A parallelizable MAC (PMAC) - general idea:

\[
\begin{array}{c}
\begin{array}{c}
m_1 \\
P(k_1) \\
F(k_1, \cdot) \\
F(k_1, \cdot) \\
\vdots \\
m_k \\
P(k_k) \\
F(k_k, \cdot) \\
F(k_k, \cdot) \\
\end{array}
\end{array}
\rightarrow
tag
\]

- Derived as ECK (04) so key is just \( k \), \( P(k, \cdot) \) are important - otherwise, adversary can permute the blocks.
- "Mask" term is of the form \( \Pi_i k_i \) where multiplication is done over \( GF(2^a) \) where \( a \) is the block size (constants \( \Pi_i \) 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)

\[
\text{Suppose we change block } i \text{ from } m[i] \text{ to } m'[i]:
\]

\[
\text{compute } \begin{array}{c}
F^{-1}(k, \text{tag}) \oplus F(k, m[i] \oplus P(k,i)) \\
\text{old value} \\
\end{array} \\
\begin{array}{c}
\oplus F(k, m'[i] \oplus P(k,i)) \\
\text{new value}
\end{array}
\]

PMAC is "incremental":

\[
\begin{array}{c}
\text{can make local updates} \\
\text{without full recomputation}
\end{array}
\]

In terms of performance:

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

\[
\begin{array}{c}
\text{Best MAC we've seen so far, but not used...} \\
\text{Reason: patents} \\
\text{[not patented anymore!]}
\end{array}
\]

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

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

\[
\text{Summary: Many techniques to build a large-domain PRF from a small-domain one (domain extension for PRF)}
\]

- Each method (ECBC, CMAC, PMAC) gives a MAC on variable-length messages
- Many of these designs (or their variants) are standardized
How do we ensure 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 $TSE = (Encrypt, Decrypt)$ is an authenticated encryption scheme if it satisfies the following two properties:
- CPA security
- ciphertext integrity

An adversary can create an $m$, $c = Encrypt(m)$, and a challenger can take $c$ and the key $k$.

If $c \notin \{c_1, c_2, \ldots, l\}$ and $Decrypt(k, c) \neq \bot$, output 1 and define $CIAdv[A, TSE] = 0$.

Define $CIAdv[A, TSE]$ to be the probability that output of above experiment is 1. The scheme $TSE$ satisfies ciphertext integrity if for all efficient adversaries $A$,

$\forall A, \exists \kappa : CIAdv[A, TSE] = \text{negl}(\kappa)$

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 $T_{SE}(\text{Encrypt, Decrypt})$ is secure against chosen-ciphertext attacks (CCA-secure) if for all efficient adversaries $A$, $\text{CCAAdv}[A, T_{SE}] = \negl$. Where we define $\text{CCAAdv}[A, T_{SE}]$ as follows:

$$\text{CCAAdv}[A, T_{SE}] = \Pr[b' = 1 | b = 0] - \Pr[b' = 1 | b = 1]$$

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

Theorem. If an encryption scheme $T_{SE}$ provides authenticated encryption, then it is CCA-secure.

Proof (Idea). Consider an adversary $A$ in the CCA-security game. Since $T_{SE}$ provides ciphertext integrity, the challenger's response to the adversary's decryption query will be $1$ with all but negligible probability. This means we can implement the decryption oracle with the “output 1” 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 active security

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

$\text{Encrypt}'((k_E, k_M), m)$: $c \leftarrow \text{Encrypt}(k_E, m)$

$\text{Sign}'(k_M, c)$: $t \leftarrow \text{Sign}(k_M, c)$

$\text{Output} (c, t)$

$\text{Decrypt}'((k_E, k_M), (c, t))$: if $\text{Verify}(k_M, c, t) = 0$, output $1$
else, output $\text{Decrypt}(k_E, c)$
Theorem. If \((\text{Encrypt}, \text{Decrypt})\) is CPA-secure and \((\text{Sign}, \text{Verify})\) is a secure MAC, then \((\text{Encrypt}', \text{Verify}')\) is an authenticated encryption scheme.

