Symmetric Cryptography

CSIE 7190 Cryptography and Network Security, Spring 2019

https://ceiba.ntu.edu.tw/1072csie_cns

cns@csie.ntu.edu.tw

Hsu-Chun Hsiao

National Taiwan University

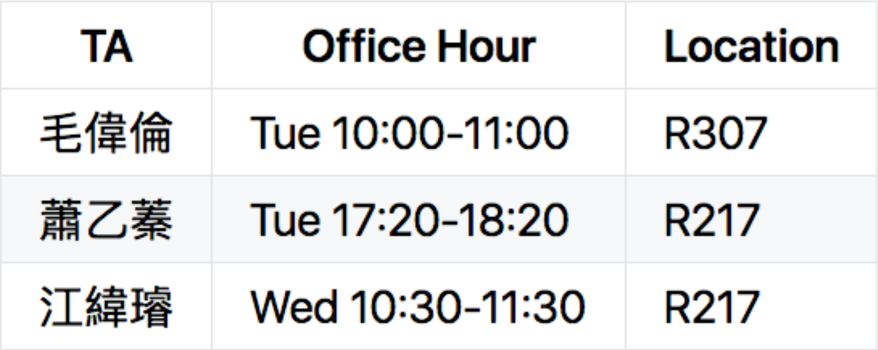
Housekeeping

3/12: HW1 out

3/19 2pm: Reading critique #4 due

4/09: final project team members and topic

TA hour updated



TA	Office Hour	Location
毛偉倫	Tue 10:00-11:00	R307
蕭乙蓁	Tue 17:20-18:20	R217
江緯璿	Wed 10:30-11:30	R217

Reading critique #4

Write a critique on one of the following:

1. L. S. Huang, A. Rice, E. Ellingsen and C. Jackson, "Analyzing Forged SSL Certificates in the Wild," IEEE Symposium on Security and Privacy, San Jose, CA, 2014.
2. Egele, Manuel, et al. "An empirical study of cryptographic misuse in android applications," ACM SIGSAC conference on Computer & communications security. ACM, 2013.

Text only, one page

Agenda

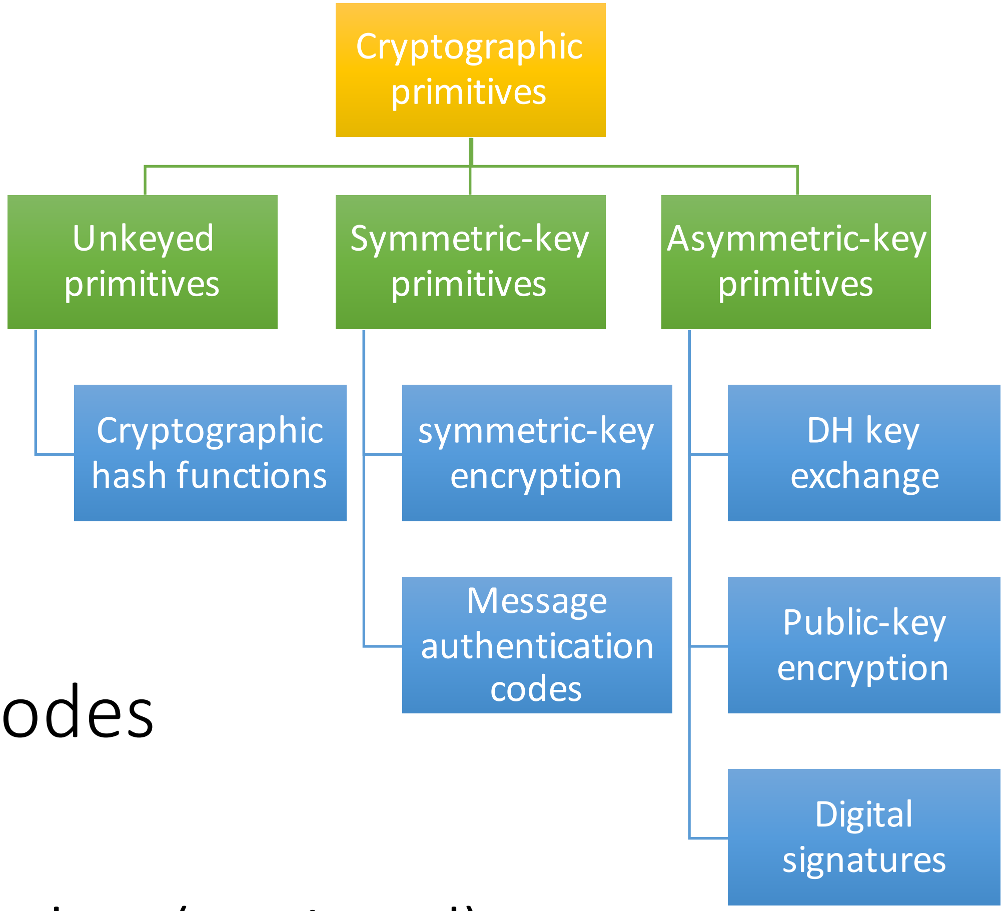
Symmetric encryption

- Security notions
- Block ciphers
- Mode of operations

Message authentication codes

Authenticated encryption

Case Study: RC4 stream cipher (optional)



* what we will cover in this course; not a complete list

Symmetric Encryption

Security notions

Block ciphers

Mode of operations

Encryption != Encoding

Encoding / Decoding (no keys)

- Binary string
- Base64
- Hex
- Morse code

Encryption / Decryption (with keys)

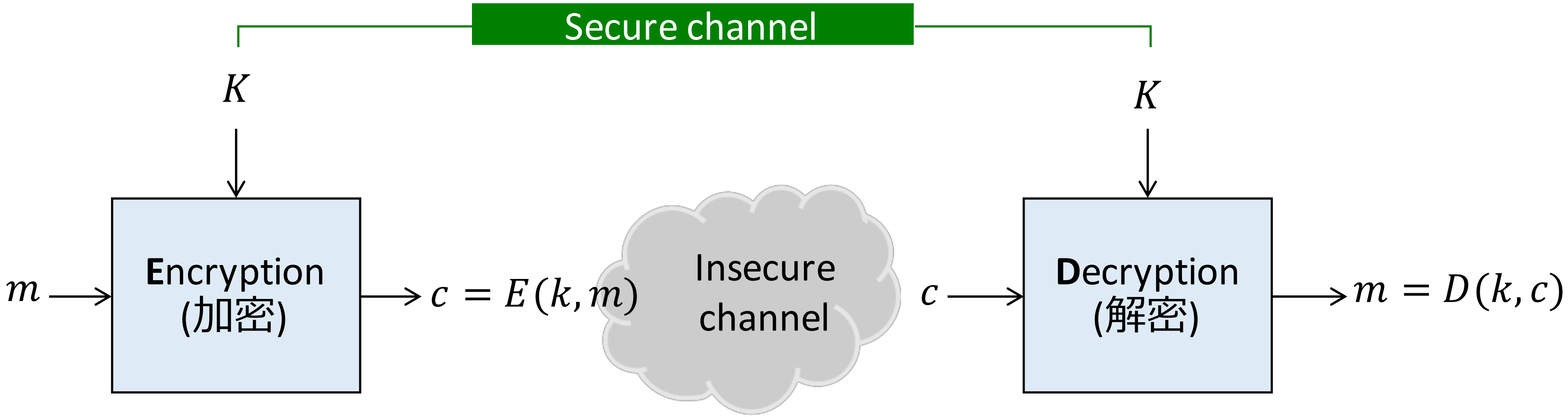
- RC4
- AES
- RSA

Symmetric-key encryption

Assumes Alice and Bob has a shared secret K

- Needs some key distribution scheme to securely share the secret key

Provides confidentiality but not message integrity



Cipher

A *cipher* $\mathcal{E}$ is a mechanism for encrypting a message using a shared secret key.

$\mathcal{E}$ consists of a pair of encryption/decryption functions:

$$\mathcal{E} = (E, D)$$

Encrypt **plaintext** m using **key** k: $c \overset{R}{\leftarrow} E(k, m)$

Decrypt **ciphertext** c using **key** k: $m \leftarrow D(k, c)$

Correctness property: for all keys k, messages m,

$$D(k, E(k, m)) = m$$

Security notions of a cipher

Perfect security

- Ciphertext leaks no information about plaintext
- Confidentiality against unbounded adversaries

Semantic security

- Ciphertext leaks ***negligible*** information about plaintext
- Confidentiality against computationally bounded adversaries

Security notions of a cipher

Perfect security

- Ciphertext leaks no information about plaintext
- Confidentiality against unbounded adversaries
- $\Pr[E(\boldsymbol{k}, m_0) = c] = \Pr[E(\boldsymbol{k}, m_1) = c]$
- It's proven that keys must be at least as long as messages: if an encryption scheme is perfectly secure, then $|\mathcal{K}| \geq |\mathcal{M}|$
- One-time pad provides prefect security

?

Security notions of a cipher

Semantic security

- Ciphertext leaks ***negligible*** information about plaintext
- Confidentiality against computationally bounded adversaries
- $|\Pr[E(\boldsymbol{k}, m_0) = c] - \Pr[E(\boldsymbol{k}, m_1) = c]\,| \leq \varepsilon$, ε is negligible
- A weaker notion than perfect security, so that we can build secure ciphers using reasonably short keys
- When encrypting several messages using the same key, two variants of security notions:
  - Semantic security against chosen plaintext attack (CPA security)
  - Semantic security against chosen ciphertext attack (CCA security)

攻擊者的目標

攻擊者的能力

Classical ciphers

Classical = 古典

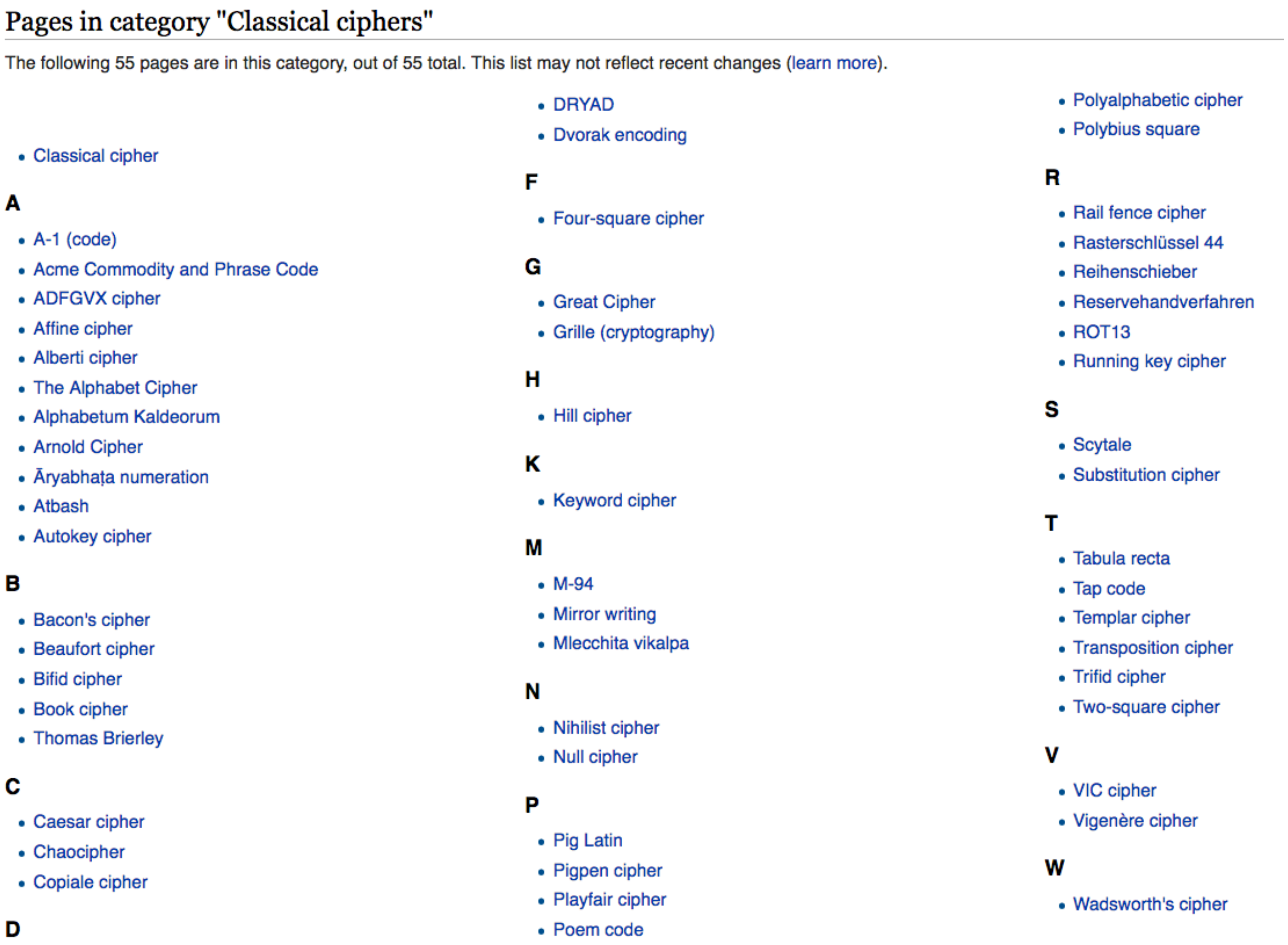
Constructed using *substitution* & *transportation* ops

- Modern ciphers also use *substitution* and *transportation* as the building blocks, but the ways of combination are much more sophisticated.

Examples: Caesar Cipher, ROT13, Affine Cipher, Rail Fence Cipher, …

Most (if not all) are broken, not semantically secure

Classical ciphers



Pages in category "Classical ciphers"

The following 55 pages are in this category, out of 55 total. This list may not reflect recent changes (learn more).

