



Private key encryption

Hyounghick Kim

Department of Software

College of Software

Sungkyunkwan University

Symmetric key crypto

- **Stream cipher** — based on one-time pad
 - Except that key is relatively short
 - Key is stretched into a long **keystream**
 - Keystream is used just like a one-time pad
 - RC4, A5/1, and etc.
- **Block cipher** — based on codebook concept
 - Block cipher key determines a codebook
 - Each key yields a different codebook
 - Employs **both “confusion” and “diffusion”**

Stream ciphers

Stream Ciphers: making OTP practical

idea: replace “random” key by “pseudorandom” key

PRG is a function $G: \{0, 1\}^s \rightarrow \{0, 1\}^n$

Generate a pseudo random key
using a random seed!

$$\text{Encryption: } c = G(k) \oplus m$$

Security will depend on specific PRG G

Quiz

Can a stream cipher have perfect secrecy?

No, since the key is shorter than the message

RC4 Stream Cipher

- A proprietary cipher owned by RSA, designed by Ron Rivest in 1987
- Became public in 1994
- Simple and effective design
- Variable key size (typical 40 to 256 bits)
- Output unbounded number of bytes
- Widely used (SSL/TLS, wireless WEP)
- Extensively studied, not a completely secure PRNG
- Newer Versions: RC5 and RC6

Stream cipher example - RC4

- Key stream generation:

- (S[] is permutation of 0,1,...,255)

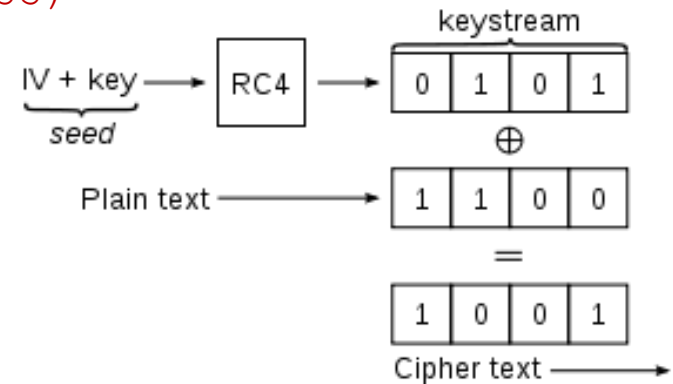
$i := i+1 \pmod{256}$

$j := j+s[i] \pmod{256}$

$\text{swap}(s[i], s[j])$

$t := s[i] + s[j] \pmod{256}$

$k := s[t]$



- Idea: systematically keep swapping and producing output bytes

Security of RC4

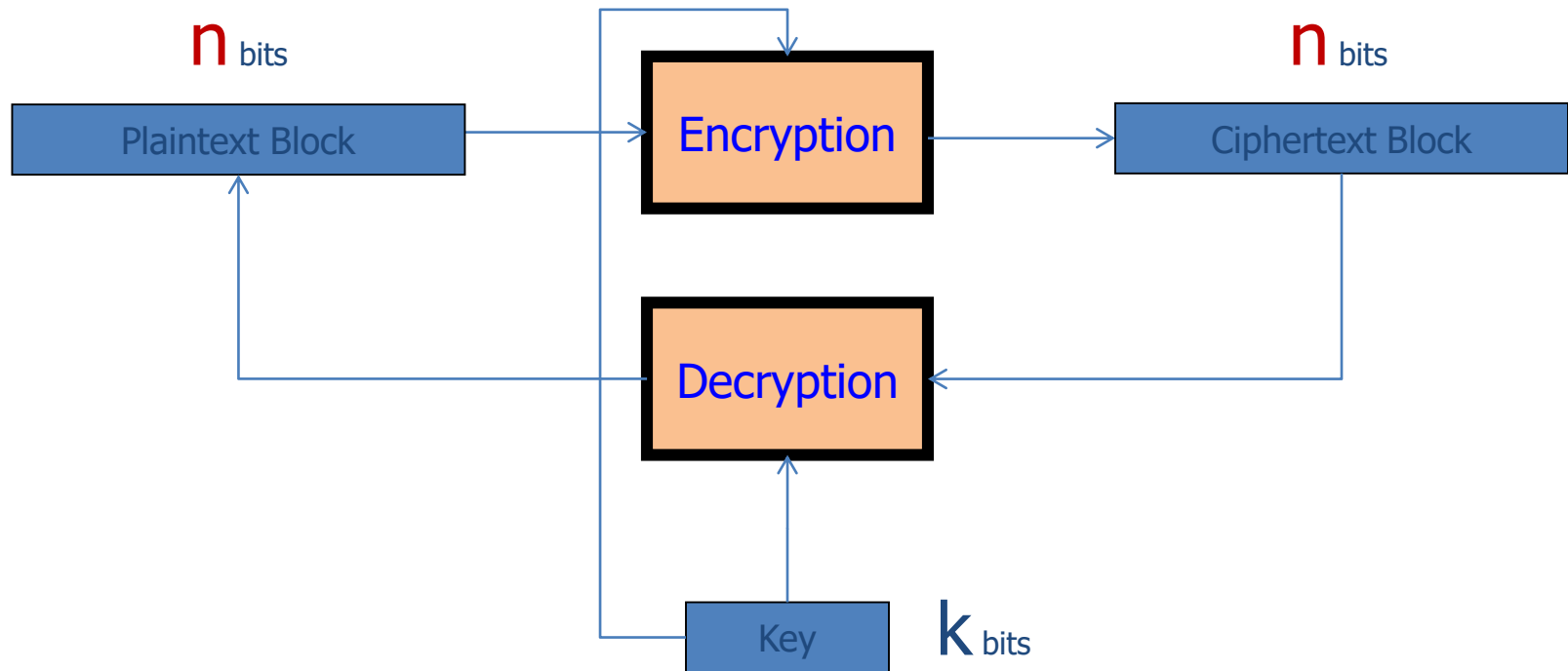
- RC4 is **not** a truly pseudorandom generator.
- The keystream generated by RC4 is biased.
 - The second byte is biased toward zero with high probability.
 - The first few bytes are strongly non-random and leak information about the input key.
- Defense: discard the initial n bytes of the keystream.
 - Called “RC4-drop[n -bytes]”.
 - Recommended values for $n = 256, 768, \text{ or } 3072$ bytes.

Trends of stream ciphers

- Stream ciphers were popular in the past
 - Efficient in hardware
 - Speed was needed to keep up with voice, etc.
 - Today, processors are fast, so software-based crypto is usually more than fast enough
- Future of stream ciphers?
 - Shamir declared “the death of stream ciphers”
 - May be greatly exaggerated...

Block ciphers

Block ciphers - basic structure

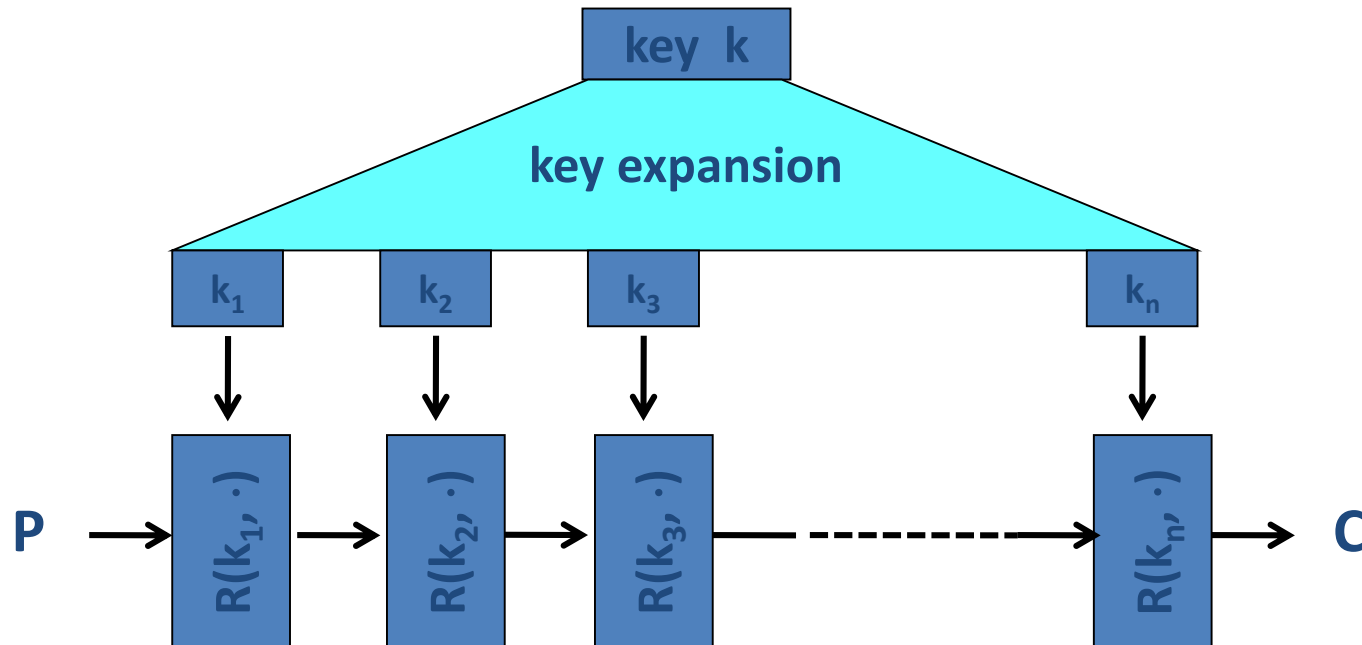


Canonical examples:

1. DES: $n = 64$ bits, $k = 56$ bits
2. AES: $n = 128$ bits, $k = 128, 192, 256$ bits

Block ciphers built by iteration

Iterate **substitution** and **permutation** (by Shannon, 1948)



$R(k, m)$ is called a round function

for DES ($n=16$), for AES-128 ($n=10$)

Cipher structures

- Feistel structure
 - This technique was devised by Horst Feistel of IBM
 - Each round uses an operation called the F-function whose input is half a block and a round key; the output is a half-block of scrambled data which is XOR-ed into the other half-block of text
 - Examples: DES
- Substitution-Permutation (SP) networks
 - Shannon's own design for a product cipher
 - 2 layers in each round: a substitution layer provides confusion, then a permutation layer provides diffusion
 - Examples: AES

To be secure, every cipher must contain *nonlinear* operations.

Feistel Cipher: Encryption

- **Feistel cipher** is a type of block cipher, not a specific block cipher
- Split plaintext block into left and right halves: $P = (L_0, R_0)$
- For each round $i = 1, 2, \dots, n$, compute
$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$
where F is **round function** and K_i is **subkey**
- Ciphertext: $C = (L_n, R_n)$

Feistel Cipher: Decryption

- Start with ciphertext $C = (L_n, R_n)$
- For each round $i = n, n-1, \dots, 1$, compute

$$R_{i-1} = L_i$$

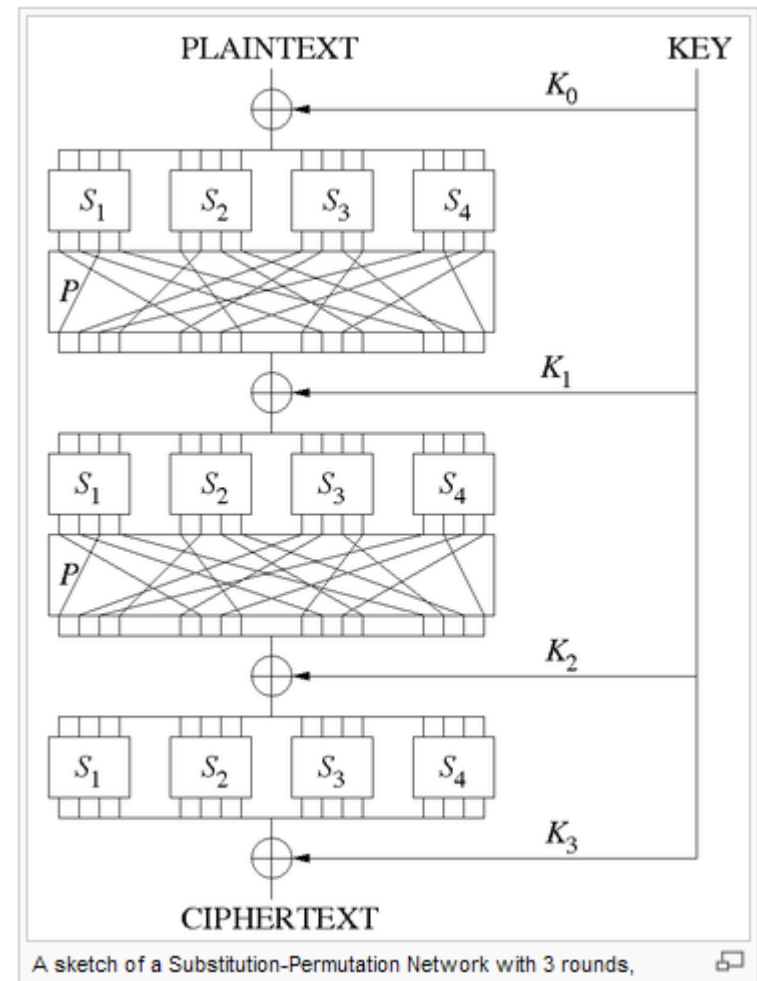
$$L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$$

where F is round function and K_i is subkey

- Plaintext: $P = (L_0, R_0)$
- Formula “works” for **any function F**
 - But only secure for certain functions F

SP networks

- More constraints on the round function: must be **invertible**
- Faster than Feistel-structure
- Parallel computation
- Typically $E \neq D$



Quiz

Q1. What is the main advantage of a Feistel cipher over an SP network?