Proof. (Sketch.) CPA-security follows by CPA-security of \((\text{Encrypt}, \text{Decrypt})\). Specifically, the MAC is computed on ciphertexts and not the messages. MAC key is independent of encryption key so cannot compromise CPA-security.
Ciphertext integrity follows directly from MAC security (i.e., any valid ciphertext must contain a new tag on some ciphertext that was not given to the adversary by the challenger).

Important notes: Encryption + MAC keys must be independent. Above proof required this (in the formal reduction, need to be able to simulate ciphertexts/MACs — only possible if reduction can change its own key).
\(\Rightarrow\) Can also give explicit constructions that are completely broken if same key is used (i.e., both properties fail to hold).
\(\Rightarrow\) In general, never reuse cryptographic keys in different schemes; instead, sample fresh, independent keys!

MAC-then-Encrypt: Let \((\text{Encrypt}, \text{Verify})\) be a CPA-secure encryption scheme and \((\text{Sign}, \text{Verify})\) be a secure MAC. We define
\[
\text{MAC-then-Encrypt to be the following scheme:}
\]
\[
\begin{align*}
\text{Encrypt}'((k_E,k_m), m): & \quad t \leftarrow \text{Sign}(k_m, m) \\
& \quad c \leftarrow \text{Encrypt}(k_E, (m,t)) \\
& \quad \text{output } c
\end{align*}
\]
\[
\begin{align*}
\text{Decrypt}'((k_E,k_m), (c,t)): & \quad \text{compute } (m,t) \leftarrow \text{Decrypt}(k_E, c) \\
& \quad \text{if } \text{Verify}(k_m, m, t) = 1, \text{ output } m, \text{ else, output } 1
\end{align*}
\]

Not generally secure! SSL 3.0 (precursor to TLS) used randomized CBC + secure MAC
\(\Rightarrow\) Simple CCA attack on scheme (by exploiting padding in CBC encryption)
\([\text{POODLE attack on SSL 3.0 can decrypt all encrypted traffic using a CCA attack}]\)

Padding is a common source of problems with MAC-then-Encrypt systems [see HW3 for an example].

In the past, libraries provided separate encryption + MAC interfaces — common source of errors
\(\Rightarrow\) Good library design for crypto should minimize ways for users to make errors, not provide more flexibility

Today, there are standard block cipher modes of operation that provide authenticated encryption
- One of the most widely used is GCM (Galois counter mode) — standardized by NIST in 2007

GCM mode: follows encrypt-then-MAC paradigm
- CPA-secure encryption + nonce-based counter mode
- MAC is a Carter-Wegman MAC
\(\Rightarrow\) “encrypted one-time MAC”
GCM encryption: encrypt message with AES in counter mode
- compute Carter-Wegman MAC on resulting message using GHASH as the underlying hash function and the block cipher as underlying PRF

Typically, use AES-GCM for authenticated encryption

Galois Hash
- key derived from PRF
- GHASH operates on blocks of 128-bits
- operations can be expressed as operations over $\mathbb{GF}(2^{128})$ - Galois field with $2^{128}$ elements
- implemented in hardware - very fast!

Oftentimes, only part of the payload needs to be hidden, but still needs to be authenticated.
- e.g., sending packets over a network: desire confidentiality for packet body, but only integrity for packet headers (otherwise, cannot route!)

AEAD: authenticated encryption with associated data
- augment encryption scheme with additional plaintext input; resulting ciphertext ensures integrity for associated data, but not confidentiality
  (will not define formally here, but follows straightforwardly from AE definitions)
- can construct directly via “encrypt-then-MAC”: namely, encrypt payload and MAC the ciphertext + associated data
- AES-GCM is an AEAD scheme