Cryptography II: Symmetric Ciphers

CSE 565: Fall 2024
Computer Security

Xiangyu Guo (xiangyug@buffalo.edu)

University at Buffalo

Announcement

- Please sign-up at course Piazza.
- Reminder of Quiz 0 (**Due 09/19**).

Review of Last Lecture

- Crypto basics
- Core application: Secure communication
  - Establish shared key: PKC
  - Transmitting msg with shared sec key: symm encryption
- Classical symm ciphers
  - Caesar; Substitution; Transposition; How they fail.
  - Modern ciphers: Combinations of the two. [C. Shannon]
- Recap on Probability
  - Uniform random var; Birthday Paradox.
- Recap on Algorithm
  - Big-O notation; Randomized Alg.

Today's topic

- Stream Ciphers
  - One-Time Pad (OTP)
  - Pseudorandom Generator (PRG)
  - Attacks on stream ciphers
- Block Ciphers
  - Design principle
  - ~~DES~~
  - ~~AES~~
  - ~~Usage & Attacks~~

Stream Ciphers

Symmetric Ciphers

- **Def**: A symm. cipher defined over $(\mathscr{K}, \mathscr{M}, \mathscr{C})$ is a pair of efficient algs (E, D) where
  - Enc. alg. $E : \mathscr{K} \times \mathscr{M} \mapsto \mathscr{C}$: Enc(Key, Ptext)=Ctext
  - Dec. alg. $D : \mathscr{K} \times \mathscr{C} \mapsto \mathscr{M}$: Dec(Key, Ctext)=Ptext
  - D(K, E(K, Ptext)) = Ptext
- E is often randomized. D is always deterministic.

One-Time Pad

- **Def**: An one-time pad (OTP) cipher (E, D) over $(\mathscr{K}, \mathscr{M}, \mathscr{C})$
  - $\mathscr{M} = \mathscr{C} = \mathscr{K} = \{0,1\}^n$
    - Key is an *uniform random* bit string as long as the message
  - Enc: $E(k, m) = k \oplus m$
  - Dec: $D(k, c) = k \oplus c$

- First proposed by F. Miller [1882], XOR version reinvented by G. Vernam [1917]
- Security from "One-time-ness" recognized only later.

One-Time Pad

Plaintext	**0101 | 1011 | 1000**
	⊕
Key (Pad)	**1100 | 1110 | 1011**
	=
Ciphertext	**1001 | 0101 | 0011**

Information Theoretic Security

- **Def**: A cipher (E, D) over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ has ***perfect secrecy*** if for <u>any</u> two *same-length plaintext* msgs $\forall m_0, m_1 \in \mathcal{M}$ $(|m_0| = |m_1|)$ and <u>any</u> ciphertext $\forall c \in \mathcal{C}$, we have
$$\Pr_{k \sim \mathcal{K}}[E(k, m_0) = c] = \Pr_{k \sim \mathcal{K}}[E(k, m_1) = c]$$
  - Basically, given only ciphertext, there's no way to tell which message (among m_0 and m_1) are encrypted.
- Strongest possible. Remain secure even if the attacker has, e.g., a quantum computer.

One-Time Pad is Secure (?)

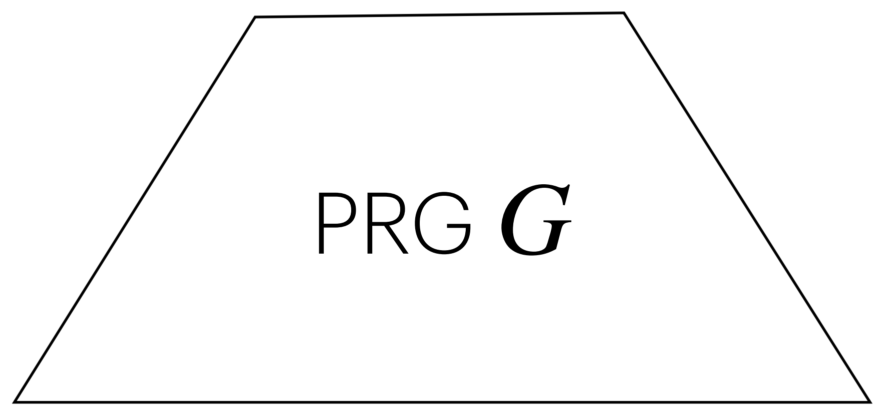
- **Thm** [C. Shannon]: OTP has perfect secrecy.
- So why is OTP not used widely in practice?
- **Fact**: perfect secrecy $\implies |\mathscr{K}| \geq |\mathscr{M}|$
  - i.e., perfect secrecy $\implies$ key-length $\geq$ msg-length
  - Not practical: How to send the key (securely) to the other party?
    - We are back at the origin: sending n-bit string securely.

Make OTP Practical

- Idea: Replace the random key with a "pseudorandom" key
- **Def**: A Pseudorandom Generator (**PRG**) is a function $G : \{0,1\}^s \mapsto \{0,1\}^n$ where
  - $n \gg s$, the *seed length*
  - G can be *efficiently* computed by a *deterministic* algorithm
- **A stream cipher is almost just a PRG + OTP.**

Make OTP Practical

s-bit Key (seed) **0101110**



n-bit Pad **1100 | 1110 | ...... | 1011**

$\oplus$

n-bit Plaintext **0101 | 1011 | ...... | 1000**

||

n-bit Ciphertext **1001 | 0101 | ...... | 0011**

Make OTP Practical

- But PRG-based stream cipher does *not* have perfect secrecy!
  - Security depend on specific PRG
    - Intuitively, a good PRG's output should "look just like" a truly random n-bit string.
    - Seems impossible (?): $|\{G(k) : k \in \{0,1\}^s| = 2^s \ll 2^n$
  - Need a new definition of security.

