Introduction to Information Security

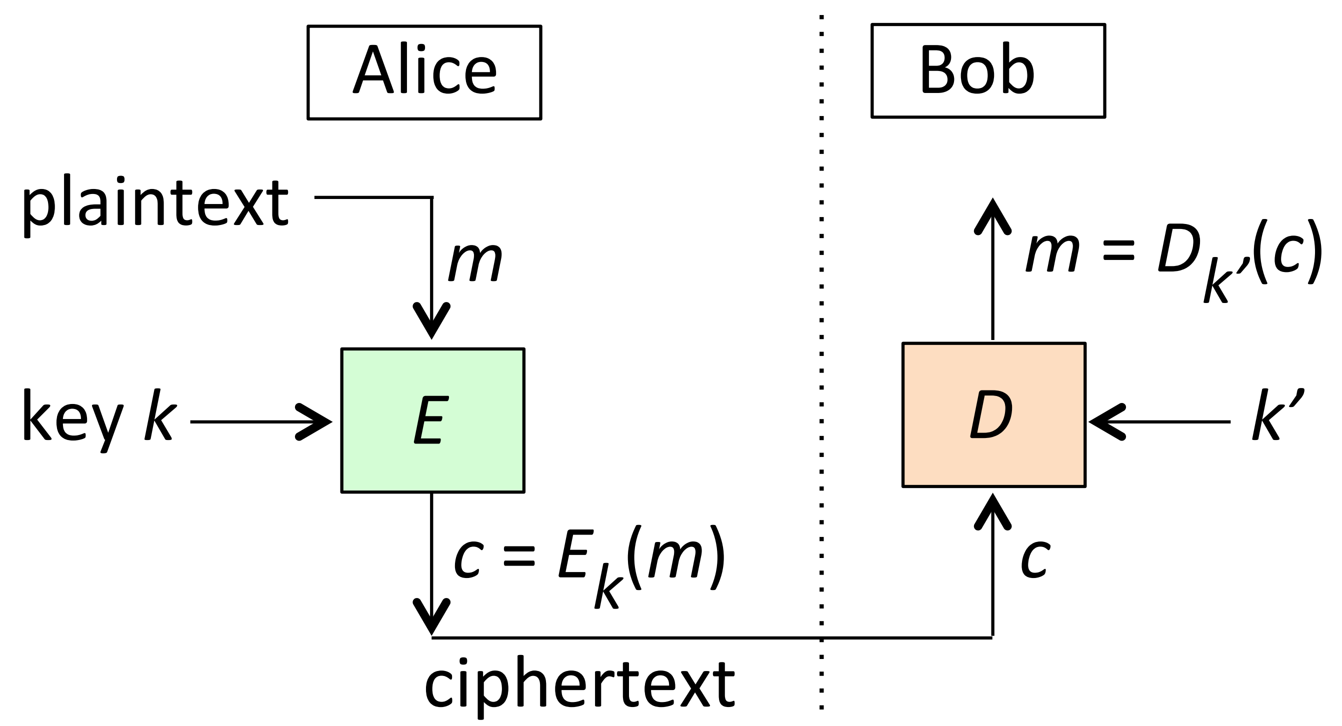
4. Symmetric-key Encryption

Kihong Heo

KAIST

Symmetric-key Encryption

- Symmetric: the encryption and decryption keys are the same
- Assume: plaintexts and ciphertexts are all bit vectors from now on (for simplicity)



Perfectly Secret Encryption

- *Ideal* encryption scheme
- Secure against an adversary with unbounded computational power (e.g., infinite time & memory)
- Two equivalent definitions

An encryption scheme (Gen, Enc, Dec) with message space $\mathscr{M}$ is perfectly secret if for every probability distribution over $\mathscr{M}$, every message $m \in \mathscr{M}$, and every ciphertext $c \in \mathscr{C}$ for which $\Pr[C = c] > 0$:

$$\Pr[M = m \mid C = c] = \Pr[M = m]$$

For every $m, m' \in \mathscr{M}$, and every $c \in \mathscr{C}$,

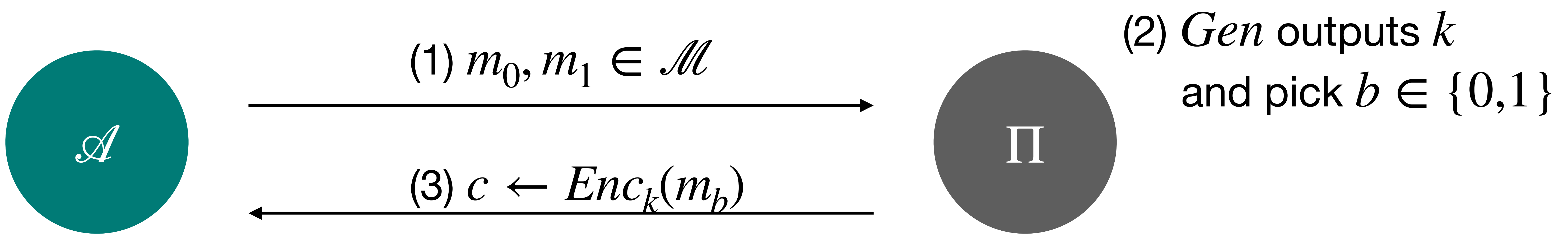
$$\Pr[Enc_K(m) = c] = \Pr[Enc_K(m') = c]$$

Perfect Indistinguishability

- Yet another equivalent definition

- Consider a game with an adversary $\mathscr{A}$ and an encryption oracle $\Pi = (Gen, Enc, Dec)$

$PrivK^{eav}_{\mathscr{A},\Pi}$



(4) $\mathscr{A}$ outputs b'

(5) $PrivK^{eav}_{\mathscr{A},\Pi}$ = 1 if $b' = b$, and 0 otherwise

- Encryption scheme Π with message space $\mathscr{M}$ is perfectly indistinguishable if for every $\mathscr{A}$

$$\Pr[PrivK^{eav}_{\mathscr{A},\Pi} = 1] = 0.5$$

Vernam Cipher

- AKA Vernam's one-time pad (Gilbert Verman, 1917)
- Fix an integer $l > 0$. The space of $\mathscr{M}$, $\mathscr{K}$, $\mathscr{C}$ are $\{0,1\}^l$
- Idea: encrypt plaintext one bit at a time using a random key
  - $m = m_1 m_2 \ldots m_l$ and $k = k_1 k_2 \ldots k_l$
- Gen: choose a key from $\mathscr{K}$ with uniform distribution
- Enc: $c_i = m_i \oplus k_i$
- Dec: $m_i = c_i \oplus k_i$
- Key k is randomly chosen and never reused: one-time pad

Proof of Perfect Secrecy

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

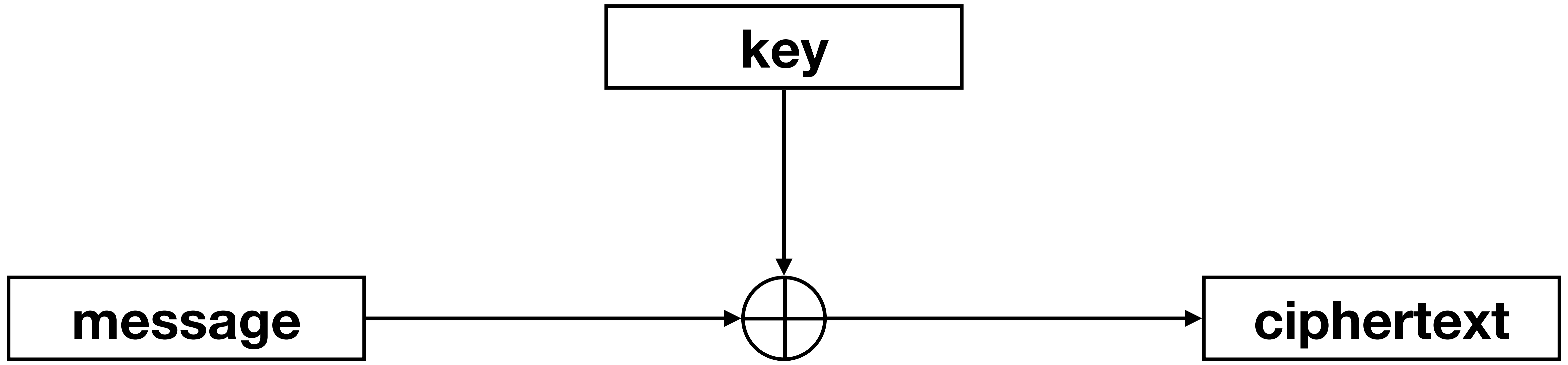
$$= \frac{2^{-l} \cdot \Pr[M = m]}{2^{-l}}$$

$$= \Pr[M = m]$$

Confidentiality of Vernam Cipher

- Unbreakable encryption scheme
  - An attacker without the key cannot recover plain text from ciphertext
  - Even given unlimited computing power and time
- So-called information-theoretically secure
  - The best thing the attacker can do is a random guess

Why don’t we use them?



Limitations of One-Time Pad

- The OTP should be truly random
- The OTP should be at least as long as the message
- Both copies of the OTPs are destroyed immediately after use

KGB (USSR)

DDR (East-Germany)

Towards Practical Encryption Schemes

- Do not rely on a **truly random** number generator → pseudo-random number generator
- Do not have a key **as large as** the message → block cipher
- Do not have the **same ciphertext** even with the same key and plaintext → prob. encryption

Computationally Secure Encryption

- Perfect secrecy: no information leaked to an adversary with unlimited computational power
  - Unnecessarily strong
- In practice, may be okay
  - leakage of a tiny amount of information
  - to an adversary with bounded computational power
- How to define
  - Tiny amount?
  - Bounded computational power?
  - Okay?

Example

- Consider a scheme with the guarantee that
  - no adversary running for at most 2^{80} cycles can break the scheme
  - with a probability better than 2^{-60}
- Is this secure?
  - Supercomputer: 2^{80} keys/year
  - Sender/receiver both struck by lightning in a year: 2^{-60}

Recall: Perfect Indistinguishability

- Yet another equivalent definition
- Consider a game with an adversary $\mathscr{A}$ and an encryption oracle $\Pi = (Gen, Enc, Dec)$

$PrivK^{eav}_{\mathscr{A},\Pi}$

$\mathscr{A}$ $\xrightarrow{(1)\ m_0, m_1 \in \mathscr{M}}$ Π

(2) Gen outputs k
and picks $b \in \{0,1\}$

$\mathscr{A}$ $\xleftarrow{(3)\ c \leftarrow Enc_k(m_b)}$ Π

(4) $\mathscr{A}$ outputs b'

(5) $PrivK^{eav}_{\mathscr{A},\Pi}$ = 1 if $b' = b$, and 0 otherwise

- Encryption scheme Π with message space $\mathscr{M}$ is perfectly indistinguishable if for every $\mathscr{A}$

$$\Pr[PrivK^{eav}_{\mathscr{A},\Pi} = 1] = 0.5$$

Computational Indistinguishability: Concrete

- Introduce two concrete parameters
  - Bounded adversary capability: time t
  - Tiny probability of failure: probability ϵ
- Encryption scheme $\Pi = (Gen, Enc, Dec)$ is (t, ϵ)-indistinguishable if for every $\mathscr{A}$ running time at most t,

$$\Pr[PrivK^{eav}_{\mathscr{A},\Pi} = 1] \leq 0.5 + \epsilon$$

- Problems?
  - Complicated formulation and proof
  - Hard to change parameters (security level)

Asymptotic Formalization

- Standard way for the estimation of the computational complexity of problems
  - Details will be covered in CS300 (Introduction to algorithms)
- Idea: describe the behavior of the algorithm based on the input size n
- Example: worst case time complexity
  - `max : int list -> int`
  - `bubble_sort : int list -> int list`
  - Exhaustive password search (i.e., brute-force, 마구잡이)
  - Shortest route that visits each city exactly once and returns to the origin

