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

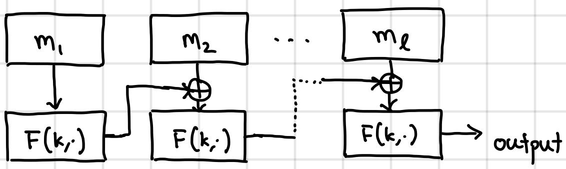
↳ AES has 128-bit output space, so can be used as a MAC

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"

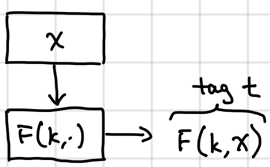
Raw-CBC is a way to build a large-domain PRF from a small-domain one

↳ 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]

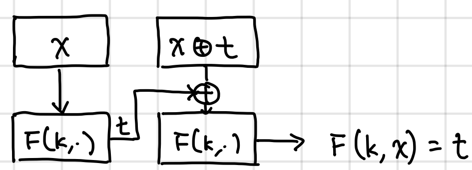
↳ includes fixed-length messages as a special case

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 $t$



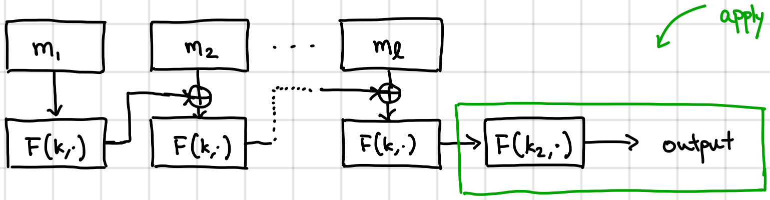

$\Rightarrow$ $t$ is a valid tag on extended message $(x, t \oplus x)$

↳ 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 (ECBC) CBC":
↳ variant used in $\underbrace{\text{ANSI X9.9, ANSI X19.9}}_{}$ standards — standards for banking / financial services

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



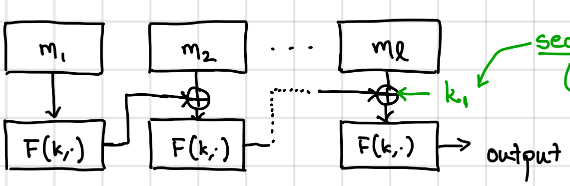

To use encrypted CBC-MAC, we need to assume message length is even multiple of block size (similar to CBC encryption)
↳ to sign messages that are not a multiple of the block size, we need to first pad the message
↳ as was the case with encryption, padding must be injective
  ↳ in the case of encryption, 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: $|\{0,1\}^{\le n}| > |\{0,1\}^n|$
- This is a bit-padding scheme [PKCS #7 that we discuss previously in the context of CBC encryption is a byte-padding scheme]

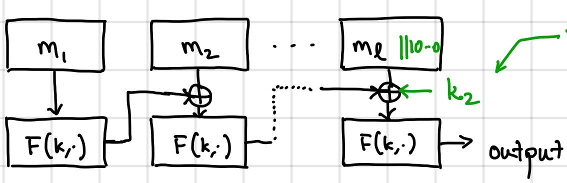
Encrypted CBC-MAC drawbacks: always need at least 2 PRF evaluations (using different keys)
messages must be padded to block size
} especially bad for authenticating short (e.g., single-byte) messages

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 $k$)

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)




secret random shift (part of the MAC key)




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

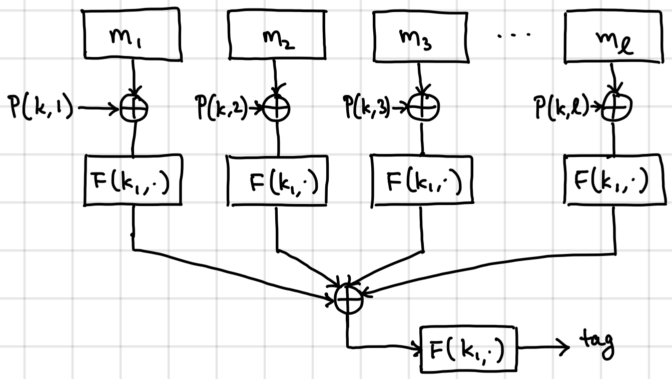
different keys needed to avoid collision between unpadded message and padded message ending in $100\cdots0$