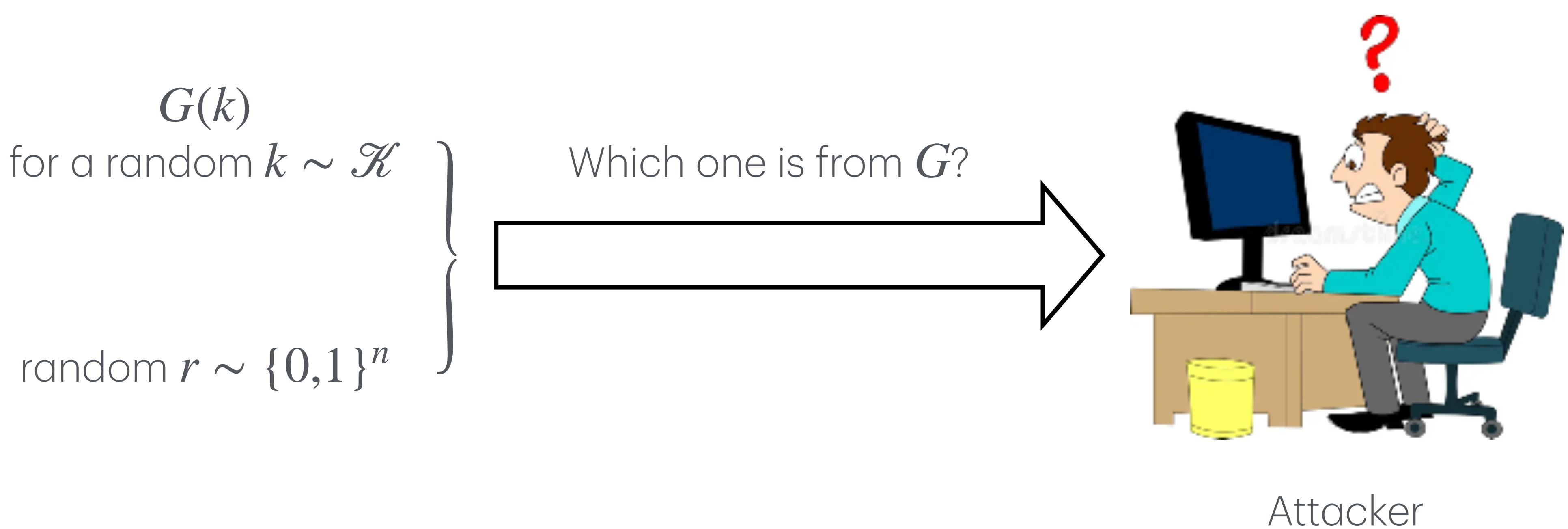
Computational Security

- A realworld attacker / adversary is **not** all-powerful
  - Finite life / computing resource
    - The attacker can only run *polynomial-time* algorithms.
  - Can be lucky, but not too lucky:
    - The attacker can do better (e.g., succeed with higher probability) than a trivial random guess, but only by a *negligible* margin.
    - "Negligible": $< 1/\text{poly}(n)$

Pseudorandom Generator

- A PRG is secure if a *computationally-bounded* attacker cannot *distinguish* its output from a truly random string.

- Specifically, the attacker succeed with prob. $< 1/2 +$ negligible, i.e., not much better than random guess.



Pseudorandom Generator

- A concrete PRG example?
  - No **provably** secure PRG known: this would imply P $\neq$ NP
  - Heuristic candidates:
    - ~~RC4~~
    - Salsa20
    - AES (CTR mode)

Attacks on Stream Ciphers

Two-Time Pad is Insecure

Never use stream cipher key more than once

$C_1 = m_1 \oplus \text{PRG}(k)$

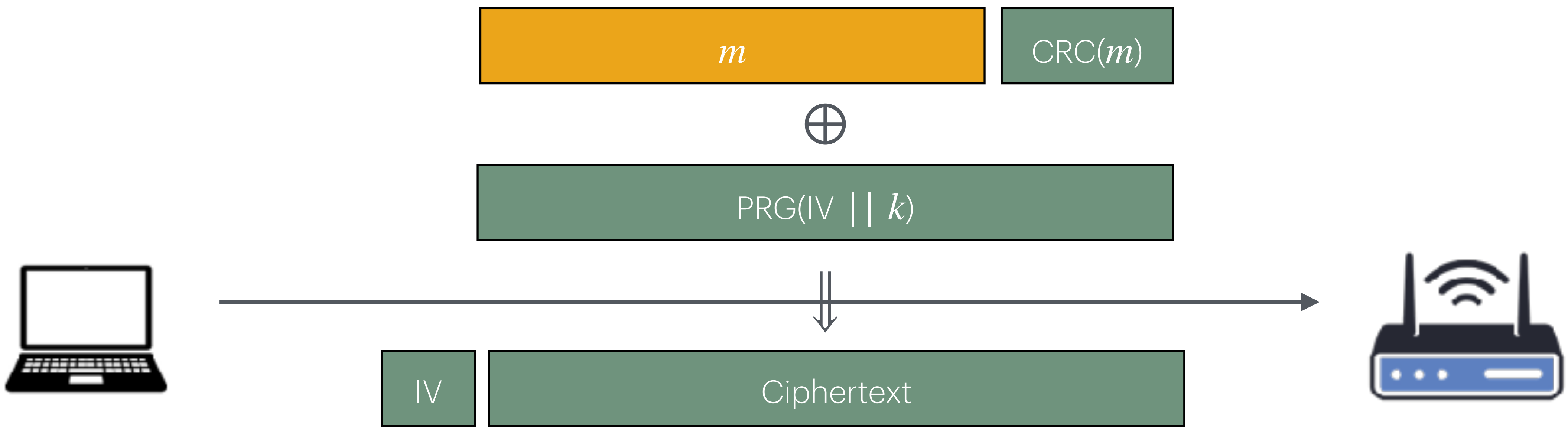
$C_2 = m_2 \oplus \text{PRG}(k)$

Attacker: $C_1 + C_2 \longrightarrow m_1 \oplus m_2$

You can recover m_1 and m_2 from $m_1 \oplus m_2$ if there's enough redundancy in the plaintext: e.g. English, ASCII encoding.