The F-function itself need not be reversible. This gives the designer extra flexibility; almost any operation he can think up can be used in the F-function

Q2. What is the main advantage of an SP network over a Feistel cipher?

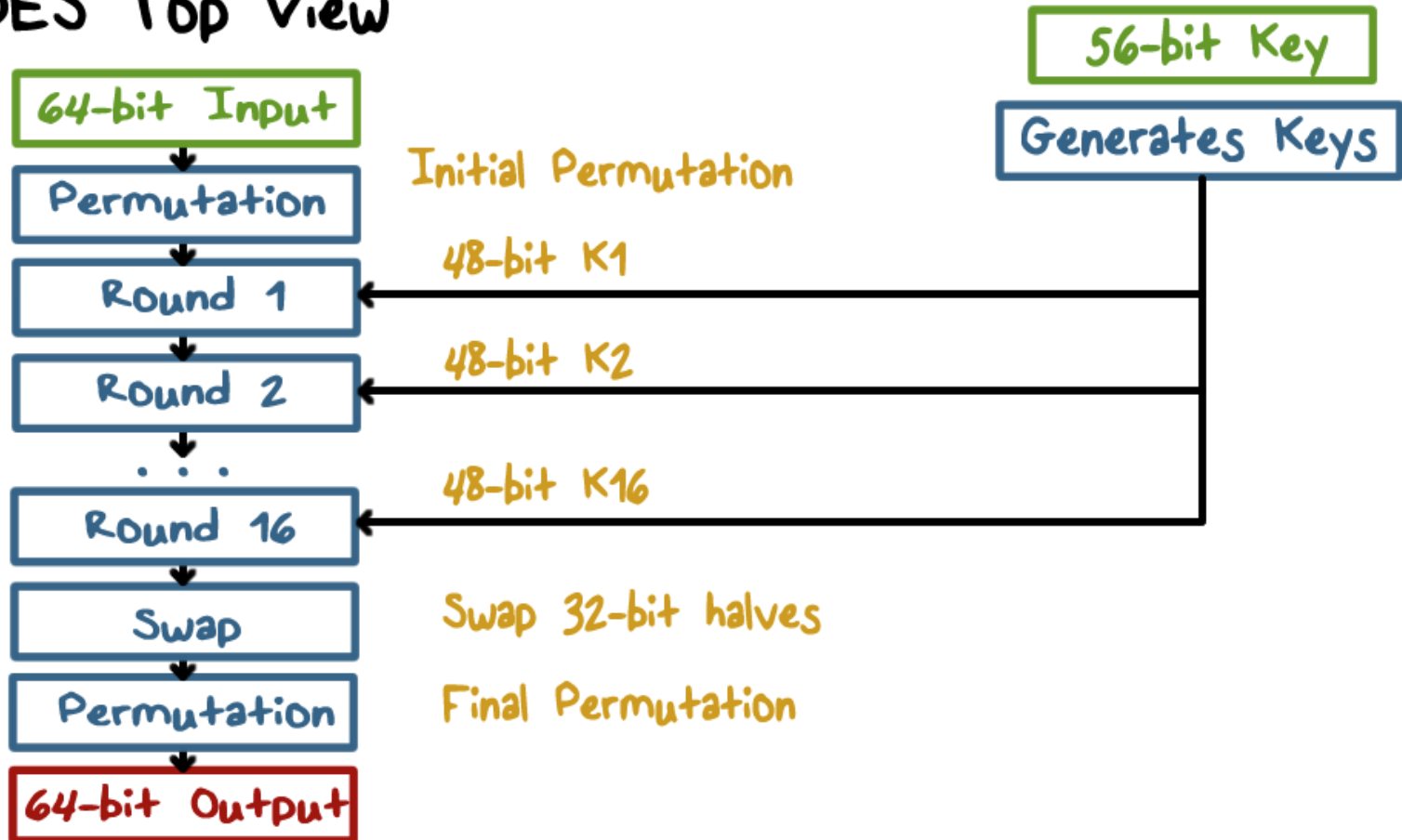
In the Feistel construction, only half the output changes in each round while an SP network changes all of it in a single round

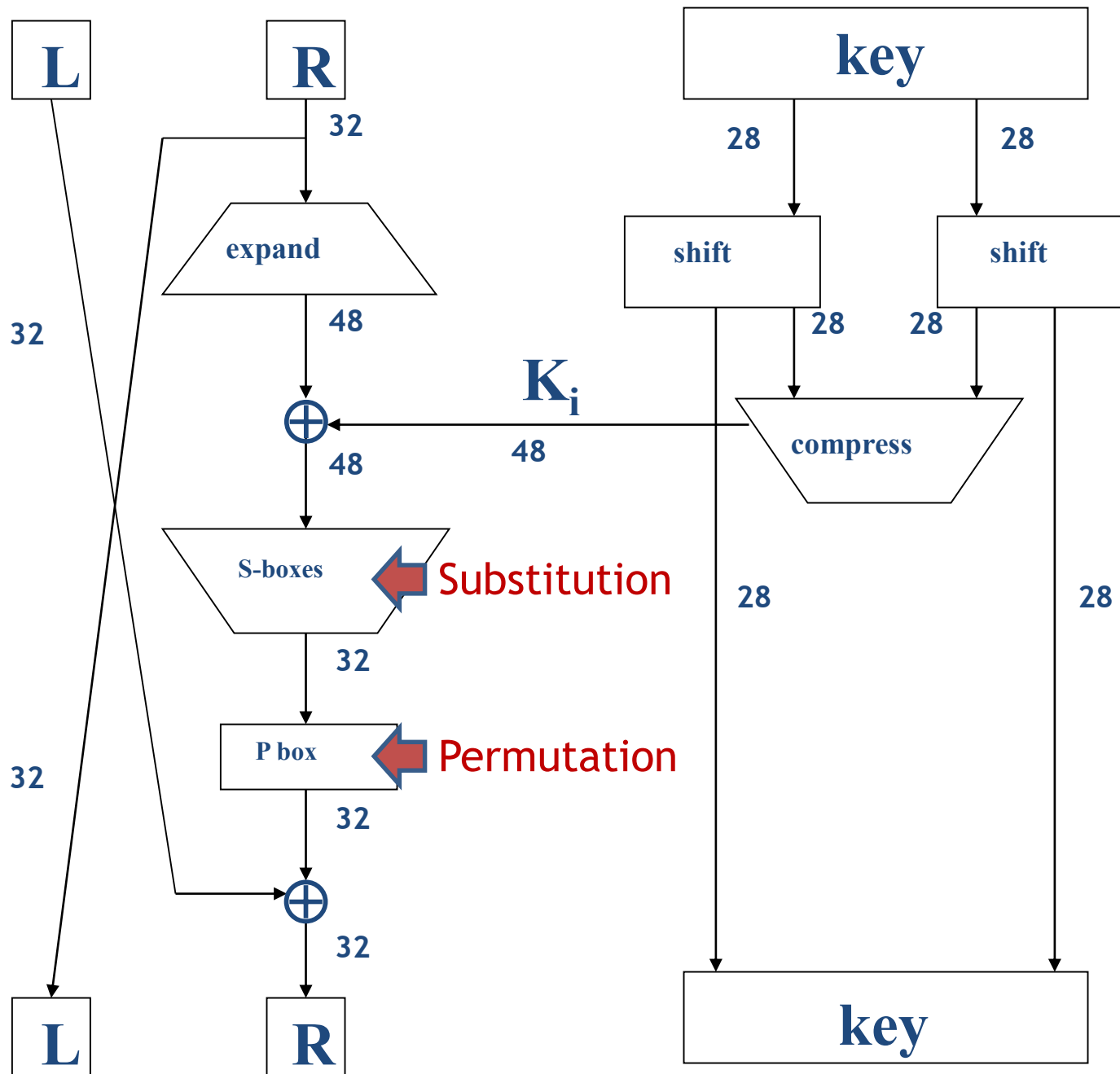
Data Encryption Standard (DES)

- DES was standardized in 1977; it's widely used in banking, and assorted embedded stuff
- Based on IBM's Lucifer cipher
- DES is a Feistel cipher with...
 - 64 bit block length
 - 56 bit key length
 - 8 bit parity
 - 16 rounds
 - 48 bits of key used each round (subkey)
- Each round is simple (for a block cipher)

Overview of DES

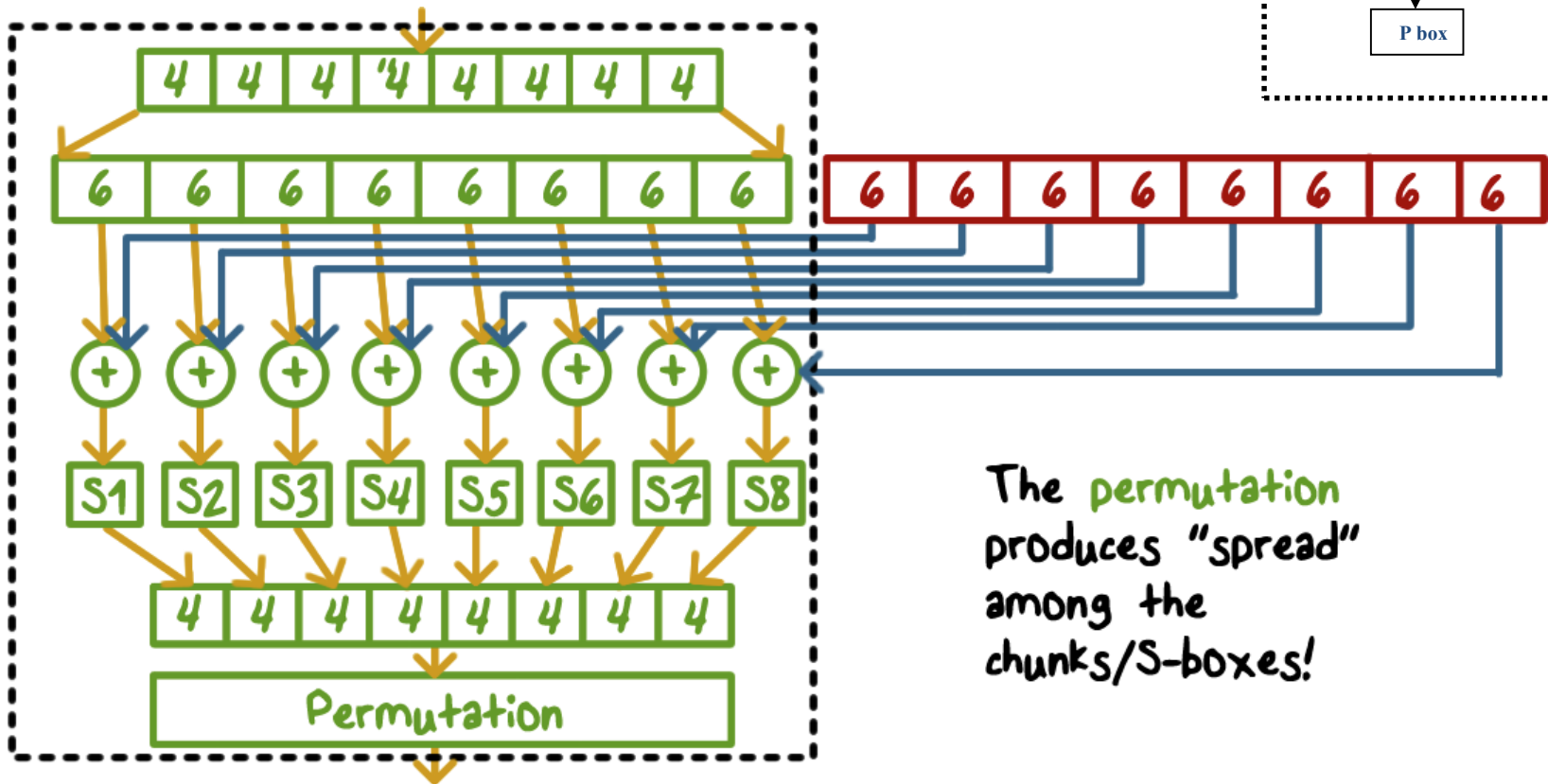
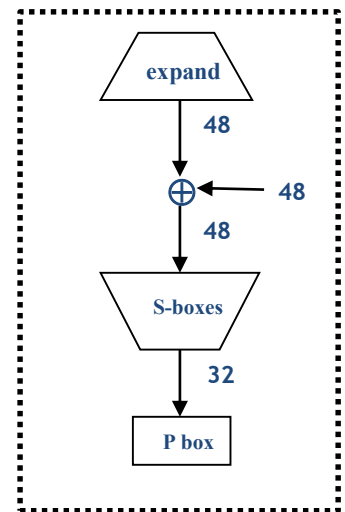
DES Top View