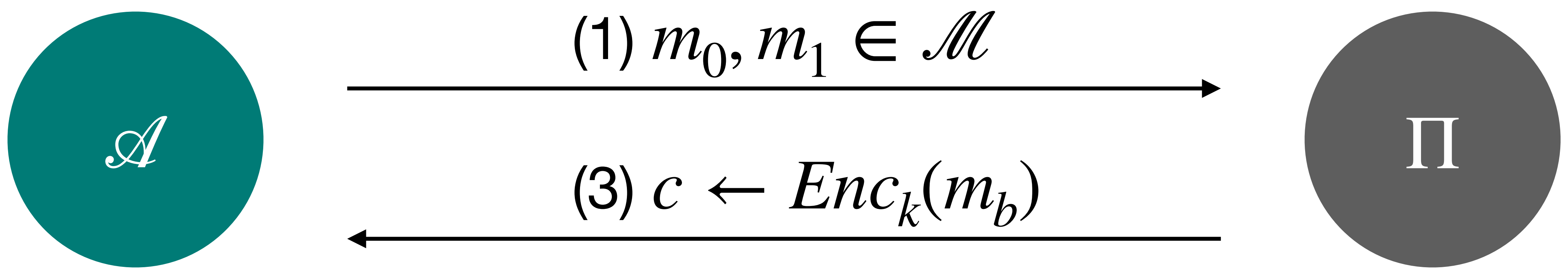
Asymptotically Secure

- Introduce an integer-valued security parameter n
  - Typically a key length
  - Parameterize both the running time of the adversary and the attack success probability
- Asymptotically secure:
  - Any probabilistic polynomial-time (PPT) adversary succeeds in breaking the scheme with at most negligible probability
  - Probabilistic: access a random bit
  - Polynomial: efficient algorithm or running in polynomial time for given n
  - Negligible: asymptotically smaller than any inverse polynomial function

Computational Indistinguishability

- Consider a game with a PPT adversary $\mathscr{A}$ and an encryption oracle $\Pi = (Gen, Enc, Dec)$

$PrivK^{eav}_{\mathscr{A},\Pi}$



(2) Gen outputs k and picks $b \in \{0,1\}$

(4) $\mathscr{A}$ outputs b'

(5) $PrivK^{eav}_{\mathscr{A},\Pi}$ = 1 if $b' = b$, and 0 otherwise

- Encryption scheme Π is computationally indistinguishable if for every PPT $\mathscr{A}$, there is a negligible function $negl$ such that for all n,

$$\Pr[PrivK^{eav}_{\mathscr{A},\Pi}(n) = 1] \leq 0.5 + negl(n)$$

Recall: Security Guarantees

- Example: What are the desired security guarantees for secure encryption?
- Impossible for an attacker
  - To recover the key? Enough?
  - To recover the entire plaintext from the ciphertext? Enough?
  - To recover any character of the plain text from the ciphertext? Enough?
  - To derive any meaningful information about the plaintext from the ciphertext? Enough?
  - To compute any function of the plaintext from the ciphertext **(semantic security)**

Semantic Security

- Semantically secure $\iff$ computationally indistinguishable

semantic security

$m \in \mathcal{M}$

$c = Enc_k(m)$

Alice (user)

Eve

Can't learn anything about m

$c =$ 9014e195b5f69df6c5f2

Bob (bank)

computational indistinguishability

$m_0, m_1 \in \mathcal{M}$

$c = Enc_k(m_b)$

Alice (user)

Eve

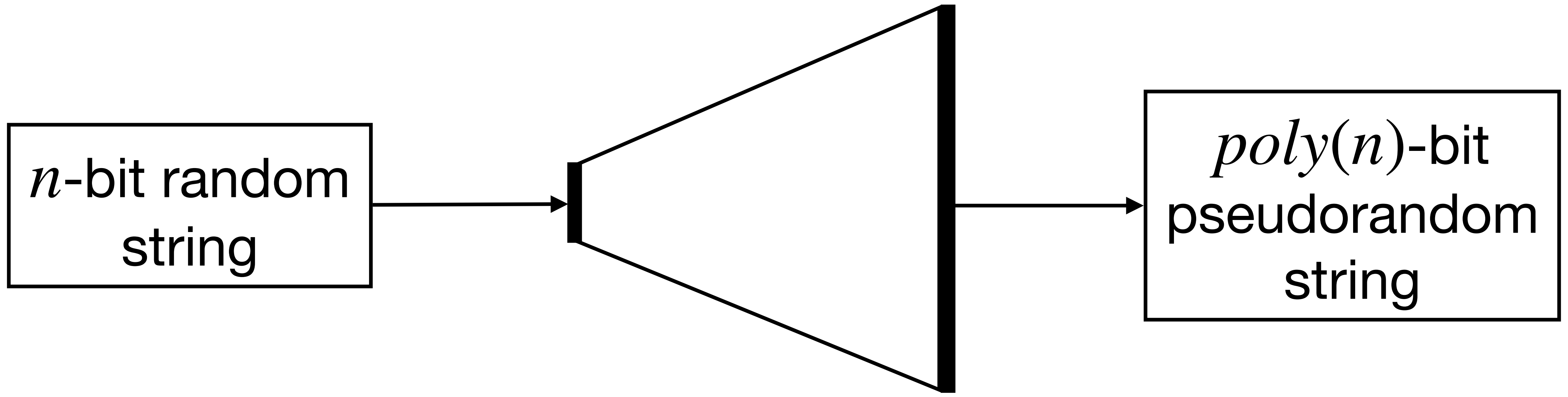
Can't tell which one is picked and encrypted

$c =$ 9014e195b5f69df6c5f2

Bob (bank)

Pseudorandom Generators (PRGs)

- An efficient algorithm that transforms a short random string (seed) into a longer "random-looking" output string
- "Random-looking"?
  - The output of PRG should look like a random string to any efficient observer
- Remember: "efficient" means "polynomial" in CS most of the time



Formal Definition of PRGs

- $G : \{0,1\}^n \rightarrow \{0,1\}^{poly(n)}$ is a pseudorandom generator if for any PPT algorithm D (distinguisher), there is a negligible function $negl$ such that

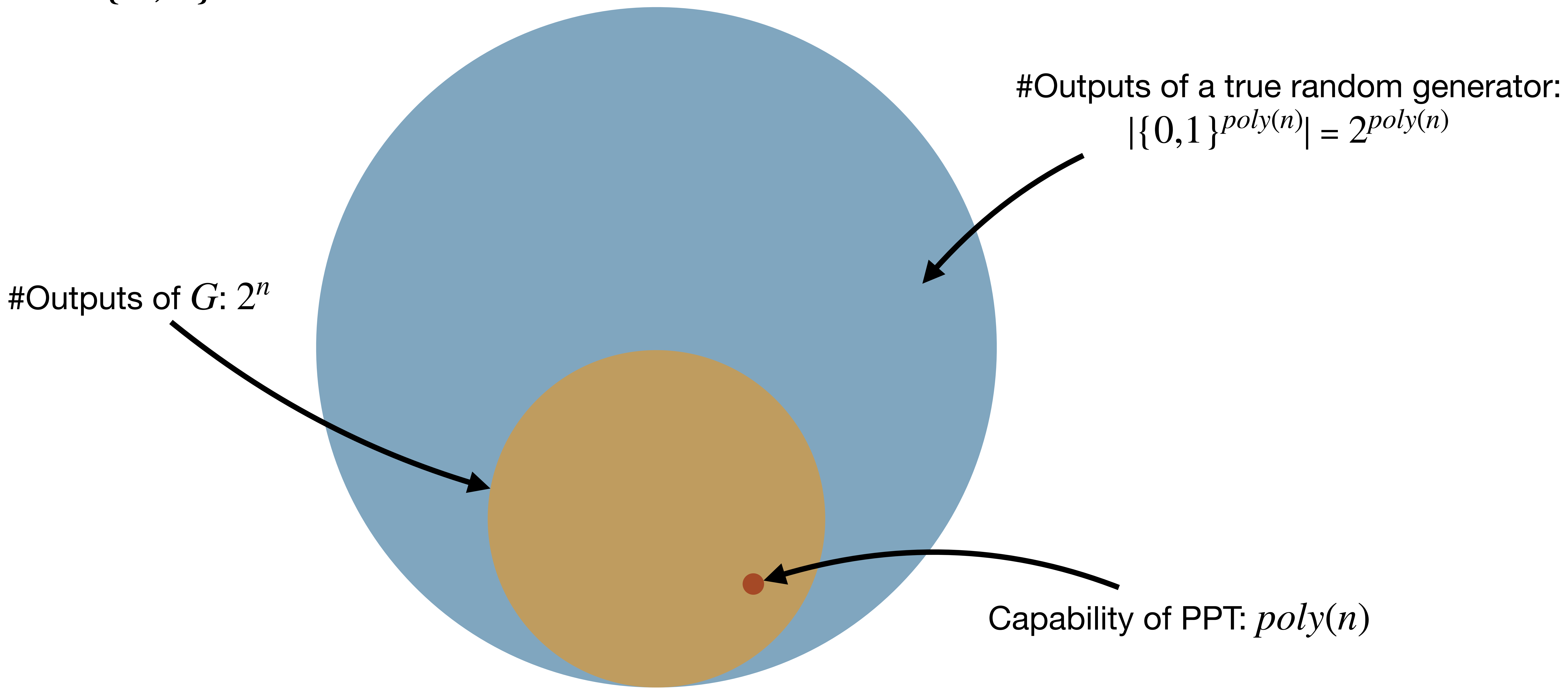
$$|\Pr[D(G(s)) = 1] - \Pr[D(r) = 1]| \leq negl(n)$$

where $D(w) = \begin{cases} 1 \text{ if } D \text{ concludes that } w = G(s), s \text{ is drawn from } \{0,1\}^n \\ 0 \text{ if } D \text{ concludes that } w \text{ is drawn from } \{0,1\}^{poly(n)} \end{cases}$

- Do such PRGs exist?
  - Don't know but YES if $P \neq NP$ (i.e., if $P = NP$ then, distinguishable by PPT)
  - Many practical PRGs in use every day (e.g., /dev/random)