Example: 802.11b WEP

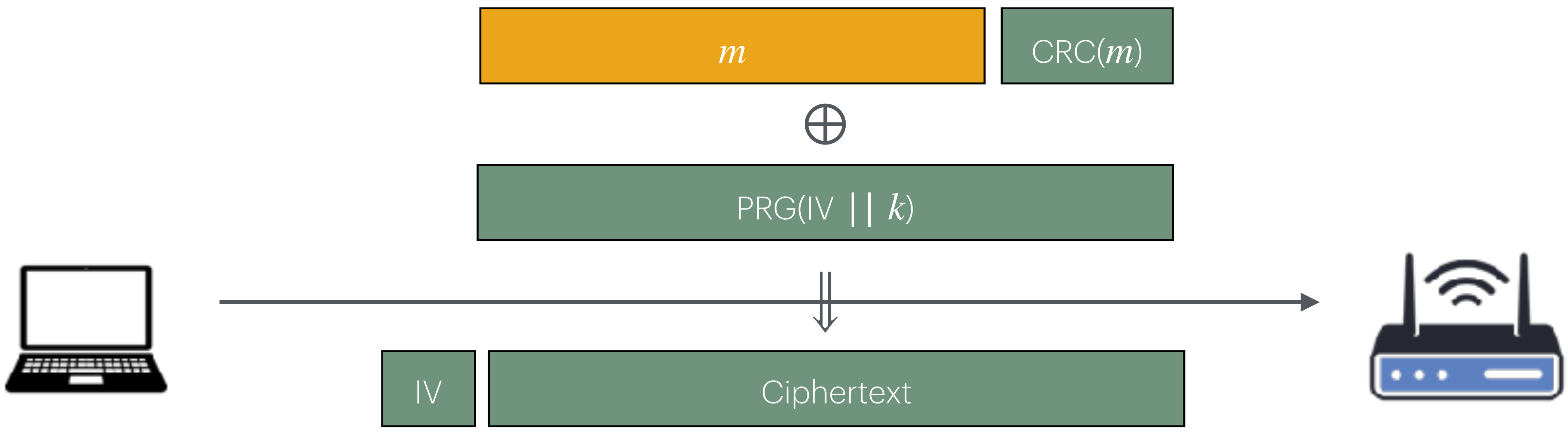
2-pad



- IV: only 24 bits long
  - Repeated IV $\Longrightarrow$ Repeated Pad after $2^{24} \approx 16M$ frames
  - On some 802.11 cards: IV resets to 0 after reboot.

Example: 802.11b WEP

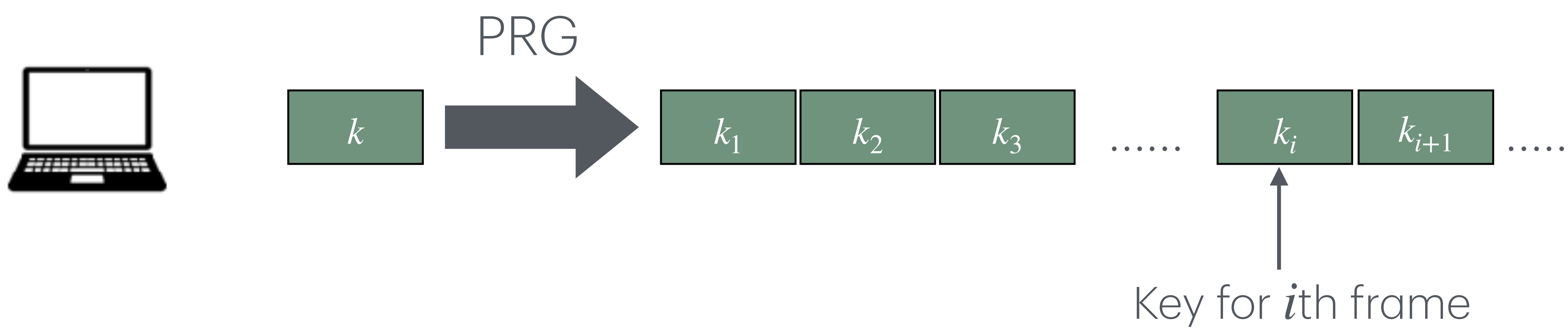
Related keys



- The PRG input is related: IV = i for the ith frame
- Not good for the **RC4** PRG used in WEP:
  - Recover key after 1M frames [Fluhrer, Mantin and Shamir 2001]
  - Now can be done with <100K frames.

Example: 802.11b WEP

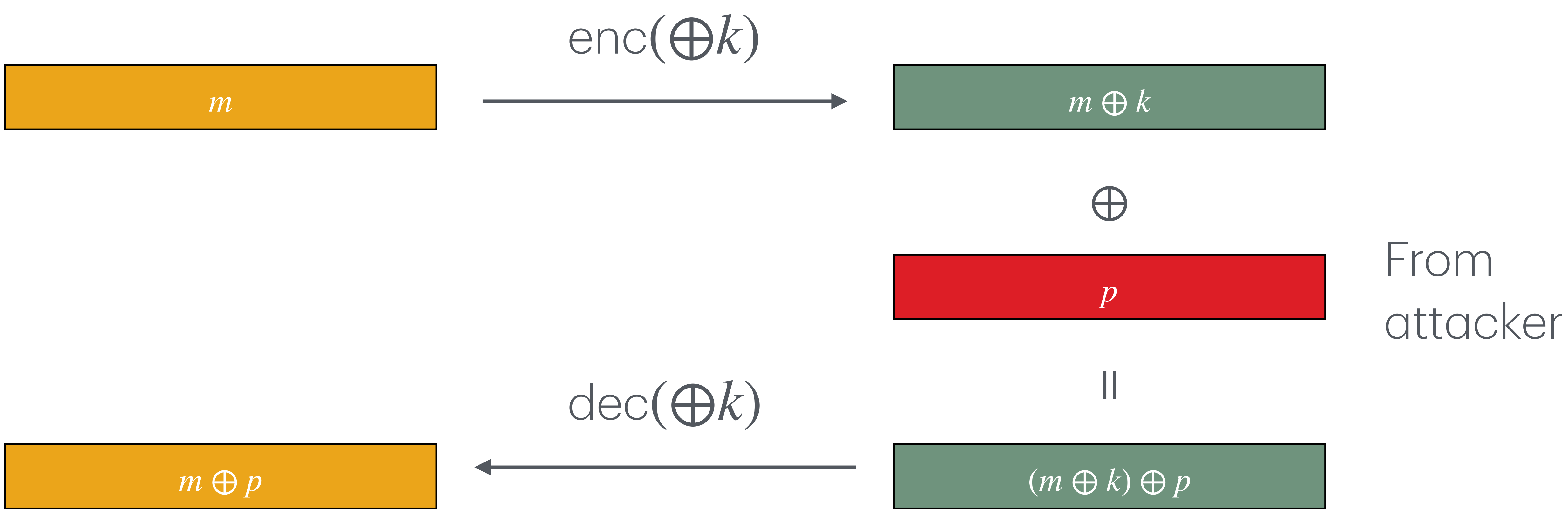
What could be done better?



