CSE 3400 - Introduction to Computer & Network Security
(aka: Introduction to Cybersecurity)

Lecture 4
Encryption – Part III
(and Pseudo-randomness)

Ghada Almashaqbeh
UConn

From Textbook Slides by Prof. Amir Herzberg
UConn

Outline

- Block ciphers.
- Pseudorandom permutations (PRPs).
- Defining security of encryption.
- Encryption modes.
- Concluding remarks.

Block Ciphers

- A pair of algorithms E_k and D_k (encrypt and decrypt with key k) with domain and range of $\{0,1\}^n$
  - Encrypt and decrypt data in blocks each of which is of size n bits.
- Conventional correctness requirement: $m = Dk(Ek(m))$
- Several schemes used in practice including DES and AES.
  - No security proofs, just resistance to cryptanalysis.
  - DES is insecure for short keys, replaced by AES.
- Security requirement of block ciphers is to be a pair of Pseudorandom Permutations (PRP).

So what is a Random Permutation?

And what is a PRP?

What is a random **permutation** ρ ?

- Random permutation ρ over finite domain D, usually: $\{0,1\}^m$
- How can we select a random permutation ρ ?
- Let $D = \{x_1, x_2, \dots, x_n\}$
- For $i = 1, \dots, n$:
  - $\rho(x_i) \xleftarrow{\$} D - \{\rho(x_1), \rho(x_2), \dots, \rho(x_{i-1})\}$
- Examples:

Domain D $\{0,1\}^2$

	ρ( )
00	10
01	11
10	00
11	01

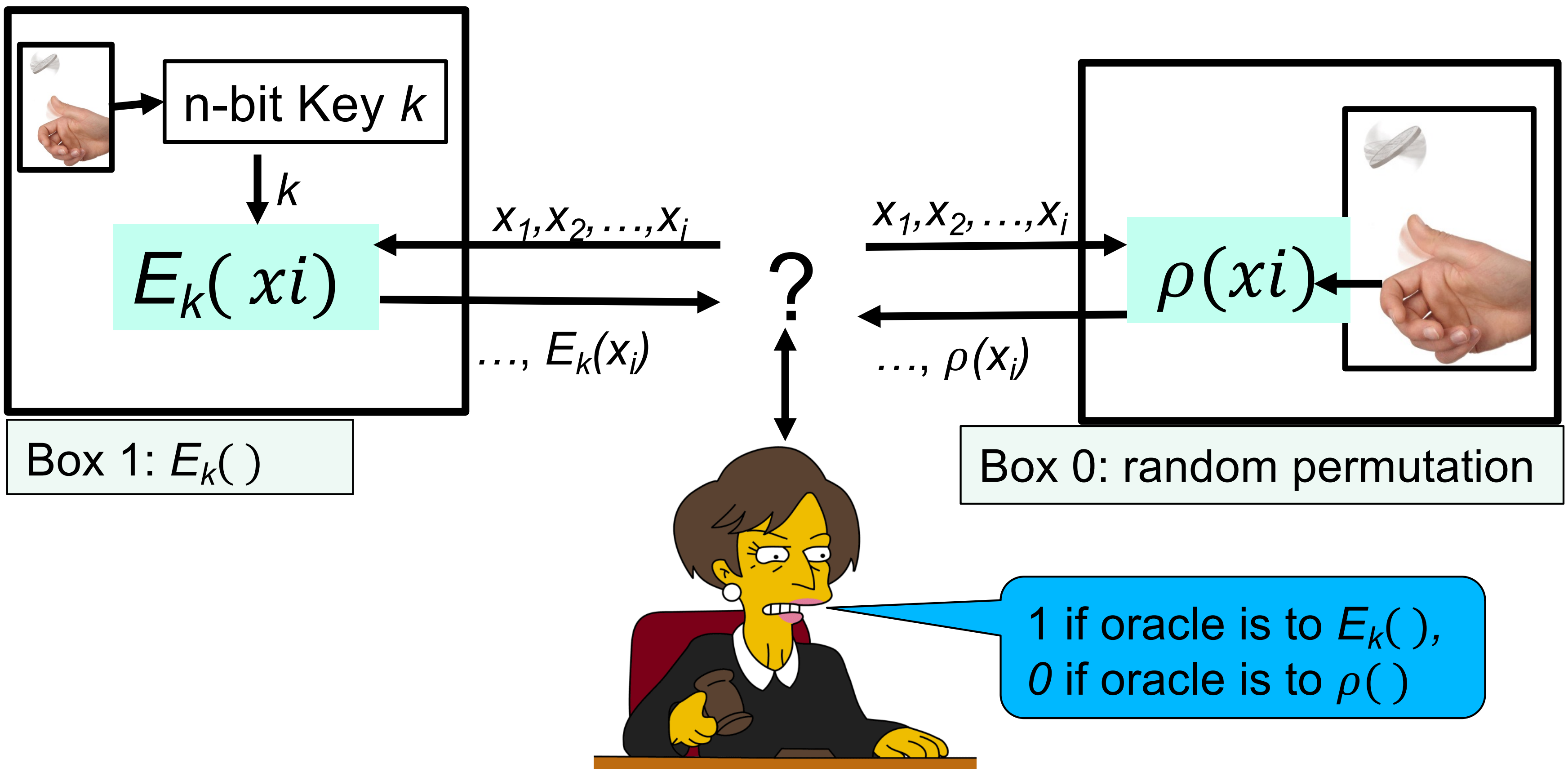
Domain D $\{0,1\}^2$

	ρ( )
00	00
01	01
10	10
11	11

Pseudo-Random Permutation (PRP)

and their Indistinguishabity Test

- *E* is a PRP over domain D, if no distinguisher D:
  - Outputs 1 (signaling PRP) given oracle to $E_k(\,)$, for random (n-bits) key *k,* and
  - Outputs 0 (signaling random) given oracle to $\rho(\,)$, a <u>random</u> permutation (over D)



Pseudo-Random Permutation (PRP)

- Pseudo-Random Permutation (PRP) $E_k(\,)$
  - Cannot be distinguished from truly random permutation over same domain
  - Against efficient adversaries (PPT), allowing negligible advantage
  - Yet practical, even efficient