- Classical cipher

A

- A-1 (code)
- Acme Commodity and Phrase Code
- ADFGVX cipher
- Affine cipher
- Alberti cipher
- The Alphabet Cipher
- Alphabetum Kaldeorum
- Arnold Cipher
- Āryabhaṭa numeration
- Atbash
- Autokey cipher

B

- Bacon's cipher
- Beaufort cipher
- Bifid cipher
- Book cipher
- Thomas Brierley

C

- Caesar cipher
- Chaocipher
- Copiale cipher

D

- DRYAD
- Dvorak encoding

F

- Four-square cipher

G

- Great Cipher
- Grille (cryptography)

H

- Hill cipher

K

- Keyword cipher

M

- M-94
- Mirror writing
- Mlecchita vikalpa

N

- Nihilist cipher
- Null cipher

P

- Pig Latin
- Pigpen cipher
- Playfair cipher
- Poem code
- Polyalphabetic cipher
- Polybius square

R

- Rail fence cipher
- Rasterschlüssel 44
- Reihenschieber
- Reservehandverfahren
- ROT13
- Running key cipher

S

- Scytale
- Substitution cipher

T

- Tabula recta
- Tap code
- Templar cipher
- Transposition cipher
- Trifid cipher
- Two-square cipher

V

- VIC cipher
- Vigenère cipher

W

- Wadsworth's cipher

A perfectly secure (yet impractical) cipher: One-time pad

XOR message with a random key of same length

$$c = E(k,m) = k \oplus m$$

$$m = D(k,c) = k \oplus c$$

http://users.telenet.be/d.rijmenants/en/onetimepad.htm

Modern ciphers

Stream ciphers

用short secret 產生看似random的long key stream

Encrypt one symbol at a time

Example: RC4

Block ciphers

把訊息拆成固定長度的短block，一個一個處理

Encrypt one block (a group of symbols) at a time

Example: DES, AES

Block Ciphers

Block cipher

A **block cipher** (a.k.a. pseudo-random permutation) is a
不可以 deterministic cipher $\mathcal{E} = (E, D)$ whose message &
ciphertext spaces are the same **finite** set $\mathcal{X}$

- Let $\mathcal{K}$ be the **key space**
- $\mathcal{E}$ is defined over $(\mathcal{K}, \mathcal{X})$

Define $f_k \coloneqq E(k,\cdot)$ for every fixed key $k \in \mathcal{K}$

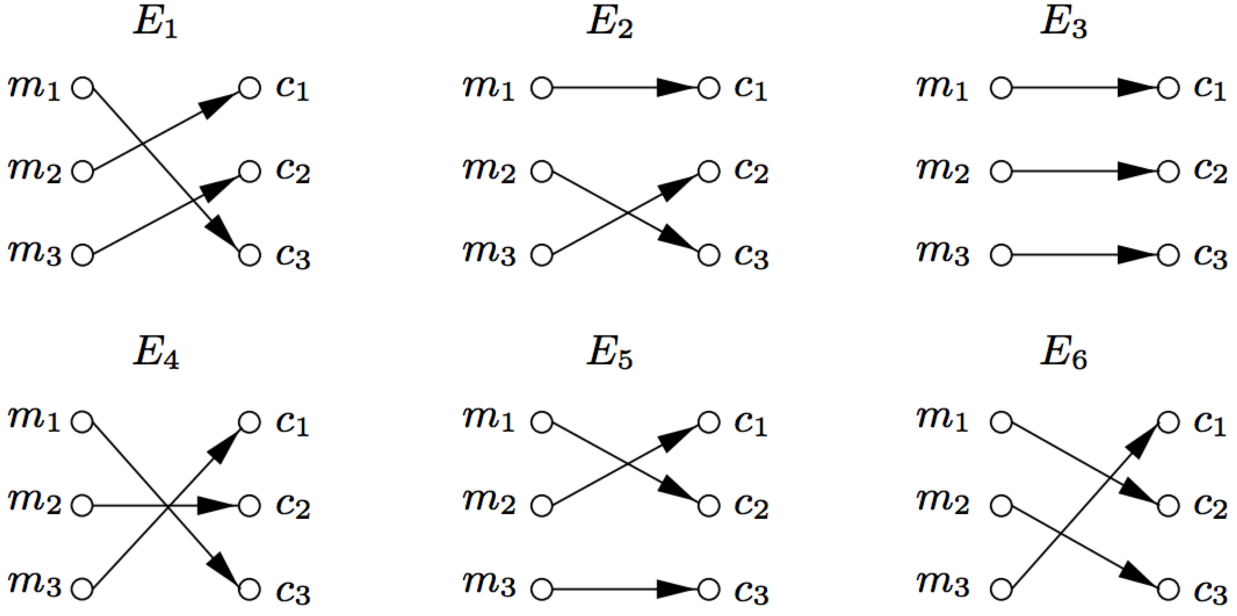
Correctness property:

- f_k is a **permutation** on $\mathcal{X}$
- $\mathrm{D}(k,\cdot)$ is the inverse permutation of f_k

Informally, a secure block cipher should ensure $E(k,\cdot)$ looks like a random permutation for randomly chosen k

Permutation from message space to ciphertext space

When $|\mathcal{X}| = 3$, # of possible permutations = ?



Permutation from message space to ciphertext space

For n-bit m and n-bit c,(i.e., $|\mathcal{X}| = 2^n$), # of possible permutations = ? $\boxed{2^n!}$

A key can be viewed as an index of such permutations

A truly random permutation would require $|\mathcal{K}| = 2^n!$

In practice, we consider pseudo-random permutation where $|\mathcal{K}| \ll 2^n!$

- Longer key, larger key space = (usually) stronger security

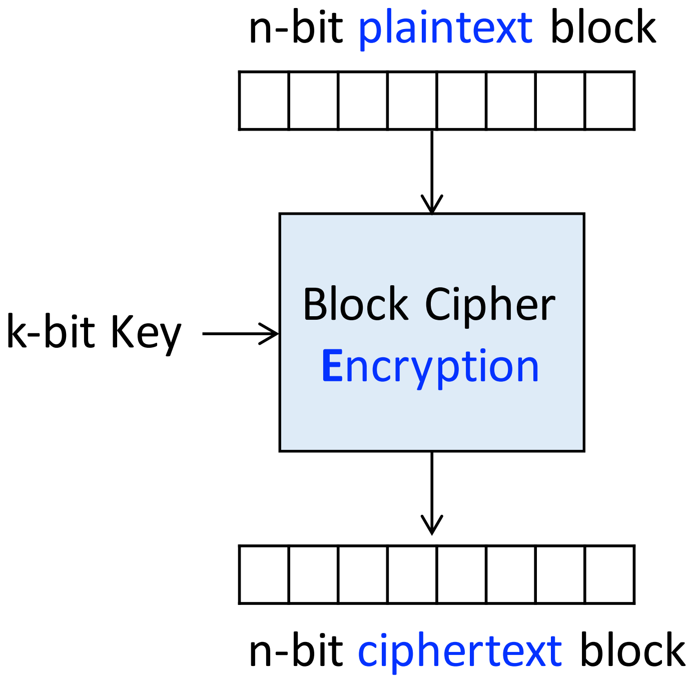
Block cipher examples

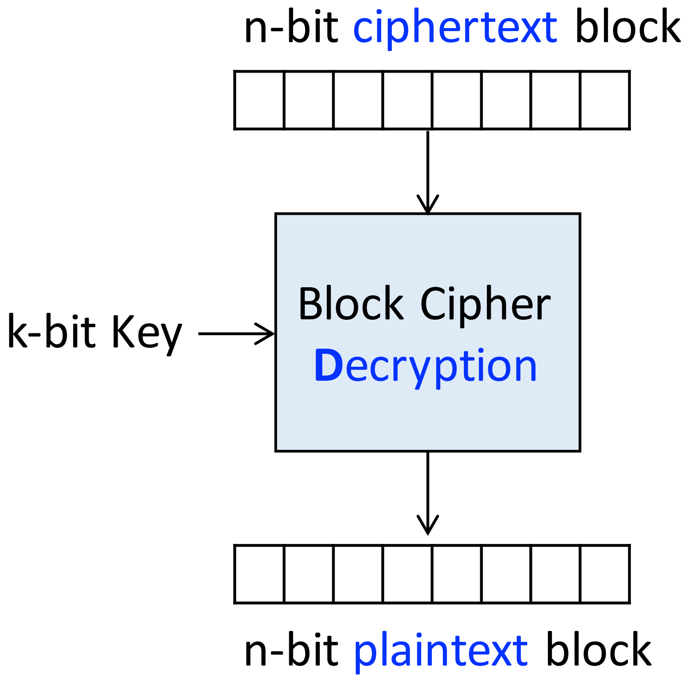
DES: n = 64, k = 54

3DES: n = 64, k = 54, 112, 168

AES: n = 128, k = 128, 192, 256

Q: Why not 2DES?





Advanced Encryption Standard (AES)

AES: $n = 128$, $k = 128, 192, 256$

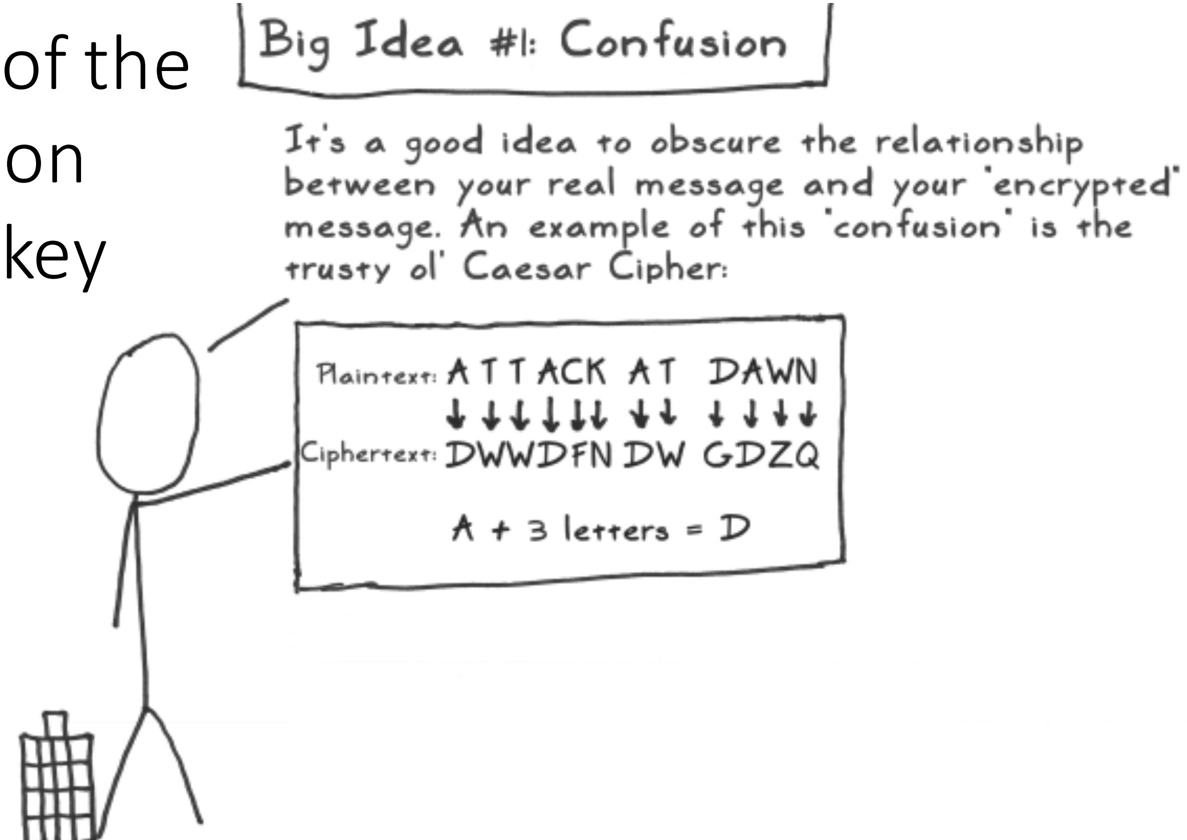
Considered secure for the time being, as no one has made it *computationally* feasible to break it

It has not been proven secure

Confusion and Diffusion in AES

Confusion and diffusion are two key properties of a secure cipher (by Claude Shannon)

Confusion: each bit of the ciphertext depends on several parts of the key

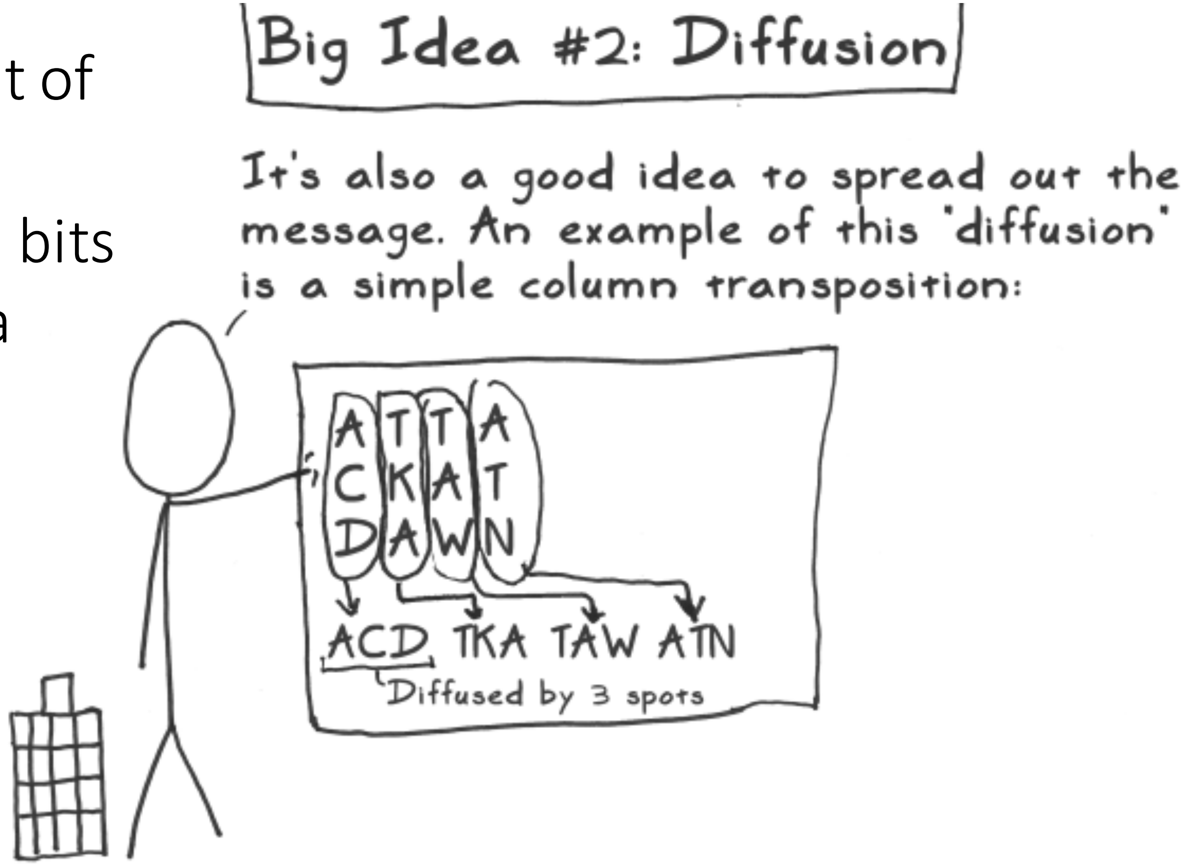

Big Idea #1: Confusion
It's a good idea to obscure the relationship between your real message and your 'encrypted' message. An example of this 'confusion' is the trusty ol' Caesar Cipher:
Plaintext: ATTACK AT DAWN
Ciphertext: DWWDFN DW GDZQ
A + 3 letters = D

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

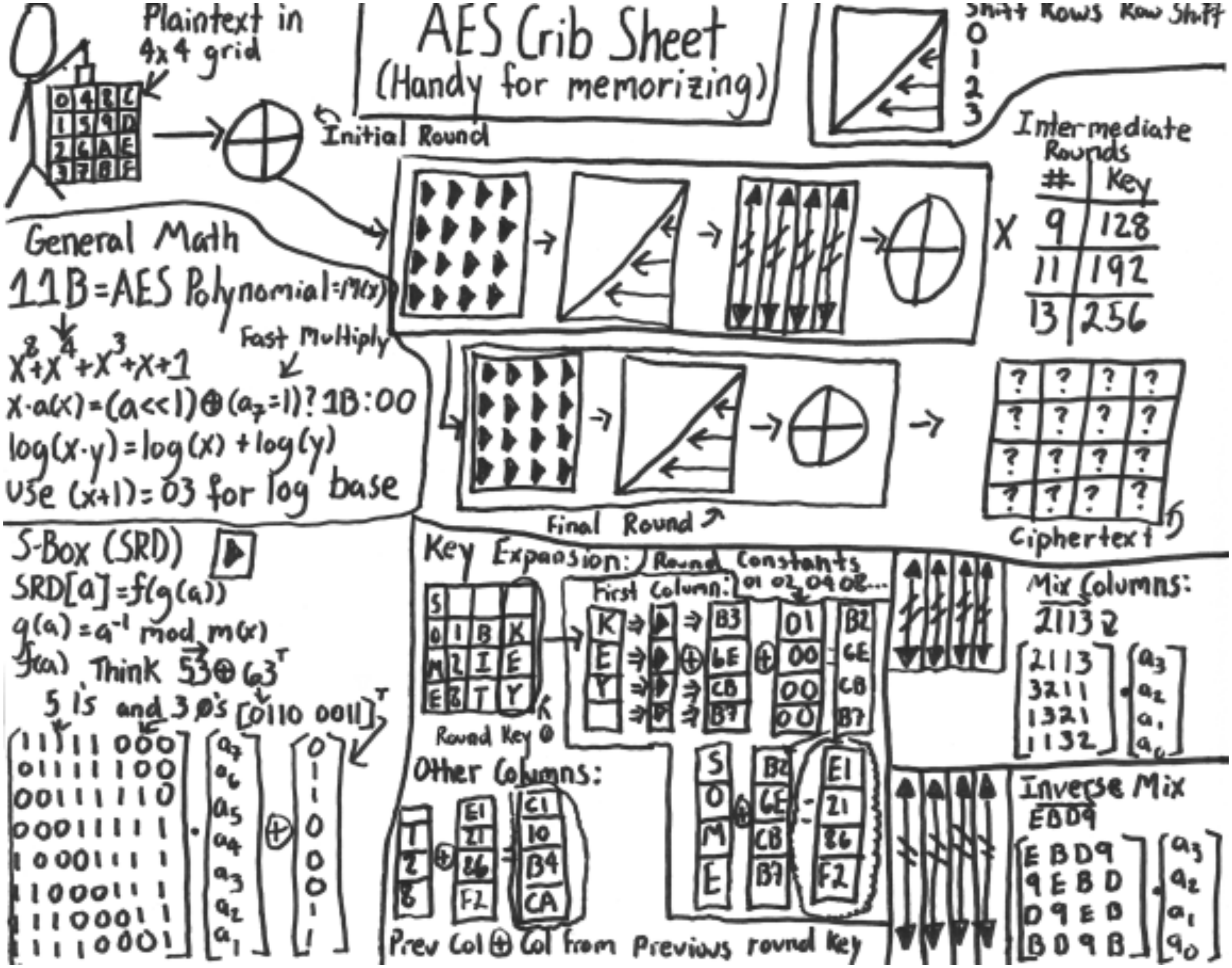
Confusion and Diffusion in AES

Confusion and diffusion are two key properties of a secure cipher (by Claude Shannon)