- Use one same key to generate the pad stream for all frames.
  - Now each frame has a pseudorandom key
  - Change key for each session.
- Better solution: use stronger encryption method (e.g. WPA2)

Attack on Integrity

OTP is Malleable

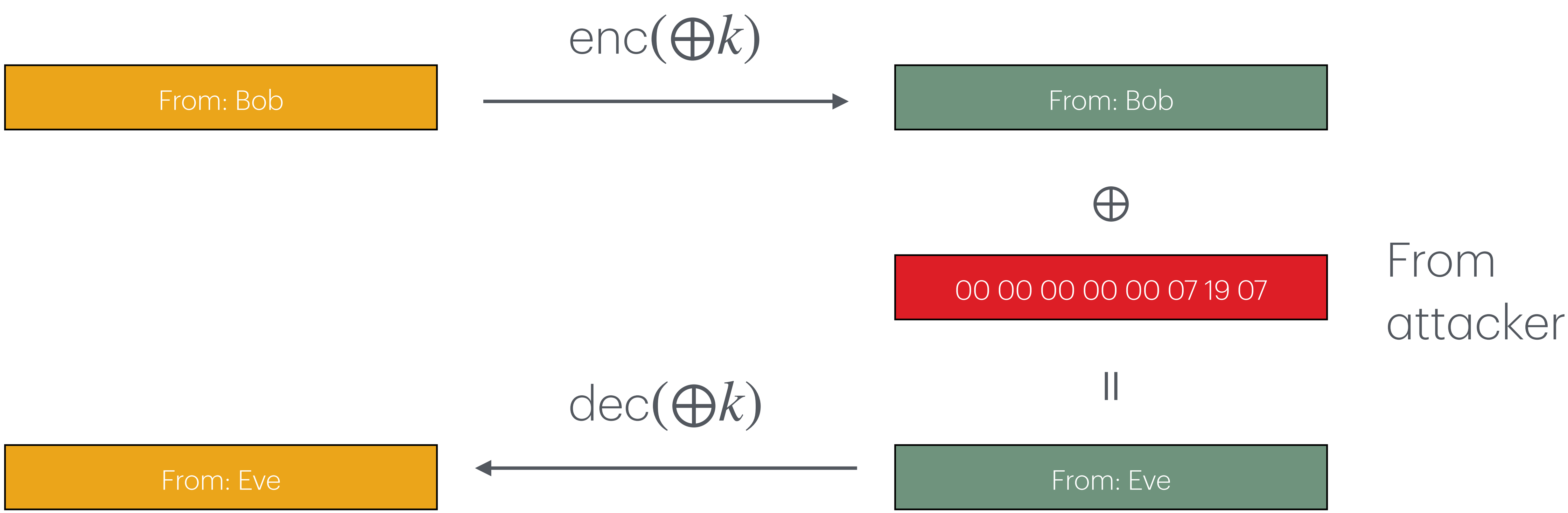


Attacker can modify the plaintext without decrypting it

- Modification **undetected**.
- **Predictable** impact on plaintext.

Attack on Integrity

OTP is Malleable



- Bob: 42 6F 62 (in ASCII)
- Eve: 45 76 65 (in ASCII)
- $p =$ Bob $\oplus$ Eve $=$ 07 19 07

Block Ciphers

Review: Simple Substitution Doesn't Work

- A large space of keys is not enough
- Mono-alphabetic
  - The same plaintext letters are always replaced by the same ciphertext letters
- Doesn't hide statistical properties of plaintext.
- Doesn't hide relationships in plaintext
- Natural languages are very redundant

Make it Harder?

- Hide statistical properties
  - Encrypt “e” with 12 different symbols, “t” with 9 different symbols, etc.
- Poly-alphbetic cipher
  - Use different substitutions
- **Transposition** (permutation)
  - Scramble order of units; reorder units of plaintext

Transposition Cipher

- Scrambling the character order by row-column transposition

1. Tile the plaintext *"MY+COOL+CIPHER+IS+SIMPLE"* in row direction.

2. Read ciphertext in column direction. The columns are ordered based on the secret key.

Key:

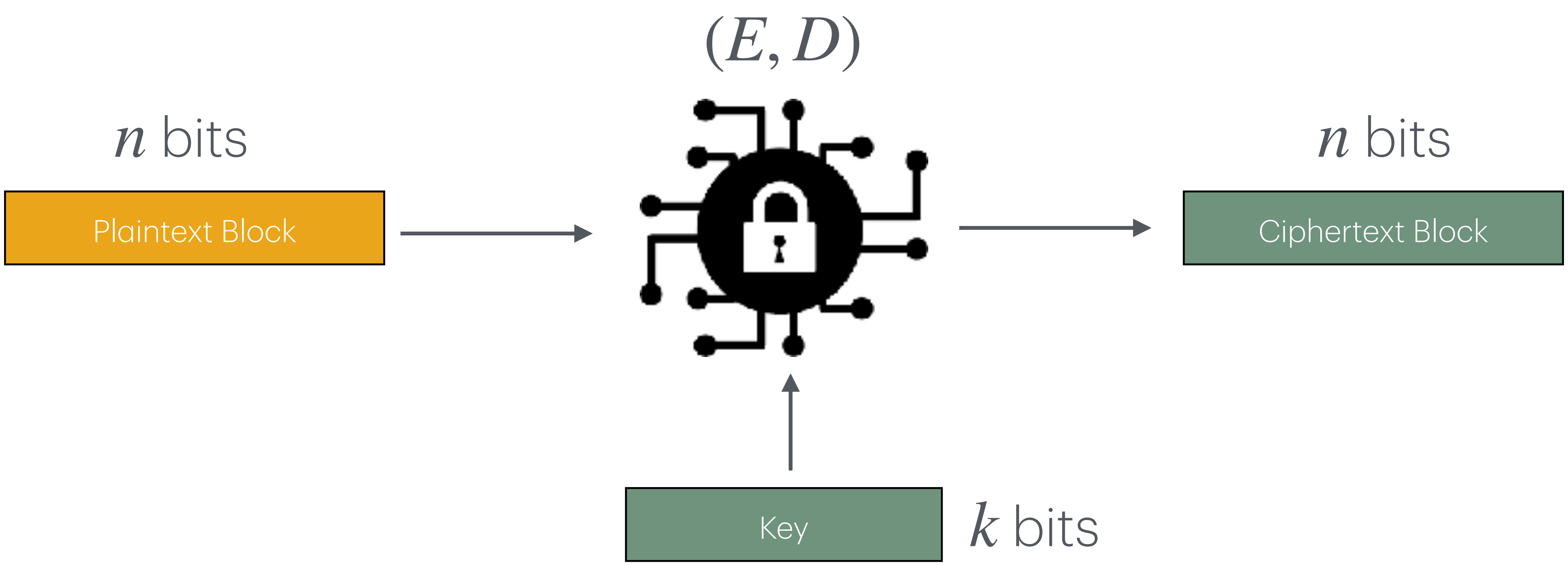
3	1	4	2	5
M	*Y*	+	*C*	*O*
O	*L*	+	*C*	*I*
P	*H*	*E*	*R*	+
I	*S*	+	*S*	*I*
M	*P*	*L*	*E*	

Ciphertext: *YLHSPCCRSEMOPIM++E+LOI+I*

From Classical to Modern Cipher

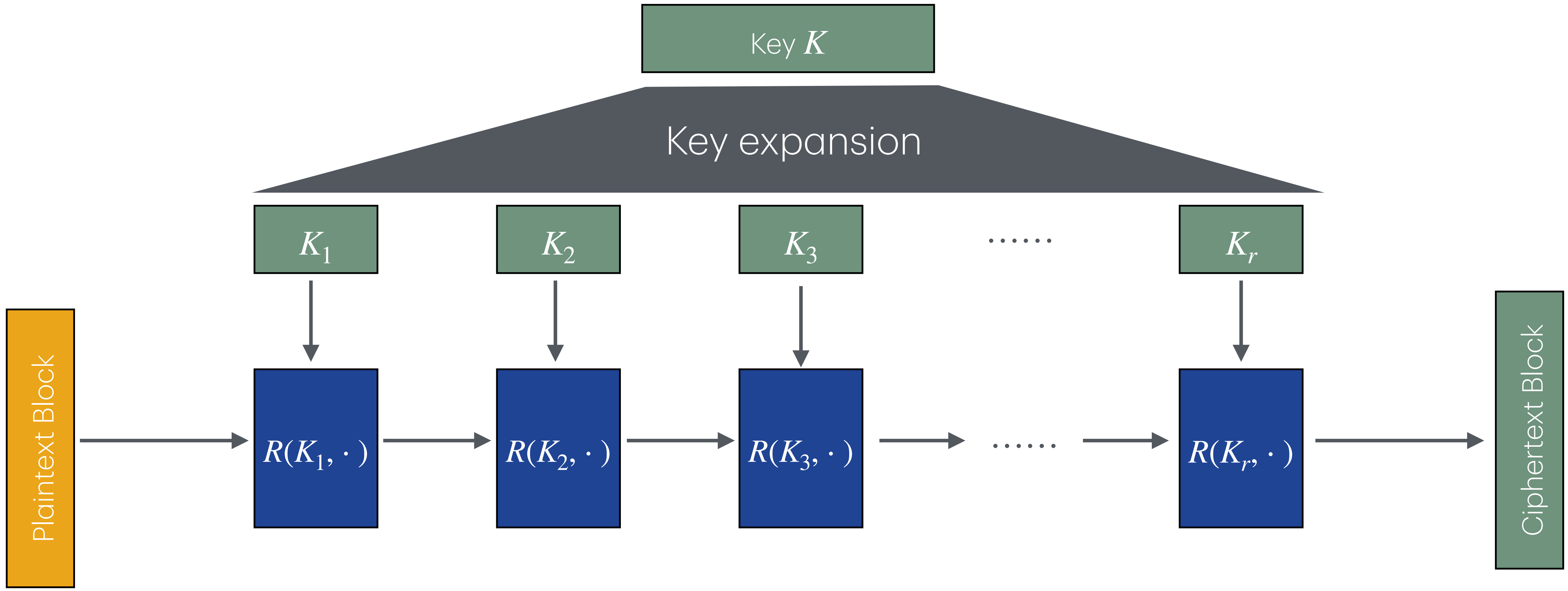
- Modern block ciphers are essentially combination of substitution (a.k.a. *"S-Box"*) and transposition (permutation, a.k.a. *"P-Box"*)
  - Combining multiple different "transformations" is more secure
    - *A Mathematical Theory of Cryptography,* Claude Shannon, 1945
  - [Shannon'45] two fundamental principles for statistical security
    - Confusion: produced by substitution
    - Diffusion: produced by transposition

What is a block cipher?