One
round
of
DES

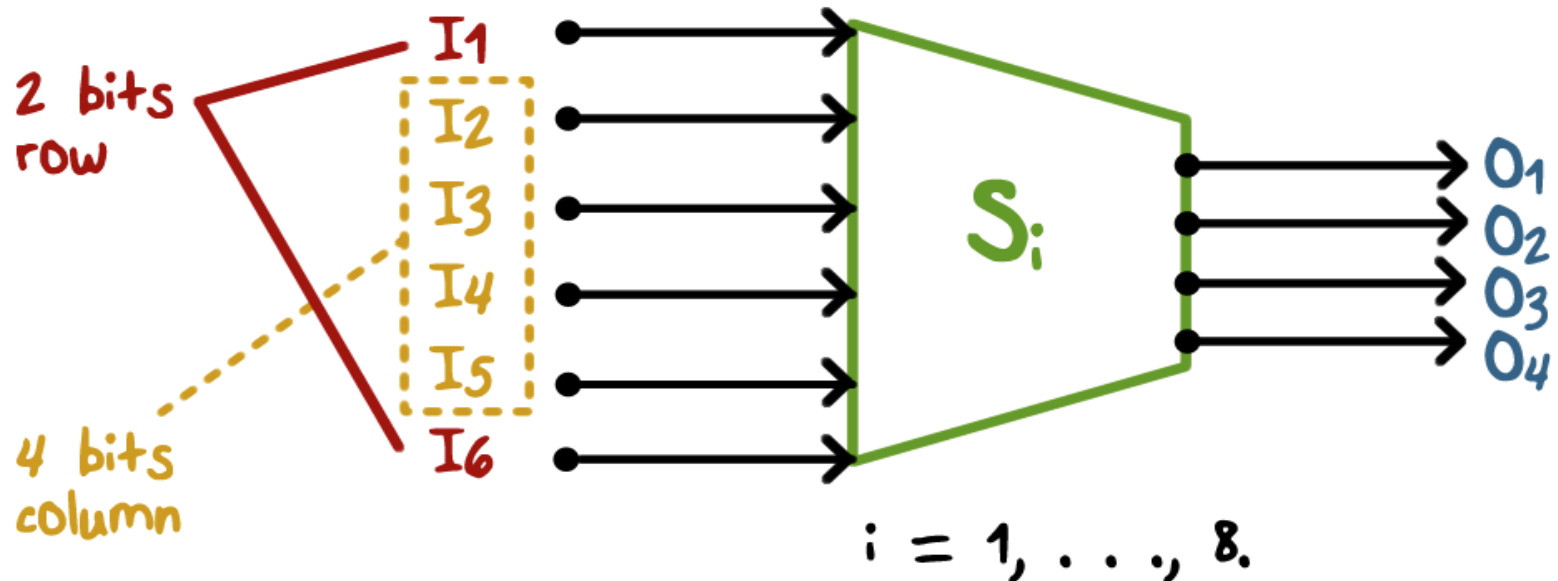
Mangler function



The **permutation** produces "spread" among the chunks/S-boxes!

S-box

- 48 bits \Rightarrow 32 bits. ($8 \times 6 \Rightarrow 8 \times 4$)
- 2 bits used to select amongst 4 substitutions for the rest of the 4-bit quantity



Quiz



For the given input, determine the output.

S ₅		Middle 4 bits of input															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Outer bits	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Input: 011011

Output:

1001

Security of DES

- Security depends heavily on **S-boxes**
 - Everything except for S-boxes in DES is linear
- Thirty+ years of intense analysis has revealed no “back door”
- Shortcut attacks exist but are not important:
 - differential cryptanalysis (2^{47} chosen texts)
 - linear cryptanalysis (2^{41} known texts)
- **56-bit key is too small** – key search is the real vulnerability!
 - COPACOBANA (120 FPGAs, 10,000\$) broke DES in 7 days.

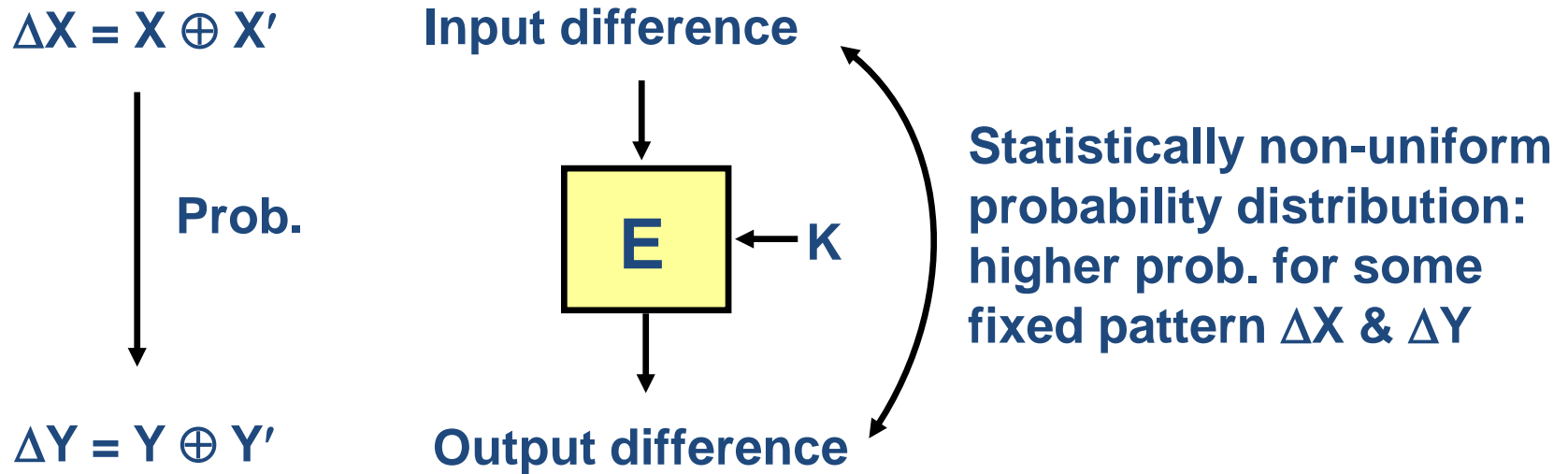


Avalanche effect

- Key desirable property of an encryption algorithm
- Where a change of one input or key bit results in changing approx half of the output bits
- If the change were small, this might provide a way to reduce the size of the key space to be searched
- DES exhibits strong avalanche

Differential cryptanalysis

- E. Biham and A. Shamir : Crypto90, Crypto92
- It is called ‘differential’ because the attacker studies how a small change in the plaintext block affects the encrypted block

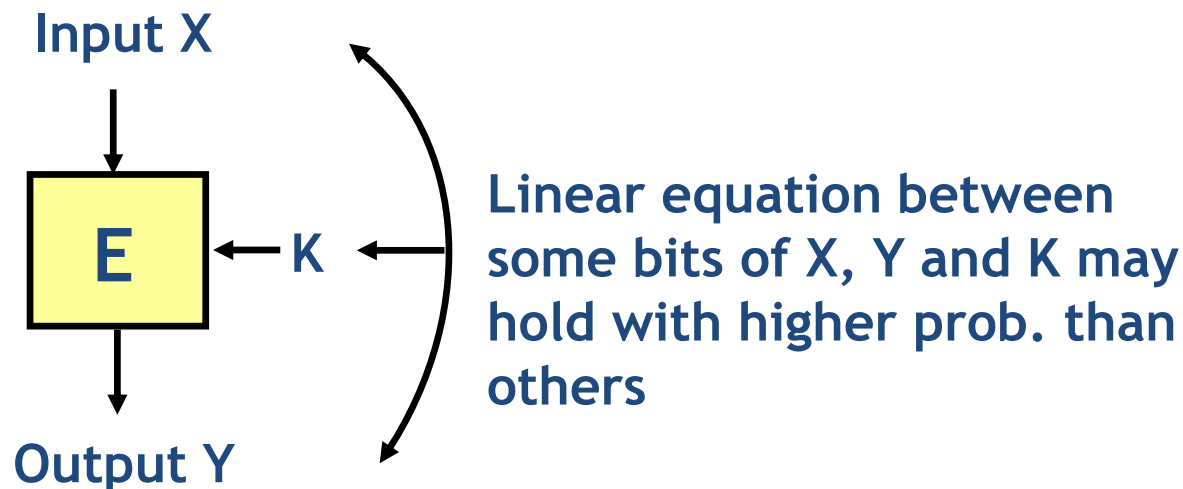


“Differential Cryptanalysis of the Data Encryption Standard”, Springer-Verlag, 1993

* <http://cs.ucsb.edu/~koc/ccs130h/notes/dc1.pdf>

Linear cryptanalysis

- Matsui : Eurocrypt93, Crypto94
- Look for correlations between key, cipher input and output



“Linear Cryptanalysis Method for DES Cipher”, Eurocrypt 93

Key space against brute-force search

- Consider brute-force search of key space; assume one key can be tested per clock cycle
- Desktop computer $\approx 2^{57}$ keys/year
- Supercomputer $\approx 2^{80}$ keys/year
- Supercomputer since Big Bang $\approx 2^{112}$ keys
- Modern key space: 2^{128} keys or more

Key length recommendation

Date	Minimum of Strength	Symmetric Algorithms	Factoring Modulus	Discrete Key	Logarithm Group	Elliptic Curve	Hash (A)	Hash (B)
(Legacy)	80	2TDEA*	1024	160	1024	160	SHA-1**	
2016 - 2030	112	3TDEA	2048	224	2048	224	SHA-224 SHA-512/224 SHA3-224	
2016 - 2030 & beyond	128	AES-128	3072	256	3072	256	SHA-256 SHA-512/256 SHA3-256	SHA-1
2016 - 2030 & beyond	192	AES-192	7680	384	7680	384	SHA-384 SHA3-384	SHA-224 SHA-512/224
2016 - 2030 & beyond	256	AES-256	15360	512	15360	512	SHA-512 SHA3-512	SHA-256 SHA-512/256 SHA-384 SHA-512 SHA3-512

NIST recommendation (2016)

(<https://www.keylength.com/en/4/>)

3DES

- Let $E : K \times M \rightarrow M$ be a block cipher
- Define $3E: K^3 \times M \rightarrow M$ as

