How do we combine confidentiality and integrity?

Systems with both guarantees are called **authenticated encryption schemes** - gold standard for symmetric encryption.

The 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, 802.11) - as we will see, not always secure.

**Definition.** An encryption scheme $T$: $(Encrypt, Decrypt)$ is an authenticated encryption scheme if it satisfies the following two properties:

- **Ciphertext integrity** [integrity]
- **Confidentiality** [confidentiality]

An adversary $A$ can choose a message $m$ and a key $k$.

**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 \( T \) (Encrypt, Decrypt) is secure against chosen-ciphertext attacks (CCA-secure) if for all efficient adversaries \( A \), \( CCA_{Adv}[A, T] = \) negl. where we define \( CCA_{Adv}[A, T] \) as follows:

\[
\text{adversary} \downarrow
\begin{align*}
\text{challenger} & \downarrow \\
& \begin{cases}
k \in \{0,1\}^n \\
L \leftarrow \{0,1\}^n \\
E[challenger](k, L) \text{ is a uniformly random key}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
& \text{c} \leftarrow \text{Encrypt}(k, m^{(0)}) \\
& \text{c} \notin \{0,1 \}^n
\end{align*}
\]

\[
\downarrow \\
\text{Decrypt}(k, c)
\]

\[
\text{CCA}_{Adv}[A, T] := | \Pr[b'=1 | b=0] - \Pr[b'=1 | b=1] |
\]

CCA-security captures above attack scenario where adversary can tamper with ciphertexts:
- Rules out possibility of transforming encryption of \( x \in \mathbb{E} \) to encryption of \( y \in \mathbb{E} \)
- Necessary for security against active adversaries (CPA-security is for security against passive adversaries)
- We will see an example of a real CCA attack in HW3.

Theorem. If an encryption scheme \( T \) provides authenticated encryption, then it is CCA-secure.

Proof (Idea). Consider an adversary \( A \) in the CCA-security game. Since \( T \) 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.

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

Takeaway: Authenticated encryption captures meaningful confidentiality + integrity properties; provides active security.

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

\[
\text{Encrypt}'((k, k_{\text{mac}}), m) := \\
\begin{cases}
\text{c} \leftarrow \text{Encrypt}(k, m) \\
\text{t} \leftarrow \text{Sign}(k_{\text{mac}}, \text{c})
\end{cases}
\]

\[
\text{Decrtypt}'((k, k_{\text{mac}}), (c, t)) := \\
\begin{cases}
\text{if } \text{Verify}(k_{\text{mac}}, \text{c}, \text{t}) = 0, \text{ output } 1 \\
\text{else, output Decrypt}(k, c)
\end{cases}
\]
Theorem. If $(\text{Encrypt}, \text{Decrypt})$ is CPA-secure and $(\text{Sign}, \text{Verify})$ is a secure MAC, then $(\text{Encrypt, Verify})$ is an authenticated encryption scheme.

Proof (Sketch). CPA-security follows by CPA-security of $(\text{Encrypt, Decrypt})$. 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).

- Can also give explicit constructions that are completely broken if same key is used (i.e., both properties fail to hold).
- In general, never reuse cryptographic keys in different schemes; instead, sample fresh, independent keys!

- MAC needs to be computed over the entire ciphertext.
- Early version of ISO 19772 for AE did not MAC IV (CBC used for CPA-secure encryption)
- MACrypt in Apple iOS (for data encryption) also problematic (HMAC not applied to encryption IV)

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

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

$$\text{Decrypt}'((kE, km), (c, t)):\ \text{compute } (m, t) \leftarrow \text{Decrypt}(kE, c)$$
$$\text{if } \text{Verify}(km, m, t) = 1, \text{ output } m, \text{ else, output } \bot$$

Not generally secure! SSL 3.0 (precursor to TLS) used randomized CBC + secure MAC

- Simple CCA attack on scheme (by exploiting padding in CBC encryption)

[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 H3D2 for an example]

In the past, libraries provided separate encryption + MAC interfaces — common source of errors

- 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) — standardised by NIST in 2007

GCM mode: follows encrypt-then-MAC paradigm

- CPA-secure encryption + nonce-based counter mode
- MAC is a Carter-Wegman MAC

Most commonly used in conjunction with AES

AES-GCM provides authenticated encryption