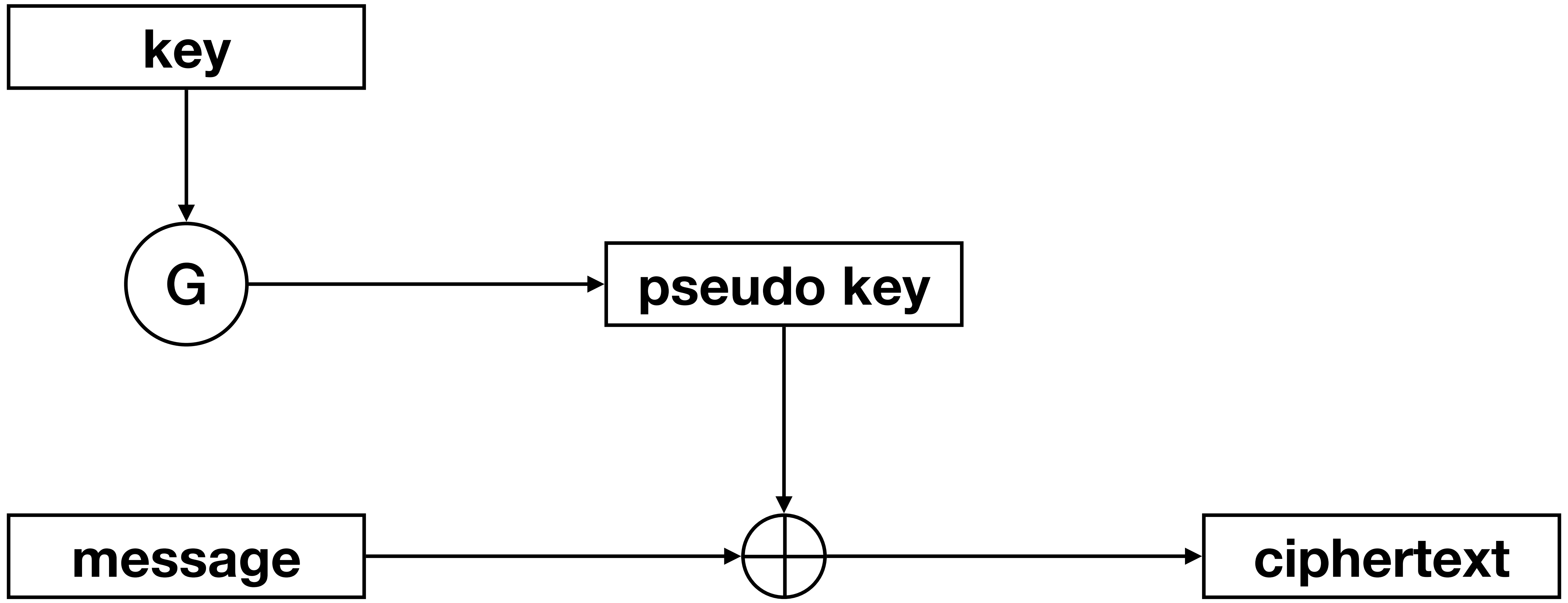
Indistinguishability

- $G : \{0,1\}^n \rightarrow \{0,1\}^{poly(n)}$



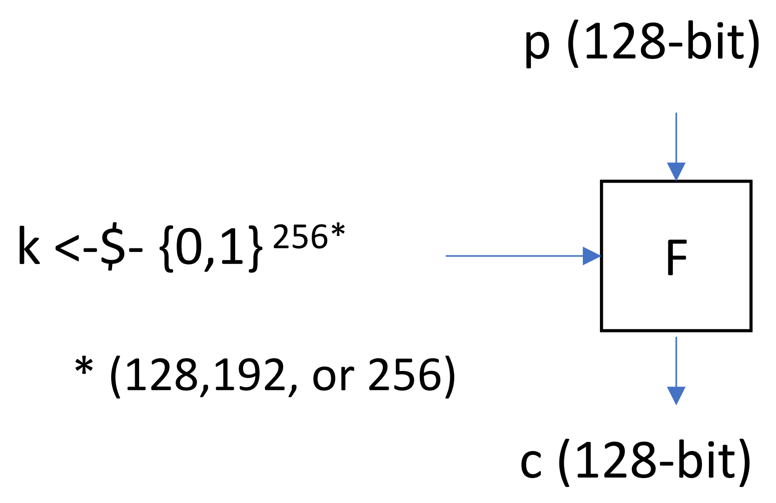
Towards Practical Encryption Schemes

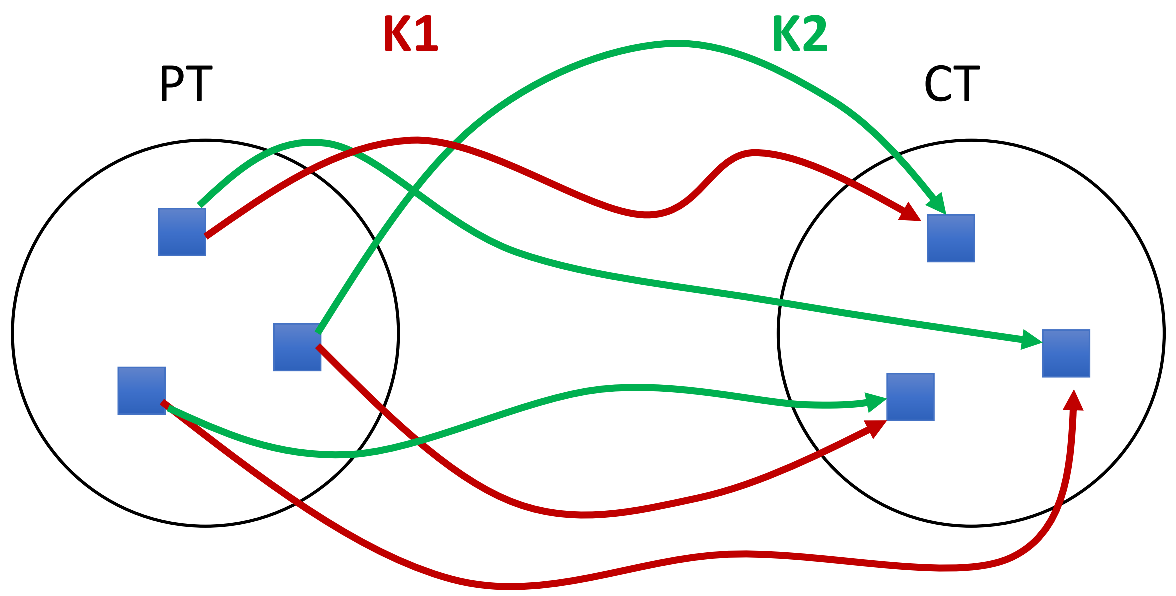
- Do not rely on a truly random number generator → pseudo-random number generator
- Do not have a key as large as the message → block cipher
- Do not have the same cipher text even with the same key and plaintext → prob. encryption



Block Cipher

- Encrypt data in blocks of fixed lengths (e.g., 128-bits)
    - C.f., Stream cipher: encrypt 1 bit of data at a time (e.g., Vernam Cipher)
- Basic building block of many encryption schemes
- Idea: key = permutation
    - For a fixed key k, a block cipher with n-bit block length is a permutation
- Example: DES, AES (Advanced Encryption Standard)





Pseudo-Random Permutation (PRP)

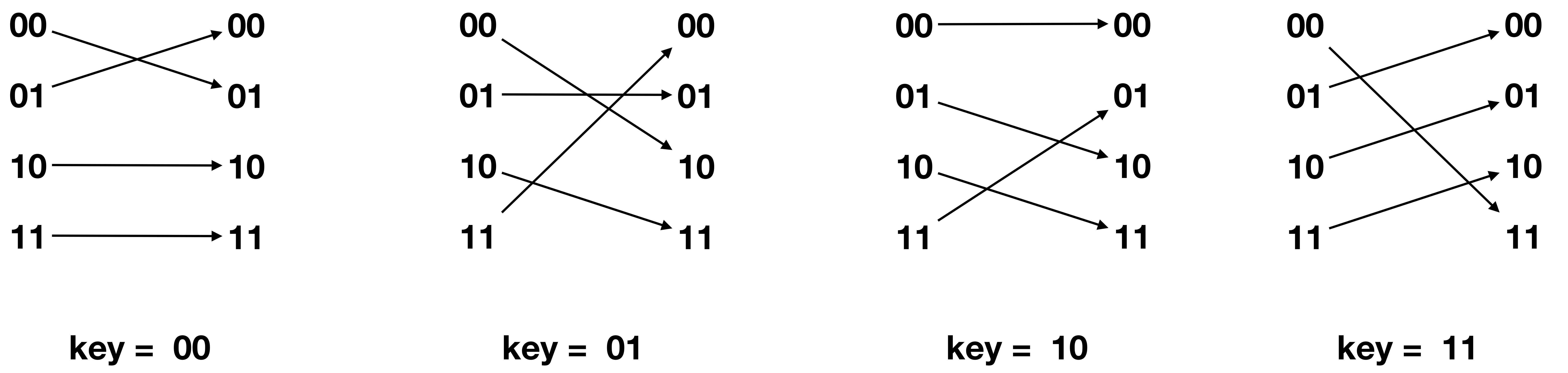
- Given a key length s and block length n
- Ideal block cipher
  - A collection $E = \{\pi_1, \ldots, \pi_{2^n!}\}$ of random permutations $\pi_i : \{0,1\}^n \rightarrow \{0,1\}^n$
- Practical block cipher using PRP $\pi : \{0,1\}^s \times \{0,1\}^n \rightarrow \{0,1\}^n$
  - Encryption $c = \pi_k(m)$ and decryption $m = \pi_k^{-1}(c)$ where k is the key
  - For any $k \in \{0,1\}^s$, π_k is a one-to-one function from $\{0,1\}^n \rightarrow \{0,1\}^n$
  - For any $k \in \{0,1\}^s$, there is an "efficient" algorithm to evaluate $\pi_k(x)$ and $\pi_k^{-1}(x)$
  - For any $k \in \{0,1\}^s$, π_k is indistinguishable from a random permutation

Indistinguishability

- How many possible π? (truly random permutation)
  - $(2^n)!$
  - If n = 3, then 30,320
  - If n = 7, then 2.856205 x 10^{215}
- How many possible π_k when the key length is s?
  - 2^s
- For larger s, π_k is indistinguishable from a random permutation

Example

- Block length: 2 bits
- Key length: 2 bits



AES

- Advanced Encryption Standard
  - Based on the Rijndael cipher developed by Rijmen and Daemen (2001)
- Symmetric key block cipher to replace DES (1977)
- Key length: 128, 192, and 256 bits
- 10 to 14 rounds of permutation
  - 10 rounds for 128-bit key, 12 for 192, 14 for 256
- 3 big ideas: confusion, diffusion, and key secrecy

Confusion

- Obscure the relationship between the plaintext and the ciphertext
- Example: Caesar cipher
  - Plaintext: `attack at dawn`
  - Ciphertext: `DWWDFN DW GDZQ`

Diffusion

- Spread out the message
- Example: column transposition
  - Plaintext: `attack at dawn`
  - Ciphertext: `ACD TKA TAW ATN`

A	T	T	A
C	K	A	T
D	A	W	N

Key Secrecy

- Kerckhoffs's principle (1883)
- A cryptosystem should be secure even if
  - Everything about the system is public (i.e., algorithm)
  - Except for the key
- Why?
  - Easier to keep small things secret than large things
  - |System design| >> |Key|

AES in a Nutshell (1)

- Consider the minimum case of 128-bit key
- Input and output: 4 x 4 matrix of bytes
- (Intermediate) State : 4 x 4 matrix of bytes