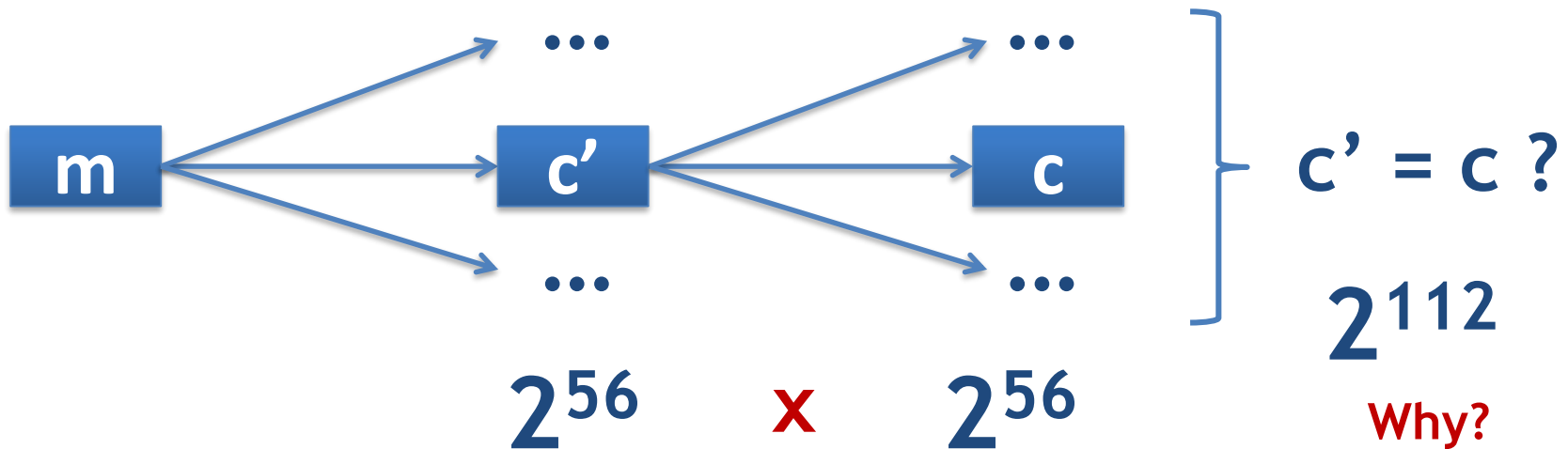
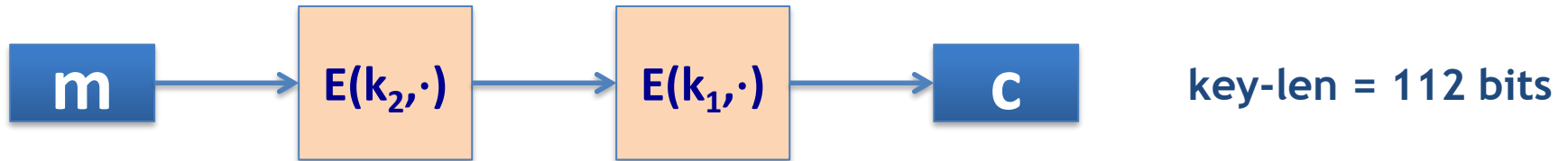
$$3E((k_1, k_2, k_3), m) = E(k_1, D(k_2, E(k_3, m)))$$

- Q. Why should we use **EDE** rather than **EEE**?
- key-size = $3 \times 56 = 168$ bits. But, $3 \times$ slower than DES.
- There exists a simple attack in time $\approx 2^{118}$

Q. What if $k_1 = k_2 = k_3$? **DES**

How about 2DES?

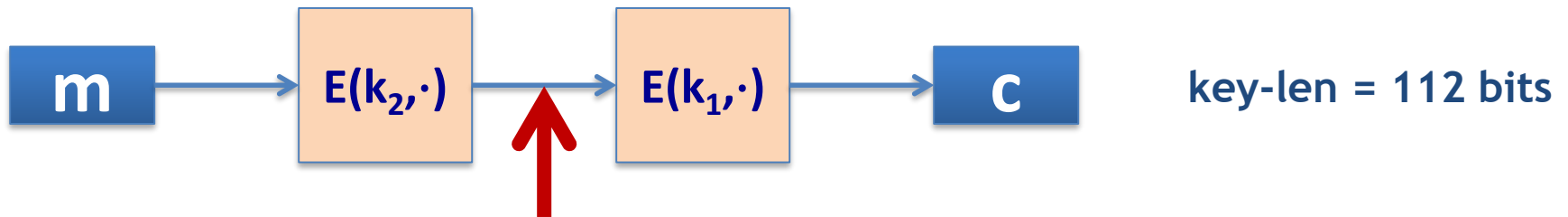
Define $2E((k_1, k_2), m) = E(k_1, E(k_2, m))$



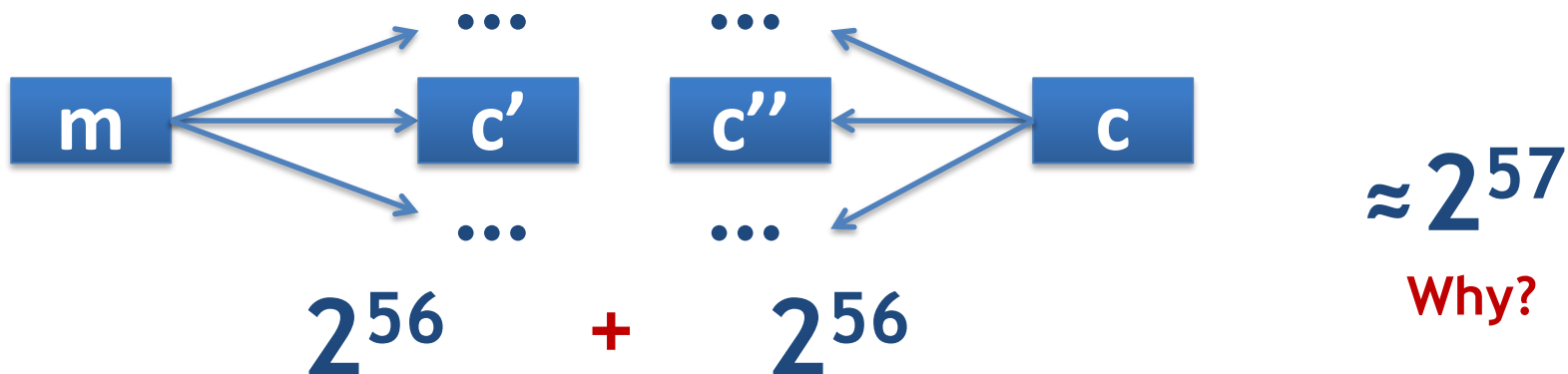
It looks good, right?

Meet in the middle attack (1)

- Define $2E((k_1, k_2), m) = E(k_1, E(k_2, m))$

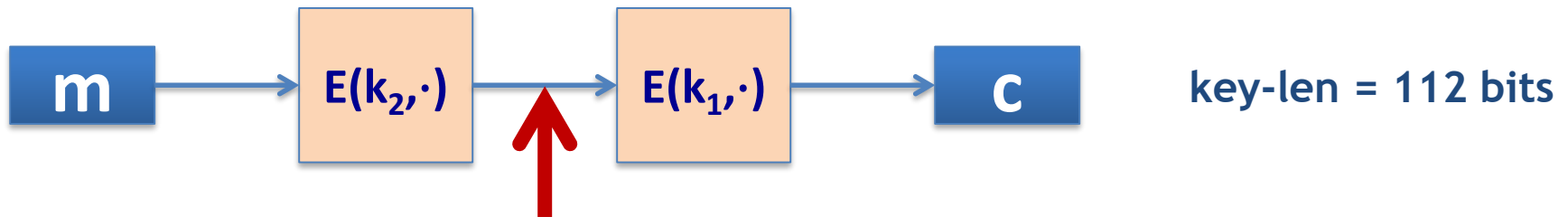


Idea: key found when $c' = c''$: $E(k_i, m) = D(k_j, c)$



Meet in the middle attack (2)

- Define $2E((k_1, k_2), m) = E(k_1, E(k_2, m))$



Assumption: the attacker knows a pair of (m, c) .

1. build table and then **sort** on 2nd column. **Q. Why?**
2. For all k all $k \in \{0, 1\}^{56}$ do: test if $D(k, c)$ is in the 2nd column. If so then **$E(k^i, m) = D(k, c) \rightarrow k_2: k^i$ and $k_1: k$**

$k^0 = 00 \dots 00$	$E(k^0, m)$	} 2^{56} entries
$k^1 = 00 \dots 01$	$E(k^1, m)$	
$k^2 = 00 \dots 10$	$E(k^2, m)$	
\vdots	\vdots	
$k^N = 11 \dots 11$	$E(k^N, m)$	

Same attack on 3DES: Time = 2^{118} , space $\approx 2^{56}$

AES contest

- 1997 : Call For AES Candidate Algorithms by NIST (National Institute of Standards and Technology)
 - ✓ 128-bit Block cipher
 - ✓ 128/192/256-bit keys
 - ✓ Worldwide-royalty free
 - ✓ More secure than Triple DES
 - ✓ More efficient than Triple DES
- 1998 : 1st Round Candidates – 15 Algorithms
 - ✓ Mars, Twofish, RC6, SAFER+, HPC, CAST256, DEAL, Frog, Magenta, Rijndael, DFC, Serpent, Crypton, E2, LOKI97
- 1999 : 2nd Round Candidates – 5 Algorithms
 - ✓ MARS, RC6, Rijndael, Serpent, and Twofish
- 2000. 10 : Rijndael selected as the finalist
- 2001. 12: official publication as FIPS PUB 197

(<http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html>,
<http://competitions.cr.yp.to/aes.html>)

1st Round candidates

Cipher	Submitted by	Country
CAST-256	Entrust	Canada
Crypton	Future Systems	Korea [†]
Deal	Outerbridge	Canada [†]
DFC	ENS-CNRS	France
E2	NTT	Japan
Frog*	TecApro	Costa Rica
HPC*	Schroeppe	USA
LOKI97*	Brown, Pieprzyk, Seberry	Australia
Magenta	Deutsche Telekom	Germany
Mars	IBM	USA [†]
RC6	RSA	USA [†]
Rijndael*	Daemen, Rijmen	Belgium [‡]
Safer+*	Cylink	USA [†]
Serpent*	Anderson, Biham, Knudsen	UK, Israel, Norway
Twofish*	Counterpane	USA [†]

* Placed in the public domain; [†] and foreign designers; [‡] foreign influence

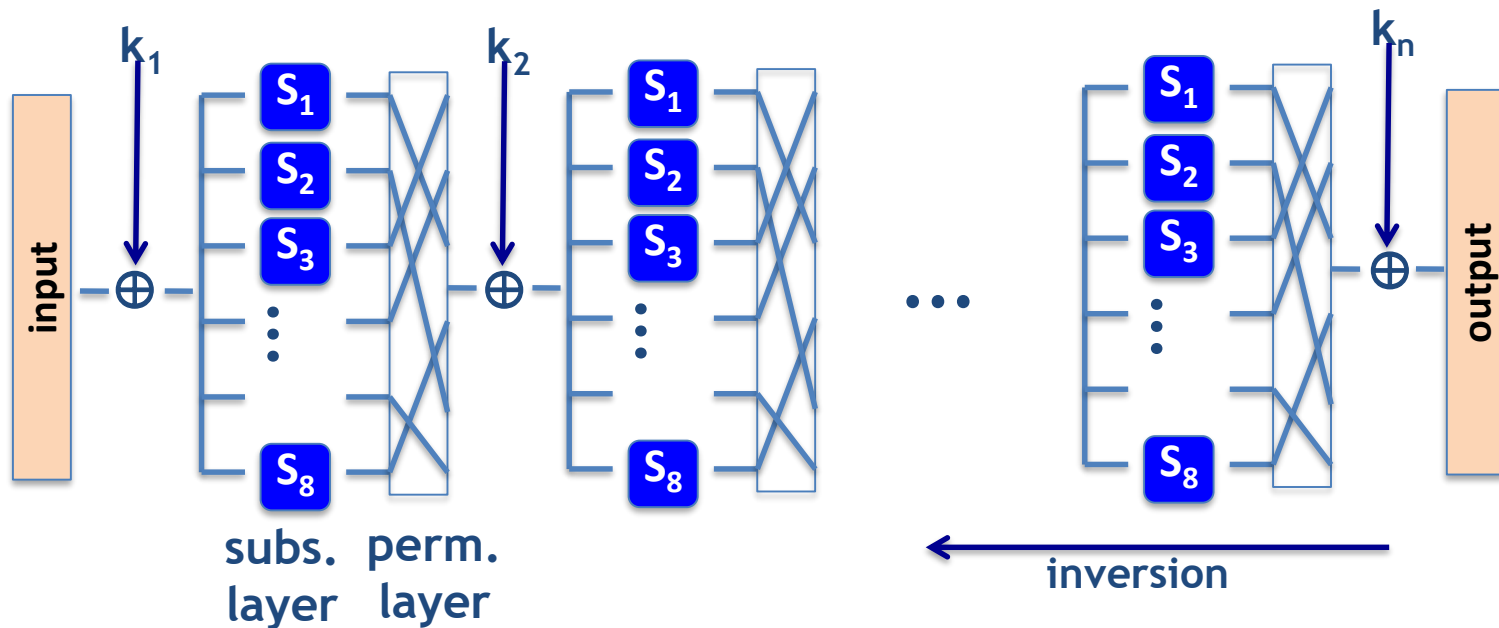
2nd Round candidates

Cipher	Submitter	Structure	Votes
MARS	IBM	Feistel	13 positive, 84 negative
RC6	RSA Lab.	Feistel	23 positive, 37 negative
Rijndael	Daemen, Rijmen	SPN	86 positive, 10 negative
Serpent	Anderson, Biham, Knudsen	SPN	59 positive, 7 negative
Twofish	Schneier et. al	Feistel	31 positive, 21 negative