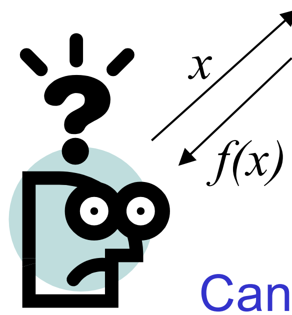
Definition 2.9. *A* pseudo-random Permutation *(*PRP*) is a polynomial-time computable function $E_k(x) : \{0,1\}^* \times D \to D \in PPT$ s.t. for all PPT algorithms $\mathcal{A}$, $\varepsilon^{PRP}_{\mathcal{A},E}(n) \in NEGL(n)$, i.e., is negligible, where the* advantage *$\varepsilon^{PRP}_{\mathcal{A},E}(n)$ of the PRP E against adversary $\mathcal{A}$ is defined as:*

$$\varepsilon^{PRP}_{\mathcal{A},E}(n) \equiv \Pr_{k \xleftarrow{\$} \{0,1\}^n}\left[\mathcal{A}^{E_k}(1^n)\right] - \Pr_{\rho \xleftarrow{\$} Perm(D)}\left[\mathcal{A}^{\rho}(1^n)\right] \qquad (2.16)$$

The probabilities are taken over random coin tosses of $\mathcal{A}$, and random choices of the key $k \xleftarrow{\$} \{0,1\}^n$ and of the function $\rho \xleftarrow{\$} Perm(D)$.

Block Cipher: Invertible PRP (E, D)

- Common definition for **block cipher**
- Invertible Pseudo-Random Permutation (PRP):
  - A pair of PRPs (E,D), s.t.: $m=D_k(E_k(m))$
  - And (E,D) is indistinguishable from (π, π^{-1})
    - where π is a random permutation
  - Note: it is deterministic, stateless → not secure encryption!
    - But used to construct encryption (soon)



$f \leftarrow E_k(\,)$ or $D_k(\,)$ [for random k]

or

$f \leftarrow$ *random n-bit permutation or its inverse*

Can't tell if (f, f^{-1}) is a random permutation+inverse, or *(E, D)* with a random key!

Example of a Block Cipher Security and Correctness

- $E_k(m) = m + k \bmod 2^n$
- In class.
  - $D_k(c)$?
  - Correctness.
  - Is it secure?

Constructing block-cipher, PRP

- Focus: constructions from a PRF $f_k(\,)$
  - PRFs seem easier to design (less restrictions)
- First: 'plain' PRP $E_k(\,)$ (not a block cipher)
- What is the simplest construction to try? $E_k(x)$=$f_k(x)$

Lemma 2.4 (The PRP/PRF Switching Lemma)**.** *Let E be a polynomial-time computable function $E_k(x) : \{0,1\}^* \times D \rightarrow D \in PPT$, and let $\mathcal{A}$ be a PPT adversary, which is limited to at most q oracle queries. Then:*

$$\left|\varepsilon^{PRF}_{\mathcal{A},E}(n) - \varepsilon^{PRP}_{\mathcal{A},E}(n)\right| < \frac{q^2}{2 \cdot |D|} \tag{2.17}$$

Where the advantage functions are as defined in Equation 2.16 and Equation 2.13.

In particular, if the size of the domain D is exponential in the security parameter n (the length of key and of the input to $\mathcal{A}$), e.g., $D = \{0,1\}^n$, then $\varepsilon^{PRF}_{\mathcal{A},E}(n) - \varepsilon^{PRP}_{\mathcal{A},E}(n) \in NEGL(n)$. In this case, E is a PRP over D, if and only if it is a PRF over D.

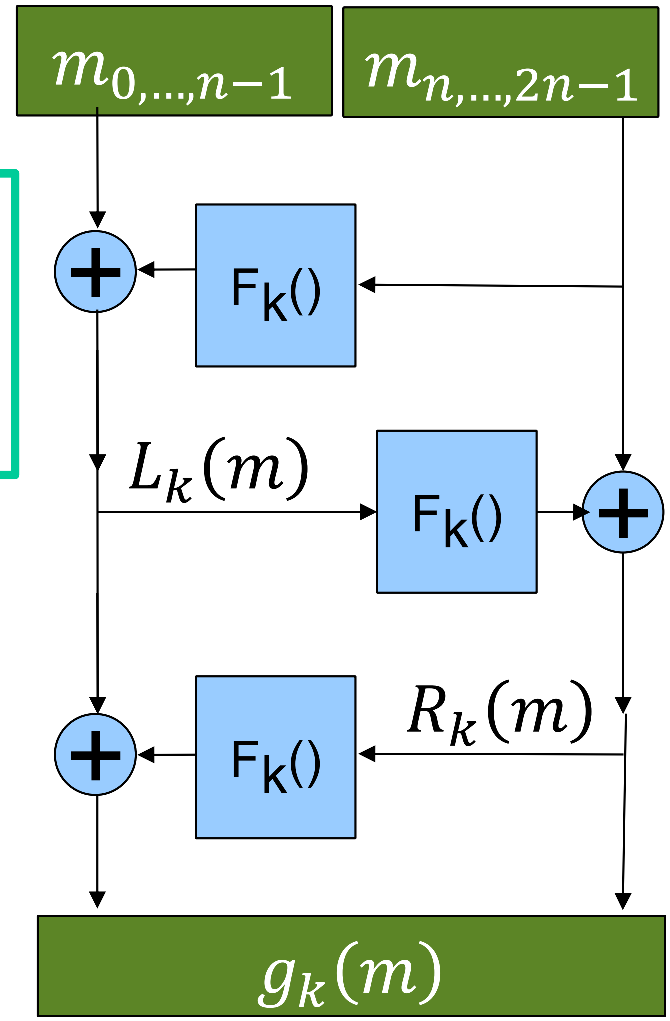
Constructing block-cipher, PRP

- Focus: constructions from a PRF $f_k(\cdot)$
  - PRFs seem easier to design (less restrictions)
- Before: ‘plain’ PRP $E_k(\ )$ (not a block cipher)
- Now: construct block cipher (invertible PRP) E_k, D_k
- Challenge: making it invertible…
- Solution: The Feistel Construction

The Feistel Block-cipher Construction

- Turn PRF F_k into a block cipher
  - Three 'rounds' suffice [LR88]