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 one key

Implication: Block size of PRF is important!
- 3DES: $|X| = 2^{64}$; need to update key after $< 2^{32}$ signing queries
- AES: $|X| = 2^{128}$; can use key to sign many more messages ($\sim 2^{64}$ messages)

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 <u>permute</u> 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 $\gamma_i$ 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 $m'[i]$:

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

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...
<u>Reason</u>: patents :( [not patented anymore!]

<u>Summary</u>: 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 <u>variable-length</u> messages
↳ Many of these designs (or their variants) are <u>standardized</u>

How do we combine confidentiality and integrity?

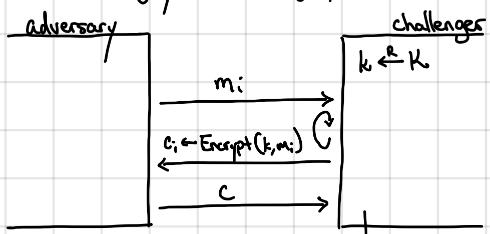
↳ Systems with both guarantees are called authenticated encryption schemes – gold standard for symmetric encryption

Two natural options:

1. Encrypt-then MAC (TLS 1.2+, IPsec) ← guaranteed to be secure if we instantiate using CPA-secure encryption and a secure MAC
2. MAC-then-encrypt (SSL 3.0/TLS 1.0, 802.11i) ← as we will see, not always secure

**Definition.** An encryption scheme $\Pi_{SE} = (\text{Encrypt}, \text{Decrypt})$ is an authenticated encryption scheme if it satisfies the following two properties:

- CPA security [confidentiality]
- ciphertext integrity [integrity]




output 1 if $c \notin \{c_1, c_2, \dots\}$ and $\text{Decrypt}(k, c) \neq \perp$

special symbol $\perp$ to denote invalid ciphertext

Define $\text{CIAdv}[\mathcal{A}, \Pi_{SE}]$ to be the probability that output of above experiment is 1. The scheme $\Pi_{SE}$ satisfies ciphertext integrity if for all efficient adversaries $\mathcal{A}$,

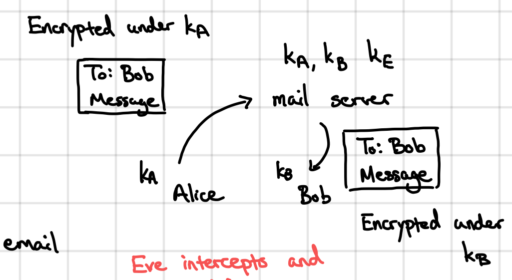
$$\text{CIAdv}[\mathcal{A}, \Pi_{SE}] = \text{negl}(\lambda)$$

security parameter determines key length

Ciphertext integrity says adversary cannot come up with a new ciphertext: only ciphertexts it can generate are those that are already valid. Why do we want this property?

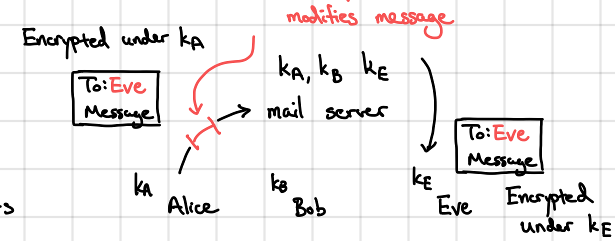
Consider the following active attack scenario:

- Each user shares a key with a mail server
- To send mail, user encrypts contents and send to mail server
- Mail server decrypts the email, re-encrypts it under recipient's key and delivers email