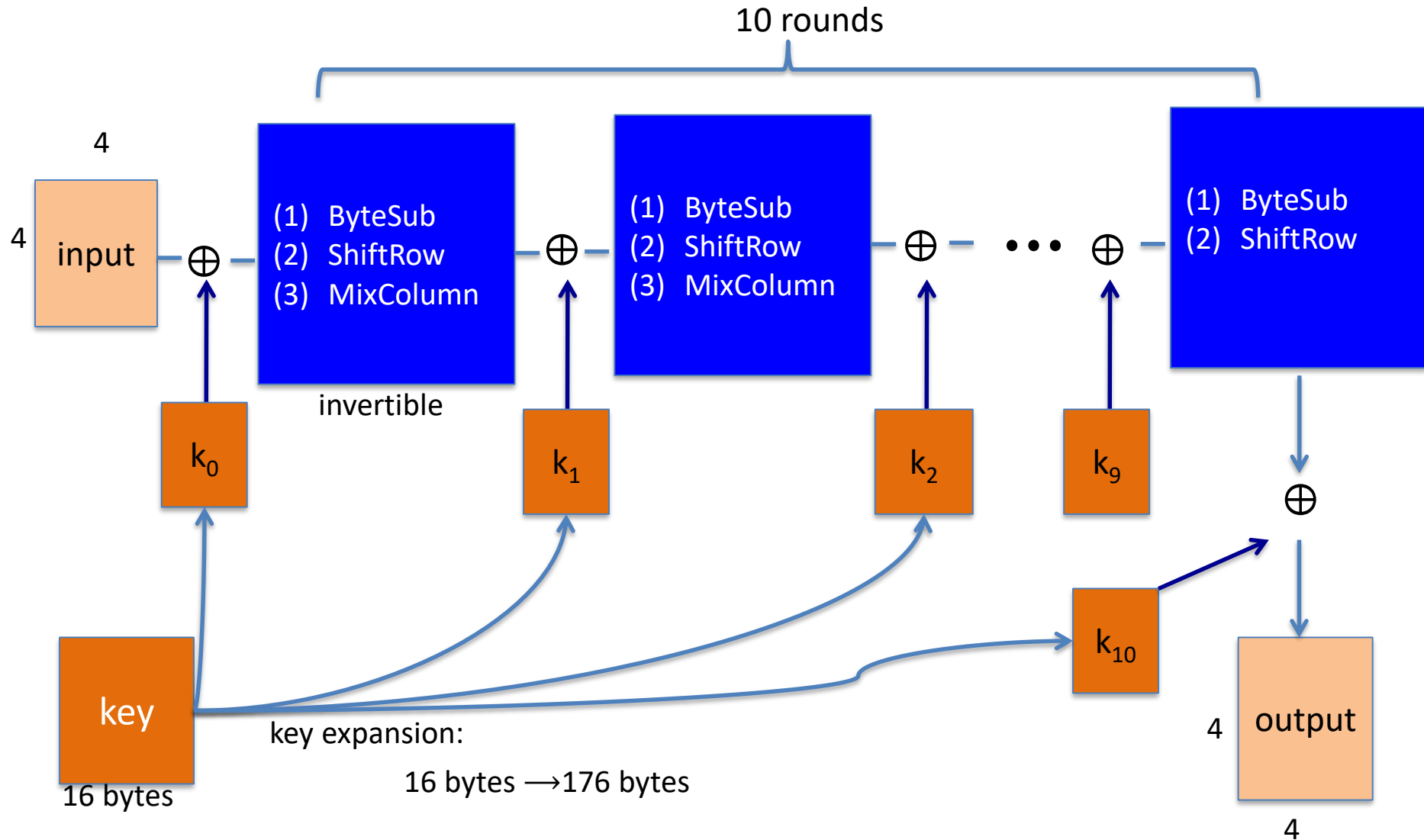
2000. 10 : Rijndael selected as the finalist

Advanced Encryption Standard (AES)

- AES is a standard symmetric encryption algorithm for US federal organizations
- AES has a **128-bit** block, arranged as 16 bytes
- AES is a Substitution-Permutation network (**not a Feistel**)



AES-128 schematic



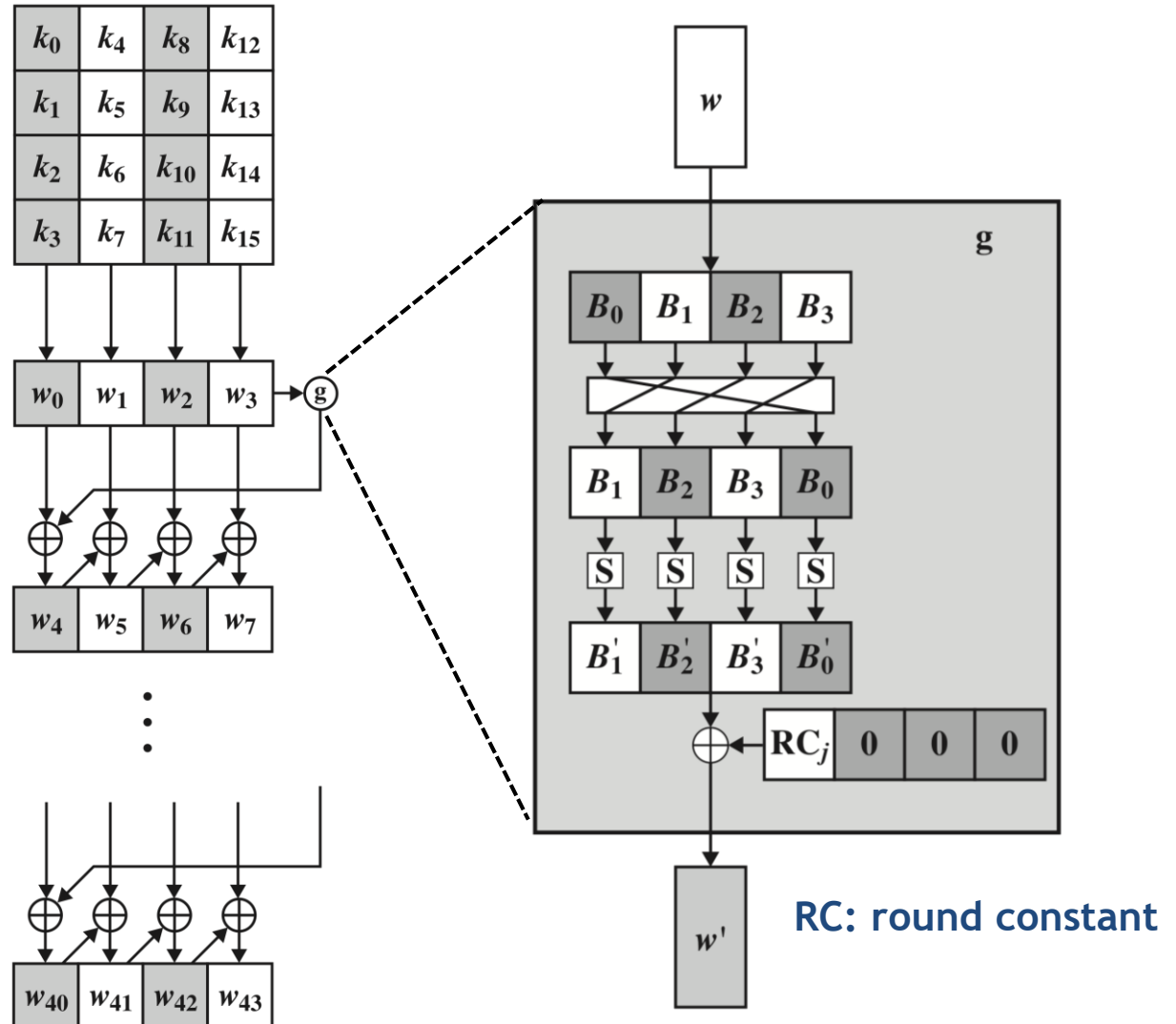
Key generation

before round 1

after round 1

⋮

after round 10



Four operations

1. *Byte Substitution* **confusion**

– predefined substitution table $s[i,j] \rightarrow s'[i,j]$

2. *Shift Row* **diffusion**


– left circular shift

3. *Mix Columns* **diffusion and confusion**

– 4 elements in each column are multiplied by a polynomial

4. *Add Round Key* **confusion**

– Key is derived and added to each column



This step is omitted for the last round

Source code for round functions

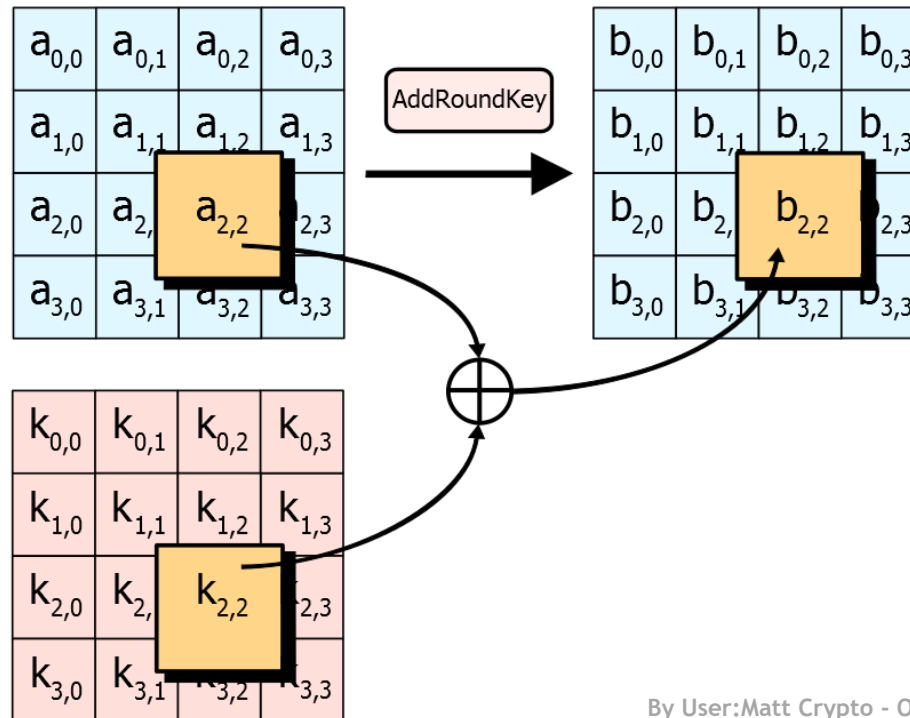
```
// first round
AddRoundKey(0);

// all rounds except for the first and last rounds
for(round=1;round<Nr;round++)
{
    SubBytes();
    ShiftRows();
    MixColumns();
    AddRoundKey(round);
}

// last round
SubBytes();
ShiftRows();
AddRoundKey(Nr);
```

Add round key

- Bitwise XOR state s with key k_0



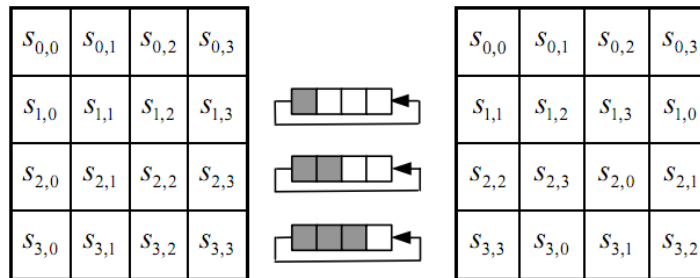
The round functions

- **ByteSub:** a 1 byte S-box. 256 byte table
(easily computable)

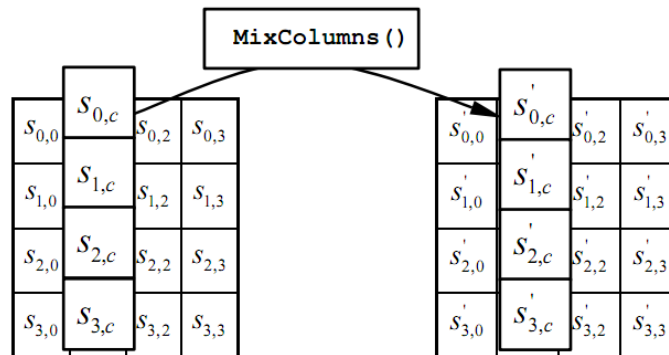
The only nonlinear elements:

$\text{ByteSub}(A_i) + \text{ByteSub}(A_j) \neq \text{ByteSub}(A_i + A_j)$, for $i, j = 0, \dots, 15$

- **ShiftRows:**

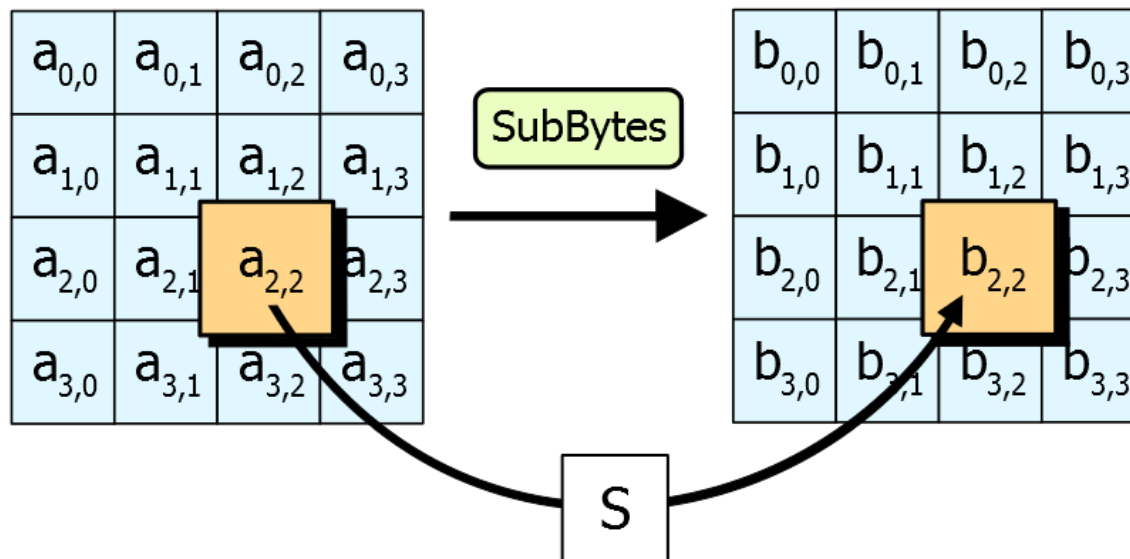


- **MixColumns:**



ByteSub

- For each round... (10 rounds total)
 - Substitute bytes
 - Use lookup table to switch positions



S-Box

	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

HEX 19 would get replaced with **HEX D4**

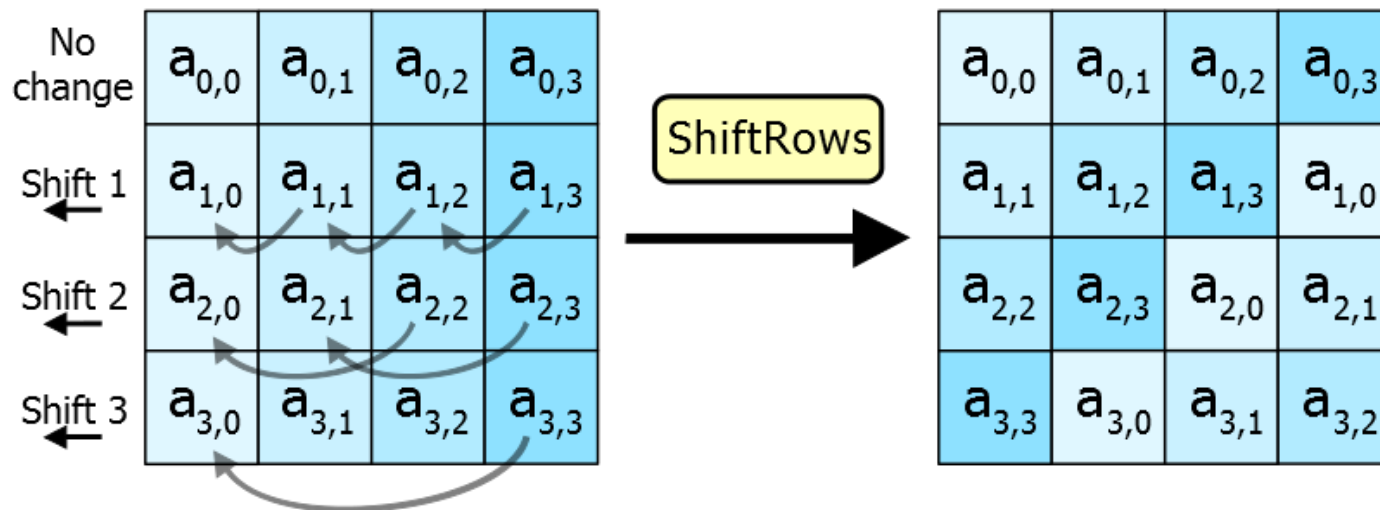
AES is a **byte**-oriented cipher

Source code for SubBytes()

```
// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
void SubBytes()
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[i][j] = getSBoxValue(state[i][j]);
        }
    }
}
```

ShiftRows

- For each round...
 - Shift rows



Source code for ShiftRows()

```
// The ShiftRows() function shifts the rows in the state.
// Each row is shifted with different offset.
// Offset = Row number. So the first row is not shifted.
void ShiftRows()
{
    unsigned char temp;

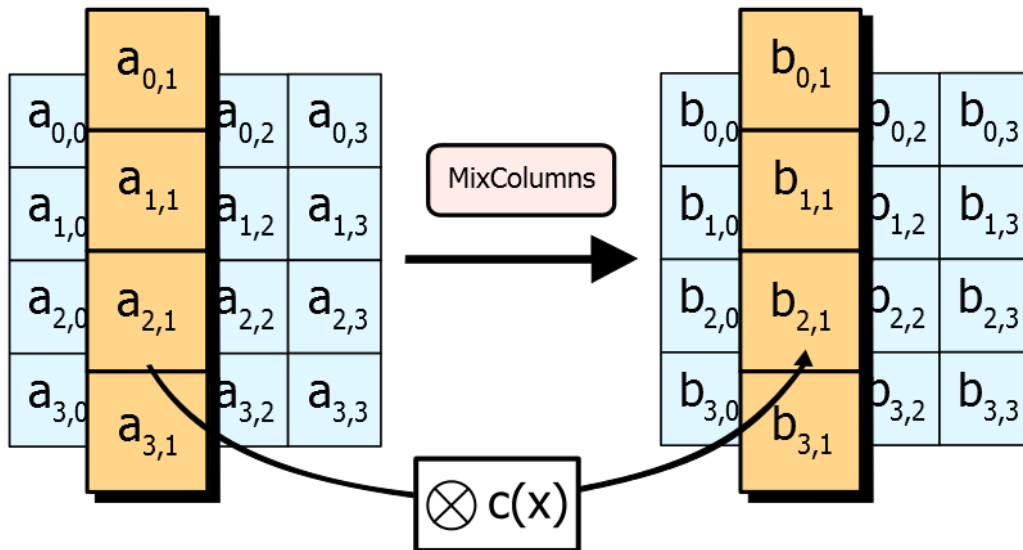
    // Rotate second row 1 columns to left
    temp=state[1][0];
    state[1][0]=state[1][1];
    state[1][1]=state[1][2];
    state[1][2]=state[1][3];
    state[1][3]=temp;

    ...
    // Skip the codes for the third and fourth rows
}
```


MixColumns

- For each round...
 - Mix columns

- Multiply by constant matrix $c(x) = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$



AES decryption

- To decrypt, process must be invertible
- Inverse of AddRoundKey is easy since “ \oplus ” is its own inverse
- MixColumn is invertible (inverse is also implemented as a lookup table)
- Inverse of ShiftRow is easy (cyclic shift the other direction)
- ByteSub is invertible (inverse is also implemented as a lookup table)

Performance

Cipher	Type	Key size	Speed (MB/sec)
MD5	Hash	--	255
DES	Block cipher	64	32
3DES	Block cipher	168	13
AES-128	Block cipher	128	109

AMD CPU (2.2 GHz), Linux, Crypto++ 5.6.0
(<http://www.cryptopp.com/benchmarks.html>)

AES in hardware

AES instructions in Intel Westmere:

- **aesenc, aesenclast:** do one round of AES
128-bit registers: xmm1=state, xmm2=round key
aesenc xmm1, xmm2 ; puts result in xmm1
- **aeskeygenassist:** performs AES key expansion
- Claim **14 x speed-up** over OpenSSL on same hardware
- <https://software.intel.com/en-us/articles/download-the-intel-aesni-sample-library>

Similar instructions on AMD Bulldozer

Known attacks on the AES

- Best key recovery attack:
four times better than exhaustive Search

[Bogdanov, Khovratovich and Rechberger, 2011]

- Related key attack on AES-256:

Given 2^{99} input/output pairs, we can recover keys in time $\approx 2^{99}$

[Biryukov and Khovratovich, 2009]

Summary of symmetric key ciphers

- Stream cipher — like a one-time pad
 - Key is stretched into a long **key stream** (using a pseudo random generator)
 - Key stream is used just like a one-time pad
 - Employs “substitution” only
 - Example: RC4, A5/1
- Block cipher
 - Employs **both “substitution” and “transposition”**
 - Examples: DES, 3DES, AES

Application: Storing a file securely

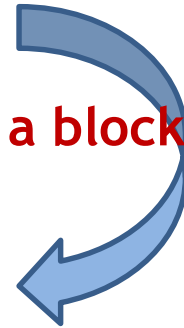


$$M = M_0, M_1, M_2, \dots, M_{N-1}$$



$$C = C_0, C_1, C_2, \dots, C_{N-1}$$

by a block cipher



What can attacker learn from captured C?

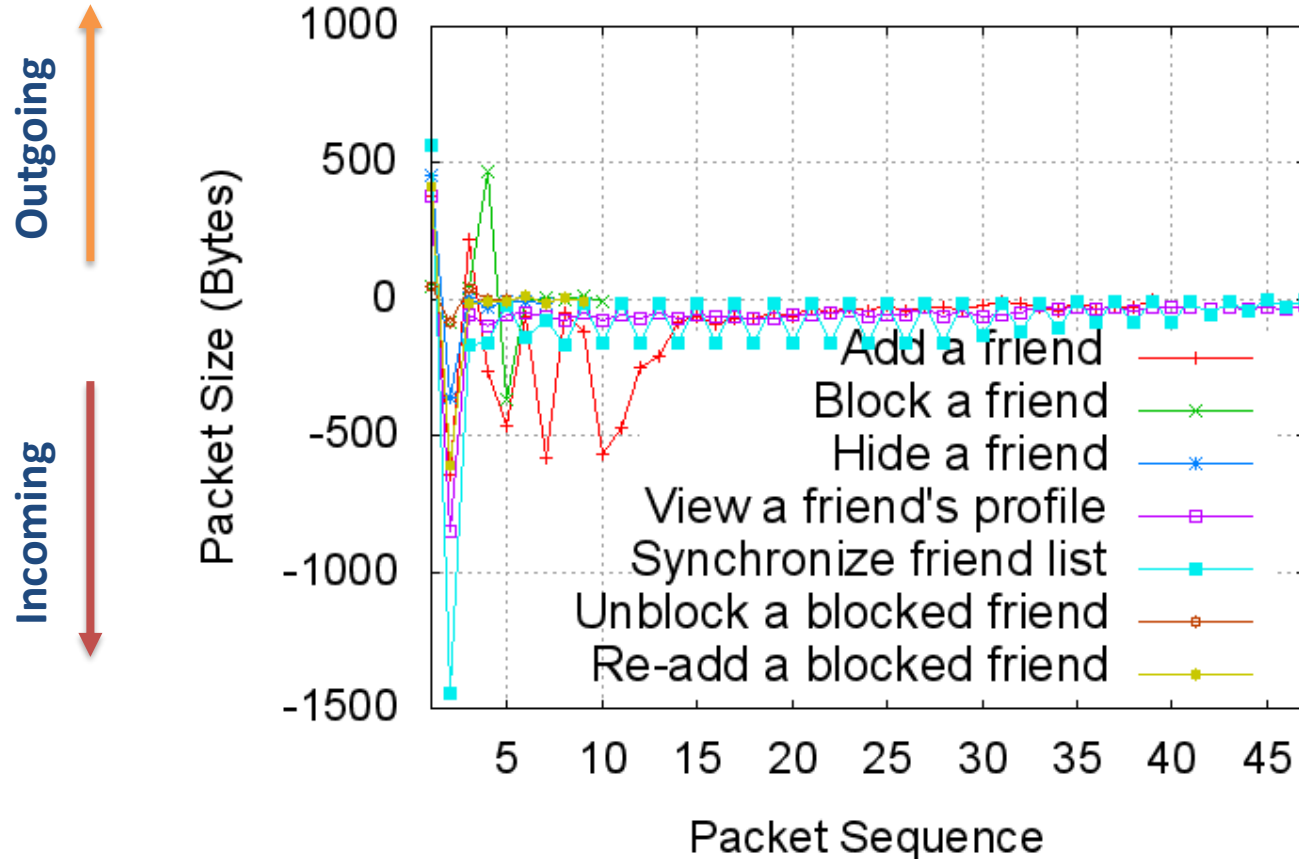
The length of M (i.e., file size)

Which blocks in M are equal

Encryption and plaintext length

- In practice, we use encryption schemes that can encrypt arbitrary-length messages.
- In general, encryption does not hide the plaintext length which might be used for traffic analysis.
- Beware that leaking plaintext length can often lead to problems!
 - Database searches (through the size of responses)
 - For example, user activities in KakaoTalk can be identified with about 99.7% accuracy.