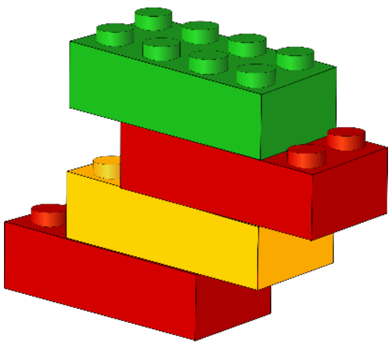
$$
\begin{aligned}
L_k(m) &= m_{0,\ldots,n-1} \oplus F_k(m_{n,\ldots,2n-1}) \\
R_k(m) &= F_k(L_k(m)) \oplus m_{n,\ldots,2n-1} \\
g_k(m) &= L_k(m) \oplus F_k(R_k(m)) \mathbin{+\!\!\!+} R_k(m)
\end{aligned}
$$



- Used in DES (but not in AES)
  - With 16 'rounds'

Crypto Building Blocks Principle

- Design and focus cryptanalysis efforts on few basic functions: simple, easy to test, replaceable
- Construct schemes from basic functions
  - Provably secure constructions: attack on scheme ➔ attack on function
  - Allows replacing broken/suspect functions
  - Allows upgrading to more secure/efficient function



- E.g., encryption from block cipher (or PRG/PRF/PRP)
  - Block-cipher, PRG,PRF,PRP: deterministic, stateless, FIL (Fixed-Input-Length)
  - Encryption: randomized/stateful, VIL (Variable-Input-Length)

We defined security for PRG, PRF and PRP. Block cipher too (informally).

But…

what about security of encryption??

A bit tricky, in fact.

Defining Secure Encryption

- Attacker capabilities:
  - Computational limitations ➔ PPT
  - Ciphertext only (CTO), Known / chosen plaintext attack (KPA/CPA), Chosen ciphertext (CCA)?
- What's a successful attack?
  - Key recovery ?
    - May be impossible yet weak cipher…
  - (Full) Message recovery?
    - What of partial exposure, e.g., m$\in${"Advance", "Retreat"}
  - <u>Prudent</u>: attacker 'wins' for <u>any</u> info on plaintext

Conservative Design Principle

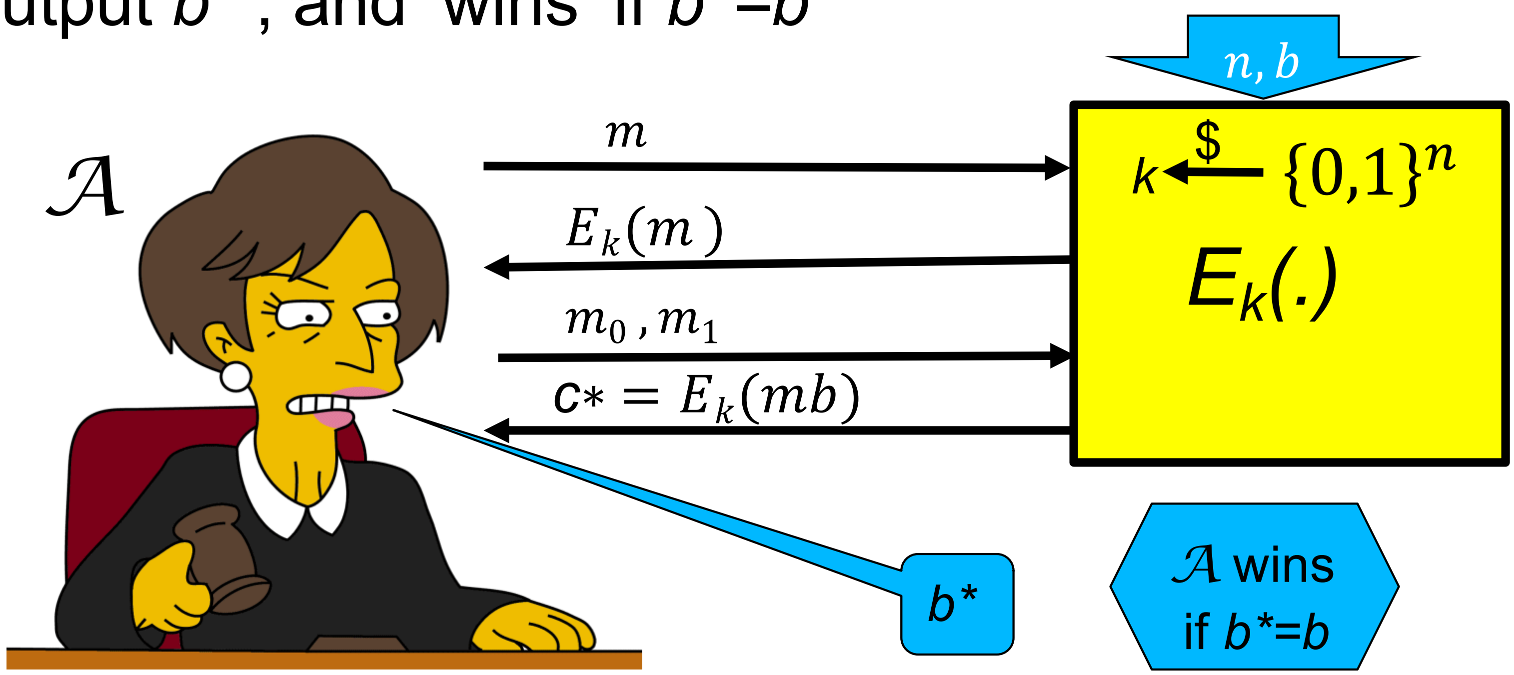
- When designing, evaluating a cryptosystem…
  - Consider most powerful attacker (CTO< KPA< CPA < CCA)
  - Be as general as possible – cover many applications
  - And `easiest’ attacker-success criteria
    - Not full message/key recovery!
  - Make it easy to use securely, hard to use insecurely!
- When designing, evaluating a system (that uses some cryptographic primitives)
  - Restrict attacker’s capabilities (e.g., avoid known/chosen plaintext)

Cryptanalysis Success Criteria for Encryption

- Learn anything at all about plaintext – how to define? Can we achieve it ?
  - Well-defined notion: ‘semantic security’ [crypto course]
- So an encryption scheme is secure if the attacker cannot learn anything about the plaintext that he did not know in advance.
- **Indistinguishability**: Eve ‘wins’ if she <u>distinguishes</u> between encryptions of (any) two messages
  - The attacker chooses these two messages.
  - We focus on indistinguishability for CPA attacker. In crypto course: equivalent to semantic security