Diffusion: changing a bit of the plaintext changes (statistically) half of the bits in ciphertext; vice versa


Big Idea #2: Diffusion
It's also a good idea to spread out the message. An example of this "diffusion" is a simple column transposition:
ATTA
CKAT
DAWN
ACD TKA TAW ATN
Diffused by 3 spots

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html



http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

Sophisticated attacks on block ciphers

Algorithmic attacks: clever analysis of the internal structure of a particular block cipher

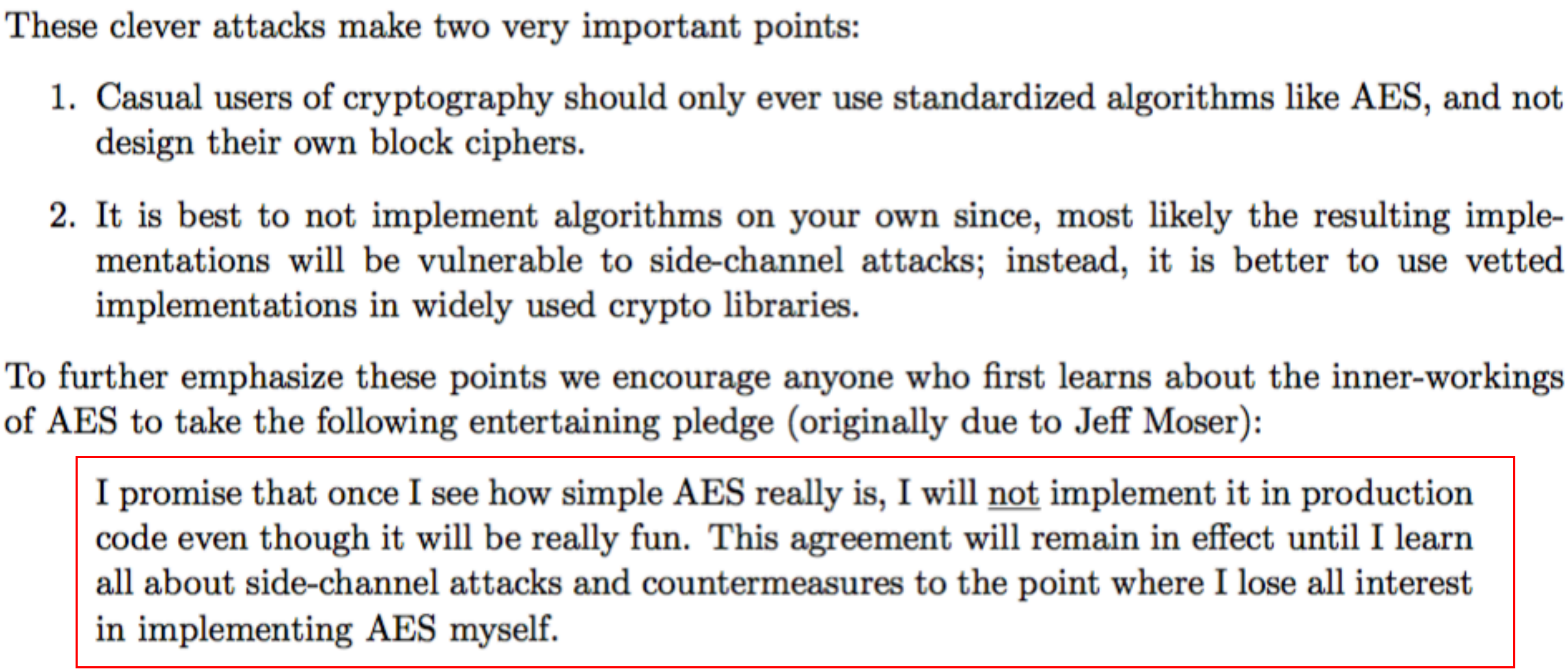
Side-channel attacks: computation is a physical process. The adversary may break a cryptosystem by measuring physical characteristics of the users' computation (e.g., running time, power consumption)

Fault-injection attack: attacks on physical implementation

Quantum mechanics: speed up computation

* Ref: Boneh & Shoup, Ch. 4.3

Why you shouldn’t implement crypto algorithms by yourself



These clever attacks make two very important points:

1. Casual users of cryptography should only ever use standardized algorithms like AES, and not design their own block ciphers.

2. It is best to not implement algorithms on your own since, most likely the resulting implementations will be vulnerable to side-channel attacks; instead, it is better to use vetted implementations in widely used crypto libraries.

To further emphasize these points we encourage anyone who first learns about the inner-workings of AES to take the following entertaining pledge (originally due to Jeff Moser):

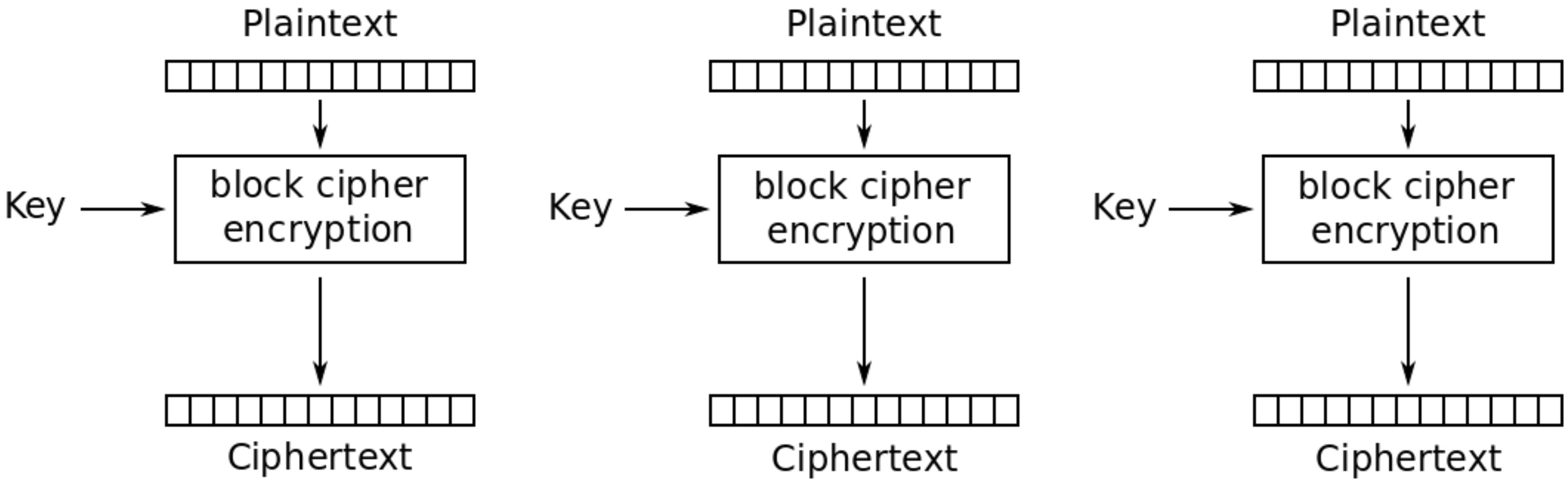
> I promise that once I see how simple AES really is, I will not implement it in production code even though it will be really fun. This agreement will remain in effect until I learn all about side-channel attacks and countermeasures to the point where I lose all interest in implementing AES myself.

* Ref: Boneh & Shoup, Ch. 4.3, pp. 120

How to encrypt a message longer than *n*?

A block cipher (e.g., AES) takes a fixed length input

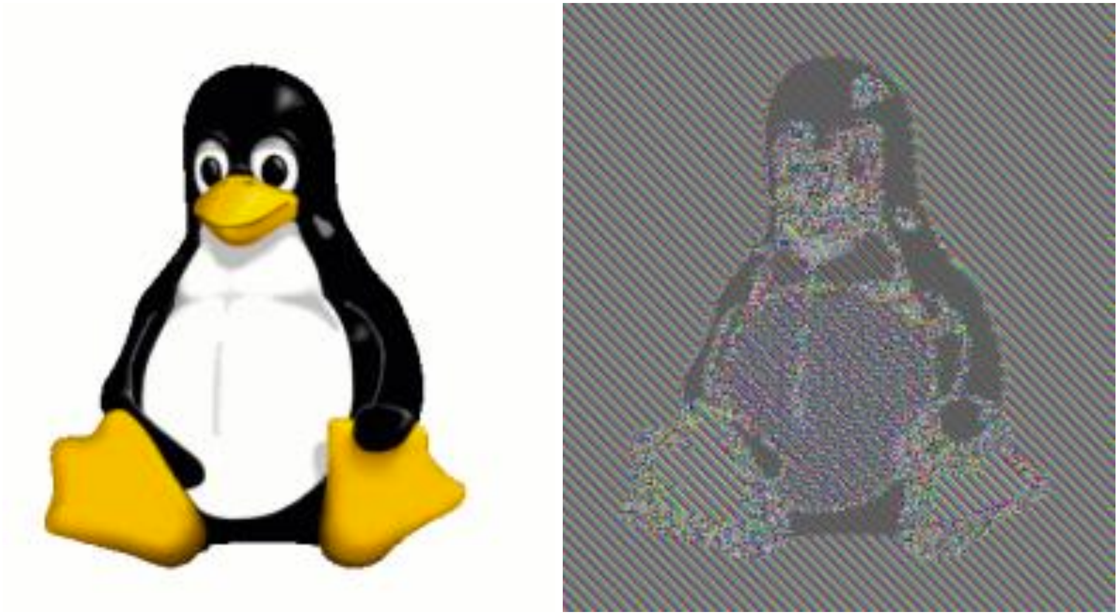
Naïve approach: Breaking messages into n-bit blocks and encrypting each of them individually?



Bad idea…

Why is it bad?

Identical plaintexts = identical ciphertexts



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Modes of Operation

Modes of Operation

Modes of operation defines how to encrypt a long message by repeatedly applying a block cipher

- ECB (not recommended, the insecure one we just saw)
- CBC
- CTR
- GCM (recommended, because it also ensures integrity)
- …

Can be applied to any block cipher

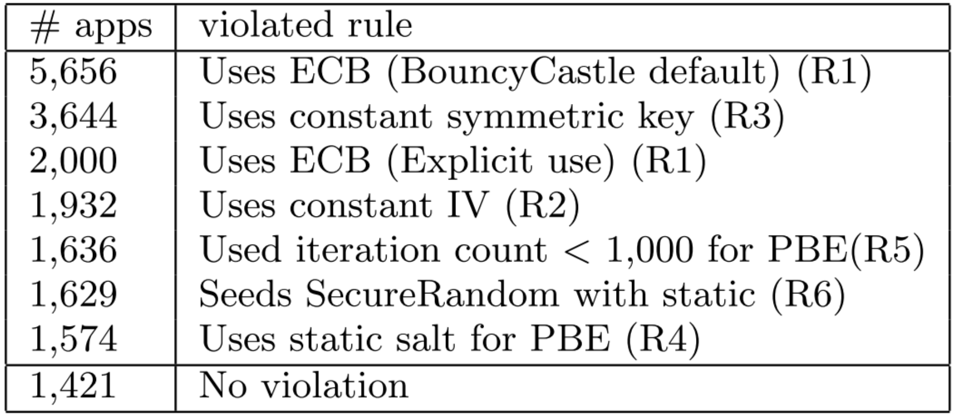
ECB mode

Does not satisfy semantic security

One of the most common crypto configuration errors

Attack examples

- Encryption oracle attack
- Cut-and-paste attack



# apps	violated rule
5,656	Uses ECB (BouncyCastle default) (R1)
3,644	Uses constant symmetric key (R3)
2,000	Uses ECB (Explicit use) (R1)
1,932	Uses constant IV (R2)
1,636	Used iteration count < 1,000 for PBE(R5)
1,629	Seeds SecureRandom with static (R6)
1,574	Uses static salt for PBE (R4)
1,421	No violation

M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of ACM CCS*, 2013.

Encryption Oracle Attack (against ECB Mode)

Oracle = 神諭、預報

- Ask a question, the oracle responds with a (sometimes obscure) answer
- We will see more oracle attacks later

Encryption Oracle

- A “service” that will return encrypted data of your chosen plaintext
- Example: server returning a cookie that is a ciphertext encrypting the user input

A chosen plaintext attack

Encryption Oracle Attack

$E(k,\cdot)$ is a block cipher encryption in ECB mode

The attacker can control A, and the oracle will return

$$E(k, P||A||S)$$

The attacker wants to recover S without knowing k, P, S

Encryption Oracle Attack

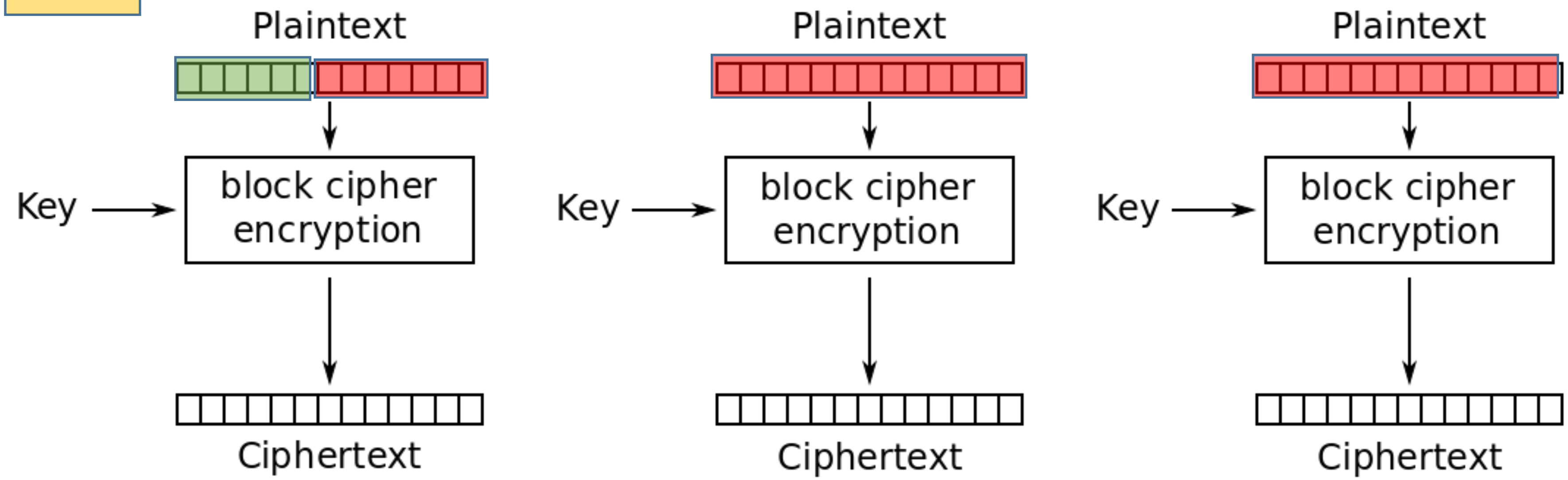
Step 1: Send long enough A s.t. it ends at the block boundary (say, A = ‘aaa...aa’, the length is L)

