**Nonce-based counter mode**: divide IV into two pieces: IV = nonce ‖ counter

↑ value that does not repeat

common choices: 64-bit nonce, 64-bit counter
96-bit nonce, 32-bit counter } only nonce needs to be sent! (slightly smaller ciphertexts)

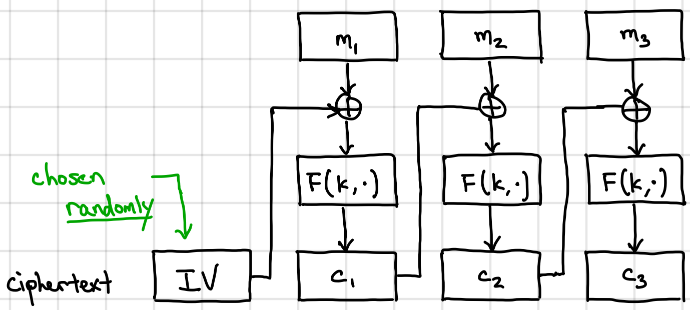
Only requirement for security is that IV does not repeat:

- Option 1: Choose randomly (either IV or nonce)
- Option 2: If sender + recipient have shared state (e.g., packet counter), can just use a counter, in which case, IV/nonce does not have to be sent

Counter mode (CTR) is parallelizable, simple-to-implement, just requires PRF — preferred mode of using block ciphers

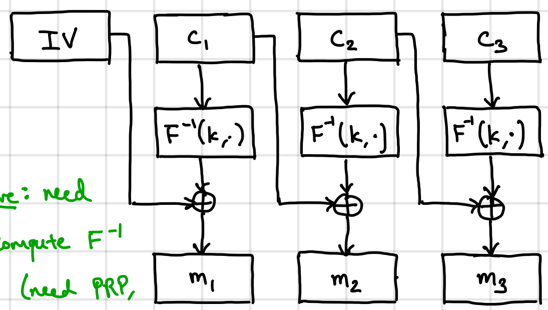
Other block cipher modes of operation:

Cipherblock chaining (CBC): common mode in the past (e.g., TLS 1.0, still widely used today)




Encryption

Observe: need to compute $F^{-1}$ here (need PRP, not just PRF)




Decryption

**Theorem**: Let $F: \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a secure PRF and let $\Pi_{CBC}$ denote the CBC encryption scheme for $\ell$-block messages ($\mathcal{M} = \mathcal{X}^{\leq \ell}$). Then, for all efficient CPA adversaries $\mathcal{A}$, there exists an efficient PRF adversary $\mathcal{B}$ such that

$$\mathrm{CPAAdv}[\mathcal{A}, \Pi_{CBC}] \leq \frac{2Q^2\ell^2}{|\mathcal{X}|} + 2 \cdot \mathrm{PRFAdv}[\mathcal{B}, F]$$

↖ $Q$: number of encryption queries
$\ell$: number of blocks in message

**Intuition**: similar to analysis of randomized counter mode:

1. Ciphertext is indistinguishable from random string if PRP is evaluated on distinct inputs
2. When encrypting, PRP is invoked on $\ell$ random blocks, so after $Q$ queries, we have $Q\ell$ random blocks.
   $\Rightarrow$ Collision probability $\leq \frac{Q^2\ell^2}{|\mathcal{X}|}$ ← this is larger than collision prob. for randomized counter mode by a factor of $\frac{\ell}{2}$ [overlap of $Q$ random intervals vs. $Q\ell$ random points]
3. Factor of 2 arises for same reason as before

**Interpretation**. CBC mode provides weaker security compared to counter mode: $\frac{2Q^2\ell^2}{|\mathcal{X}|}$ vs. $\frac{4Q^2\ell}{|\mathcal{X}|}$

Concretely: for same parameters as before (1 MB messages, $2^{-32}$ distinguishing advantage):

$$Q \leq \sqrt{\frac{|\mathcal{X}| \cdot 2^{-32}}{2\ell^2}} = \sqrt{\frac{2^{128} \cdot 2^{-32}}{2(2^{16})^2}} = \sqrt{2^{63}} = 2^{31.5} \quad (\sim 1 \text{ billion messages})$$

↳ $2^{7.5}$ ~ 180x smaller than using counter mode

**Padding in CBC mode:** each ciphertext block is computed by feeding a message block into the PRP

⇒ message must be an even multiple of the block size

⇒ when used in practice, need to pad messages

Can we pad with zeroes? Cannot decrypt! What if original message ended with a bunch of zeroes?