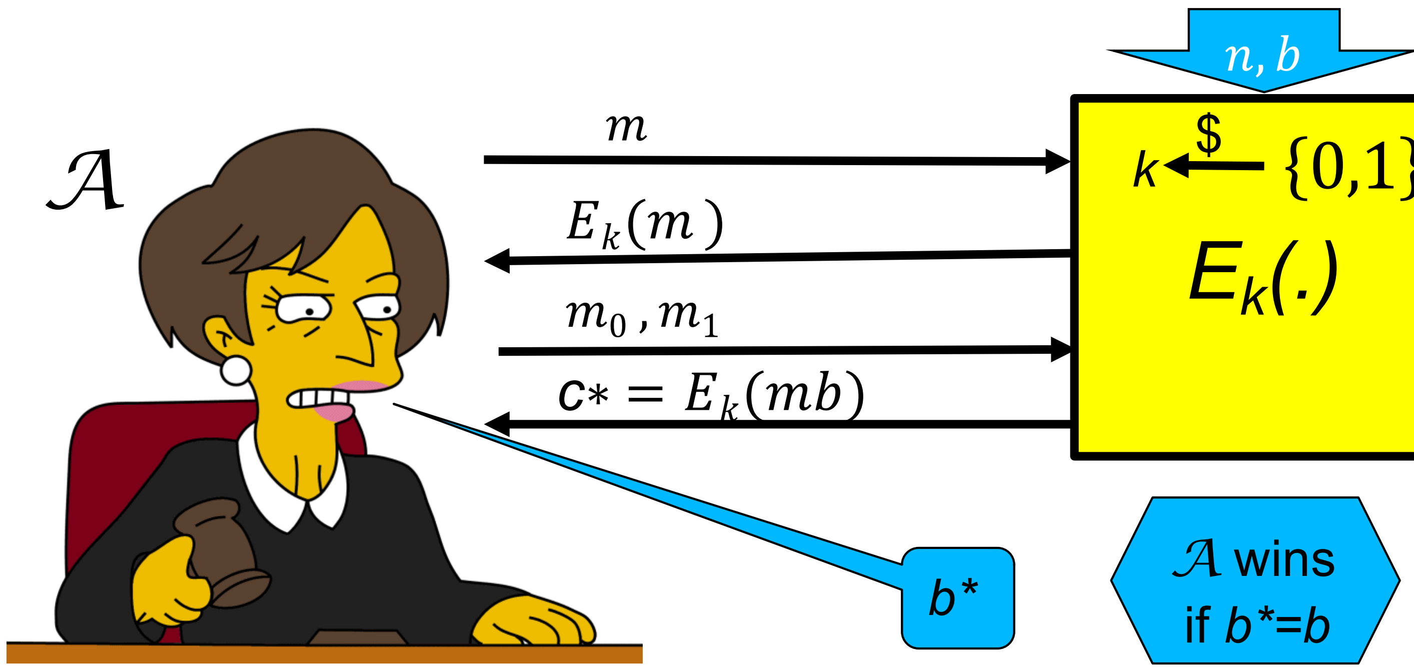
IND-CPA-Encryption Test (1st try)

- Flip coins to select random bit *b* and key *k*
- $\mathcal{A}$ (adversary) gives message *m*, receives $E_k(m)$
  - Repeat if desired (with different messages *m*)
  - Chosen Plaintext Attack
- $\mathcal{A}$ gives two messages (m_0,m_1), receives $c^*=E_k(m_b)$
- $\mathcal{A}$ output b^* , and 'wins' if $b^*=b$



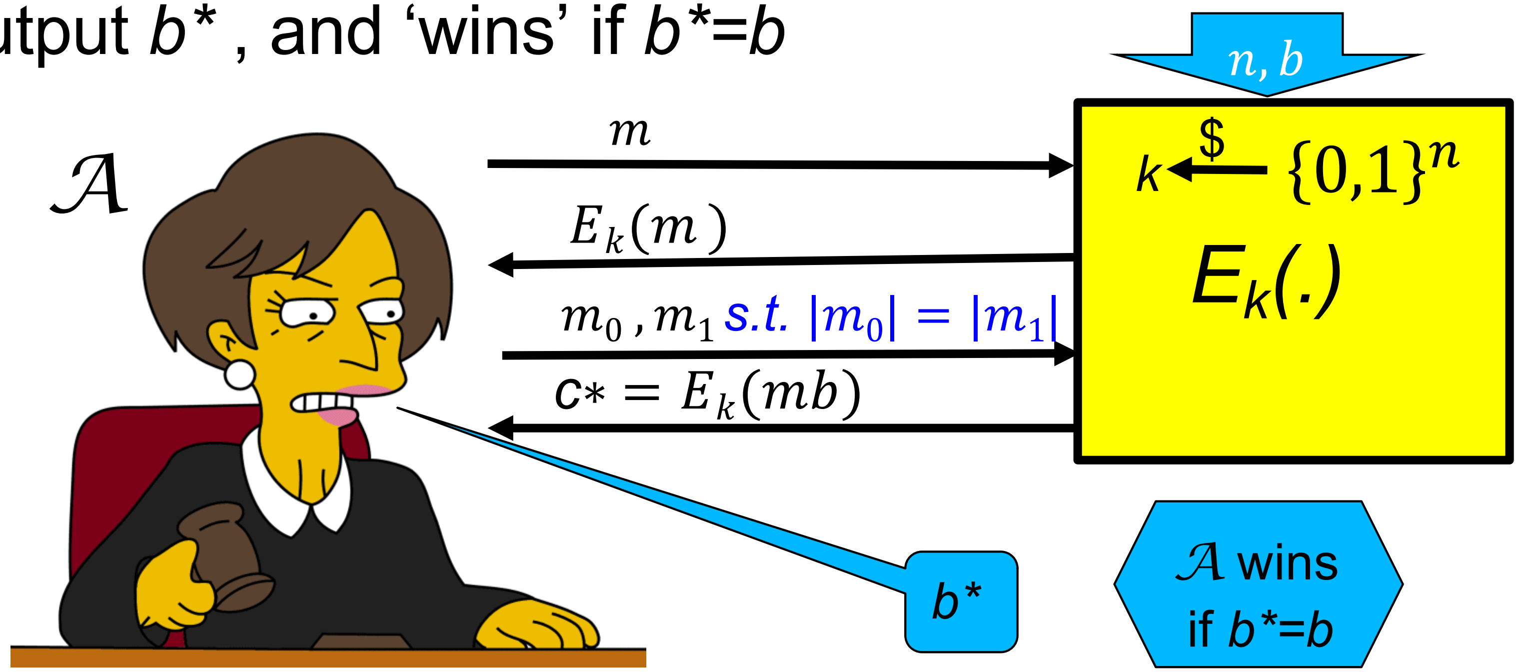
IND-CPA-Encryption Test (1st try): too easy