- Canonical examples:
  - ~~DES: $n = 64$ bits, $k = 56$ bits~~
  - 3DES: $n = 64$ bits, $k = 168$ bits
  - AES: $n = 128$ bits, $k = 128, 192, 256$ bits

What is a block cipher?



- $R(K_i, m)$: **round function**
  - DES: round $r = 16$, 3DES: round $r = 48$
  - AES: round $r = 10/12/14$

Block Cipher Design Principles

- **Round function**: Confusion-diffusion paradigm
    1. Split a block into small chunks
    2. Define a substitution on each chunk separately (confusion)
    3. Mix outputs from different chunks by rearranging bits (diffusion)
    4. Repeat to strengthen the result

Substitution-Permutation Network (SPN)

Round Function Examples

One SPN *round*:

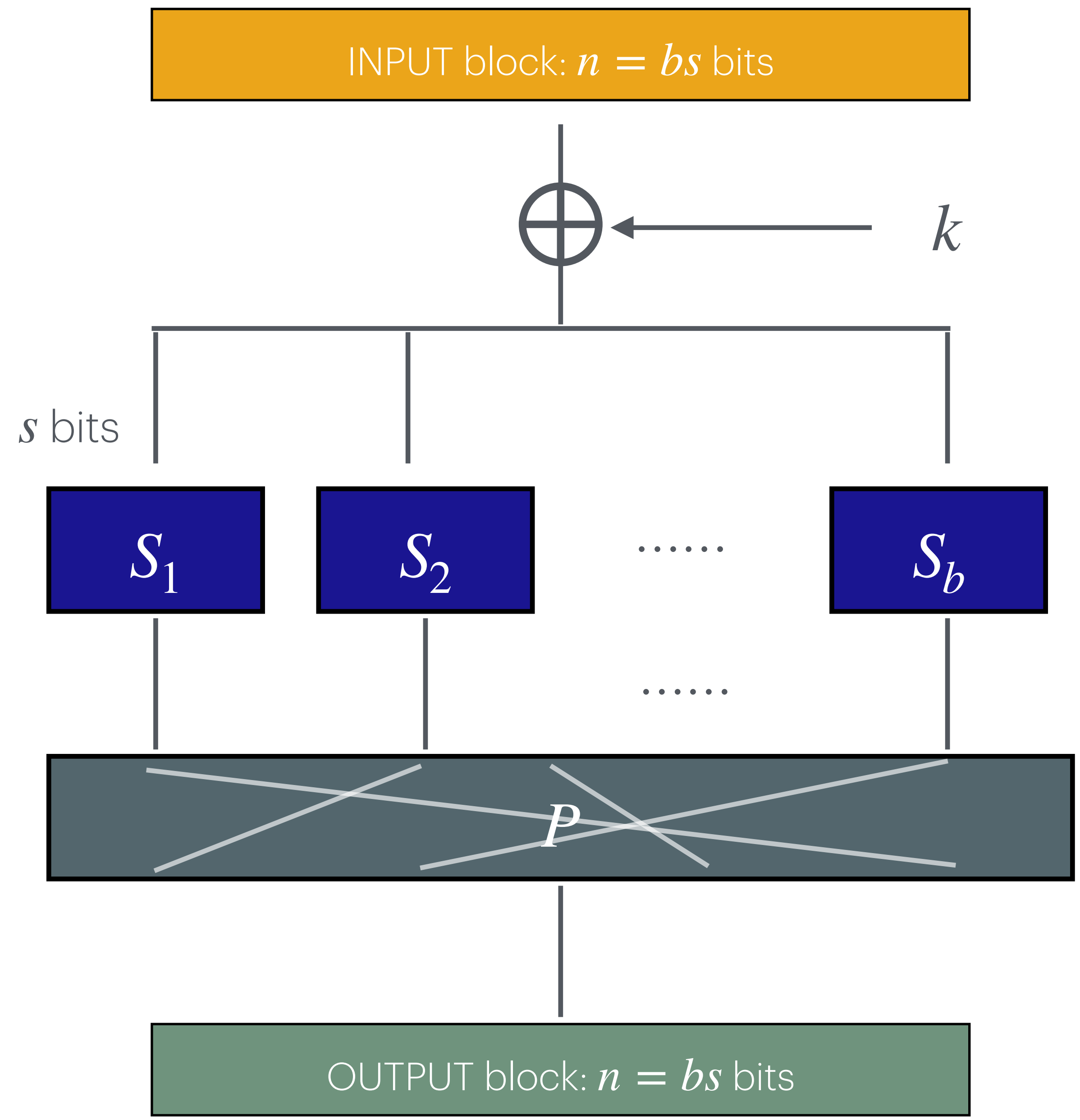
1. Split a block into b chunks

2. S-Box: substitute each block with another block

3. P-Box: Mix outputs from different chunks by permuting bits

- Every step is **reversible**.