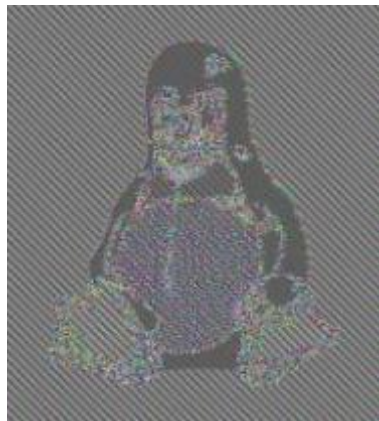
Traffic analysis in KakaoTalk



“Encryption Is Not Enough: Inferring user activities on KakaoTalk with traffic analysis”, WISA 2015

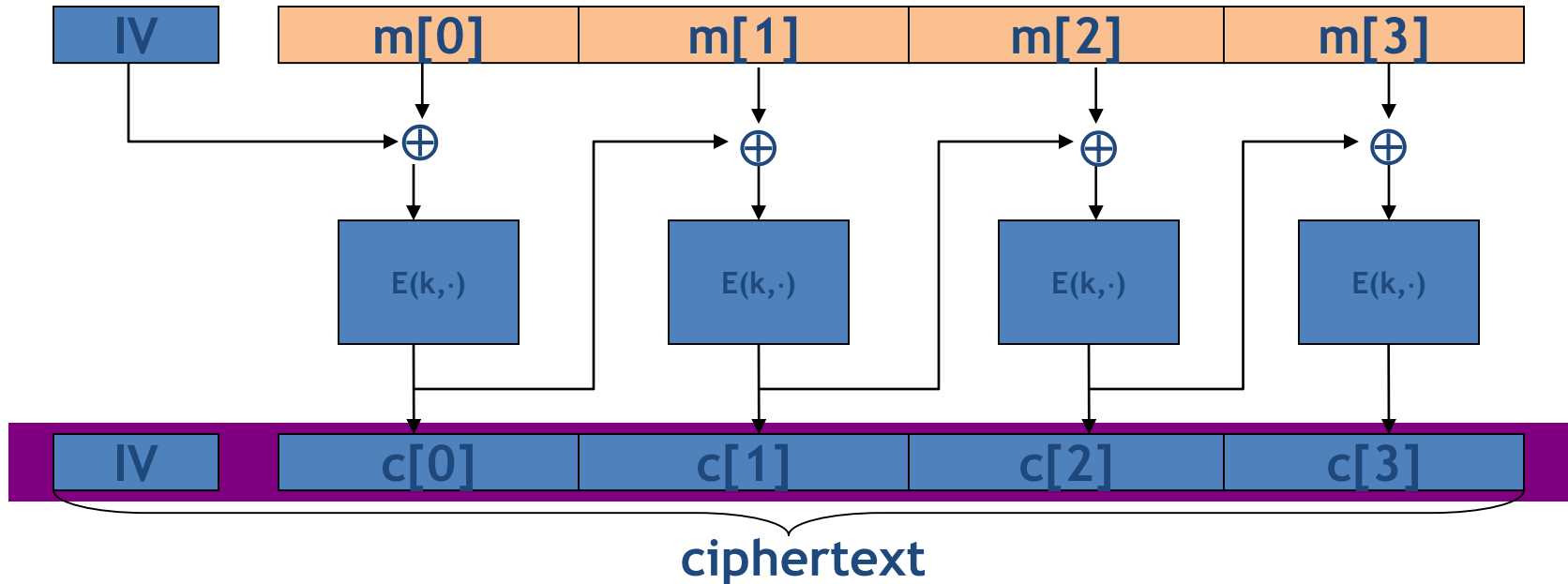
Modes of operation - ECB

ECB – electronic codebook – mode just encrypts a block at a time



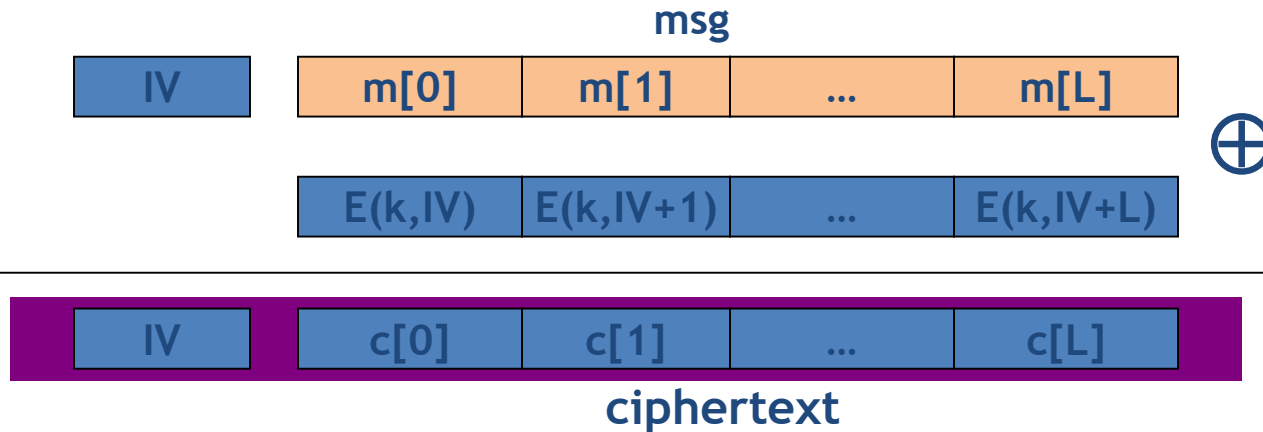
Patterns can still be fairly obvious!

Modes of operation - CBC



- Cipher block chaining (CBC) was the traditional mode for bulk encryption
- If attacker can predict IV, CBC is not secure against Chosen Plaintext Attack.
 - Attacker uses $M \oplus IV$ instead of M .
- Error propagates

Modes of operation - CTR



- Counter mode (encrypt a counter to get keystream)
- Unlike CBC, one encryption per block – and **parallelizable!**
- Random access is possible
- Efficient for software and hardware
- Used in various protocols (e.g., SSH, IPSEC ...)


Quiz

Q. Suppose Alice forgets the value she used for IV (initialization vector), has ciphertext (encrypted with CBC) and key. Can she recover plaintext m ?

1. No
- 2. Almost everything except $m[0]$
3. Almost everything except $m[0]$ and $m[1]$
4. Can only recover $m[n-1]$

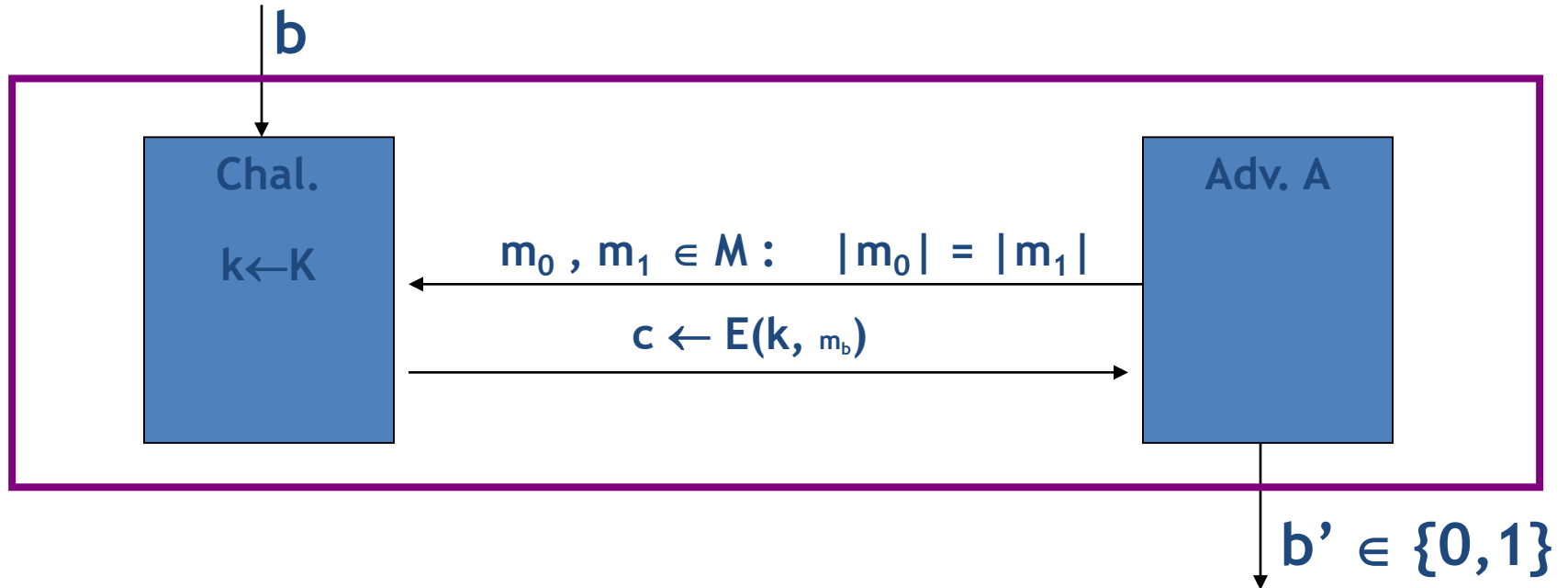
Quiz

Q. If Alice wants to quickly encrypt a large file by using many processors, which mode is preferred?

1. CBC or CTR
2. ECB
3. CBC
-  4. CTR

Semantic security for one-time key

- $\mathbb{E} = (E, D)$ a cipher defined over (K, M, C)
- For $b=0,1$ define $\text{EXP}(b)$ as:



- Def: \mathbb{E} is sem. sec. for one-time key if for all “efficient” A :

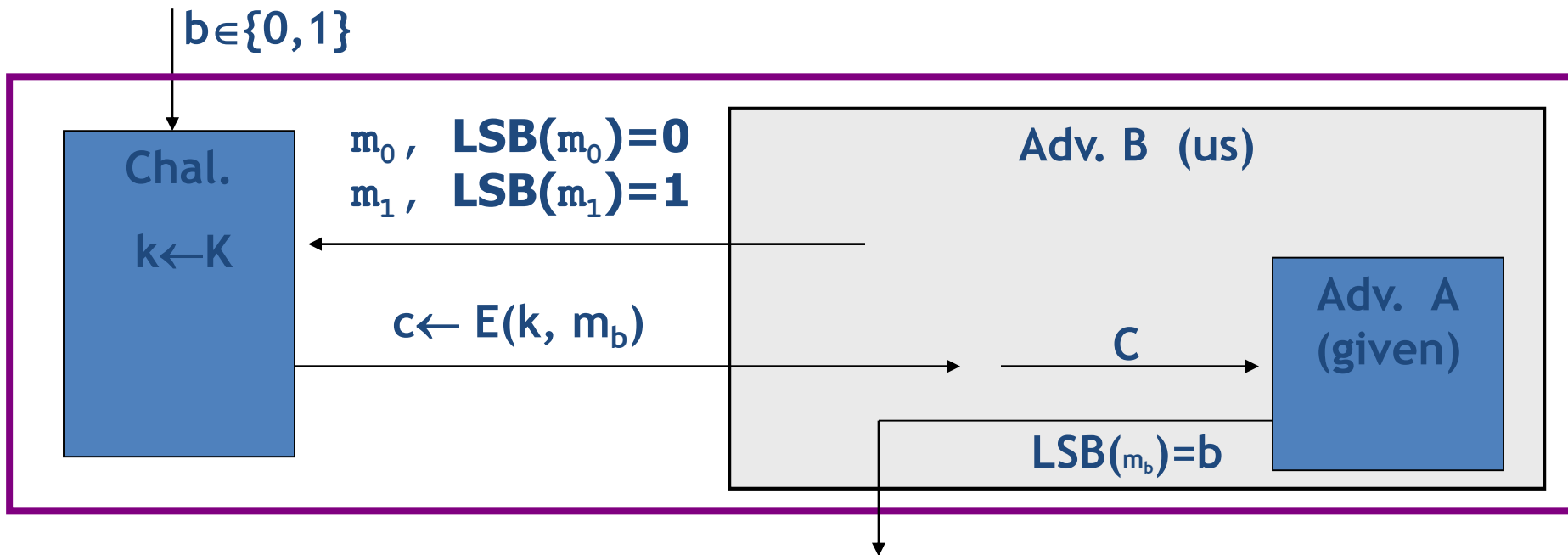
$$\text{Adv}_{ss}[A, \mathbb{E}] = | \Pr[\text{EXP}(0)=1] - \Pr[\text{EXP}(1)=1] |$$

is “negligible.”