- You can try until observing a repeated pattern

P 🟩

A 🟥

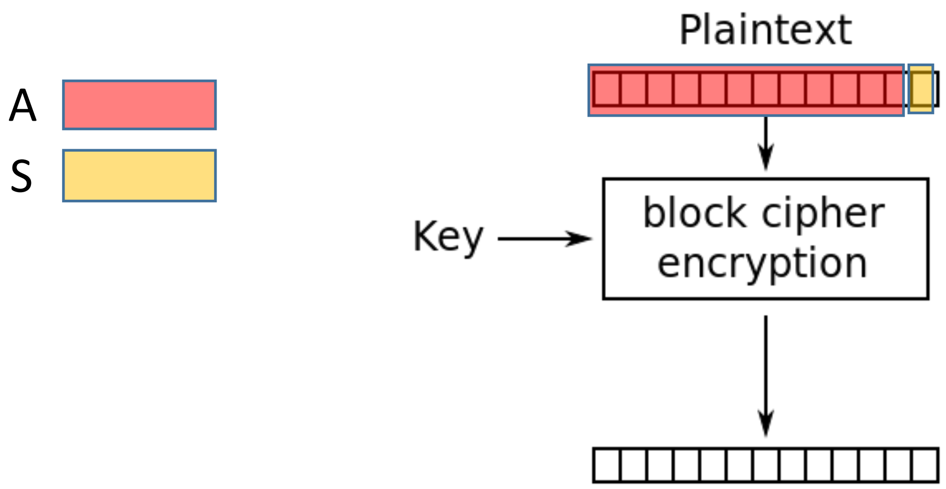
S 🟨



Encryption Oracle Attack

Step 2: Send a $L-1$ number of ‘a’

- Obtain the encryption block of $E(k, aaa \ldots a || s_0)$
- s_0 is the first byte of S

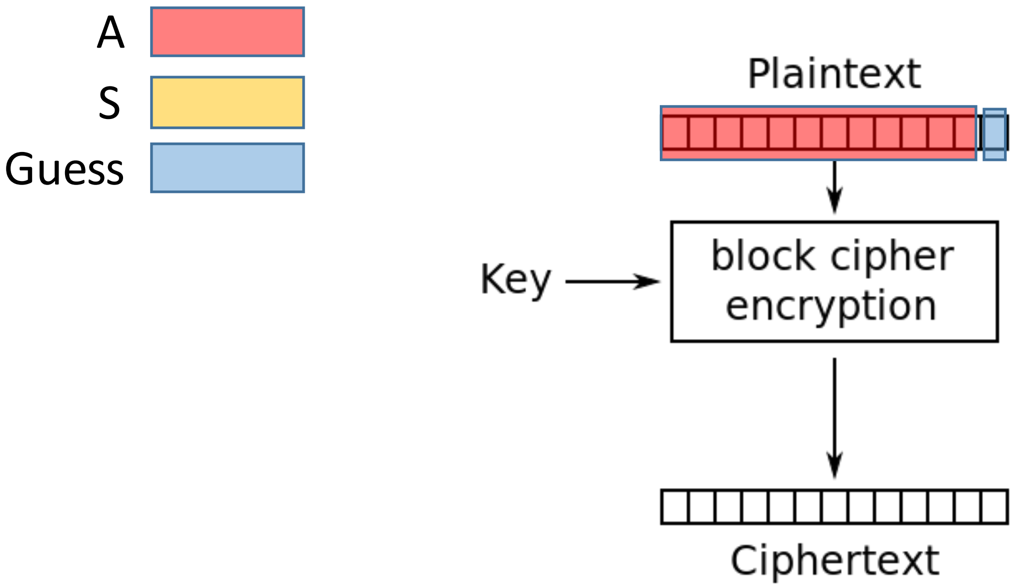


$$E(k, aaa \ldots a || s_0)$$

Encryption Oracle Attack

Step 3: Guess s_0, the first byte of S

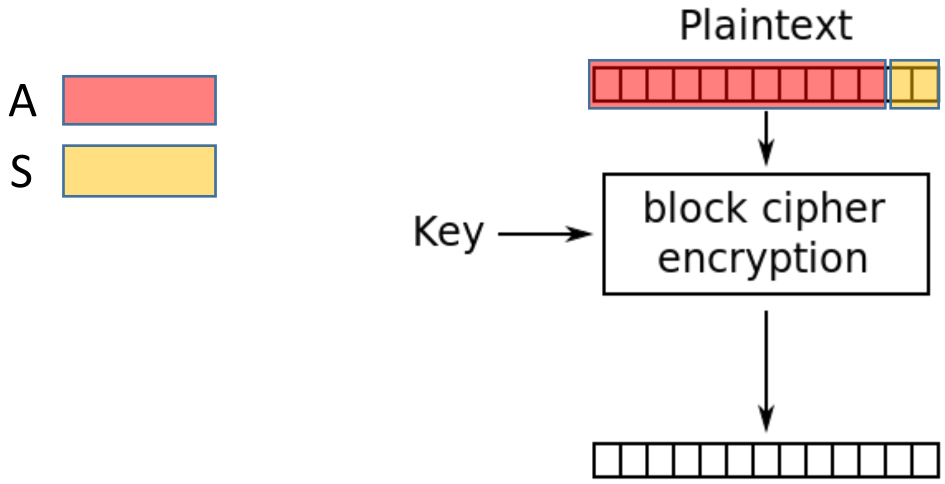
- Send $L-1$ number of ‘a’ plus a guess of s_0 🟦
- Enumerate all possible s_0 (only 256 possibilities) to see if it equals to $E(k, aaa \ldots a||s_0)$



Encryption Oracle Attack

Step 4: Send a $L-2$ number of ‘a’

- Obtain the encryption block of $E(k, aaa \ldots a||s_0||s_1)$
- $s_{0,}\, s_1$ are the first two bytes of S



$E(k, aaa \ldots a||s_0||s_1)$

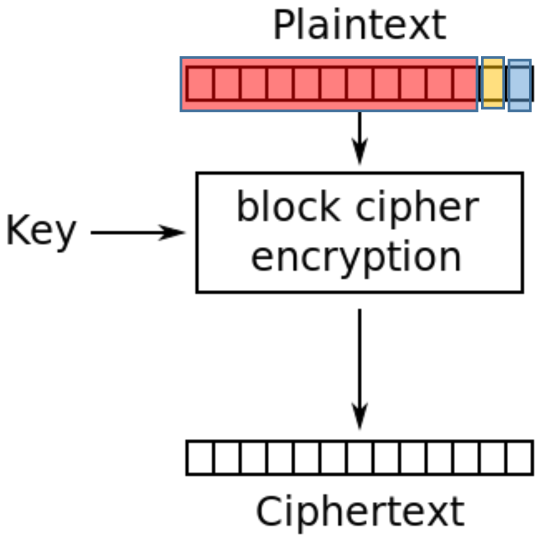
Encryption Oracle Attack

Step 5: Guess s_1, the second byte of S

- Send $L-2$ number of ‘a’, known s_0, and the guess of s_1 🟦
- Enumerate all possible s_1 (only 256 possibilities) to see if it equals to $E(k, aaa \ldots a||s_0||s_1)$

A 🟥

S 🟨



Encryption Oracle Attack

Recover one byte at a time

Repeat the process until recovering full S

O(256*# of bytes)

Cut-and-paste Attack (against ECB Mode)

$E(k,\cdot)$ is a block cipher encryption in ECB mode

The attacker can control A, and the oracle will return

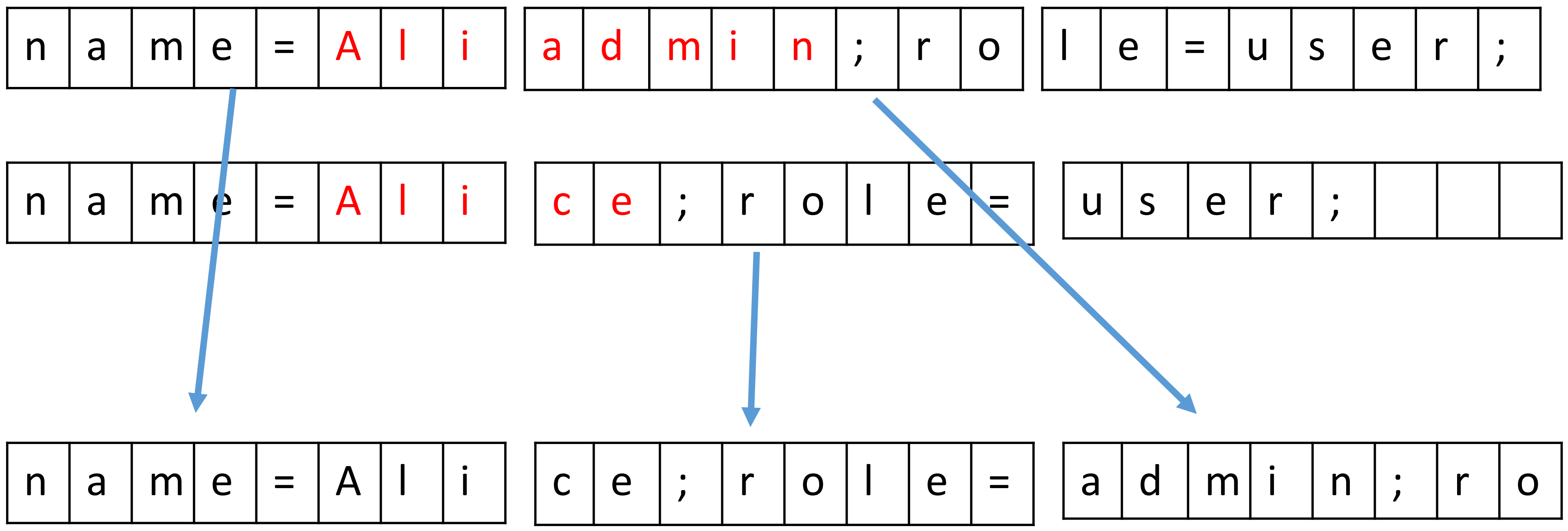
$$E(k, P||A||S)$$

You want to obtain $E\left(k, P\middle|\middle|A\middle|\middle|S'\right)$ for a specific S' without knowing P, k

Example

- A cookie is $E(k, name = alice||role = user)$ username is user-controllable, but not the role
- You want to obtain $E(k, name = alice||role = admin)$

Cut-and-paste Attack



name=Ali admin;ro le=user;

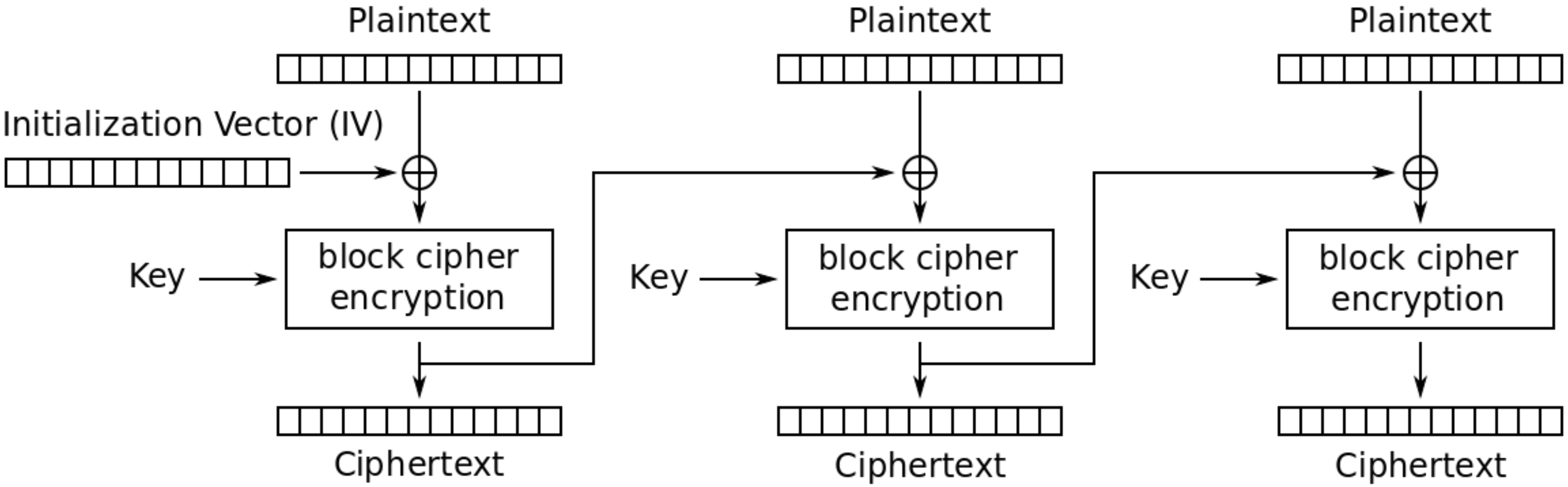
name=Ali ce;role= user;

name=Ali ce;role= admin;ro

CBC mode

Encryption: $c_i = E(k, m_i \oplus c_{i-1}), c_0 = IV$

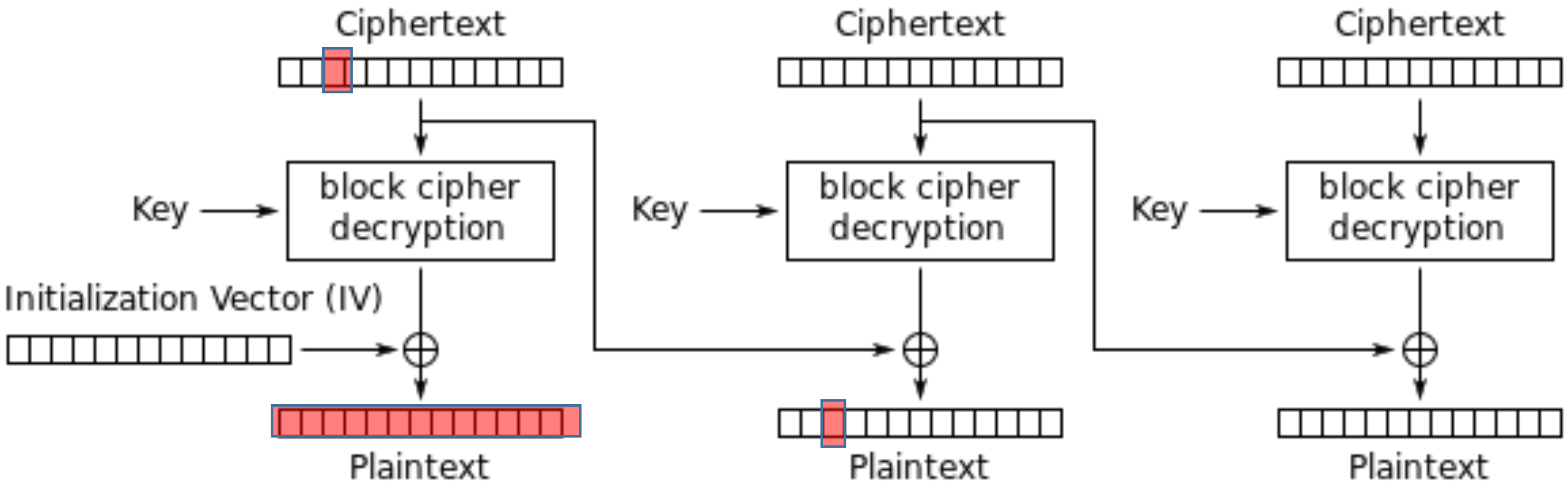
IV is sent with the ciphertext but should be unpredictable



Cipher Block Chaining (CBC) mode encryption

CBC mode

Decryption: $m_i = D(k, c_i) \oplus c_{i-1}, c_0 = IV$



Cipher Block Chaining (CBC) mode decryption

Mode of operation: CBC mode

A ciphertext block depends on all plaintext blocks up to this point

- Cut-and-paste may not work

An initialization vector (IV) adds randomness, ensuring unique ciphertext

- IV doesn’t need to be secret
- IV should be unpredictable and unique

Same plaintext block -> different ciphertext block

Attack examples

- BEAST attack
- CBC padding oracle attack

Problem with Predictable IV in CBC mode

The attacker can perform *chosen-plaintext attack* and guess the plaintext

- We will talk more about CPA later

BEAST (Browser Exploit Against SSL/TLS) attack

In TLS 1.0, the last ciphertext block of a message is reused as the IV for the next message.