**Requirement:** padding must be invertible

CBC padding in TLS 1.0: if $k$ bytes of padding is needed, then append $k$ bytes to the end, with each byte set to $k-1$

(for AES-CBC) if 0 bytes of padding is needed, then append a block of 16 bytes, with each byte equal to 15

↳ dummy block needed to ensure pad is invertible

[injective functions must expand: $|\{0,1\}^{\leq 256}| > |\{0,1\}^{256}|$]

↳ called PKCS#5/PKCS#7 (public-key cryptography standards)

Need to pad in CBC encryption can be exploited in "padding oracle" attacks

Padding in CBC can be avoided using idea called "ciphertext stealing" (as long as messages are more than 1 block)

**Comparing CTR mode to CBC mode:**

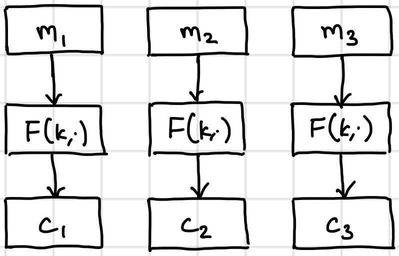
| CTR mode | CBC mode |
| --- | --- |
| 1. no padding needed (shorter ciphertexts) | 1. padding needed |
| 2. parallelizable | 2. sequential |
| 3. only requires PRF (no need to invert) | 3. requires PRP |
| 4. tighter security | 4. less tight security (re-key more often) |
| 5. IVs have to be non-repeating (and spaced far apart) | 5. requires unpredictable IVs |

Notes on CTR mode:
- [easy to implement: IV = nonce ‖ counter] ↑ only needs to be non-repeating (can be predictable)

Notes on CBC mode:
- padding needed ← imagine 1 byte messages (e.g., encrypted key strokes over SSH) → 1 block + 1 byte with CTR, 2 blocks with CBC
  - interesting traffic analysis attack: each keystroke is sent in separate packet, so # packets leaks info on length of user's password!
- requires PRP ← requires more structured primitive, more code to implement forward and backward evaluation
- requires unpredictable IVs ← TLS 1.0 used predictable IVs (see HW1 for an attack); SSH v1 used a 0 IV (even worse!)

**Bottom-line:** use randomized or nonce-based counter mode whenever possible: simpler, easier, and better than CBC!

A tempting and <u>bad</u> way to use a block cipher: ECB mode (electronic codebook)




Scheme is deterministic! Cannot be CPA secure!

Not even semantically secure!

$(m_0, m_0)$ vs. $(m_0, m_1)$ where $m_1 \neq m_0$

- $(m_0, m_0)$: ciphertext blocks output are same
- $(m_0, m_1)$: ciphertext blocks output are different

**Encryption:** simply apply block cipher to each block of the message

**Decryption:** simply invert each block of the ciphertext

NEVER USE ECB MODE FOR ENCRYPTION!

**Message integrity:** Confidentiality alone not sufficient, also need message integrity. Otherwise adversary can tamper with the message (e.g., "Send \$100 to Bob" → "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 $\Pi_{MAC} = (\text{Sign}, \text{Verify})$:

$$\text{Sign}: \mathcal{K} \times \mathcal{M} \to \mathcal{T}$$
$$\text{Verify}: \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{0,1\}$$

} Must be efficiently-computable

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

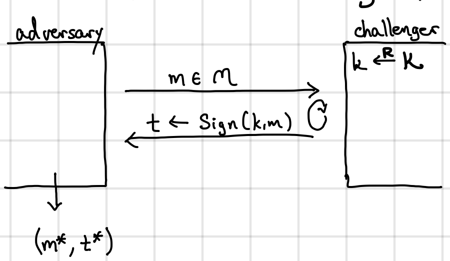
$$\Pr[\text{Verify}(k, m, \text{Sign}(k,m)) = 1] = 1$$

↑ 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 chosen message attacks (EUF-CMA) [adversary gets to choose messages to be signed] 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, $\lambda$ 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 $m_1, \ldots, m_Q$ 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:

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

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.