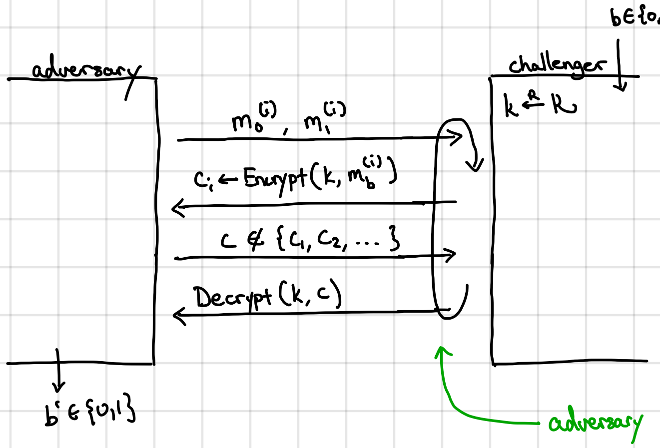



If Eve is able to tamper with the encrypted message, then she is able to learn the encrypted contents (even if the scheme is CPA-secure)

↳ More broadly, an adversary can tamper and inject ciphertexts into a system and observe the user's behavior to learn information about the decrypted values – against active attackers, we need stronger notion of security



**Definition.** An encryption scheme $\Pi_{SE} = (\text{Encrypt}, \text{Decrypt})$ is secure against chosen-ciphertext attacks (CCA-secure) if for all efficient adversaries $\mathcal{A}$, $\text{CCAAdv}[\mathcal{A}, \Pi_{SE}] = \text{negl.}$ where we define $\text{CCAAdv}[\mathcal{A}, \Pi_{SE}]$ as follows:




adversary can make arbitrary encryption and decryption queries, but cannot decrypt any ciphertexts it received from the challenger (otherwise, adversary can trivially break security)
↳ called an "admissibility" criterion

$$\text{CCAAdv}[\mathcal{A}, \Pi_{SE}] = \left| \Pr[b'=1 \mid b=0] - \Pr[b'=1 \mid b=1] \right|$$

CCA-security captures above attack scenario where adversary can tamper with ciphertexts
- ↳ Rules out possibility of transforming encryption of $x \| z$ to encryption of $y \| z$
- ↳ Necessary for security against <u>active</u> adversaries [CPA-security is for security against <u>passive</u> adversaries]
- ↳ We will see an example of a real CCA attack in HW1

**Theorem.** If an encryption scheme $\Pi_{SE}$ provide authenticated encryption, then it is CCA-secure.

**Proof (Idea).** Consider an adversary $\mathcal{A}$ in the CCA-security game. Since $\Pi_{SE}$ provides ciphertext integrity, the challenger's response to the adversary's decryption query will be $\perp$ with all but negligible probability. This means we can implement the decryption oracle with the "output $\perp$" function. But then this is equivalent to the CPA-security game.
[Formalize using a hybrid argument]

simple counter-example: concatenate unused bits to end of ciphertext in a CCA-secure scheme (stripped away during decryption)

**Note:** Converse of the above is not true since CCA-security $\not\Rightarrow$ ciphertext integrity.
- ↳ However, CCA-security + plaintext integrity $\Rightarrow$ authenticated encryption

**Take-away:** Authenticated encryption captures meaningful confidentiality + integrity properties; provides <u>active</u> security

**Encrypt-then-MAC:** Let (Encrypt, Verify) be a CPA-secure encryption scheme and (Sign, Verify) be a secure MAC. We define Encrypt-then-MAC to be the following scheme:

$\text{Encrypt}'((k_E, k_M), m)$: $c \leftarrow \text{Encrypt}(k_E, m)$
$t \leftarrow \text{Sign}(k_M, c)$
output $(c, t)$

independent keys

$\text{Decrypt}'((k_E, k_M), (c, t))$: if $\text{Verify}(k_M, c, t) = 0$, output $\perp$
else, output $\text{Decrypt}(k_E, c)$