Problem with Predictable IV in CBC mode

$E(k,\cdot)$ is a block cipher encryption in CBC mode

Suppose the attacker can predict the next IV and can perform chosen-plaintext attacks (i.e., obtaining the ciphertext of the plaintext you choose)

1. The attacker observes c and IV I, and wants to guess m

$$c = E(k, m \oplus I)$$

2. The attacker predicts the next IV to be I', and would like to check whether the previous plaintext is m'.
3. The attacker submits plaintext $m' \oplus I \oplus I'$, and obtains ciphertext c'
4. If $c' = c$, then the guess is correct $(m' = m)$

若 next IV = I', $c' = E(m' \oplus I \cancel{\oplus I' \oplus I'})$

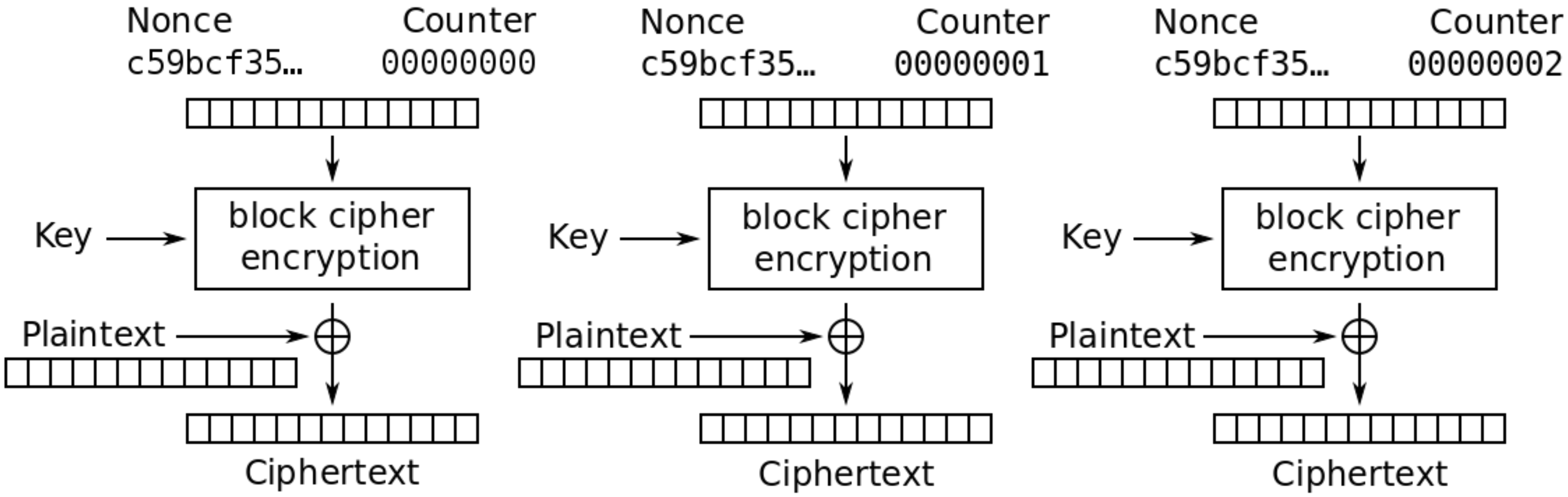
$= E(m' \oplus I)$

Mode of operation: CTR mode

Introduced by Whitfield Diffie and Martin Hellman in 1979

Turns a block cipher into a stream cipher

不會受到 padding attack

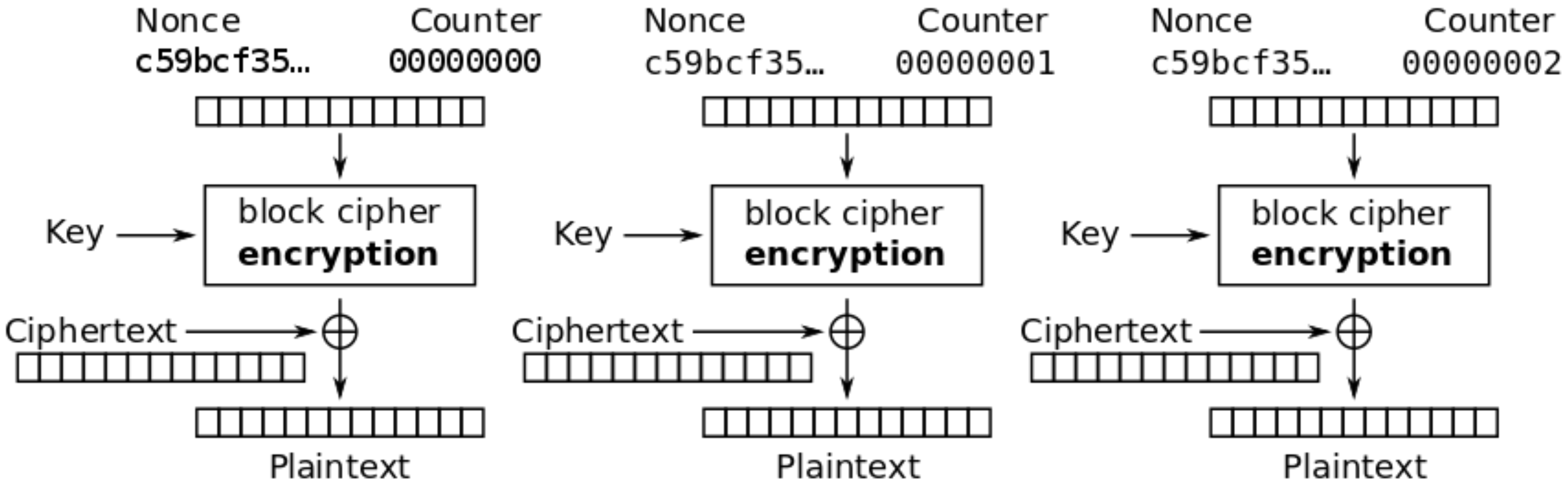


沒有對 plaintext 做 constraint

Counter (CTR) mode encryption

Mode of operation: CTR mode

Note: we also use the encryption function here!



Counter (CTR) mode decryption

因為我們對 Nonce 做 encrypt，plaintext 長度不用是 n 的整數倍

What happen when plaintext size is not a multiple of n?

Some modes of operation don’t require the plaintext size to be divisible by n

- E.g., CTR, OFB

Sometimes we can apply the ciphertext stealing (CTS)

- Not an attack
- No expansion of ciphertext
- E.g., CBC, ECB

Padding to the block boundary

- Padding might introduce additional security vulnerabilities as we will discuss later

CTR vs. CBC

CTR is better than CBC

- Parallelism and pipelining
- Shorter ciphertext length
- Encryption only

彼此沒有 dependency，因為用 Nonce 加密

Both require a random IV

- CTR: IVs are sufficiently spread out
- CBC: IVs are unpredictable

每一次 rand seed 的 Nonce 要很不一樣

More on Security Notions

Recall “Semantic Security”

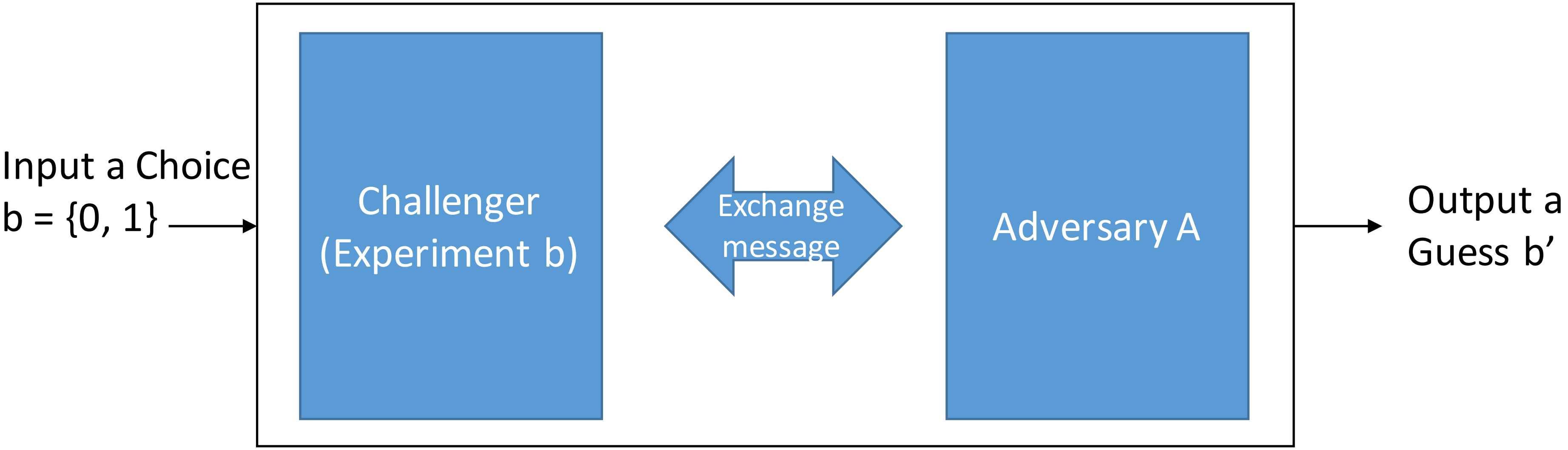
Semantic security

- Ciphertext leaks ***negligible*** information about plaintext
- $|\Pr[E(\boldsymbol{k}, m_0) = c] - \Pr[E(\boldsymbol{k}, m_1) = c]\,| \leq \varepsilon$, ε is negligible

What does negligible mean? How do we mathematically model this?

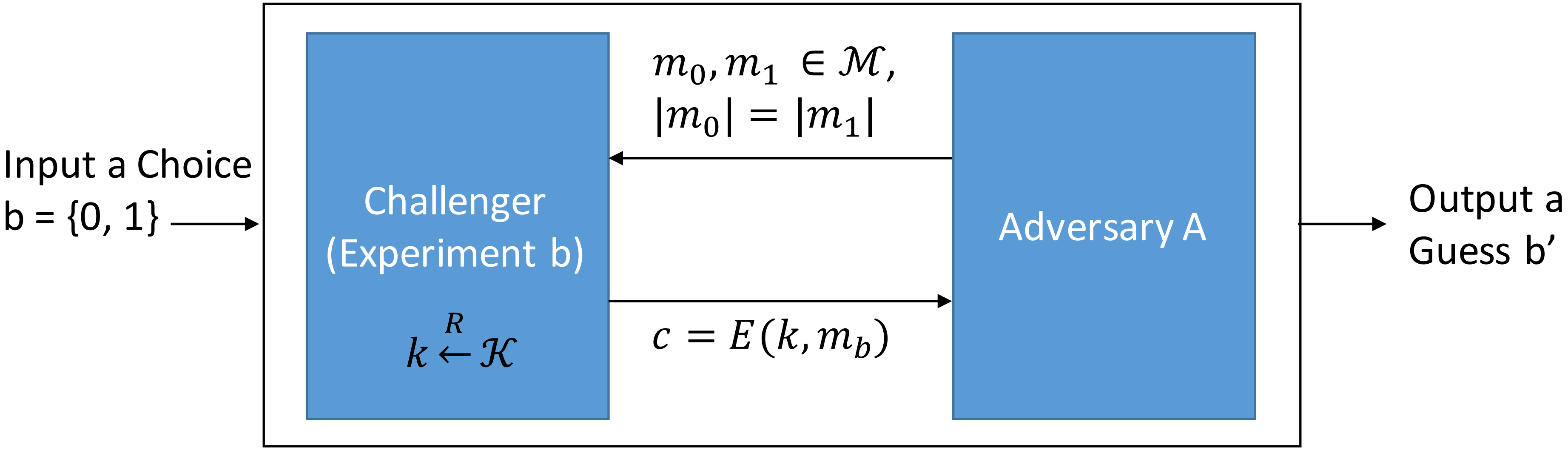
Game-based security definition

Attack game played between two parties:
the challenger and an adversary



Attacker’s **advantage** = |Pr[Exp(0) = 1] – Pr[Exp(1) = 1]|

Semantic security for one-time key



不管你做 0 或做 1，你猜對的機率都差不多

Attacker's **advantage** = Adv$_{SS}[A, \mathcal{E}]$ = |Pr[Exp(0) = 1] – Pr[Exp(1) = 1]|

A cipher $\mathcal{E} = (E, D)$ is semantic secure if the advantage is negligible ($\leq \varepsilon$) for all computationally bounded adversary

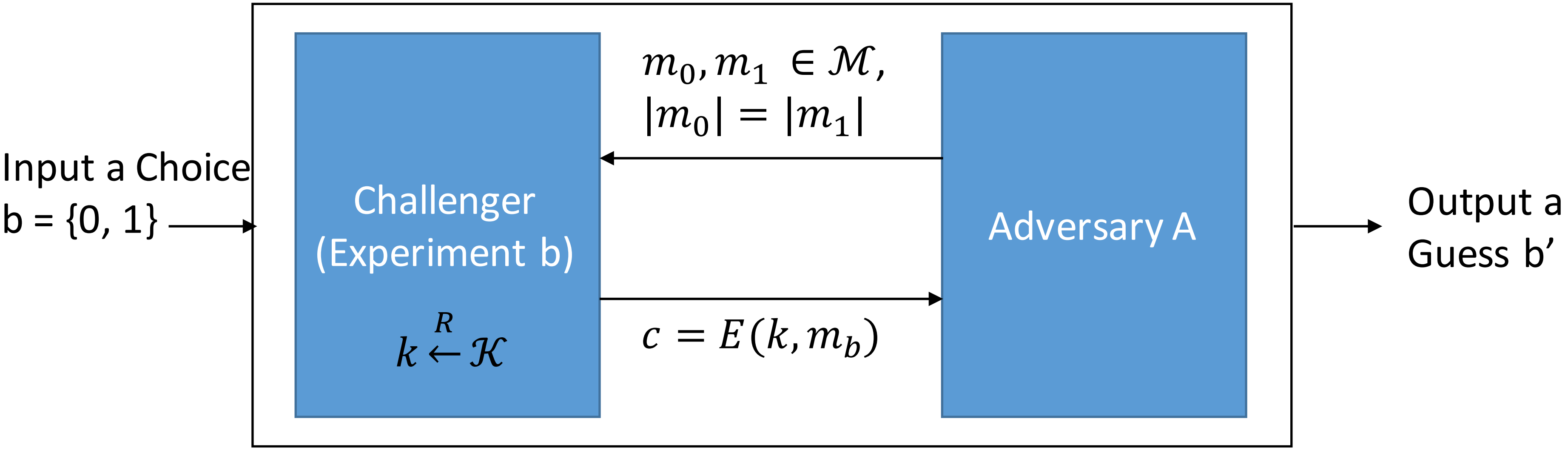
ECB is not semantic secure

Suppose $\mathcal{E}$ is a block cipher in ECB mode

What can the adversary send to win this game?