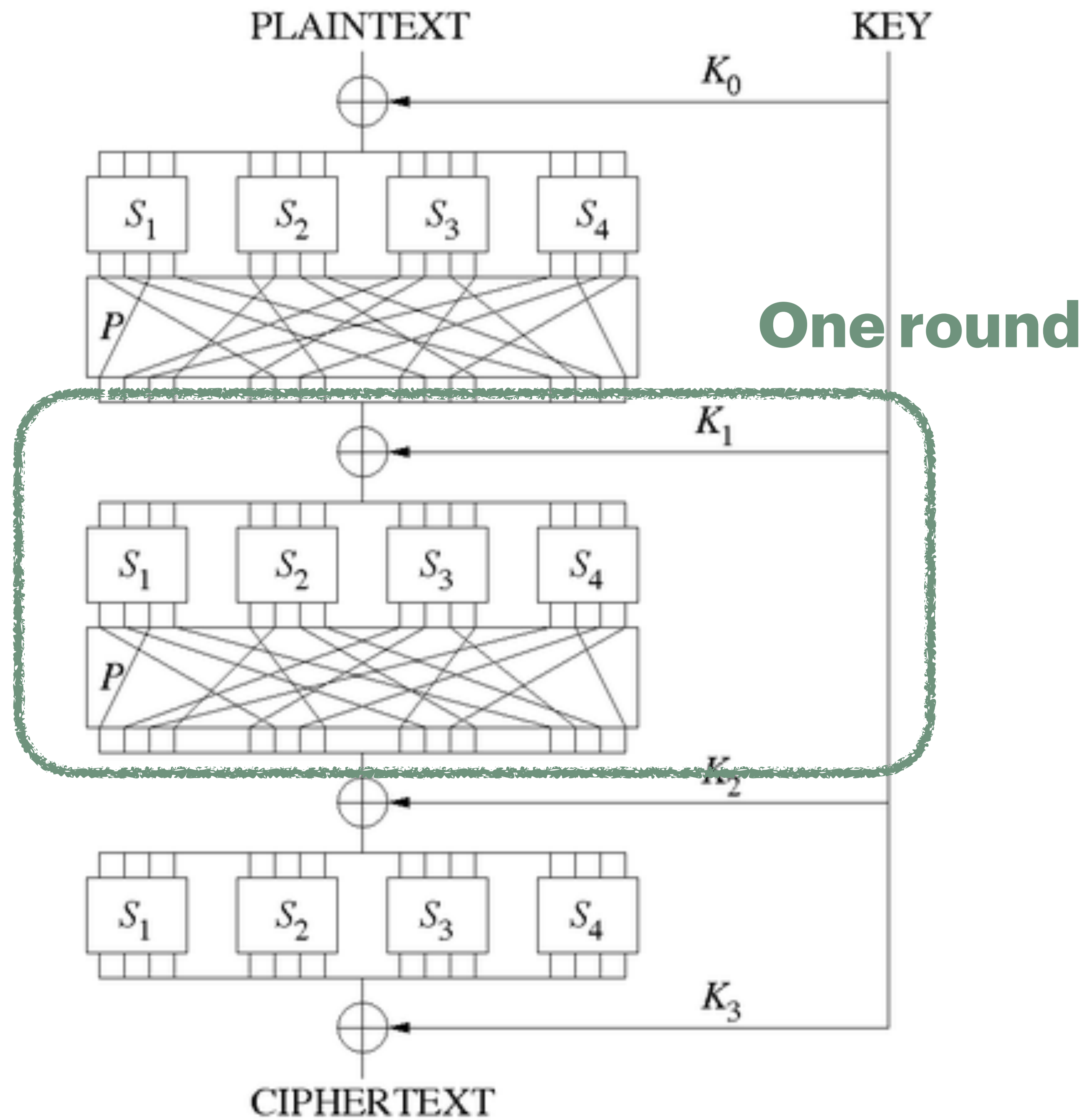
- Decryption: run backwards.



Substitution-Permutation Network (SPN)

Round Function Examples

- Concatenating multiple *rounds*
- View each round as func g
  - Input: round key K_i and previous round's output s_{i-1}
  - Output: s_i
- Plaintext: s_0
- Ciphtertext: s_N, where N is the number of rounds
- Decryption: run g^{-1} iteratively