- This test is too easy!! The adversary can easily win!!
- How?
- Hint: messages can be arbitrary binary strings
  - Namely, $m, m_0, m_1 \in \{0,1\}$^*
  - **Solution:** let $m_0=0$, $m_1=11111111111111111$
  - If $c^*=E_k(m_b)$ is `short', output $b^*=0$; if 'long', output $b^*=1$



IND-CPA-Encryption Test (fixed)

- Flip coins to select random bit *b* and key *k*
- $\mathcal{A}$ (adversary) gives message *m*, receives $E_k(m)$
  - Repeat if desired (with another message)
  - Chosen Plaintext Attack
- $\mathcal{A}$ gives messages (m_0,m_1) s.t. $|m_0|=|m_1|$, receives $E_k(m_b)$
- $\mathcal{A}$ output b^* , and 'wins' if $b^*=b$



IND-CPA-Encryption Test (fixed)

- Or, as pseudo-code:

$T^{IND-CPA}_{\mathcal{A},\langle E,D\rangle}(b,n)$ {

$k \xleftarrow{\$} \{0,1\}^n$

$(m_0,m_1) \leftarrow \mathcal{A}^{E_k(\cdot)}(\text{`Choose'}, 1^n)$ s.t. $|m_0| = |m_1|$

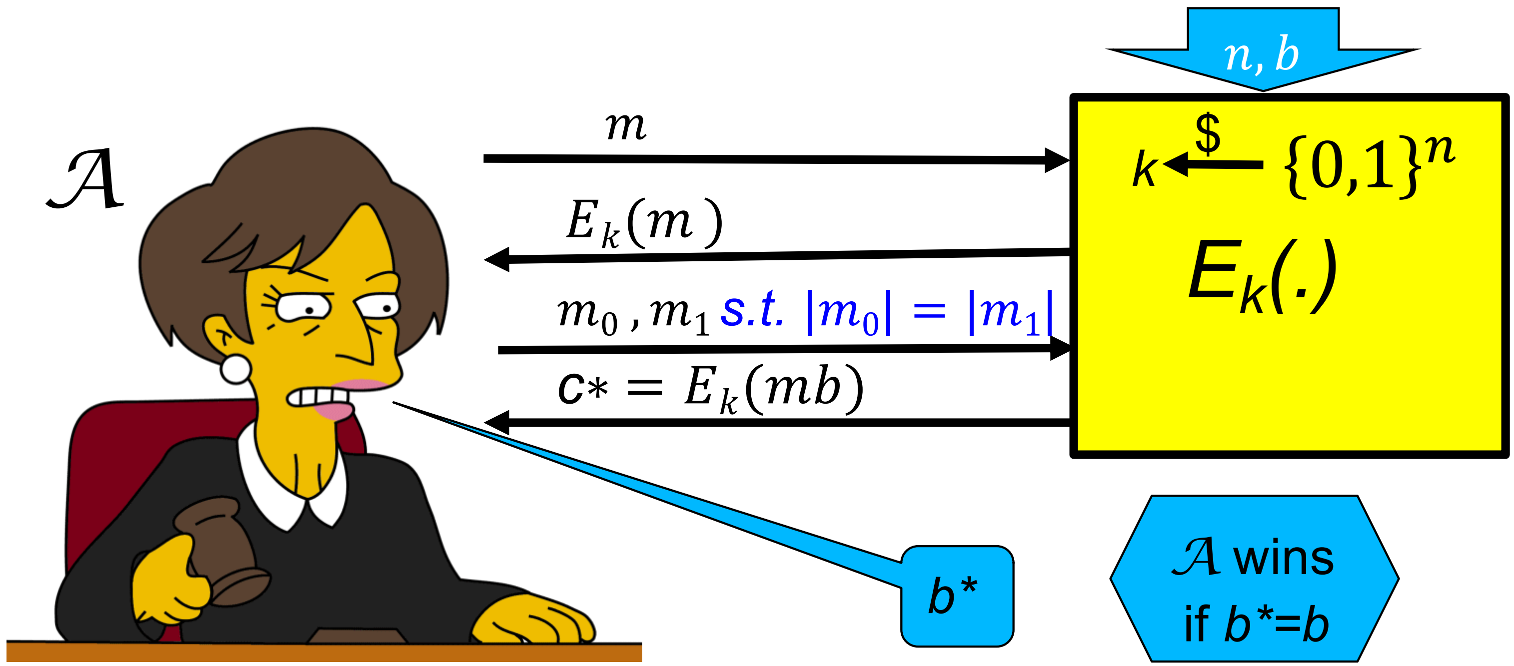
$c^* \leftarrow E_k(m_b)$

$b^* = \mathcal{A}^{E_k(\cdot)}(\text{`Guess'}, c^*)$

Return b^*

}

Oracle notation



Definition: IND-CPA Encryption

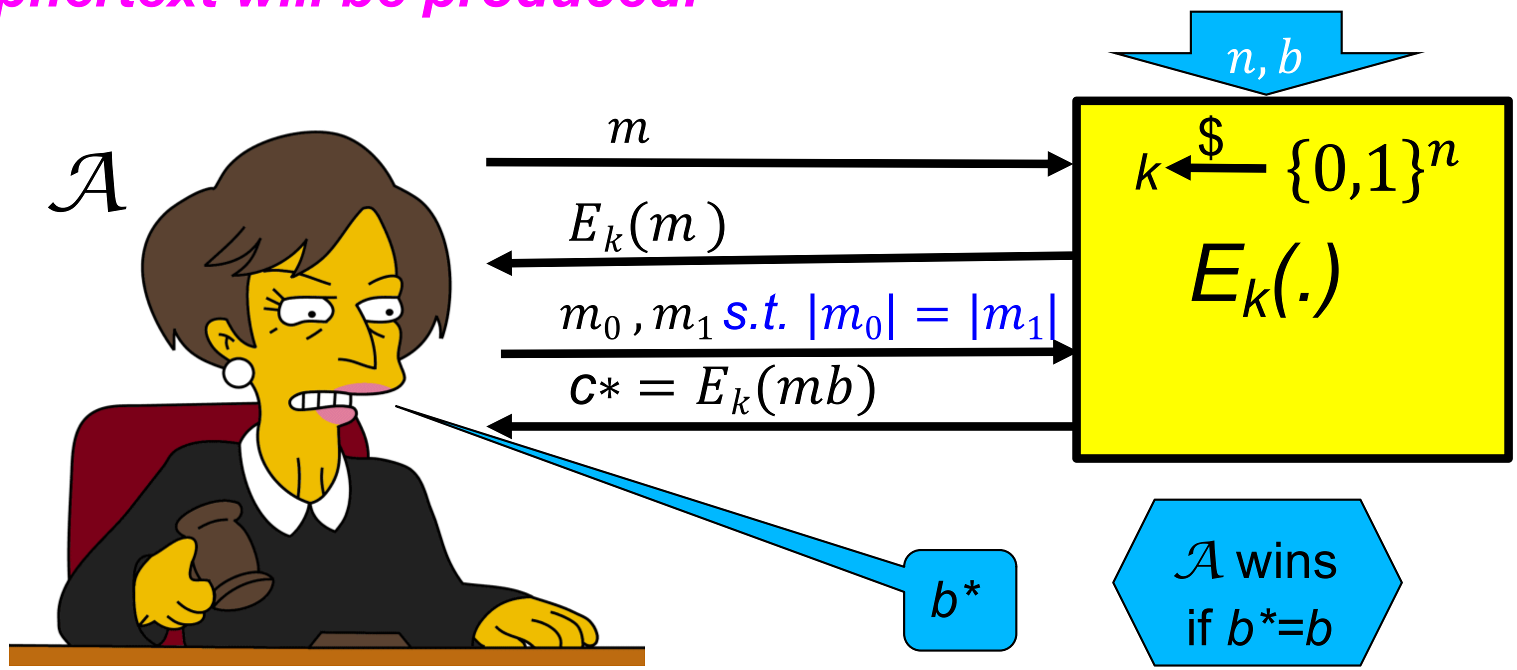
- Shared key cryptosystem (E, D) is **IND-CPA**, if every efficient adversary A has negligible advantage:

$$\varepsilon^{IND-CPA}_{\langle E,D\rangle,\mathcal{A}}(n) \equiv \Pr\left[T^{IND-CPA}_{\mathcal{A},\langle E,D\rangle}(1,n)=1\right] - \Pr\left[T^{IND-CPA}_{\mathcal{A},\langle E,D\rangle}(0,n)=1\right]$$

$$
\begin{array}{l}
T^{IND-CPA}_{\mathcal{A},\langle E,D\rangle}(b,n)\ \{ \\
\quad k \xleftarrow{\$} \{0,1\}^n \\
\quad (m_0, m_1) \leftarrow \mathcal{A}^{E_k(\cdot)}(\text{'Choose'}, 1^n) \text{ s.t. } |m_0| = |m_1| \\