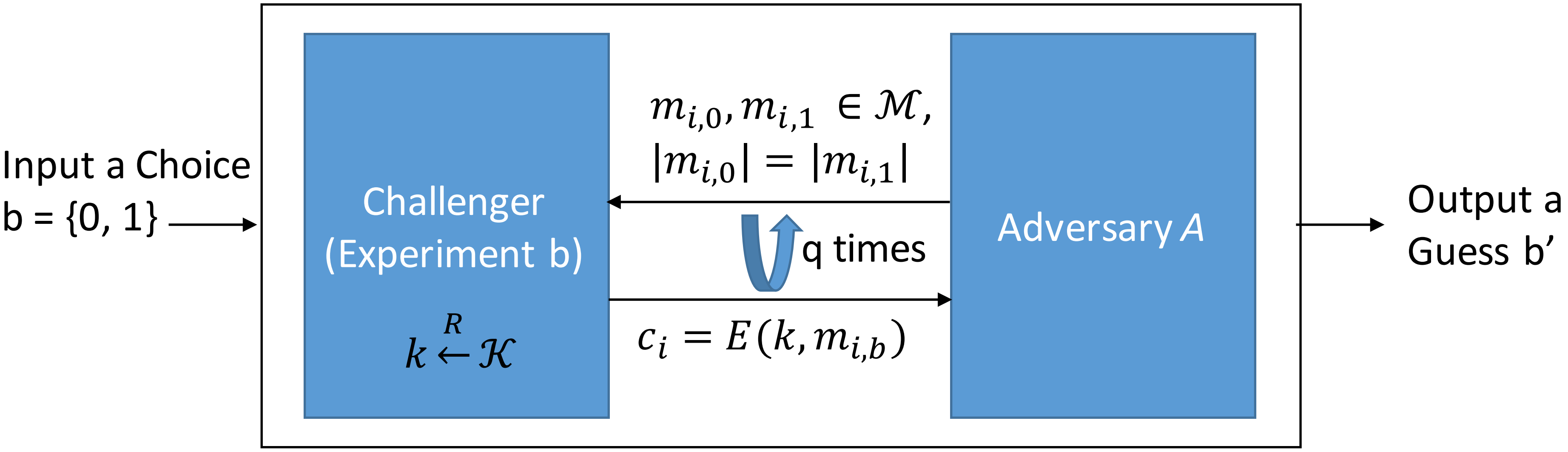
$$|\Pr[\mathrm{Exp}(0) = 1] - \Pr[\mathrm{Exp}(1) = 1]| > \varepsilon$$

Adversary 選兩個 m 一樣的 block，如果丟進 ECB 那麼輸出的 c 會一樣，那 epsilon 會很大



Semantic security for many-time key

Indistinguishability under Chosen plaintext attack (IND-CPA)



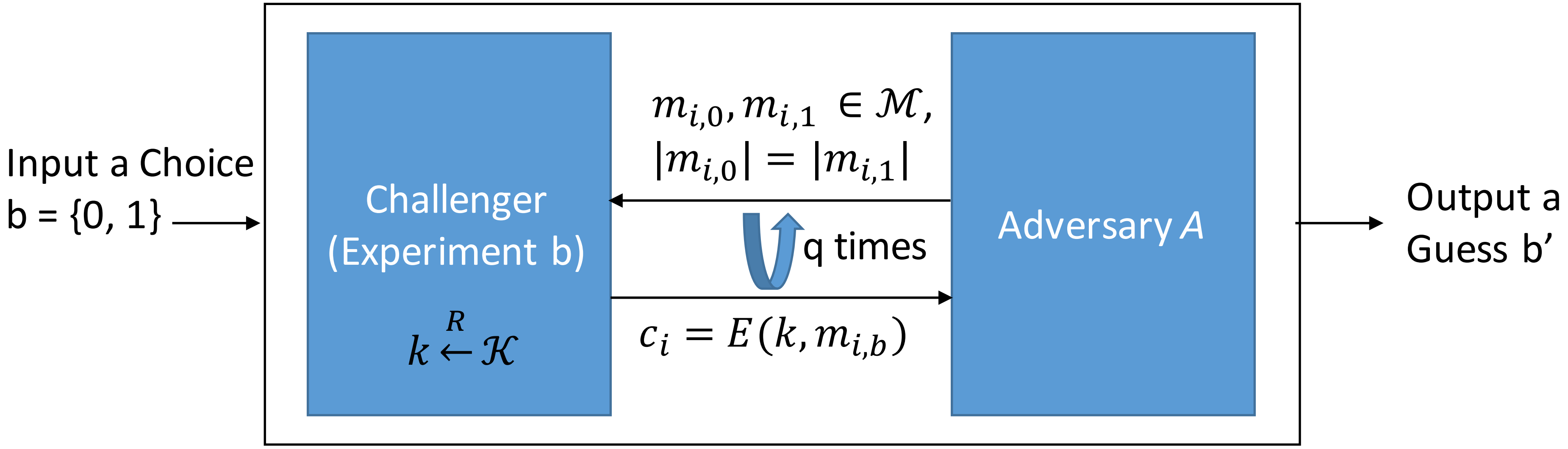
Attacker's **advantage** $\mathrm{Adv}_{\mathrm{CPA}}[A, \mathcal{E}]$ = |Pr[Exp(0) = 1] – Pr[Exp(1) = 1]|
A cipher $\mathcal{E} = (E, D)$ is semantic secure under CPA if the advantage is negligible ($\leq \varepsilon$) for all computationally bounded adversary

Deterministic cipher is insecure under CPA

Suppose $\mathcal{E}$ is a deterministic cipher (encrypting the same plaintext will always generate an identical ciphertext)

What can the adversary send to win this game?

$$|\Pr[\mathrm{Exp}(0) = 1] - \Pr[\mathrm{Exp}(1) = 1]| > \varepsilon$$



Adversary m_{i, 0} 一直選一樣的，m_{i, 1} 一直選不一樣的，因為一樣的 message 透過 ECB 算出來的 ciphertext 仍然會一樣，因此 adversary 只要觀察第 i round 的 c_i 是否和之前 c_j（j < i）不同，就能猜出 challenger 是加密 0 還是 1，那麼猜出 b’ 也是輕而易舉的事。

Message Authentication Codes

Message Authentication Codes (MAC)

Message authentication codes (MAC), or keyed hash

Provides **integrity** and **authenticity** against an active adversary 爛

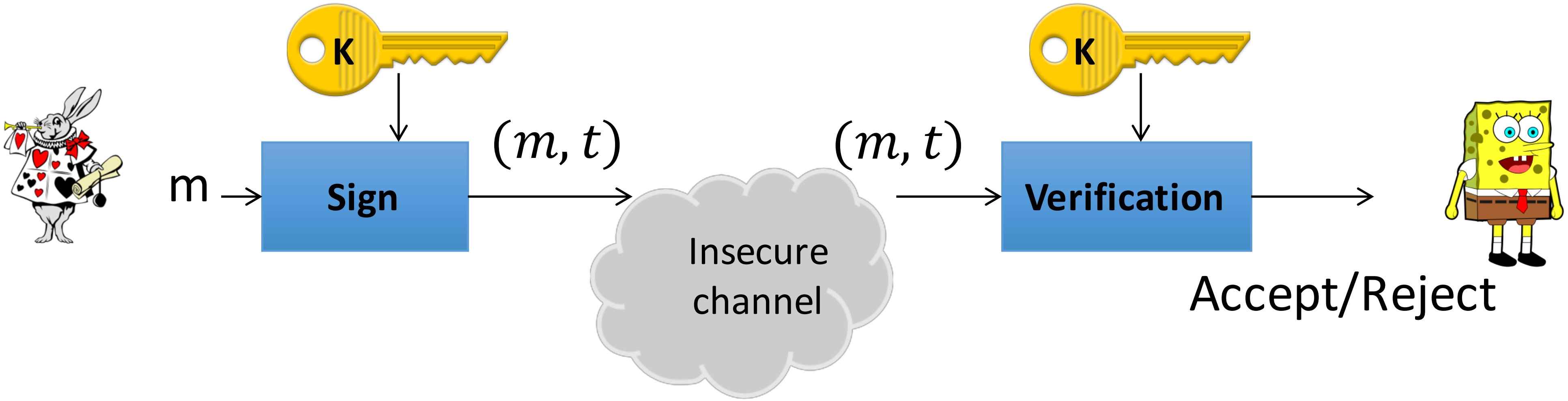
- Integrity: *m* was not modified
- Authenticity: *m* was created by the key owner (which implies integrity)

Message Authentication Codes (MAC)

Alice and Bob have a shared key k

Alice computes the tag $t = S(k, m)$ and sends (m, t)

Bob verifies the received (m, t) by recomputing t using the same key k



Message Authentication Codes (MAC)

A *MAC* $\mathcal{I}$ is a mechanism for authenticating a message using a shared secret key.

$\mathcal{I}$ consists of a pair of signing/verification functions:

$$\mathcal{I} = (S, V)$$

Sign message m using **key** k: $t \overset{R}{\leftarrow} S(k, m)$

Verify tag t using **key** k: $r \leftarrow V(k, m, t)$,
$r \in \{accept, reject\}$

Correctness property: for all keys k, messages m,

$$\Pr\left[V\left(k, m, S(k, m)\right) = accept\right] = 1$$

Message Authentication Codes (MAC)

Similar to hash, MAC should be **easy to compute** and **compress** to a fixed size output

A secure MAC should be existentially unforgeable under a chosen message attack:

- The attacker can submit several m and obtain pairs of (m, t), but it remains computationally infeasible to compute a new message-tag pair.

A *MAC* $\mathcal{I}$ is a mechanism for authenticating a message using a shared secret key.

$\mathcal{I}$ consists of a pair of signing/verification functions:

$$\mathcal{I} = (S, V)$$

Sign message m using key k: $t \overset{R}{\leftarrow} S(k, m)$

Verify tag t using key k: $r \leftarrow V(k, m, t)$,
$r \in \{accept, reject\}$

Correctness property: for all keys k, messages m,

$$\Pr[V(k, m, S(k, m)) = accept] = 1$$

MAC implementation

Attacker 可以做出 S(k, m’) = H(k ǁ m ǁ m’) = MAC’，因此他可以告訴 Bob 這是我簽的 t’ = S(k, m’)，然後把 t’, m’ 拿給 Bob 做驗證，Bob 計算 r = V(k, m’, t’) 不疑有他，就認為這是 Alice 簽的，殊不知其實是 Eve 簽的。

Bad design: $S(k, m) = H(k||m)$

Does this satisfy existentially unforgeable under a chosen message attack?

Recall that some hash functions are vulnerable to the length extension attack!

Better design: HMAC

$$S(k, m) = H(K \oplus opad || H(K \oplus ipad || M))$$

ipad = 3636..36, opad = 5C5C..5C

Authenticated Encryption

Authentication & Encryption

How to achieve both authenticity and confidentiality?

假設 x1, x2, ..., xn，n 人，確認傳訊息的人，共有 shared key 是合法的使用者

Generic composition

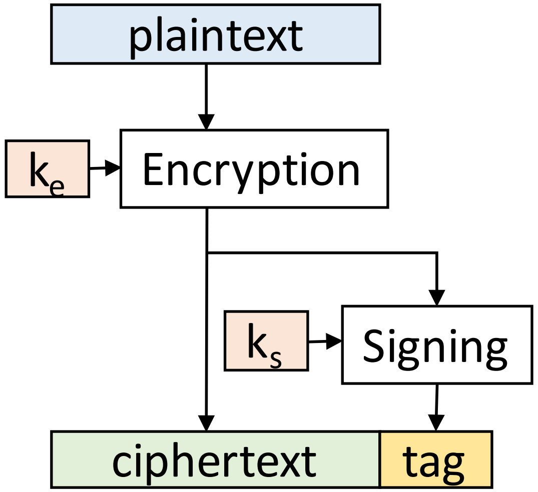
- Combine a secure cipher and a secure MAC
- 3 possible combinations but some are insecure
- Examples: GCM encryption mode

Integrated schemes

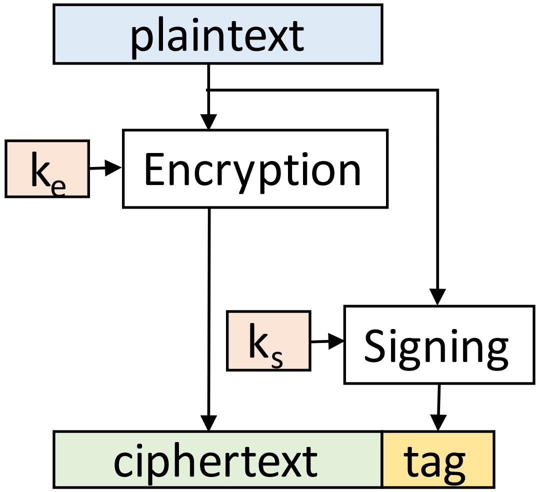
- Build directly from a block cipher or a PRF
- Examples: OCB encryption mode

Combining encryption & MAC

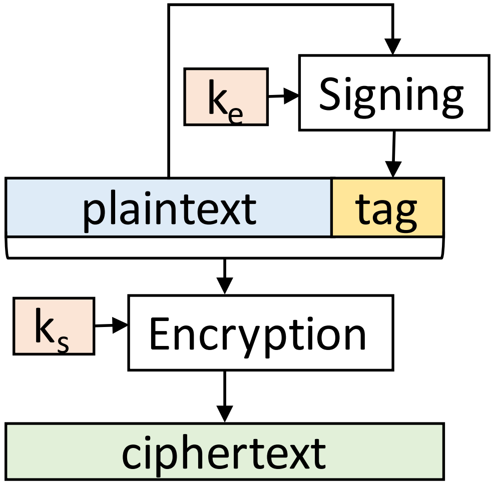
3 possible compositions, which one is better?



Encrypt-then-MAC



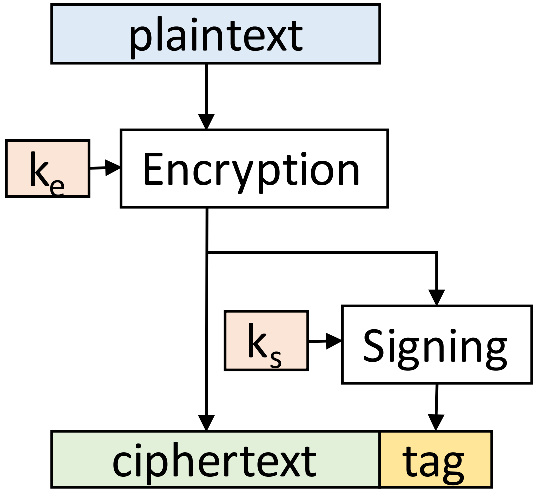
Encrypt-and-MAC



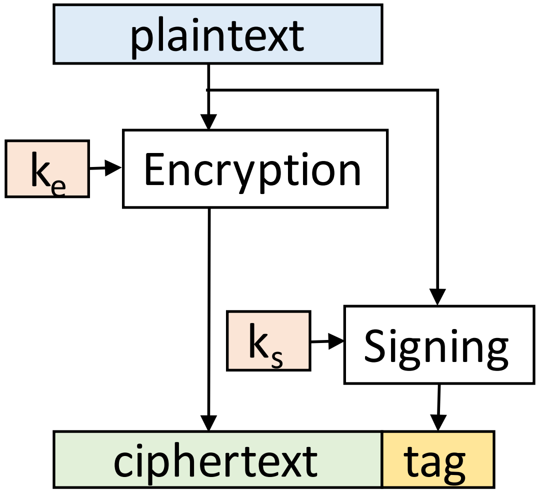
MAC-then-Encrypt

Combining encryption & MAC

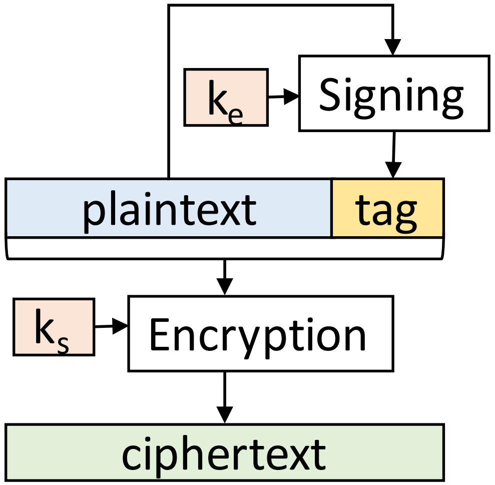
3 possible compositions, which one is better?



Encrypt-then-MAC



Encrypt-and-MAC



MAC-then-Encrypt

MAC-then-Encrypt is bad

MAC-then-Encrypt cannot protect the integrity of *padding*

CBC Padding Oracle Attack possible

MAC-then-encrypt used in SSL/TLS

- POODLE, BEAST and Lucky 13 attacks

TLS 1.3 will only support Authenticated Encryption (e.g., GCM mode)

PKCS#7 Padding

Padding to the block boundary

The value of each added byte is the number of bytes that are added.