input bytes

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

→

State array

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

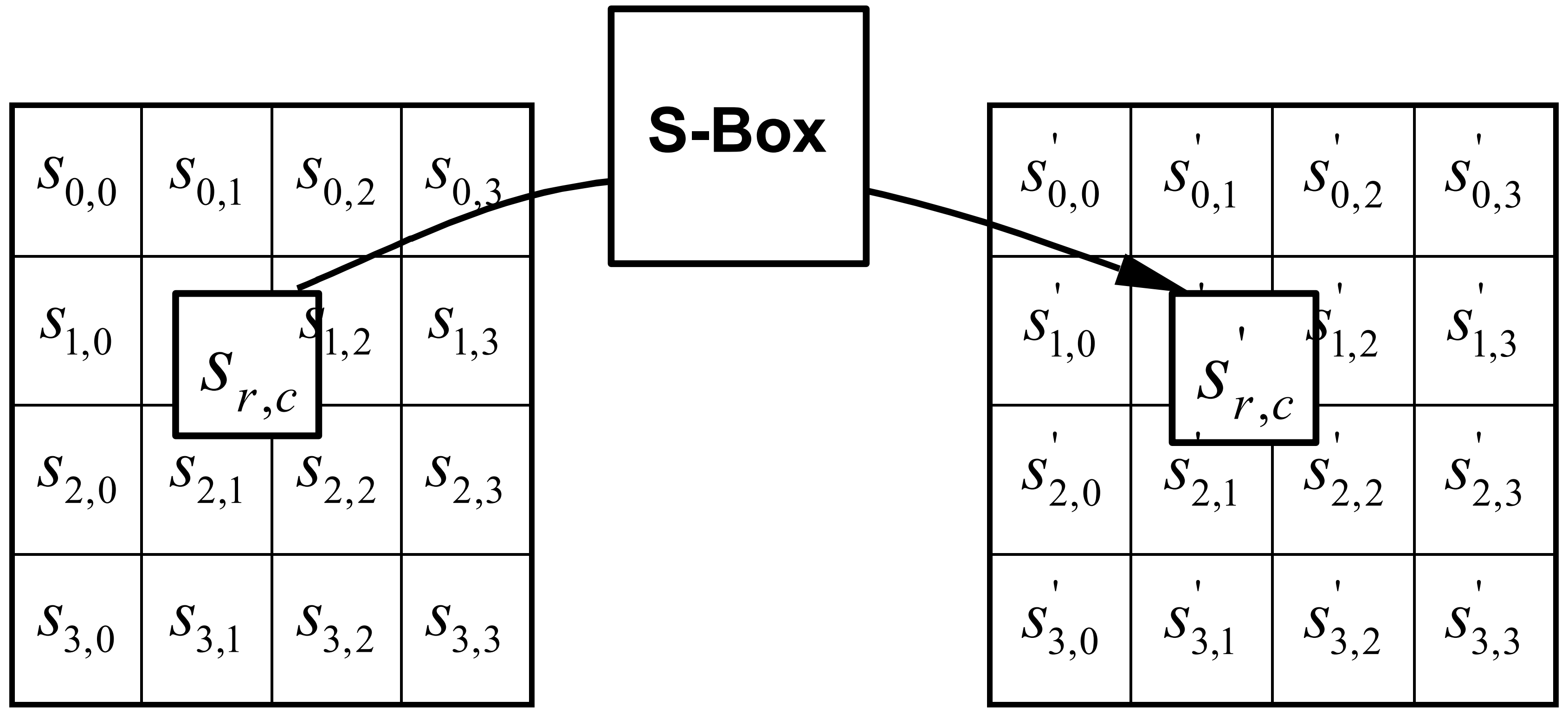
→

output bytes

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

AES in a Nutshell (2): SubBytes

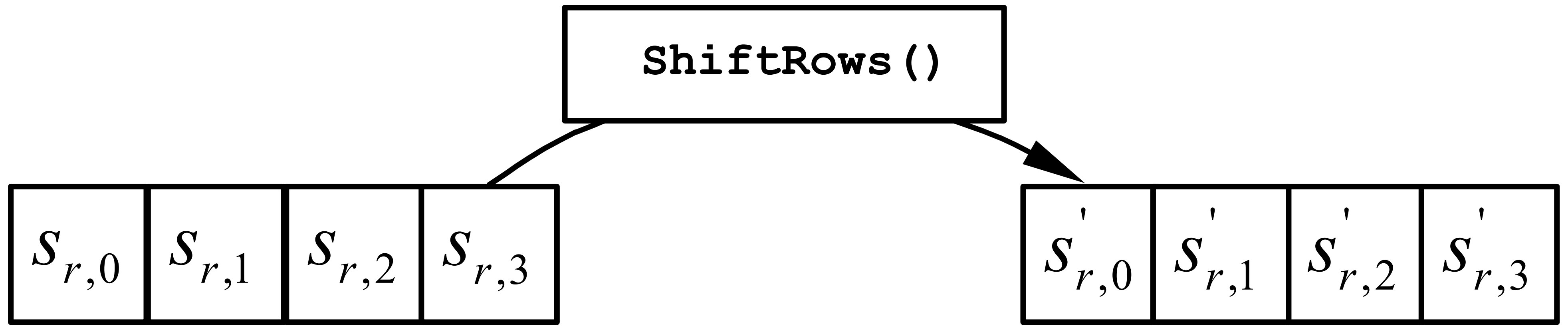
- Non-linear byte substitution for **confusion**
- Independent operation on each byte of the state using a substitution table (S-box)
- Example: if $s_{1,1} = 53$ then $s'_{1,1} = ed$

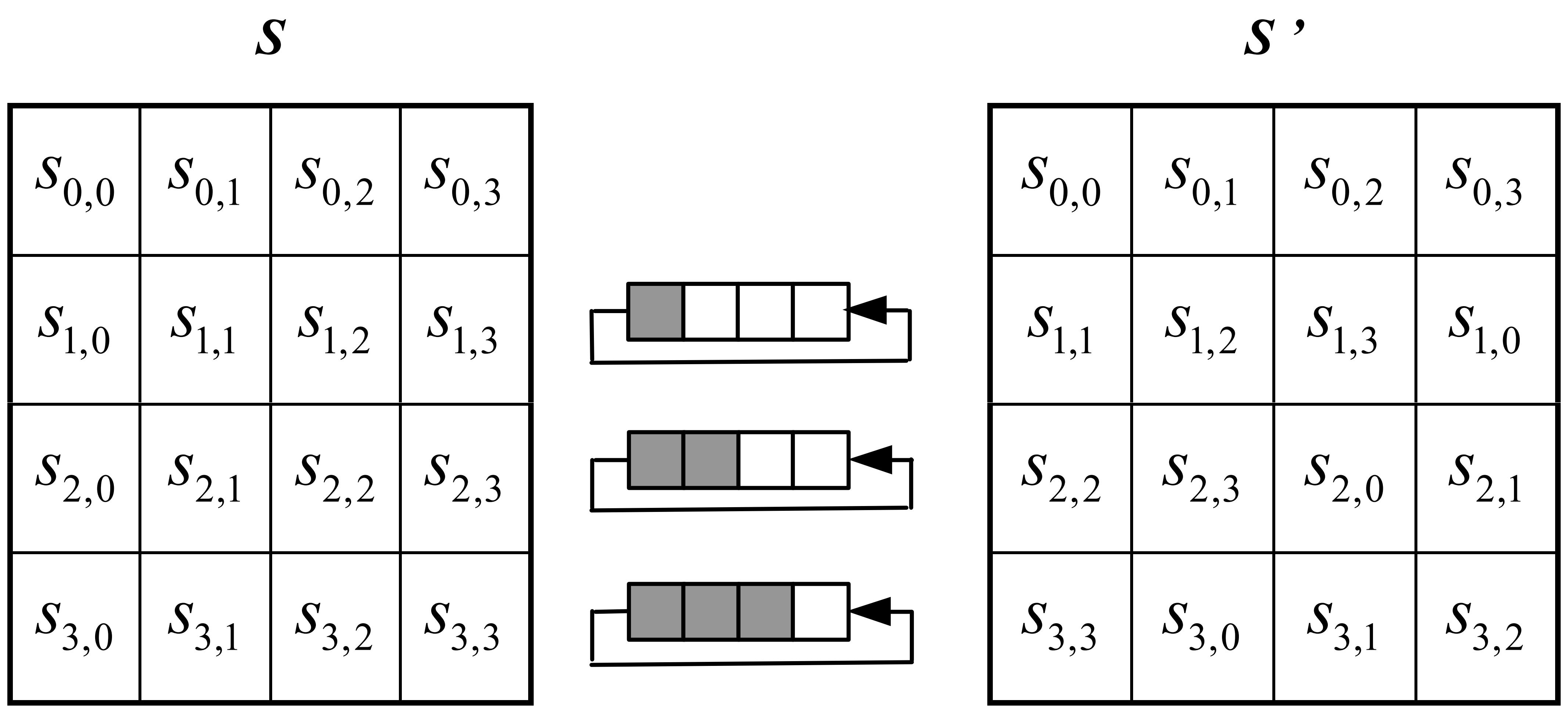


x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

AES in a Nutshell (3): ShiftRows

- Cyclic shift over different numbers of bytes for **diffusion**
  - i-th row: i-byte shift
- Example: if s_2 = 0a23 then s_2' = 230a





AES in a Nutshell (4): MixColumns

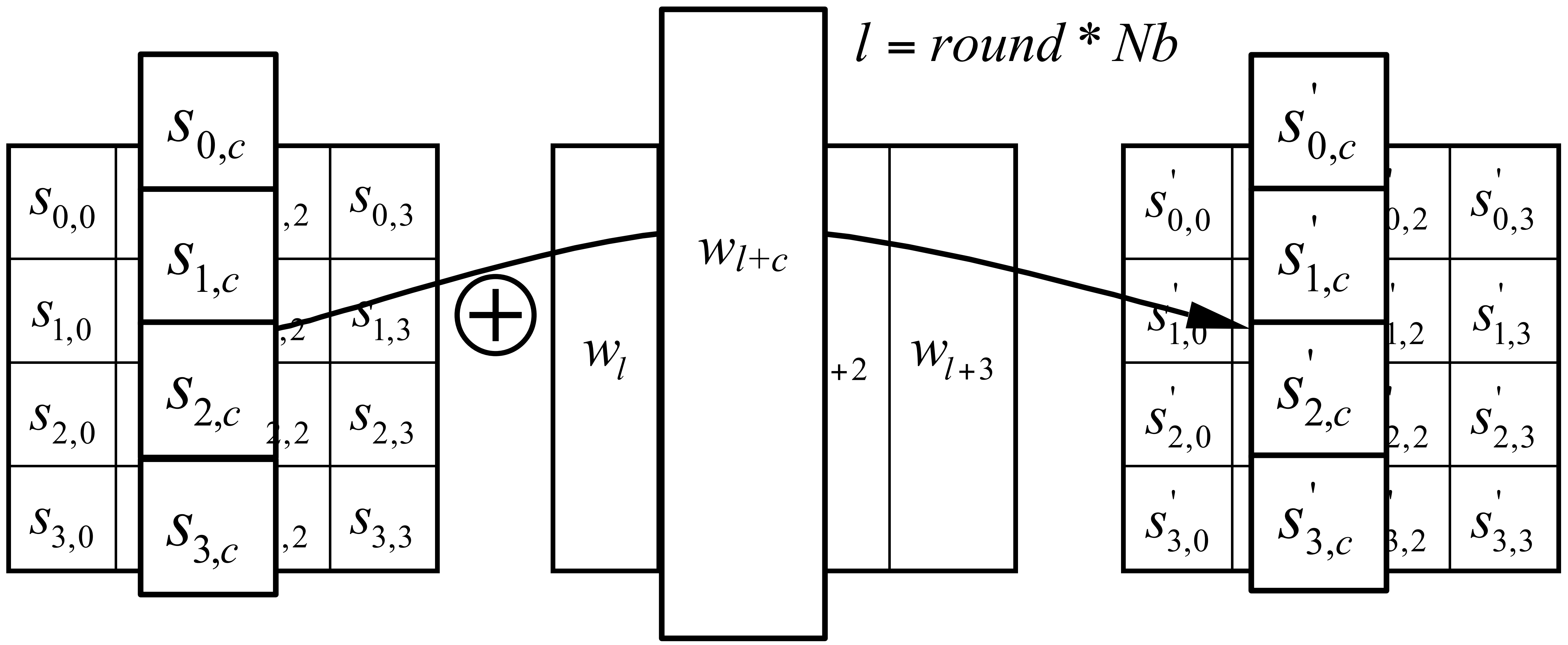
- Matrix multiplication on each column for **diffusion**
  - Multiplied by a fixed array

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$\begin{aligned} s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}). \end{aligned}$$

AES in a Nutshell (5): AddRoundKey

- A round key is added to the state by a simple bitwise XOR for **secrecy**
- The round key is determined by the key schedule algorithm



AES in a Nutshell (6): Put it All Together

- Encryption
    - For each round: AddRoundKey ∘ MixColumns ∘ ShiftRows ∘ SubBytes
- Decryption: the inverse of the encryption
    - For each round: $\text{SubBytes}^{-1} \circ \text{ShiftRows}^{-1} \circ \text{MixColumns}^{-1} \circ \text{AddRoundKey}^{-1}$