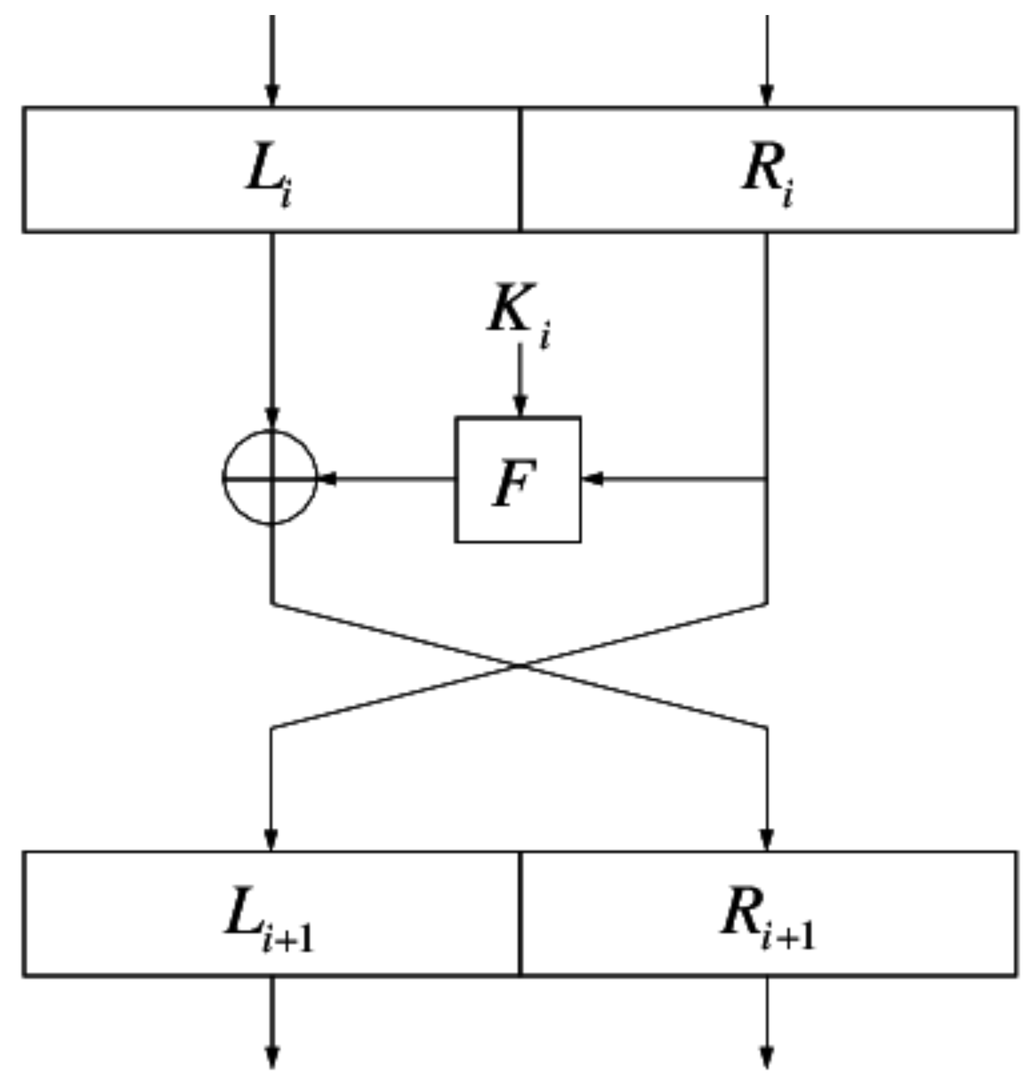


Feistel Network

Round Function Examples

One Feistel round:

- Only encrypt half of the Input block
  - So *one round alone* does not provide security
- Security provided by a *Pseudorandom Function* F
  - “Like” PRG used in stream cipher
- Lastly, swap the two half

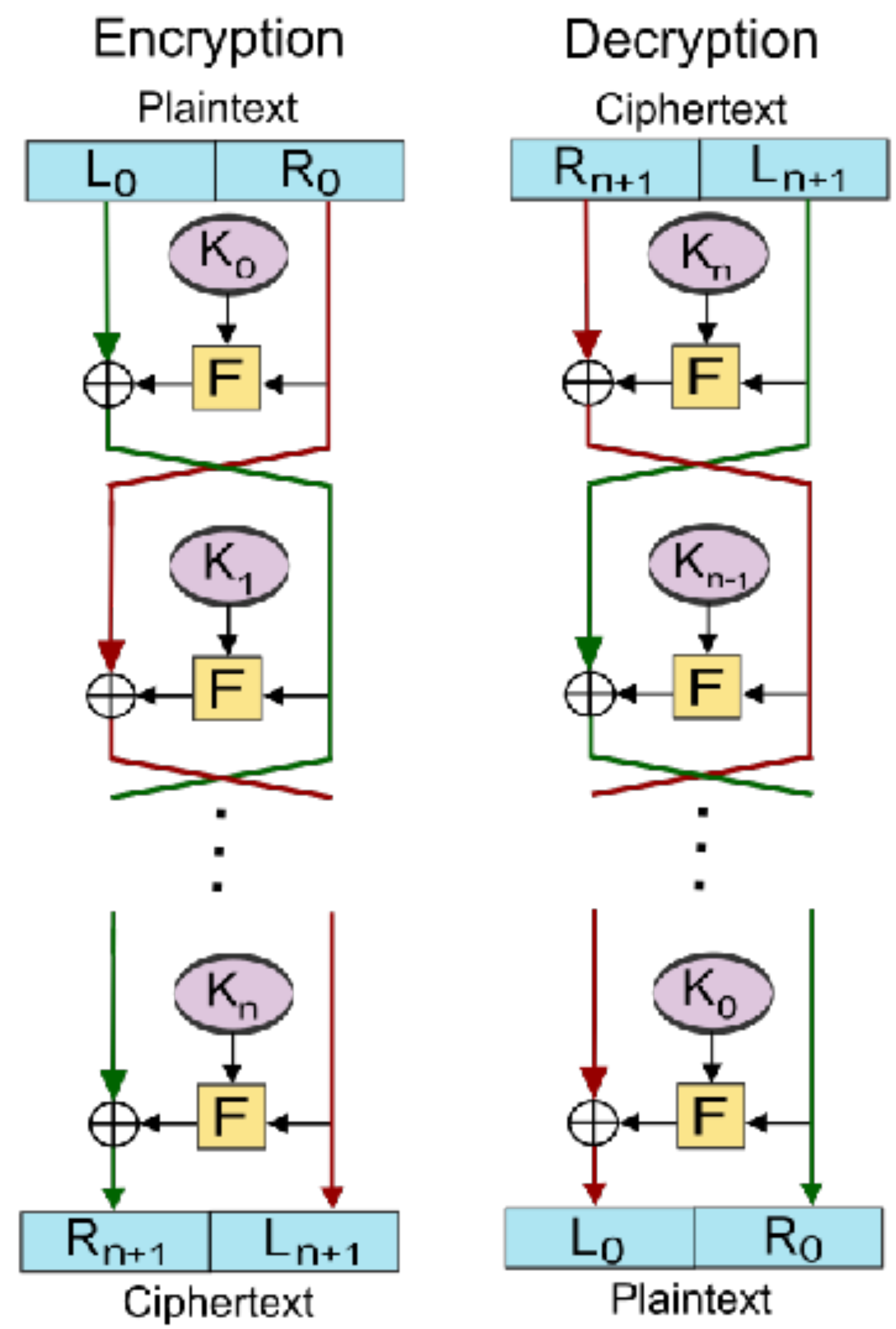


Decrypt: run again with L, R swapped

Feistel Network

Round Function Examples

- Concatenating multiple rounds
  - **Theorem** [Luby-Rackoff] $\geq$3 Feistel rounds with a "secure F" gives a secure block cipher (a.k.a, a "*secure pseudorandom permutation*")
- In practice, F is often implemented as a small (*not necessarily invertible*) **SPN**



Principle for Round Functions

- In both types of networks, the substitution and permutation algorithms must be carefully designed
  - choosing *random* substitution/permutation strategies leads to significantly weaker ciphers
- Each bit difference in S-box input creates at least 2-bit difference in its output
- Mixing permutation ensures that difference in one S-box propagates to at least 2 S-boxes in next round

Acknowledgement

- The slides of this lecture is developed heavily based on
  - Slides from Prof Dan Boneh's lecture on Cryptography
  - Slides from Prof Ziming Zhao's lecture on Symm. Encryption

Questions ?