\quad c^* \leftarrow E_k(m_b) \\
\quad b^* = \mathcal{A}^{E_k(\cdot)}(\text{'Guess'}, c^*) \\
\quad \text{Return } b^* \\
\}
\end{array}
$$

Can IND-CPA encryption be deterministic?

- No!! But why? Suppose $E_k(m)$ is deterministic…
- Assume messages are words.
- $\mathcal{A}$ can ask E_k to encrypt m_0 and m_1 and then check which one is equal to the challenge ciphertext → always wins!
- Conclusion: IND-CPA Encryption **must be randomized**
  - ***Even if you encrypt the same m over and over again, a new ciphertext will be produced.***



What's next?

Present a secure cryptosystem?

… provably secure w/o assumptions ?

Unlikely: Proof of security → $P \neq NP$

(similar argument to PRF)

Instead, let's build secure encryption from PRFs !
(I.e.: PRF is secure → encryption is IND-CPA)

Actually, we'll use **block cipher** (recall the PRF/PRP switching lemma) to build encryption schemes under what is called "Modes of operation."

Examples

- Let F be a PRF.
  - $E_k(m) = F_k(0^n)$ *xor* m
  - $E_k(m) = (r, F_k(r)$ *xor* $m)$ where r is a random string freshly generated for each message.

Encryption: Modes of Operation

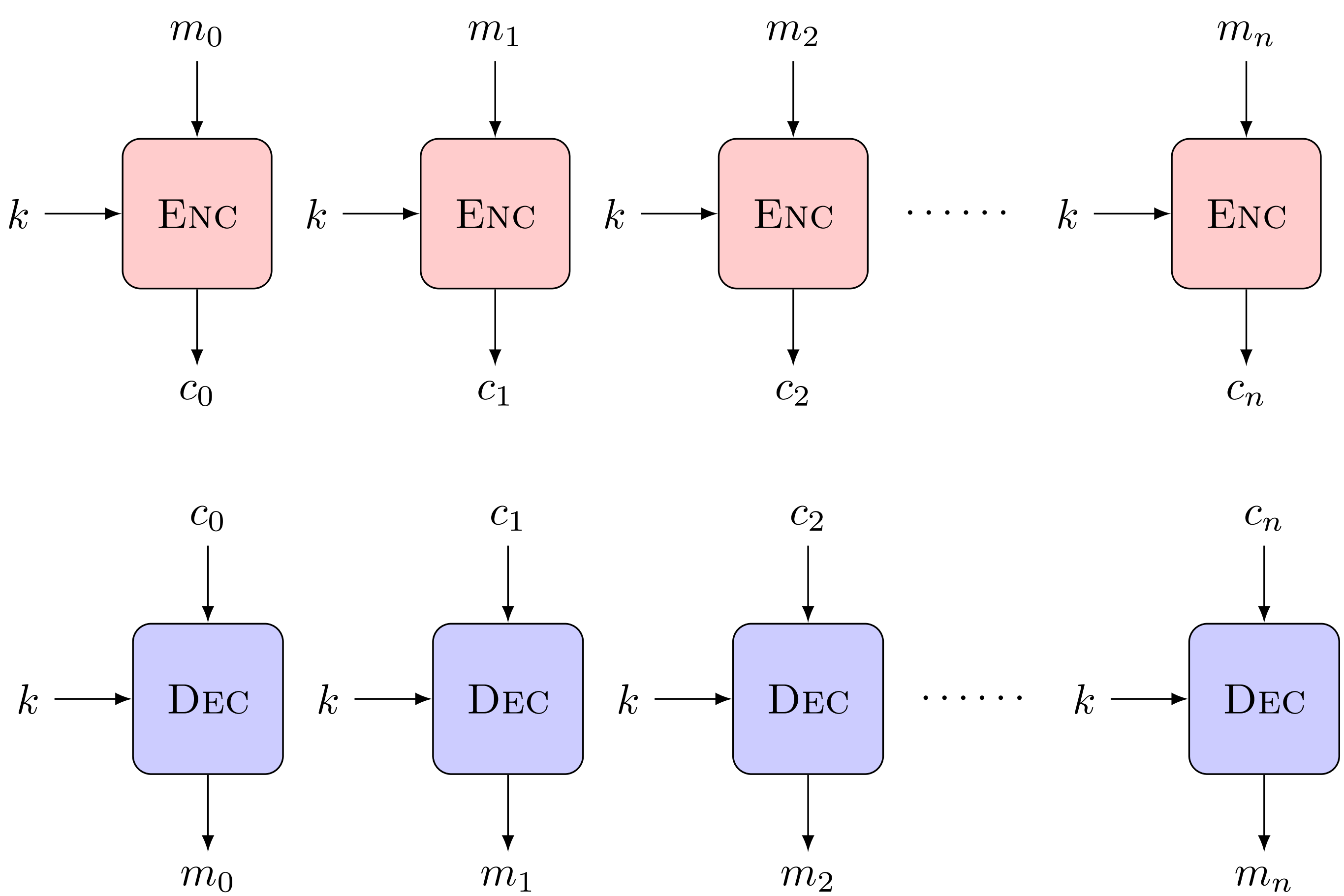
- `Modes of operation': use block cipher (PRP), to encrypt long (Variable Input Length, VIL) messages
- Randomize/add state for security
  - Often: use random or stateful *Initialization Vector (IV)*
- Use long keys
  - Better security (at least against exhaustive search)
- Assume plaintext is in blocks: $m_0||m_1||...$
  - An integer number of blocks, each block is n bits.