Practical Use of Block Cipher

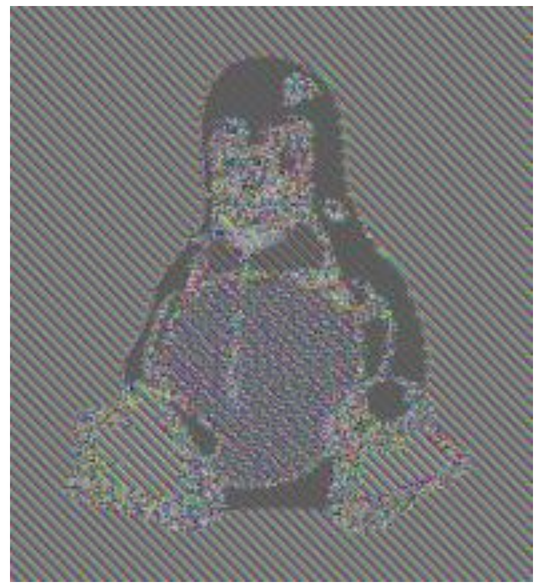
- If |plaintext| = block length?
  - Encrypt the plaintext using π_k
- If |the last plaintext block| < block length?
  - Padding with “filler” characters
- Then, the encryption algorithm is as follows:

1. Pad the plaintext with filler characters
2. Split the padded plaintext into equal-size blocks
3. Apply π_k for each block and concatenate them

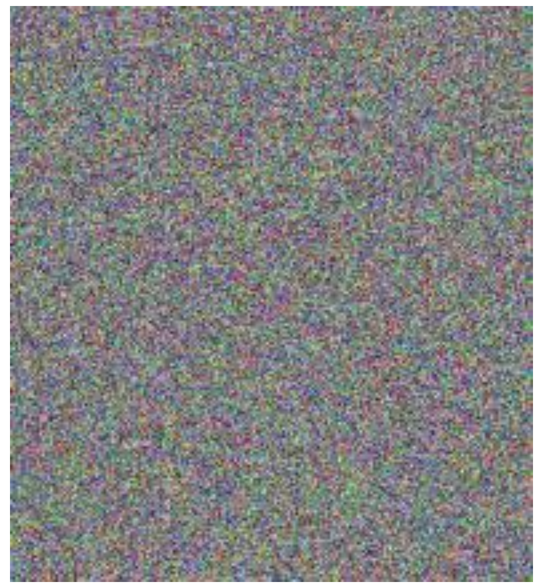
Problem

- Identical plaintext blocks → identical cipher text blocks

Plaintext



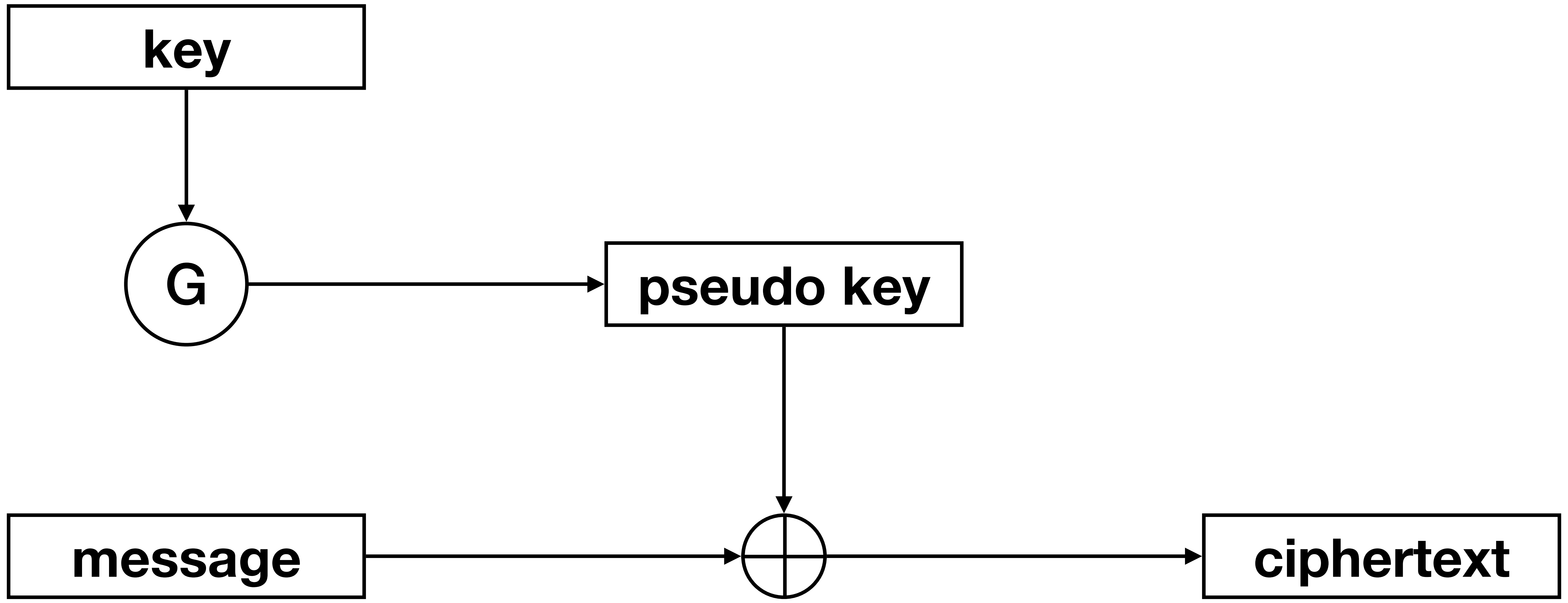
Ciphertext of the Naive block cipher



Ciphertext we want!

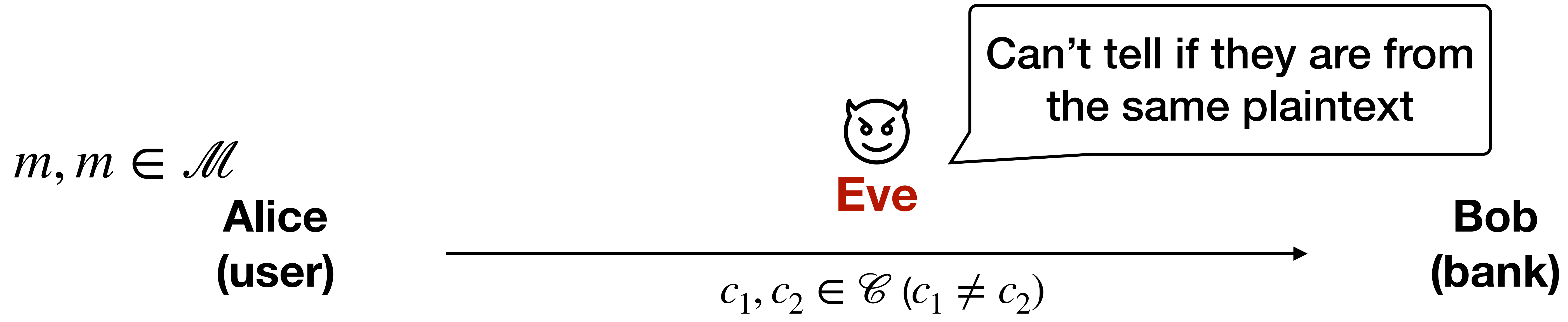
Towards Practical Encryption Schemes

- Do not rely on a truly random number generator → pseudo-random number generator
- Do not have a key as large as the message → block cipher
- Do not have the same cipher text even with the same key and plaintext → prob. encryption



Probabilistic Encryption

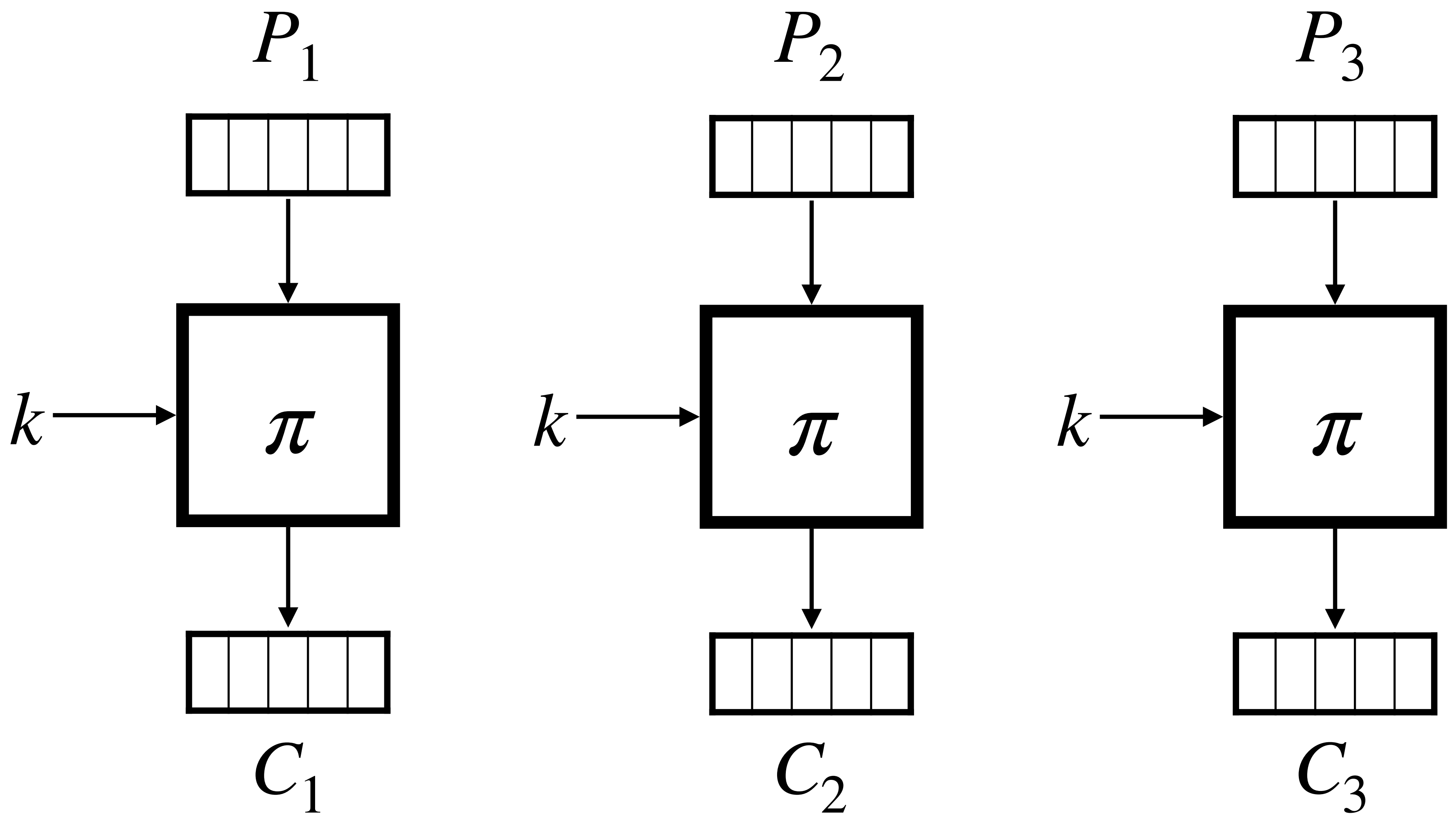
- Probabilistic encryption: different cipher texts for the same plaintext
  - All state-of-the-art encryption schemes are probabilistic
- How to generate different c_i for the same m?
- How to obtain the same m from different c_i?