1-byte padding: 01

2-byte padding: 02 02

3-byte padding: 03 03 03

…

16-byte padding: 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16

CBC Padding Oracle Attack

Suppose we have access to a padding oracle that tells us whether the padding format is correct

- How?
- Error messages, timing difference, …

Example: We knowc_1, c_2, c_3 and want to get m_3

- Manipulate c_2 such that the padding oracle to tell us whether our guess about m_3 is correct
- Correct guess -> no padding error
- Wrong guess -> (very likely) padding error

CBC Padding Oracle Attack

若 m_l = 我們猜的 z_{−1}
那麼 m_l XOR z_{−1} 會消掉，我們會看
到 m_3 最後一個 bit output 出 0x01

m_ℓ

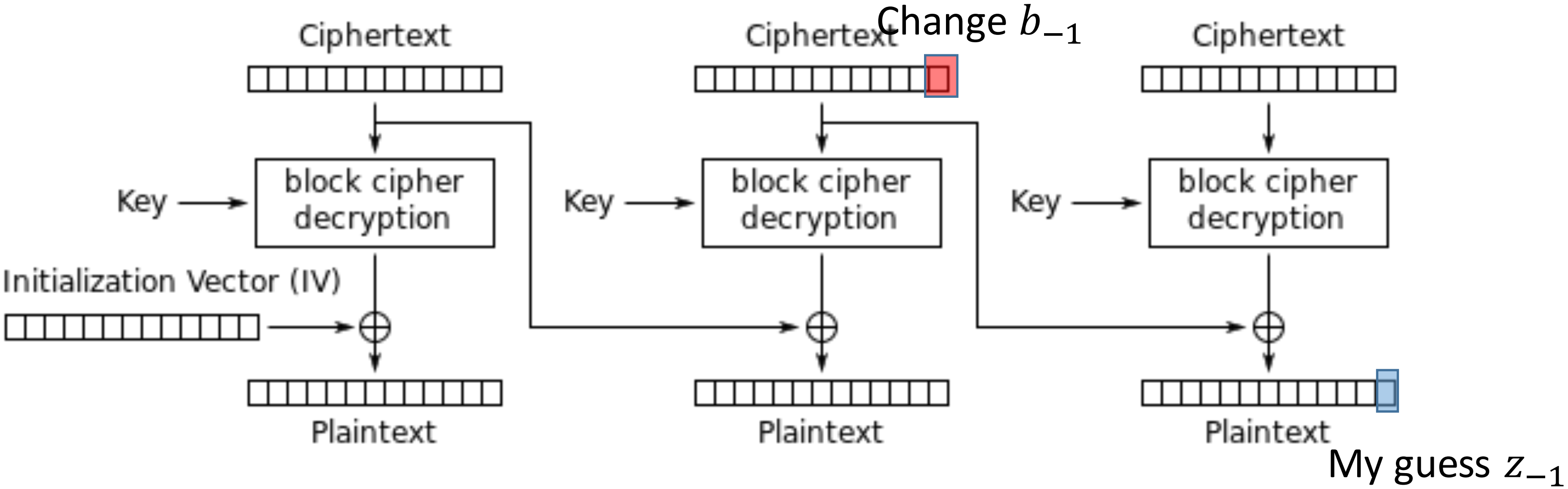
Guess the last byte of m_3, say z_{-1}

Change the last byte of c_2, say b_{-1}, into

$$b_{-1} = b_{-1} \oplus z_{-1} \oplus 0x01$$

If guess is correct, then no padding error: because the last byte of m_3 becomes 0x01 $m_\ell \oplus z_{-1} \oplus 0x01$

If guess is wrong, very *likely* to have a padding error



Cipher Block Chaining (CBC) mode decryption

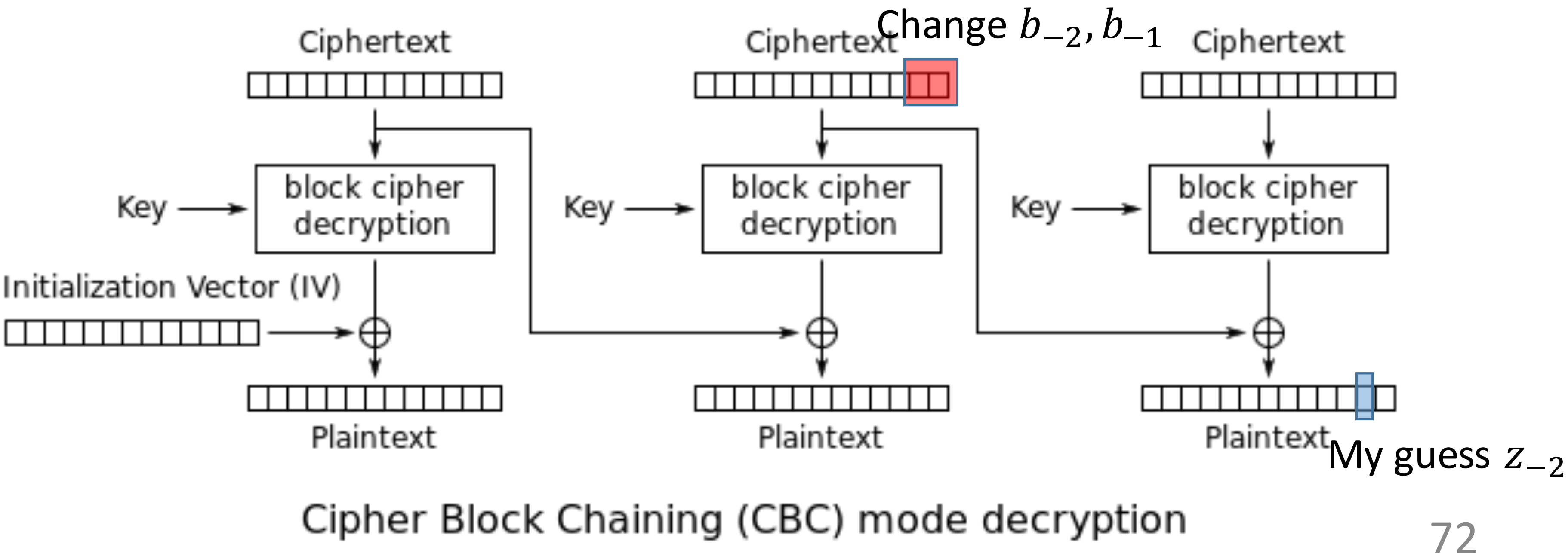
CBC Padding Oracle Attack

Guess the second to the last byte of m_3, say z_{-2}

Change the last two bytes of c_2 into

$$b_{-1} = b_{-1} \oplus z_{-1} \oplus 0x02$$
$$b_{-2} = b_{-2} \oplus z_{-2} \oplus 0x02$$

If the guess is correct, then the last two bytes of m_3 becomes 0x02 0x02, thus no padding error



Cipher Block Chaining (CBC) mode decryption

CBC Padding Oracle Attack

Repeat the process until recovering the full block

#of attempts required: O(256*#of bytes)

Known attacks against SSL/TLS

- **Lucky Thirteen attack**: constructing the padding oracle using a timing side-channel attack against the message authentication code (MAC)
- **POODLE attack** (Padding Oracle On Downgraded Legacy Encryption) against SSL 3.0

How to prevent padding oracle attacks?

Ensure no information leak about the padding

- E.g., server should not provide reasons why decryption failed

Using ciphers without padding oracle attack, e.g., RC4?

- Bad, RC4 itself is insecure

Encrypt-then-MAC: Verify message authentication code (MAC) or signature before checking padding (good)

Authenticated encryption: doing encryption and authentication together (better)

Stream Cipher: RC4

RC4

The most widely used stream cipher

- 18% of TLS connections were still using RC4 as of February 2015

Advantages: speed and simplicity

- Efficient implementations in both software and hardware

Can be used for encryption and as a pseudorandom number generator

RC4

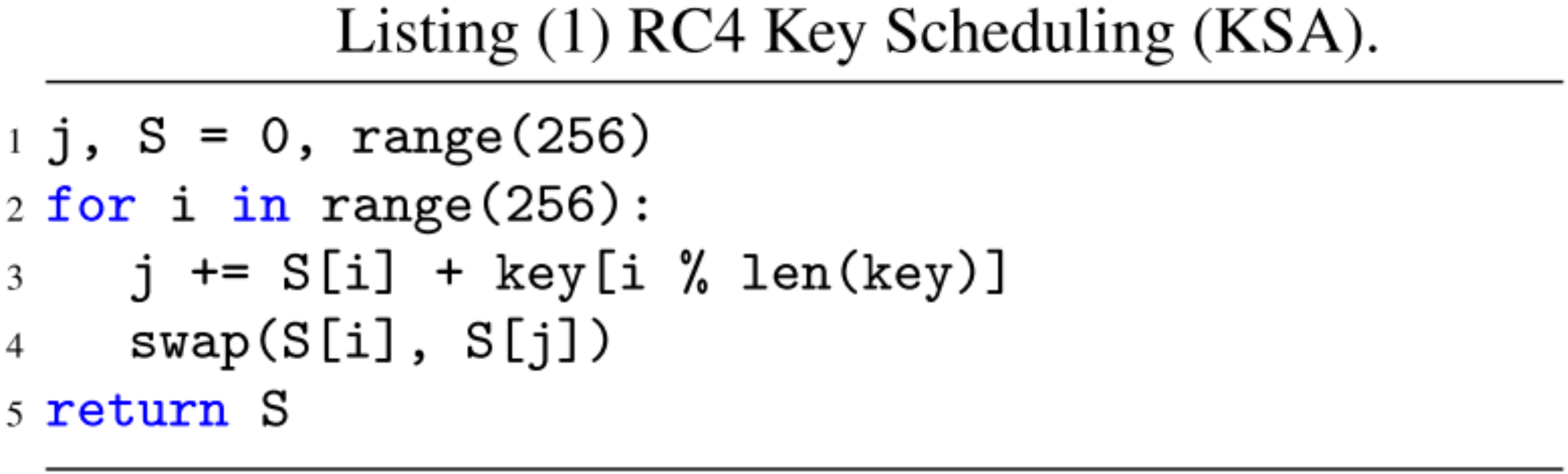
Key Scheduling Algorithm (KSA)

- Given a key, compute a secret internal state which is a permutation of all the 256 possible 8-bit words

Pseudo Random Generation Algorithm (PRGA)

- Given the secret internal state, generate a keystream

RC4

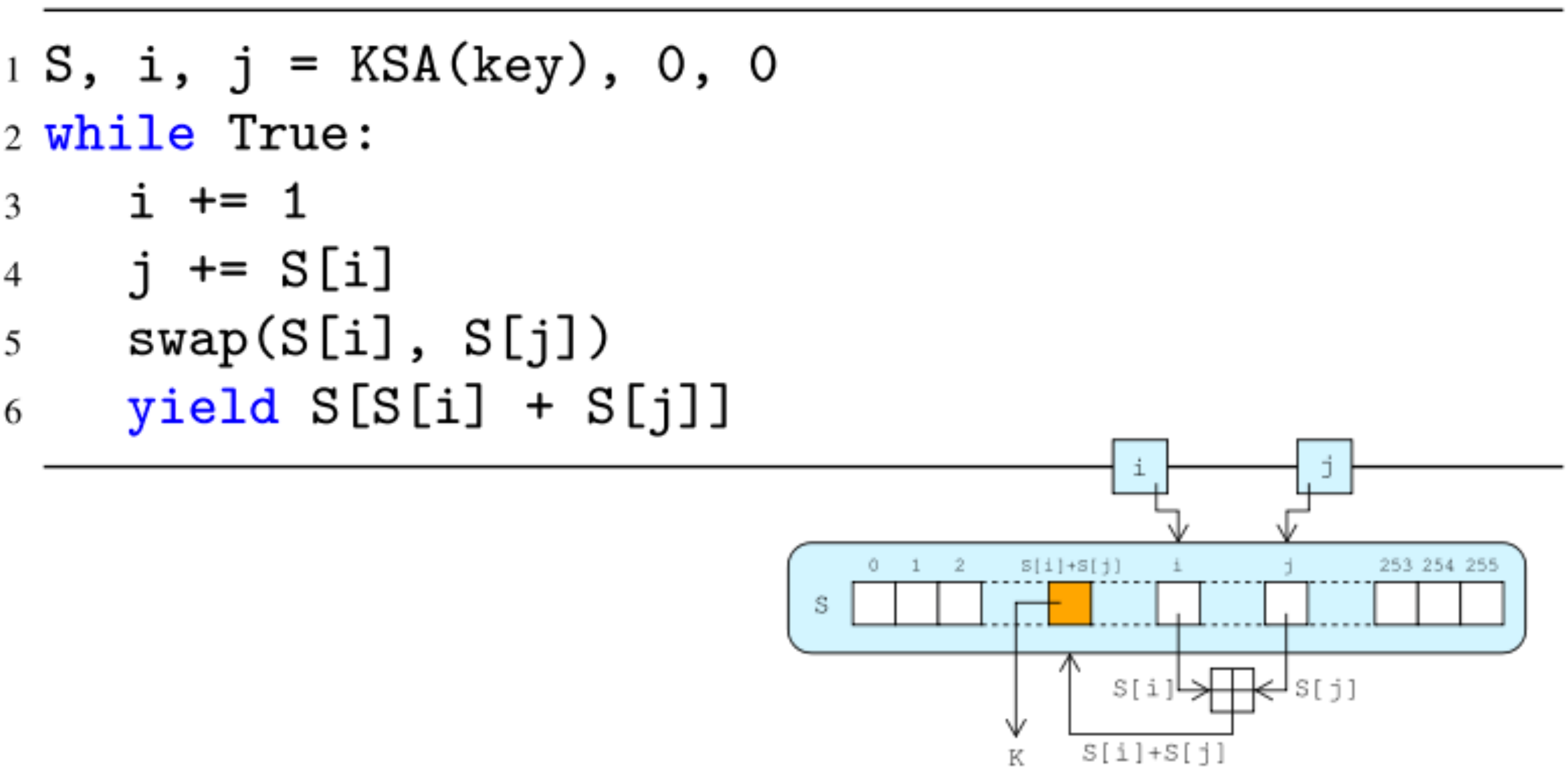


Listing (1) RC4 Key Scheduling (KSA).

```
j, S = 0, range(256)
for i in range(256):
    j += S[i] + key[i % len(key)]
    swap(S[i], S[j])
return S
```

RC4

Listing (2) RC4 Keystream Generation (PRGA).



```
S, i, j = KSA(key), 0, 0
while True:
    i += 1
    j += S[i]
    swap(S[i], S[j])
    yield S[S[i] + S[j]]
```

RC4

Since RC4 is a stream cipher, each keystream should appear like a random sequence (and be unpredictable)

But sadly not true in reality

- RC4 inherently has statistical biases
- Some protocols use a small key space and thus render repetitive keystreams (e.g., WEP)

Protocols using RC4 are considered insecure (e.g., WEP)