Encryption Modes of Operation

Mode	Encryption	Flip $c_i[j] \Rightarrow$	Properties
Electronic code book (ECB)	$c_i = E_k(m_i)$	Corrupt m_i	Deterministic (distinguishable)
Per-Block Random (PBR)	$r_i \xleftarrow{\$} \{0,1\}^n$, $c_i = (r_i, m_i \oplus E_k(r_i))$	Flip $m_i[j]$	Long ciphertext
Counter (CTR) [simplified]	$c_i = m_i \oplus E_k(i)$	Flip $m_i[j]$	Fast online, **stateful** (i)
Output Feedback (OFB)	$r_0 \xleftarrow{\$} \{0,1\}^n, r_i = E_k(r_{i-1})$, $c_0 \leftarrow r_0,\ c_i \leftarrow r_i \oplus m_i$	Flip $m_i[j]$	Fast online (precompute)
Cipher Feedback (CFB)	$c_0 \xleftarrow{\$} \{0,1\}^n$, $c_i \leftarrow m_i \oplus E_k(c_{i-1})$	Corrupt m_i, flip $m_{i+1}[j]$	Can decrypt in parallel
Cipher-Block Chaining (CBC)	$c_0 \xleftarrow{\$} \{0,1\}^n$, $c_i \leftarrow E_k(m_i \oplus c_{i-1})$	Flip $m_{i-1}[j]$, corrupt m_i	Can decrypt in parallel

Electronic Code Book mode (ECB) I

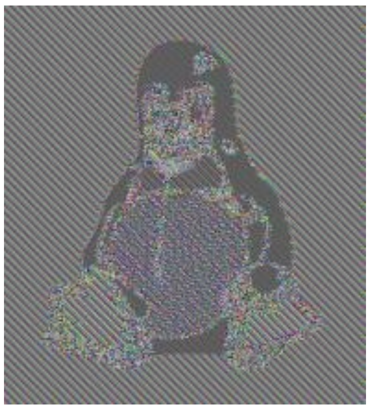
- Encryption $c_i=E_k(m_i)$, decryption $m_i=D_k(c_i)$
  - *Each m_i is n bit block and same for ci*



Electronic Code Book mode (ECB) II

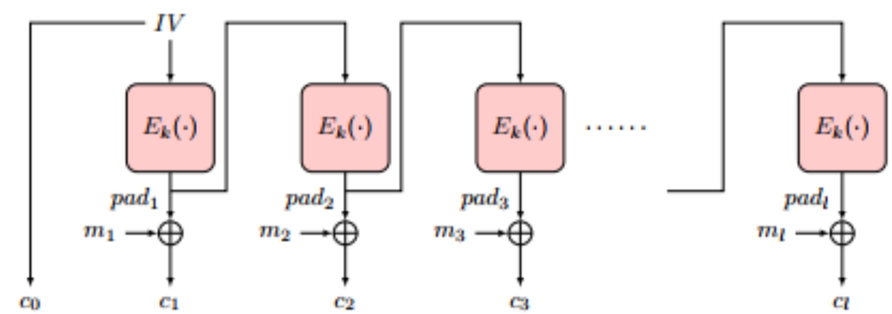
- Encryption $c_i\!=\!E_k(m_i)$, decryption $m_i\!=\!D_k(c_i)$

Insecure!! (do not use it!) Which of these is ECB encryption? Why?

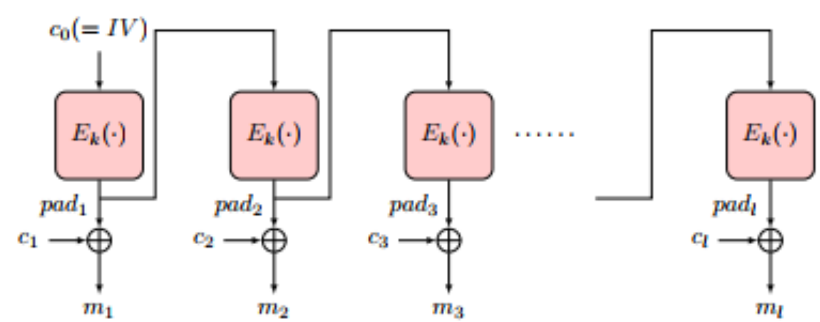


Output-Feedback (OFB) Mode

- Goal: encrypt long (multi-block) messages, with **less random bits**
- How? Use random bits only for first block (`initialization vector')
  - To encrypt next blocks of message, use output of previous block
  - Namely, a **block-by-block stream cipher**
- Encryption: $pad_0 \leftarrow IV$,
  $pad_i \leftarrow E_k(pad_{i-1})$,
  $c_0 \leftarrow pad_0,\ c_i \leftarrow pad_i \oplus m_i$