Block Cipher Mode of Operation

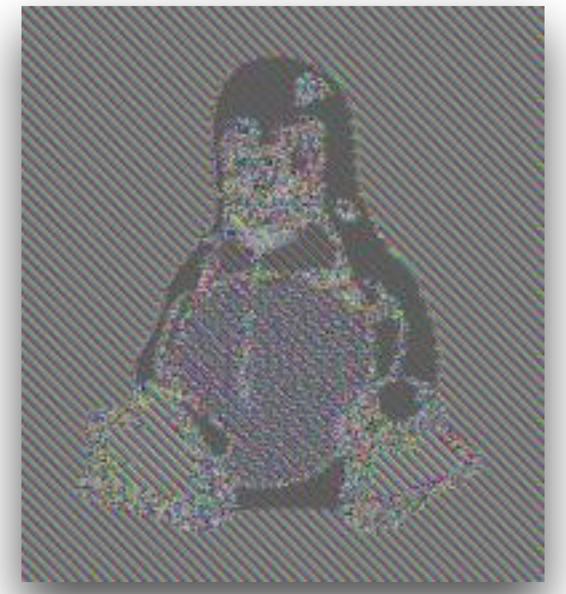
- Determine how to repeatedly apply a single-block operation to a sequence of blocks
- Pseudo-random permutation only guarantees the confidentiality of a single block
- Different modes of operations
  - ECB: Electronic Code Book
  - CBC: Cipher Block Chaining
  - CFB: Cipher FeedBack
  - OFB: Output FeedBack
  - CTR: CounTeR mode

Electronic Code Book Mode (ECB)

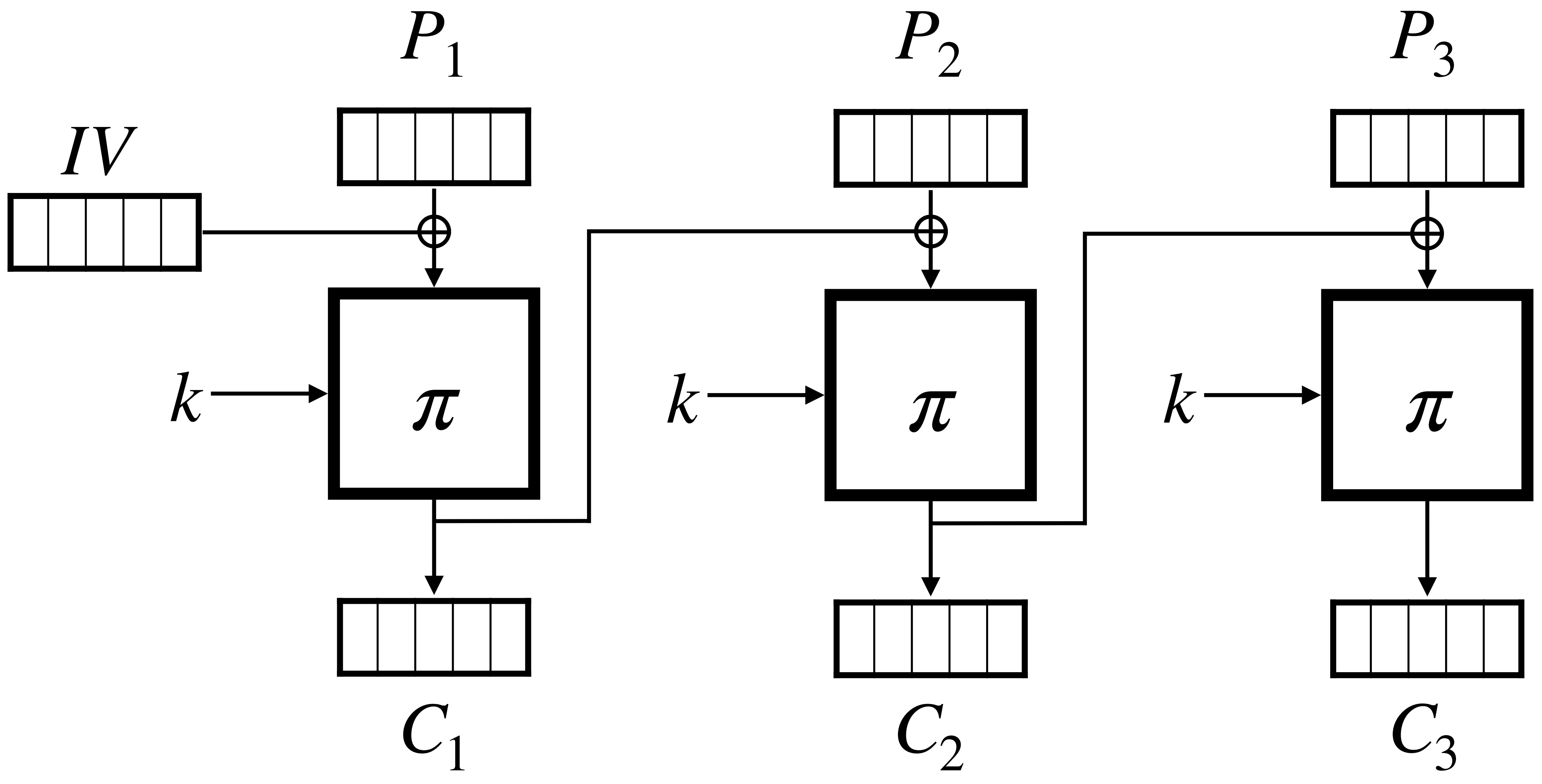


$$C_i = \pi_k(P_i)$$
$$P_i = \pi_k^{-1}(C_i)$$

- Advantages
  - Simple and efficient (i.e., parallelizable) to compute
- Disadvantages
  - Same plaintext always corresponds to same cipher text



Cipher Block Chaining Mode (CBC)

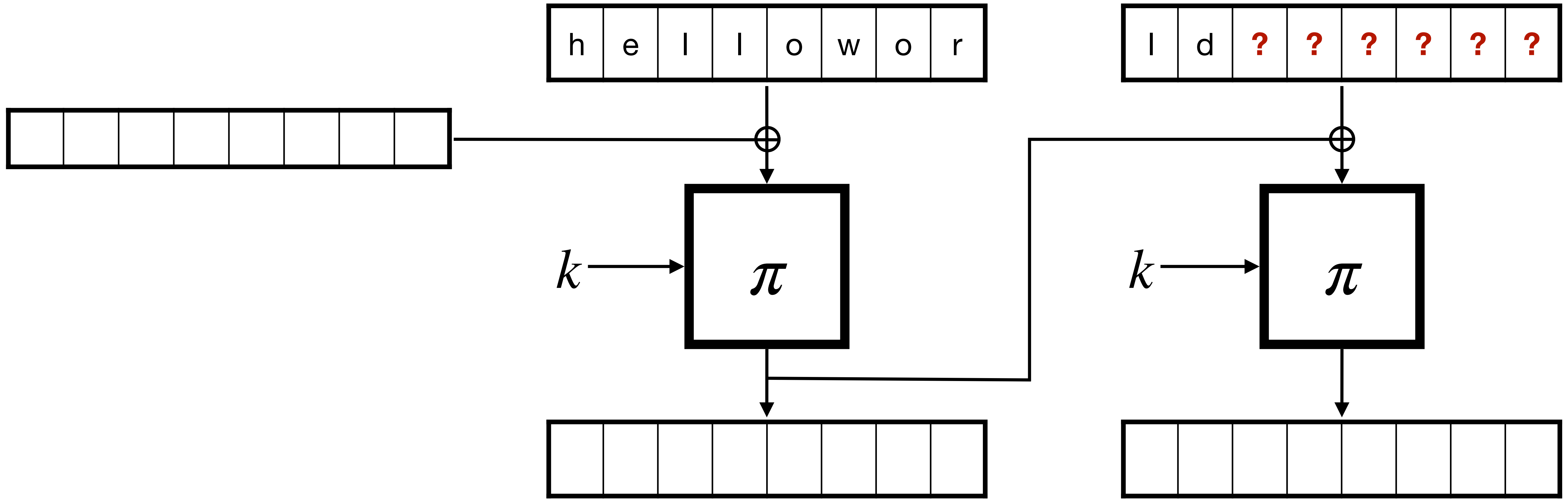


$C_i = \pi_k(P_i \oplus C_{i-1})$
$P_i = \pi_k^{-1}(C_i) \oplus C_{i-1}$
$C_0 = IV$ (Initialization Vector)

- Advantages
  - Semantic security
- Disadvantages
  - Cannot be parallelized

Padding

- Block cipher: a fixed block size
- What if the message size is not a multiplication of the block size?
- Example: 64-bit block (8 bytes)

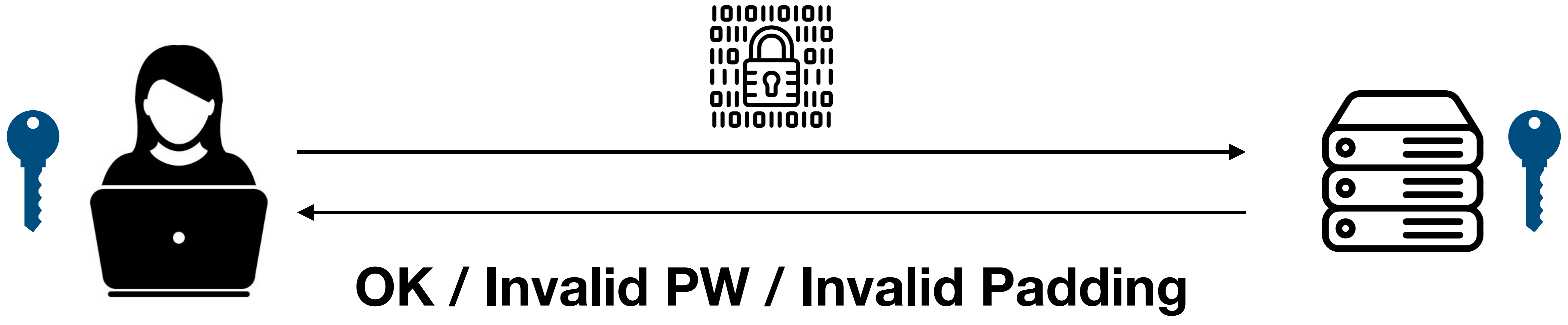


Padding Schemes

- What kind of padding scheme can you imagine?
- Zero padding: padded with zero
  - | 00 11 22 33 44 55 66 77 | 88 99 **00 00 00 00 00** |
  - Not reversible
- PKCS#5 (and PKCS#7): padded with the number of bytes that are added
  - | 00 11 22 33 44 55 66 77 | 88 99 **05 05 05 05 05** |
  - Most commonly used
- Many others

Padding Oracle

- A service that checks whether the plaintext is correctly padded or not
- Usually for providing detailed error messages
  - E.g., Invalid data, Invalid padding, etc



Padding Oracle Attack

- An attacker can obtain the plaintext using the oracle
- Discovered in 2002

September 13, 2010, 7:58AM

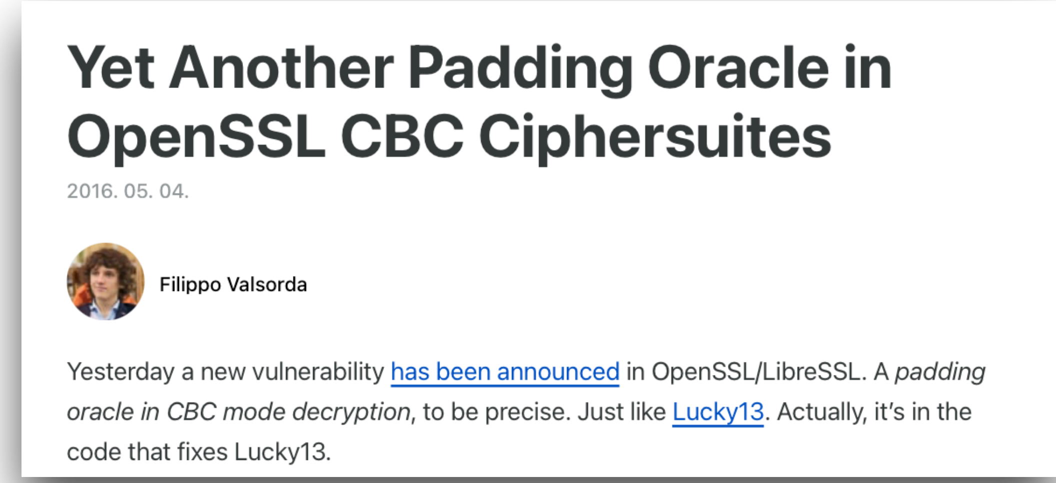
'Padding Oracle' Crypto Attack Affects Millions of ASP.NET Apps

by Dennis Fisher

Share Recommend (23) Print E-mail 41 Comments

A pair of security researchers have implemented an attack that exploits the way that ASP.NET Web applications handle encrypted session cookies, a weakness that could enable an attacker to hijack users' online banking sessions and cause other severe problems in vulnerable applications. Experts say that the bug, which will be discussed in detail at the Ekoparty conference in Argentina this week, affects millions of Web applications.

