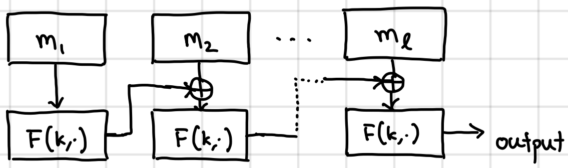
__Implication__: Any PRF with large output space can be used as a MAC.

↳ 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"

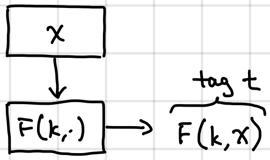
Raw-CBC is a way to build a __large-domain__ PRF from a __small-domain__ one

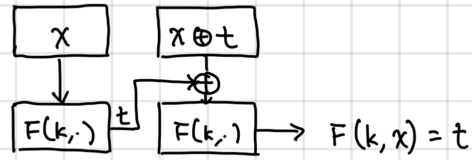
↳ Can show security for "prefix-free" messages [more precisely, raw-CBC is a prefix-free PRF: pseudorandom as long as PRF never evaluated on two values where one is a prefix of other]

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 $(x, t \oplus x)$

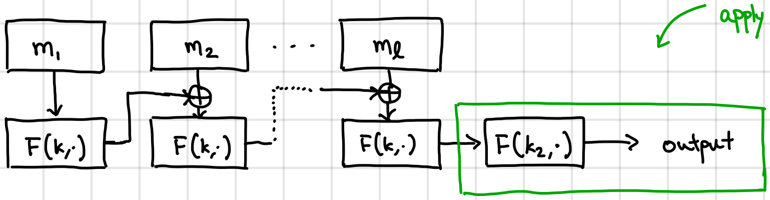
↳ Adversary succeed with advantage 1

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 $\overbrace{\text{ANSI X9.9, ANSI X19.9}}^{\text{standards for banking / financial services}}$ standards

apply another PRF with a different key to the output of raw CBC — critical for security (using the same key not 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 pad 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 security [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]

- Note: 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 bit-padding scheme [PKCS #7 that we discuss previously in the context of CBC encryption is a byte-padding scheme]

Encrypted CBC-MAC drawbacks: always need at least 2 PRF evaluations (using different 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?

- Option 1: Prepend 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)

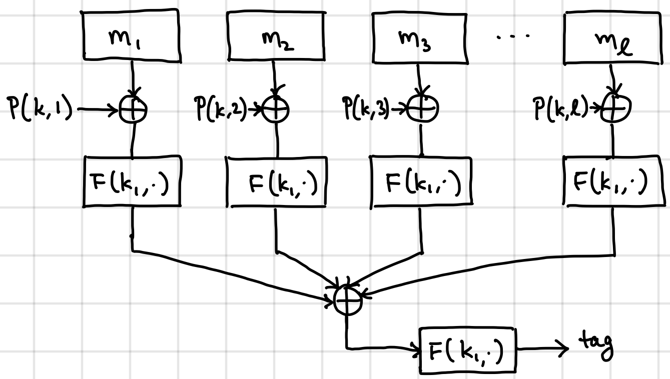
- Option 2: Apply a random secret shift to the last block of the message

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

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

  ↳ basis for CMAC (standardized by NIST in 2005)

A parallelizable MAC (PMAC) — general idea:



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

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

↳ "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, CMAC, PMAC) gives a MAC on variable-length messages

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