In 2015, RFC 5746 prohibits RC4 for all versions of TLS

RC4: examples of short-term biases

$\Pr[Z_2 = 0] \sim 2^{-7}$

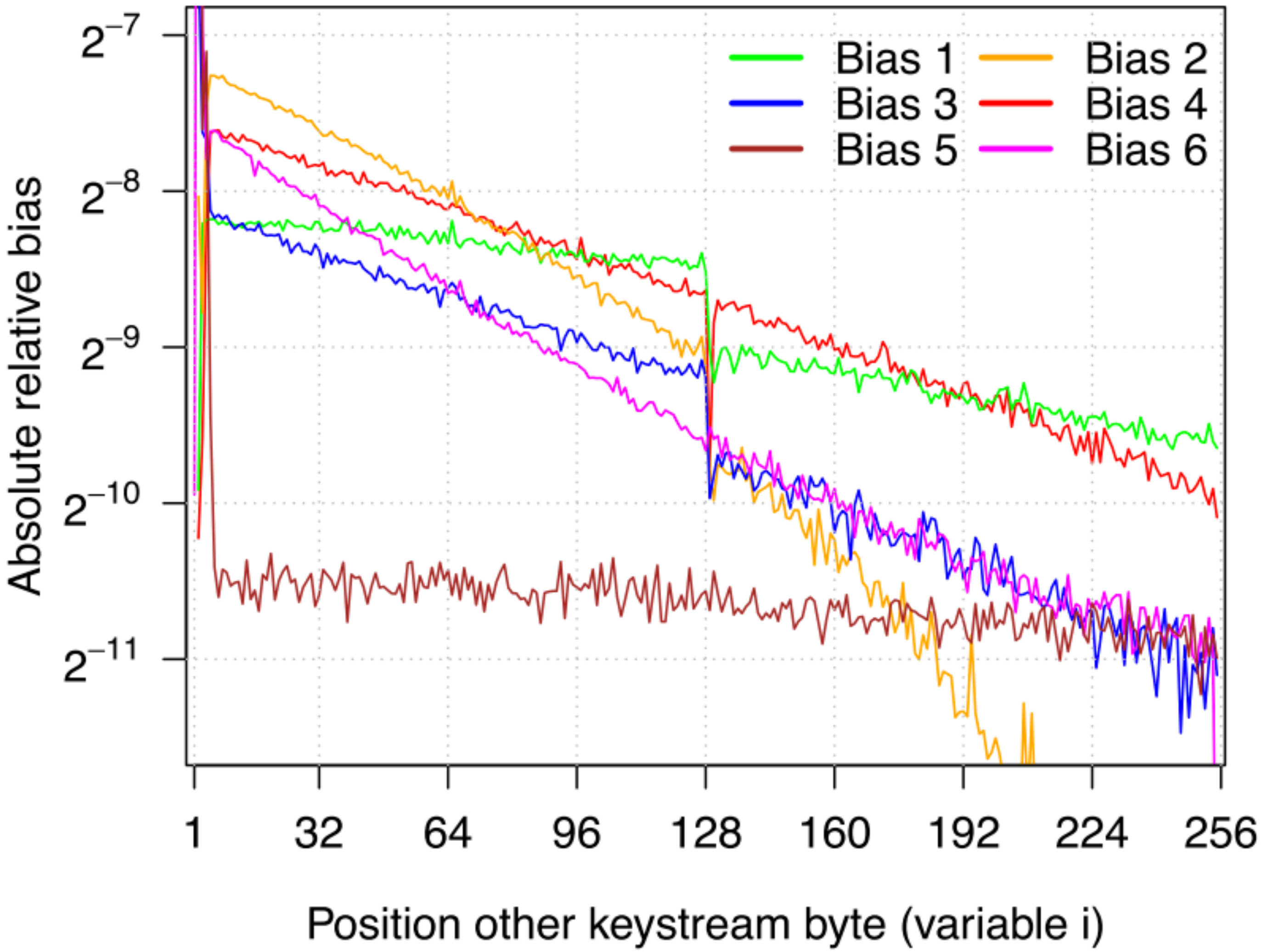
- The second keystream byte is twice as likely to be zero compared to uniform

$\Pr[Z_1 = Z_2]= 2^{-8}(1-2^{-8})$

$\Pr[Z_1 = Z_2 = 0] \approx 3 \cdot 2^{-16}$

keystream byte Z_w has a positive bias towards $256-w$, where w is the key length

for all positions $1 \leq r \leq 256$ there is a positive bias toward value r



1) $Z_1 = 257 - i \wedge Z_i = 0$

2) $Z_1 = 257 - i \wedge Z_i = i$

3) $Z_1 = 257 - i \wedge Z_i = 257 - i$

4) $Z_1 = i - 1 \wedge Z_i = 1$

5) $Z_2 = 0 \wedge Z_i = 0$

6) $Z_2 = 0 \wedge Z_i = i$

RC4: Long-term biases

$\Pr[(Z_r,Z_{r+1})=(Z_{r+g+2},Z_{r+g+3})]=2^{-16}(1+2^{-8}e^{(-4-8g)/256})$

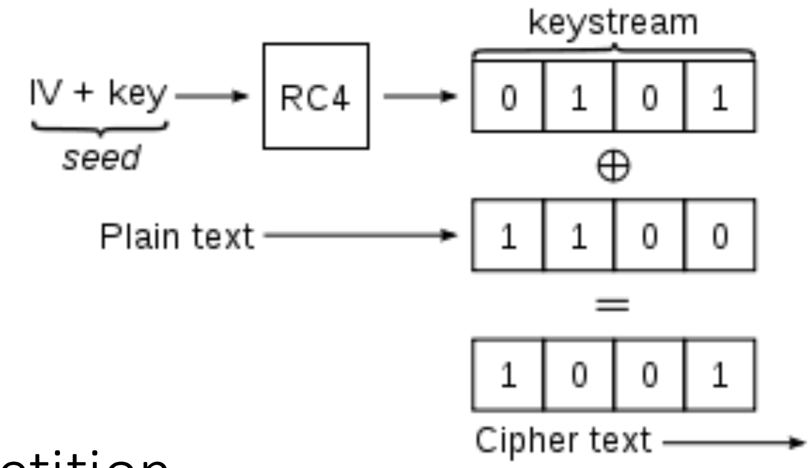
$\Pr[(Z_{w256},Z_{w256+2})=(0,0)]=2^{-16}(1+2^{-8})$

$\Pr[(Z_{w256},Z_{w256+2})=(128,0)]=2^{-16}(1+2^{-8})$

RC4 application: Wired Equivalency Privacy (WEP)

Provide data confidentiality comparable to traditional wired network

Per-packet encryption



P = M || CRC(M)

RC4_KEY = IV || WEP_KEY

- IV: 24 bits, aim to prevent repetition
- WEP_KEY: 40 or 104 bits, derived from password

C = IV || $\text{RC4}_{\text{RC4\_KEY}}$(P)

Key stream reuse

IV is is only 24 bits

WEP key is identical for a long time

For a 24-bit IV, we will find two packets with the same IV (and thus the RC4 keystream) after 1.17*2^{12} ~ 5000 packets

- Birthday attack

Fluhrer, Mantin and Shamir (FMS) attack

A key recovery attack

Related key vulnerability

- A small part of the secret key determines a large number of bits of the initial permutation (KSA output)
- Pseudo Random Generation Algorithm (PRGA) translates these patterns in the initial permutation into patterns in the prefix of the output stream
- With certain weak IVs, an attacker knowing the first byte of the keystream and the first m bytes of the key can derive the $(m+1)^{th}$ byte of the key due to a weakness in PRNG
  - 1st byte of plaintext: usually 0xAA

Fluhrer, Scott, Itsik Mantin, and Adi Shamir. "Weaknesses in the key scheduling algorithm of RC4." Selected areas in cryptography. Springer Berlin Heidelberg, 2001.

Weak IVs in WEP

IV is 24 bits for WEP

A weak IV has the form of (A+3, N-1, X)

- A = Byte in the keystream to be attacked
- N = 256
- X = any value between 0-255

FMS attack to discover (A+4)th byte of the RC4 key (i.e. the (A+1)th byte of the WEP key):

- Perform A+3 steps of Key Scheduling Algorithm
  - This can be done given that the first A+3 bytes are known
- If j < 2, abort
- Pr[(A+4)th byte of RC4 key = 1st keystream byte - j - S[A+3]] ≥ 5%

Collect packets with weak IVs

Passive collection: may need to wait for several days

Active replay: replay (encrypted) ARP request packets and collect ARP response packets, which will be in separate packets with new IVs

Several advanced techniques to reduce the required number of packets

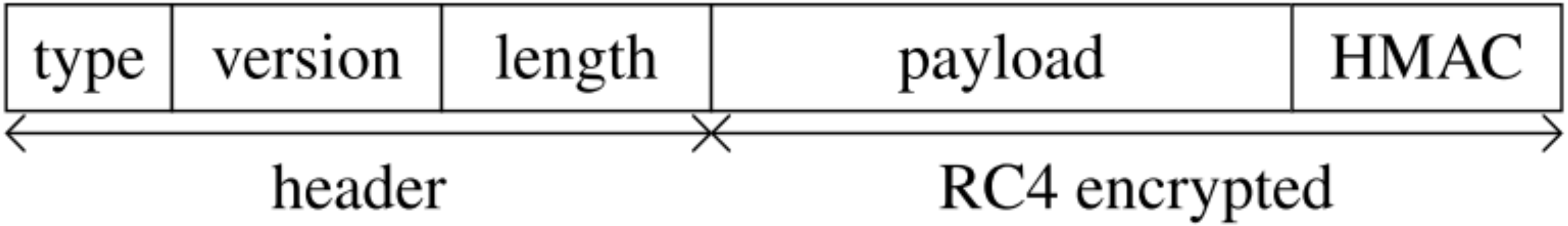
```
AIRCRACK-NG(1)         General Commands Manual          AIRCRACK-NG(1)

NAME
       aircrack-ng - a 802.11 WEP / WPA-PSK key cracker

SYNOPSIS
       aircrack-ng [options] <.cap / .ivs file(s)>

DESCRIPTION
aircrack-ng is an 802.11 WEP and WPA/WPA2-PSK key cracking program.
It can recover the WEP key once enough encrypted packets have been cap-
tured with airodump-ng. This part of the aircrack-ng  suite  determines
the  WEP key using two fundamental methods. The first method is via the
PTW approach (Pyshkin, Tews, Weinmann). The main advantage of  the  PTW
approach  is  that  very few data packets are required to crack the WEP
key. The second method is the FMS/KoreK method.  The  FMS/KoreK  method
incorporates  various  statistical  attacks to discover the WEP key and
uses these in combination with brute forcing.
Additionally, the program offers a dictionary  method  for  determining
the WEP key. For cracking WPA/WPA2 pre-shared keys, a wordlist (file or
stdin) or an airolib-ng has to be used.
```

RC4 NOMORE attack

https://www.rc4nomore.com/

Mathy Vanhoef and Frank Piessens. "All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS." USENIX Security Symposium, 2015. https://www.rc4nomore.com/

- Recovered a HTTPS cookie in 75 hours
- Attacked a WPA-TKIP network within an hour



type	version	length	payload	HMAC

header

RC4 encrypted

RC4 NOMORE attack: Decrypting secure cookies

Since secure cookies guarantee only confidentiality but not integrity, the insecure HTTP channel can be used to overwrite, remove, or inject secure cookies

Inject known data around a cookie, enabling use of the ABSAB biases

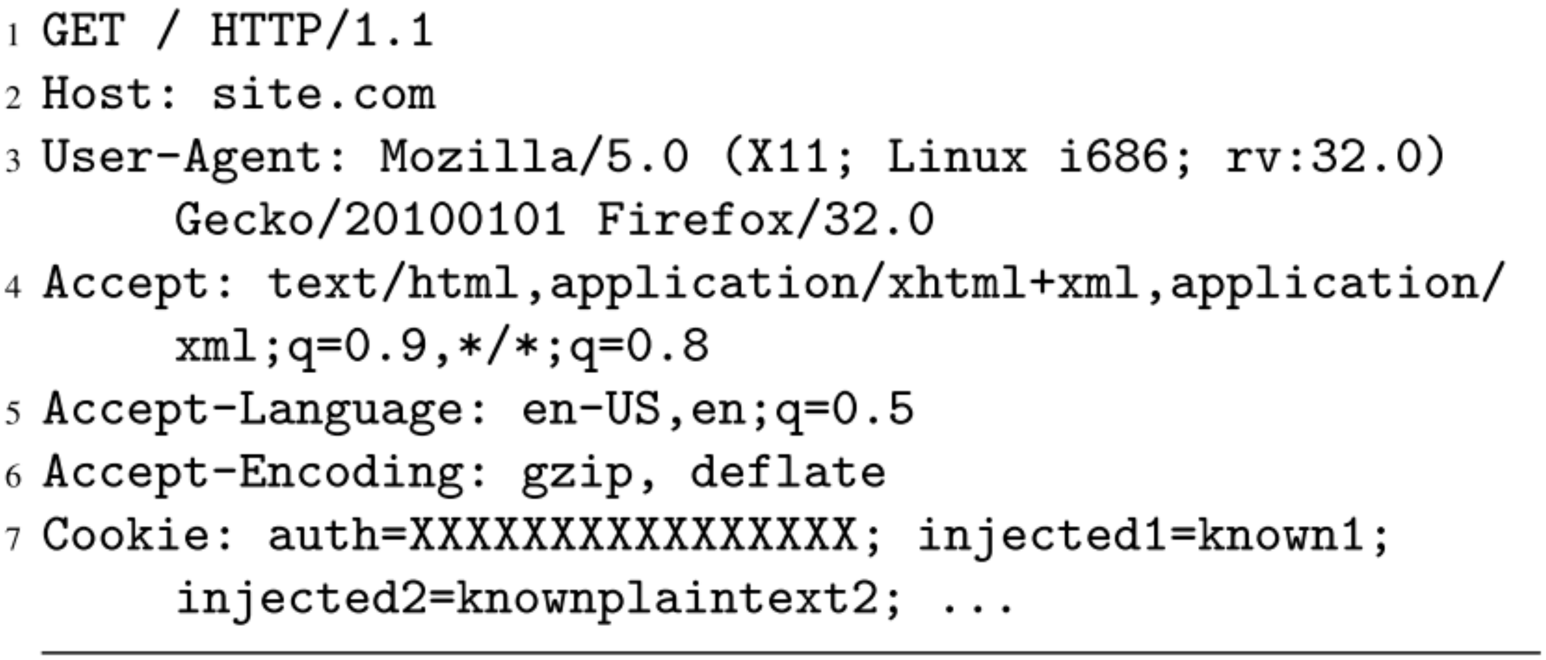
- $\Pr[(Z_r,Z_{r+1})=(Z_{r+g+2},Z_{r+g+3})]=2^{-16}(1+2^{-8}e^{(-4-8g)/256})$

RC4 NOMORE attack: Decrypting secure cookies

Remove all cookies except auth cookie

Surround auth cookie by known plaintext at both sides

Use persistent connection (`keep-alive)` to gather more packets using the same configuration and key



```
GET / HTTP/1.1
Host: site.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:32.0)
     Gecko/20100101 Firefox/32.0
Accept: text/html,application/xhtml+xml,application/
     xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: auth=XXXXXXXXXXXXXXXX; injected1=known1;
     injected2=knownplaintext2; ...
```

RC4 NOMORE attack: Brute-force cookie

A cookie has at most 90 unique characters

Need roughly $9 \cdot 2^{27}$ ciphertexts for 94% success rate

- = 75 hours, assuming 4450 requests per seconds

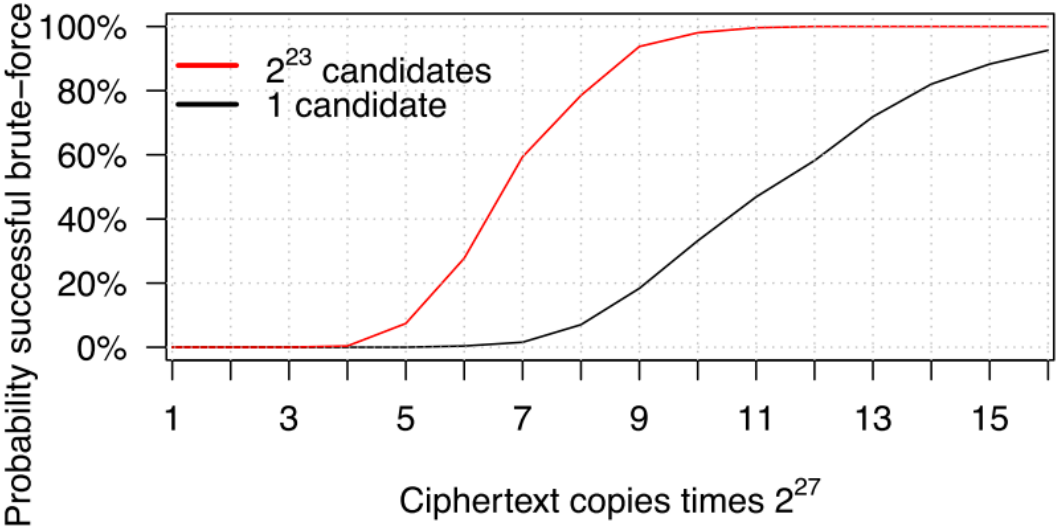


Figure 10: Success rate of brute-forcing a 16-byte cookie using roughly 2^{23} candidates, and only the most likely candidate, dependent on the number of collected ciphertexts. Results based on 256 simulations each.