The problem lies in the way that ASP.NET, Microsoft's popular Web framework, implements the AES encryption algorithm to protect the integrity of the cookies these applications generate to store information during user sessions. A common mistake is to assume that encryption protects the cookies from tampering so that if any data in the cookie is modified, the cookie will not decrypt correctly. However, there are a lot of ways to make mistakes in crypto implementations, and when crypto breaks, it usually breaks badly.



Yet Another Padding Oracle in OpenSSL CBC Ciphersuites

2016. 05. 04.

Filippo Valsorda

Yesterday a new vulnerability has been announced in OpenSSL/LibreSSL. A *padding oracle in CBC mode decryption*, to be precise. Just like Lucky13. Actually, it's in the code that fixes Lucky13.

Home › Vulnerabilities

Microsoft Resolves Padding Oracle Vulnerability in Azure Storage SDK

By Ionut Arghire on July 19, 2022

Share Tweet 추천 10개 RSS

As part of its July 2022 Patch Tuesday fixes, Microsoft has released an update for the Azure Storage SDK, to address a padding oracle vulnerability in client-side encryption.

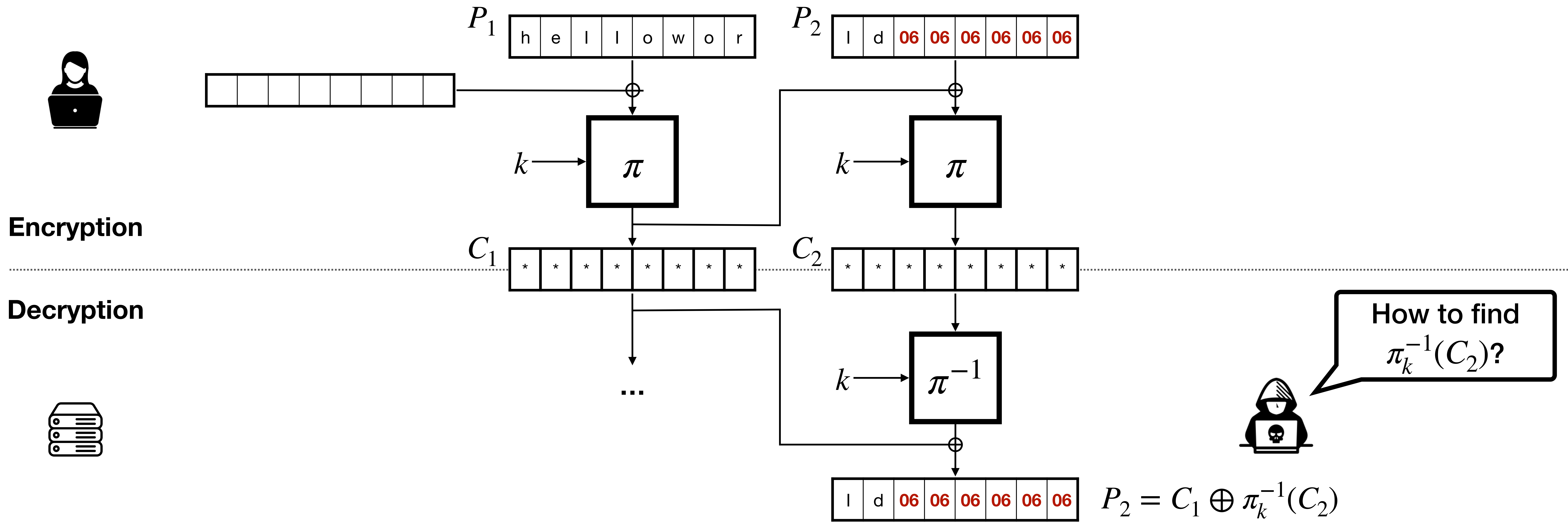
The Azure Storage SDK includes all of the necessary resources that Python, .NET, or Java developers need to build Azure applications that leverage cloud computing resources.

The SDK supports client-side encryption with a customer-managed key that is stored in Azure Key Vault or in a different key store. The previous SDK release uses cipher block chaining (CBC) mode for the encryption.

*Serge Vaudenay, Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS…, Eurocrypt 2022

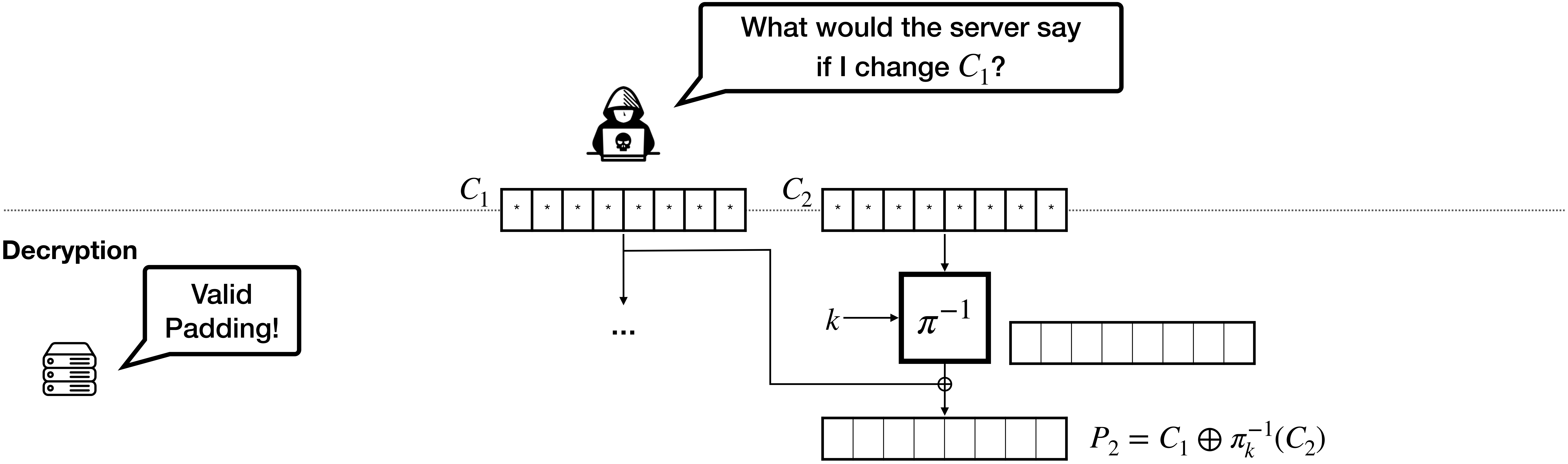
Example (1)

- Assume 64-bit (8 bytes) block size



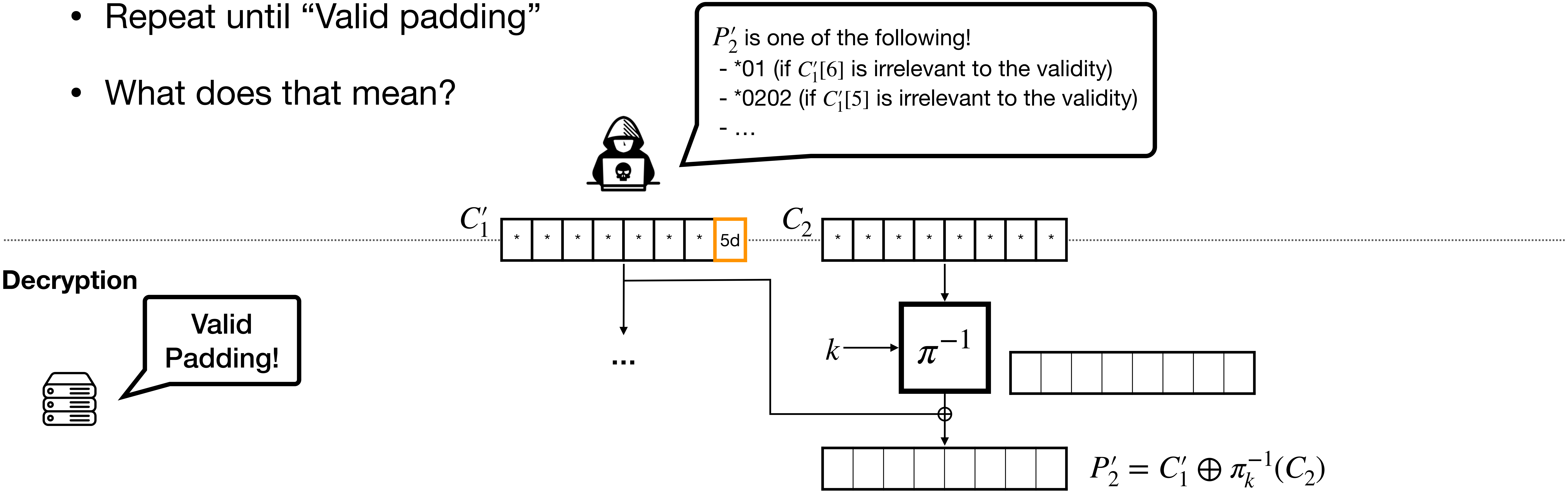
Example (2)

- Assume 64-bit (8 bytes) block size



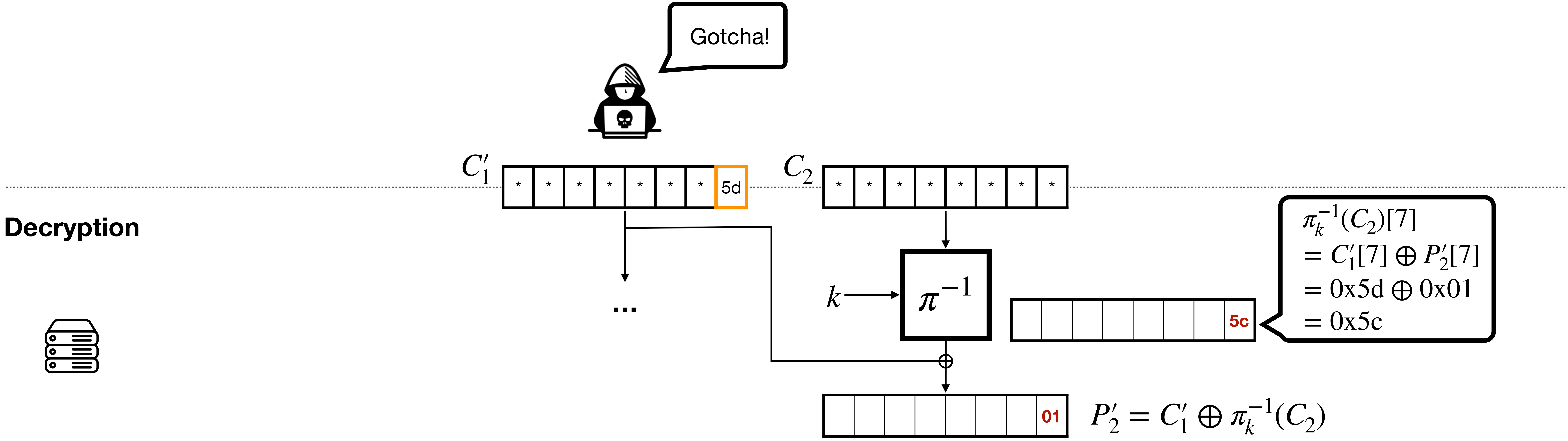
Example (3)

- Construct C_1' by randomly changing the last byte of C_1 and send $C_1' || C_2$ to the oracle
- Repeat until “Valid padding”
- What does that mean?