- Decryption:
  $pad_0 \leftarrow c_0$,
  $pad_i \leftarrow E_k(p_{i-1})$,
  $m_i \leftarrow pad_i \oplus c_i$

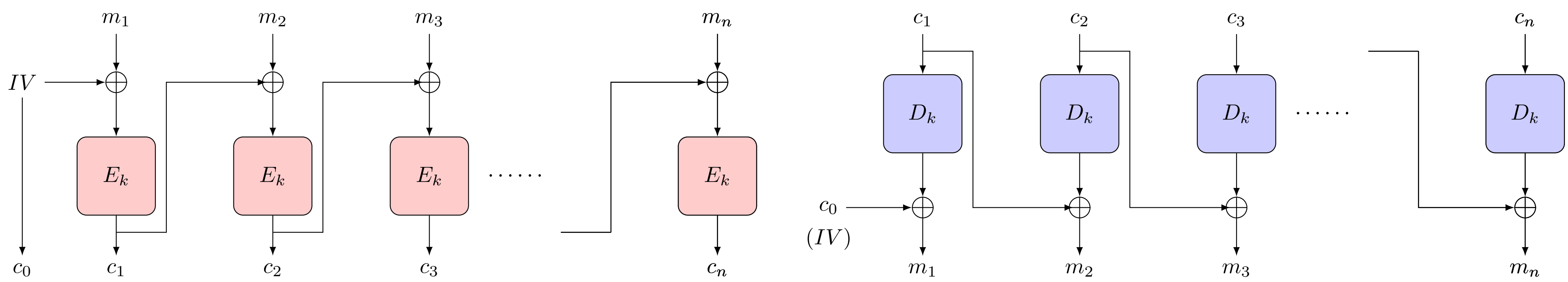


Output-Feedback (OFB) Mode

- **Offline pad computation:** compute pad in advance
  - Online computation: only (parallelizable) XOR !
- Bit errors are bitwise **localized**
  - Corrupting a one bit in the ciphertext corrupts only one bit in the plaintext.

Cipher Block Chaining (CBC) Mode

- Random first block c_0 (`initialization vector', *IV*)
- $i>0$: $c_i = E_k(c_{i-1} \oplus m_i)$, $m_i = c_{i-1} \oplus D_k(c_i)$



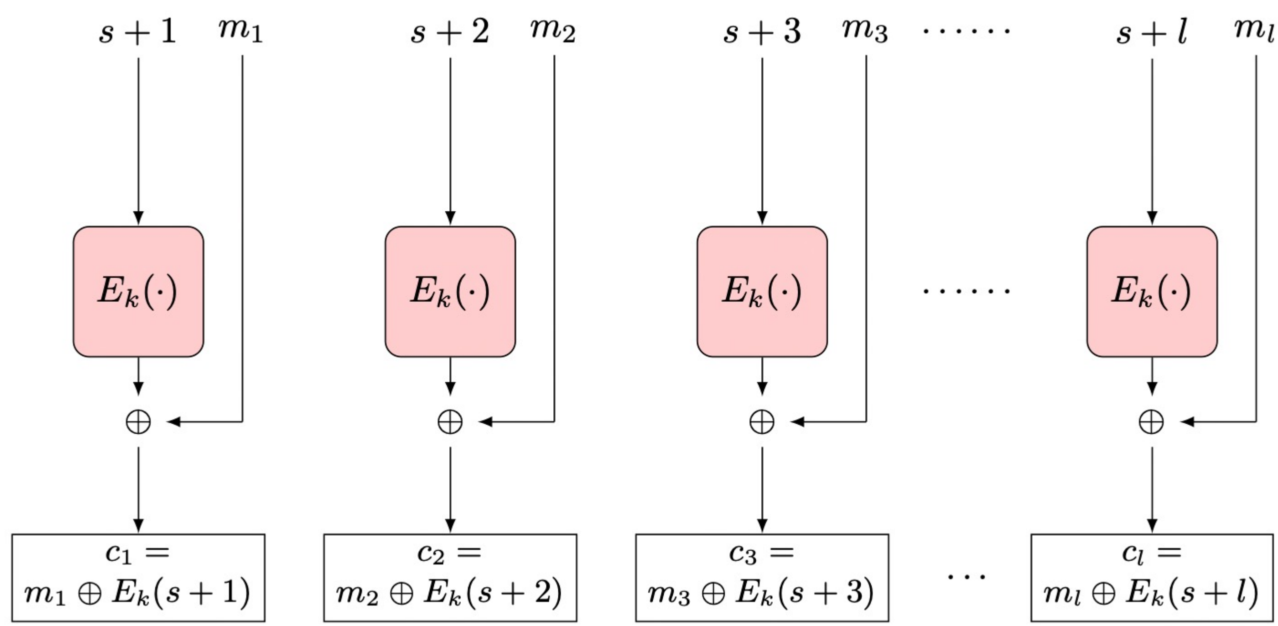
- Parallel decryption
  - But no offline precomputing
  - How about encryption? Sequential (it is a chain!)
- Error propagation:
  - flip bit in $c[i]$ ➔ flip bit in $m[i+1]$ and corrupt $m[i]$

Security of CBC mode

- Theorem: If block-cipher E is a (strong) <u>pseudo-random permutation</u> ➔CBC is IND-CPA-secure encryption
- Proof: omitted (crypto course ☺)
- Observation: CBC is Not IND-CCA-Secure
  - CCA (Chosen ciphertext attack), intuitively: attacker can choose ciphertext and get its decryption, except for the \`challenge ciphertext’
  - Definition, details: crypto course
  - Exercise: show CBC is Not IND-CCA-Secure
  - Other variants of CBC exists that are CCA secure.

Counter (CTR) Mode

- Random counter (or `initialization vector', *IV, or s*)
  - $i>0$: $c_i = E_k(s+i) \oplus m_i$
  - $m_i = E_k(s+i) \oplus c_i$
- Parallel encryption and decryption with offline precomputing



- If a PRF is used for the PRP (for E_k), then it is CPA (provably secure).
- Error propagation:
  - flip bit in c_i ➔ flip bit in m_i

Covered Material From the Textbook

- Sections 2.6, 2.7, and 2.8, excluding:
  - 2.7.3
  - PBR from 2.8.2,
  - “Encode-then-Encrypt considered harmful.” from 2.8.3
  - 2.8.4,
  - 2.8.6.

Thank You!