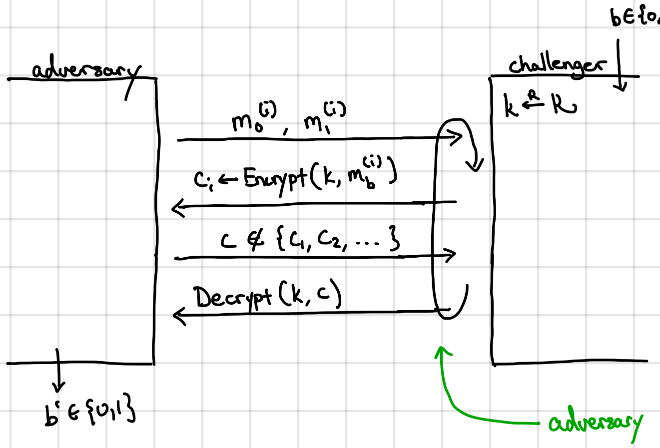
Definition. An encryption scheme $\Pi_{SE} = (\text{Encrypt}, \text{Decrypt})$ is secure against chosen-ciphertext attacks (CCA-secure) if for all efficient adversaries $\mathcal{A}$, $\text{CCAAdv}[\mathcal{A}, \Pi_{SE}] = \text{negl.}$ where we define $\text{CCAAdv}[\mathcal{A}, \Pi_{SE}]$ as follows:



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 Π_{SE} provide authenticated encryption, then it is CCA-secure.

Proof (Idea). Consider an adversary $\mathcal{A}$ in the CCA-security game. Since Π_{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:

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

independent keys

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

Theorem. If (Encrypt, Decrypt) is CPA-secure and (Sign, Verify) is a secure MAC, then (Encrypt', Verify') is an authenticated encryption scheme

Proof. (Sketch). CPA-security follows by CPA-security of (Encrypt, Decrypt). Specifically, the MAC is computed on ciphertexts and <u>not</u> 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 <u>independent</u>. Above proof required this (in the formal reduction, need to be able to simulate ciphertexts / MACs — only possible if reduction can chose its own key).
  - ↳ Can also give explicit constructions that are <u>completely broken</u> if same key is used (i.e., both properties fail to hold)
  - ↳ In general, never <u>reuse</u> cryptographic keys in different schemes; instead, sample fresh, independent keys!
- MAC needs to be computed over the <u>entire</u> ciphertext
  - Early version of ISO 19772 for AE did not MAC IV (CBC used for CPA-secure encryption)
  - RNCryptor in Apple iOS (for data encryption) also problematic (HMAC not applied to encryption IV)

  means first block (i.e., "header") is <u>malleable</u>

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

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

$$\text{Decrypt}'((k_E, k_M), (c,t)): \quad \text{compute } (m,t) \leftarrow \text{Decrypt}(k_E, c)$$
$$\text{if Verify}(k_M, m, t) = 1, \text{ output } m, \text{ else, output } \perp$$

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 <u>all</u> encrypted traffic using a CCA attack]

Padding is a common source of problems with MAC-then-Encrypt systems [see HW2 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, <u>not</u> provide more flexibility

Today, there are standard block cipher modes of operation that provide <u>authenticated encryption</u>
- 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 is nonce-based counter mode
- MAC is a Carter-Wegman MAC

Most commonly used in conjuction with AES (AES-GCM provides authenticated encryption)

Carter-Wegman MAC ("encrypted MAC"): very lightweight, randomized MAC:

- Let $H: K_H \times M \to \{0,1\}^n$ be a keyed hash function
- Let $F: K_F \times R \to \{0,1\}^n$ be a PRF

security relies on a mild assumption on the hash function and can be realized unconditionally
↳ security relies only on PRF security

The Carter-Wegman MAC is defined as follows:

$\text{Sign}((k_H, k_F), m)$: $r \xleftarrow{R} R$
$t \leftarrow H(k_H, m) \oplus F(k_F, r)$
output (r, t)

$\text{Verify}((k_H, k_M), (r,t))$: output 1 if $F(k_F, r) \oplus t = H(k_H, m)$ and 0 otherwise

very simple construction!
but tags are longer (need both a nonce and a PRF output)

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

Galois Hash

key derived from PRF evaluation at 0^n

↳ GHASH operates on blocks of 128-bits
operations can be expressed as operations over $GF(2^{128})$ — Galois field with 2^{128} elements
implemented in hardware — very fast!

Typically, use AES-GCM for authenticated encryption

$GF(2^{128})$ is defined by the polynomial $g(x) = x^{128} + x^7 + x^2 + x + 1$
↳ elements are polynomials over $\mathbb{F}_2$ with degree less than 128 [e.g. $x^{127} + x^{52} + x^2 + x + 1$]
(can be represented by 128-bit string: each bit is coefficient of polynomial)
↳ can add elements (xor) and multiply them (as polynomials) — implemented in hardware
(also used for evaluating the AES round function)

↳ $\text{GHASH}(k, \overbrace{m}^{(m[1], m[2], \dots, m[\ell])}) := m[1]k^{\ell} + m[2]k^{\ell-1} + \cdots + m[\ell]k$ [values $m[1], \dots, m[\ell]$ give coefficients of polynomial, evaluate at point k]

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