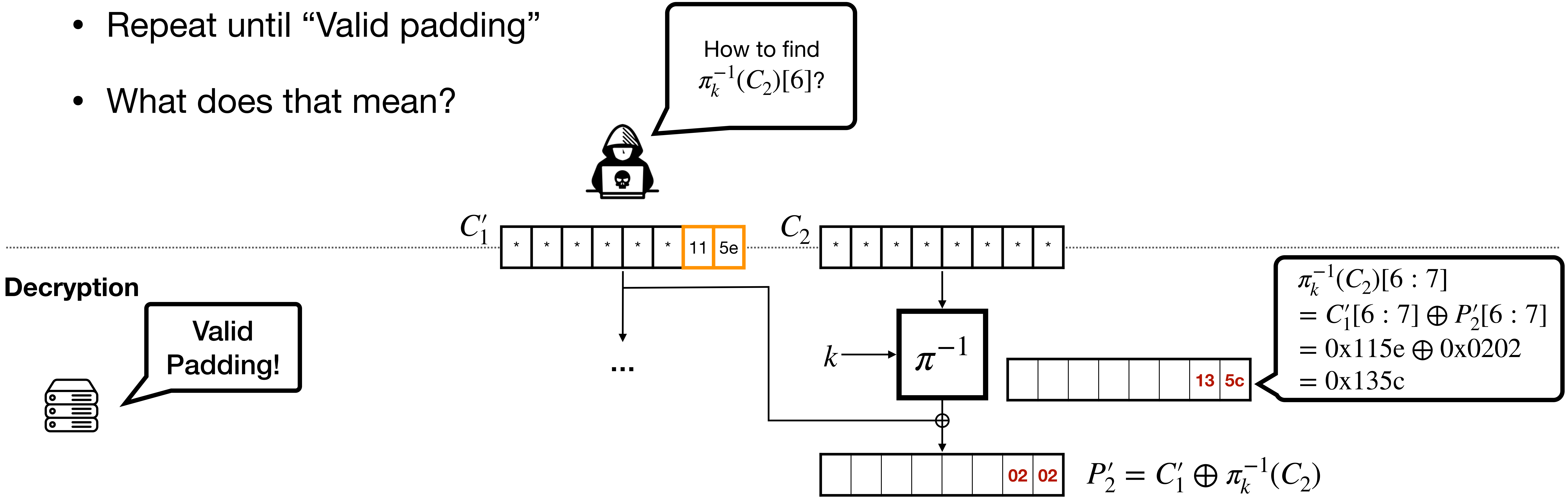
Example (4)

- Suppose we are sure that $P'_2[7] = 0\text{x}01$



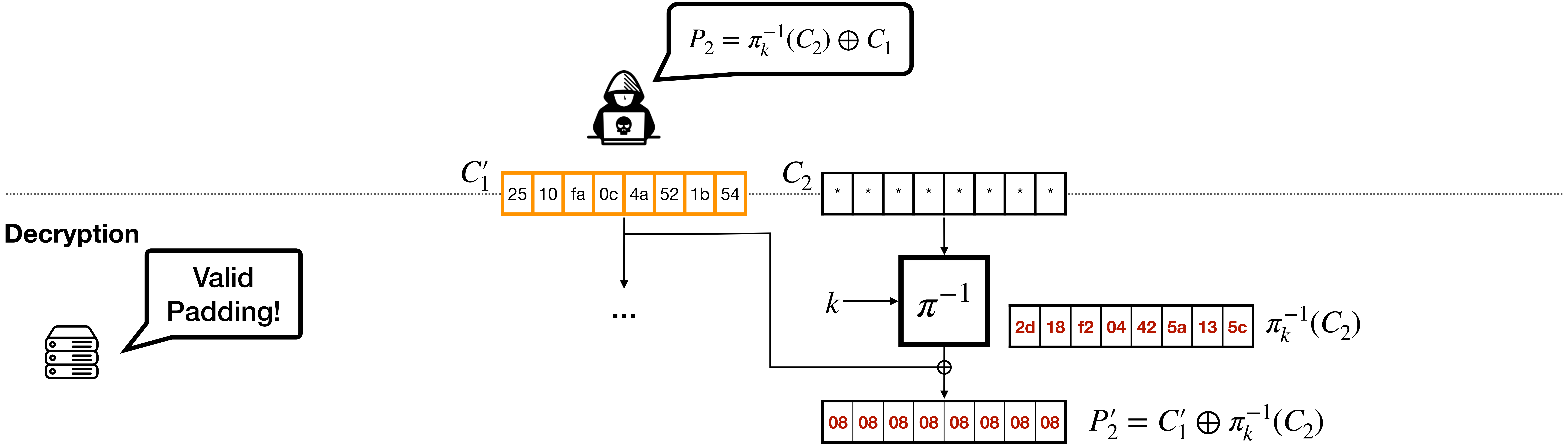
Example (5)

- Construct C_1' by randomly changing the last two bytes of C_1 and send $C_1' \,||\, C_2$ to the oracle
- Repeat until "Valid padding"
- What does that mean?



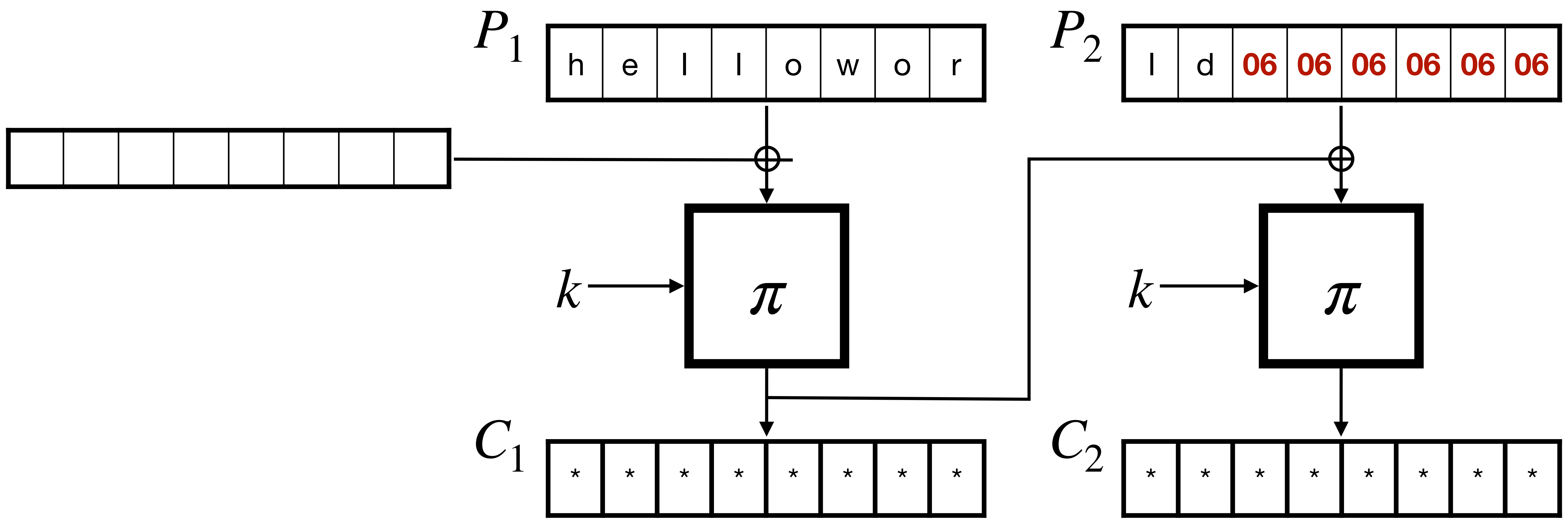
Example (6)

- Finally, $\pi_k^{-1}(C_2)$ will be discovered by the attacker
- Apply the same attack for all the other blocks

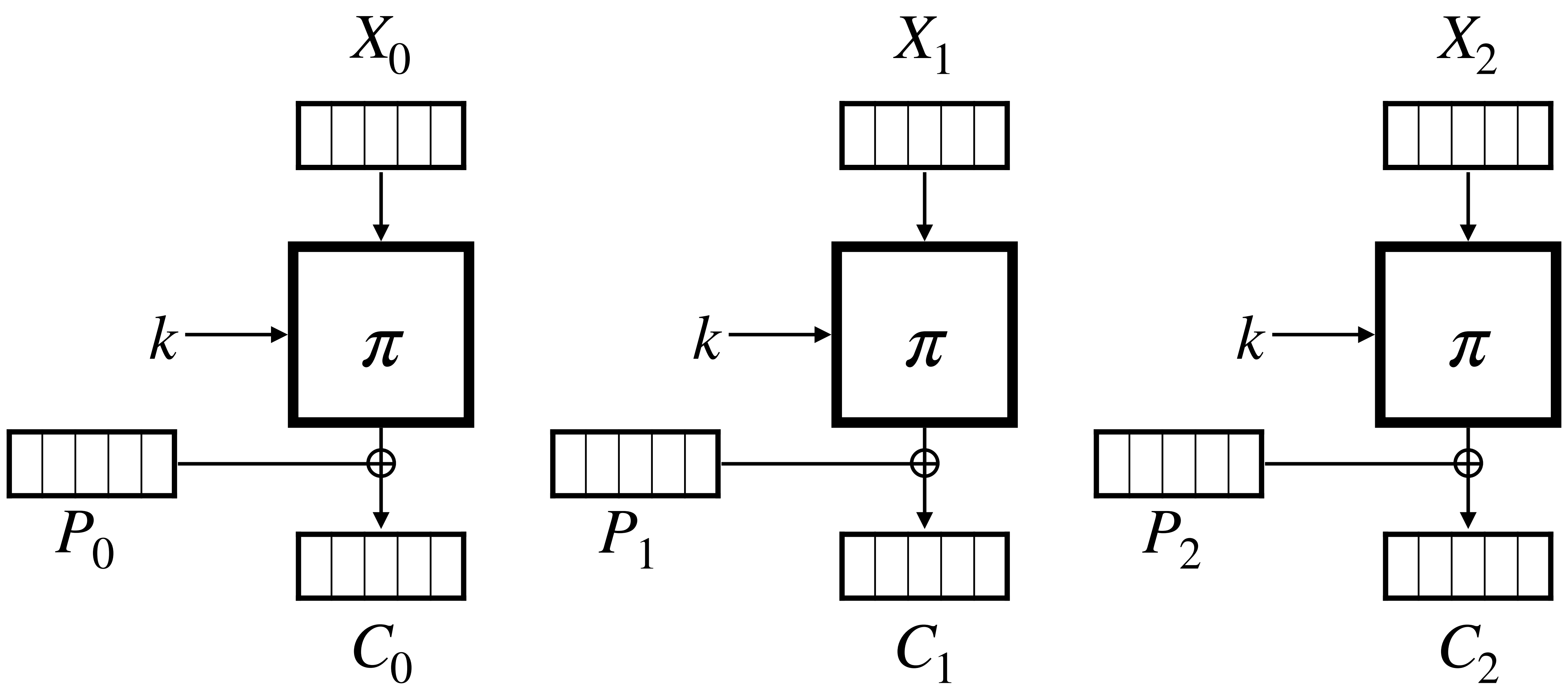


Lessons Learned

- Be careful when you design a secure service based on cryptography
- What if we do not allow such an oracle?
  - Security vs usability
- “Chaining” is not a good idea



Counter Mode (CTR)



$X_0 = IV$
$X_i = X_0 + i$
$C_i = \pi_k(X_i) \oplus P_i$
$P_i = \pi_k(X_i) \oplus C_i$

- Advantages
    - Semantic security and parallelization
- Disadvantages
    - Maintenance of synchronous counters

Summary

- Symmetric-key cryptography: the same key for encryption and decryption
- Vernam cipher (one-time pad): unbreakable but impractical
- Block cipher: basic building block of many schemes using pseudo-random permutation
- Block cipher mode of operations

	Advantages	Disadvantages
ECB	Simple Parallelizable enc / dec	Pattern leackage
CBC	Semantic security	Only dec parallelizable Padding oracle attack
CTR	Semantic security Parallelable enc / dec	Counter maintenance