Semantic security (cont.)

Sem. Sec. \Rightarrow no “efficient” adversary learns info about plaintext from a single ciphertext.

Example: suppose efficient A can deduce LSB of plaintext from ciphertext.
Then $\mathbb{E} = (E, D)$ is not semantically secure.

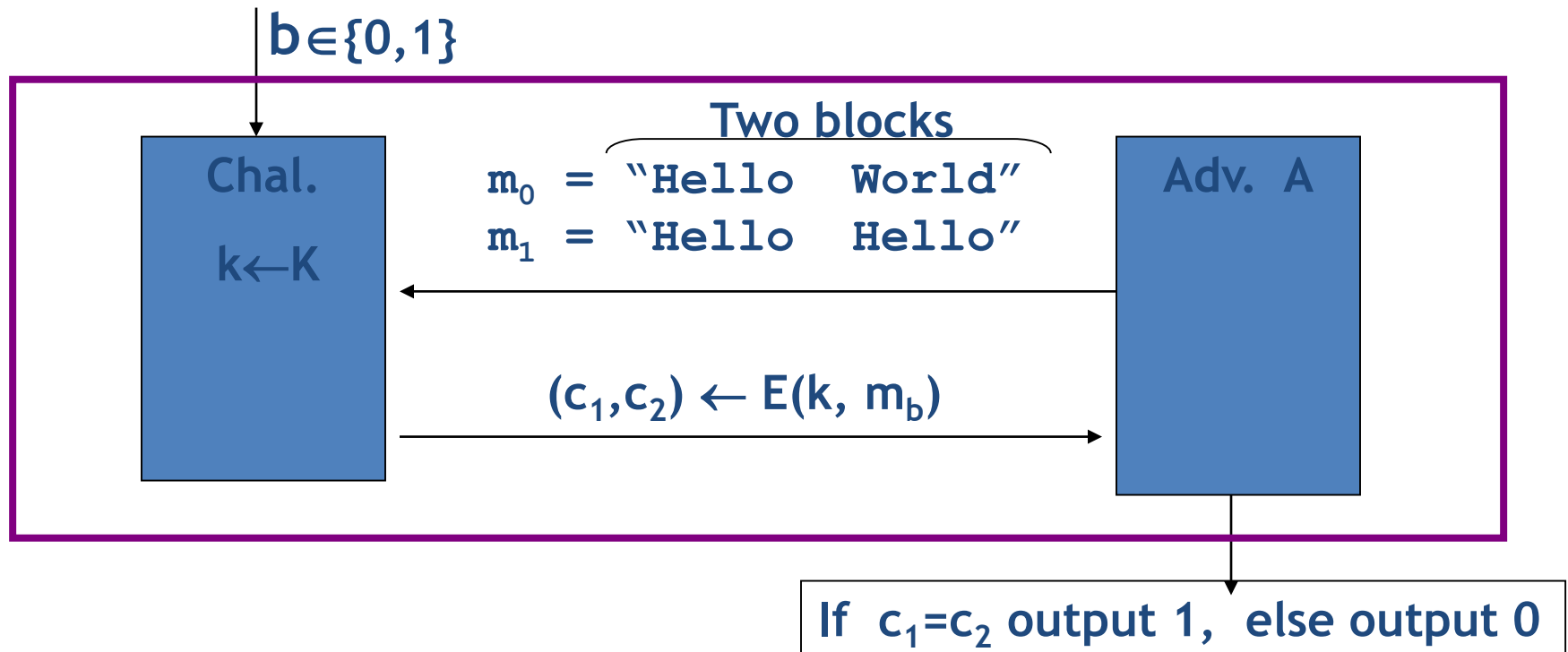


Then $\text{Adv}_{ss}[B, \mathbb{E}] = 1 \Rightarrow \mathbb{E}$ is not sem. sec.

ECB is not Sem. Sec.

Electronic Code Book (ECB):

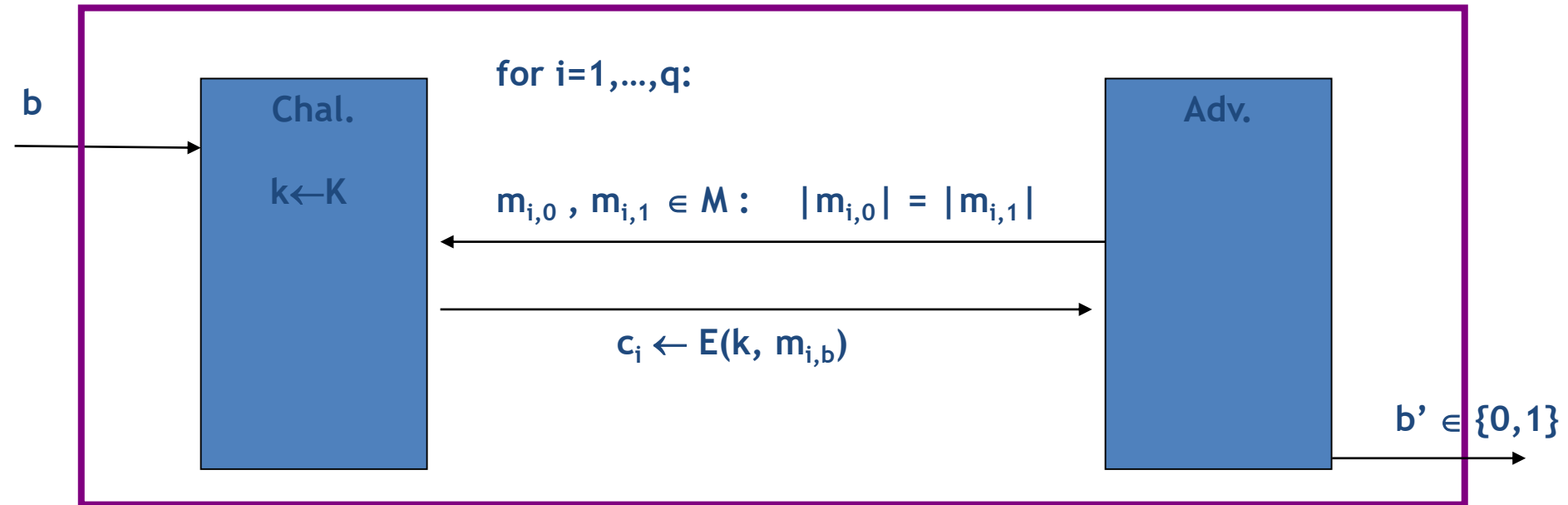
- Not semantically secure for messages that contain more than one block.



Then $\text{Adv}_{ss}[A, \text{ECB}] = 1$

Semantic security for many-time key (CPA security)

- CIPHER $\mathbb{E} = (E, D)$ defined over (K, M, C) .
- For $b=0,1$ define $\text{EXP}(b)$ as:



- Def: \mathbb{E} is sem. sec. under CPA if for all “efficient” A :

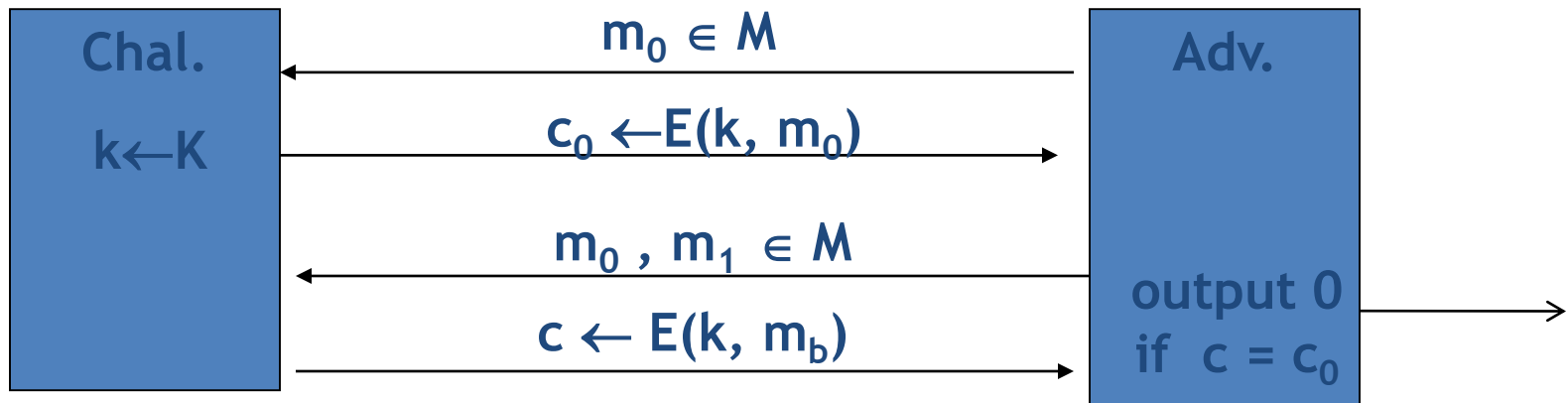
$$\text{Adv}_{\text{CPA}}[A, \mathbb{E}] = \left| \Pr[\text{EXP}(0)=1] - \Pr[\text{EXP}(1)=1] \right|$$

is “negligible.”

Security for many-time key

Fact: stream ciphers are insecure under CPA.

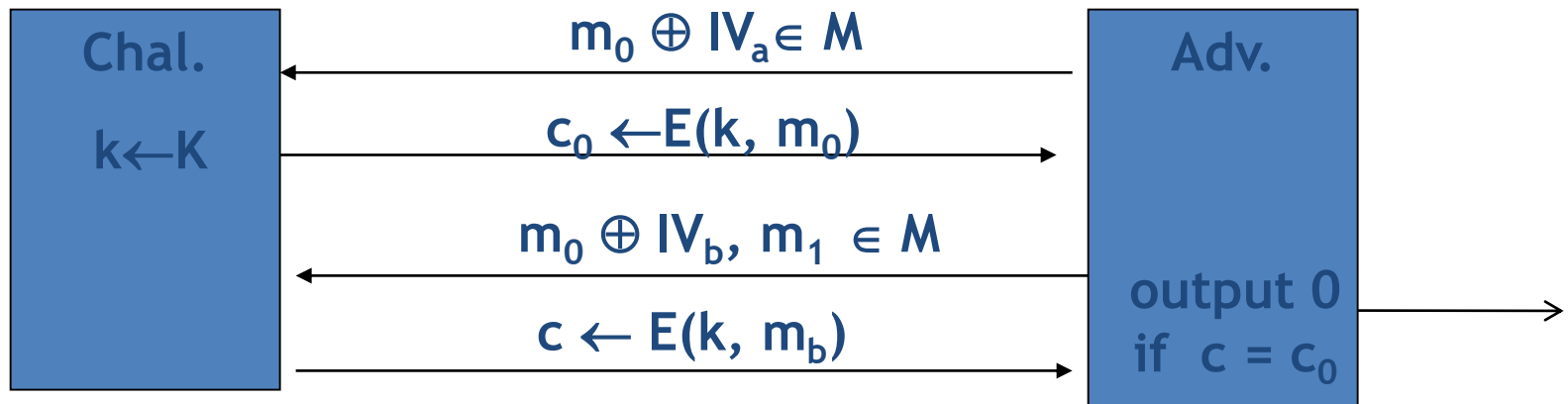
- More generally: if $E(k, m)$ always produces same ciphertext, then cipher is insecure under CPA.



If secret key is to be used multiple times \Rightarrow

given the same plaintext message twice,
the encryption alg. must produce different outputs.

CBC is not Sem. Sec. (when IV is predictable)



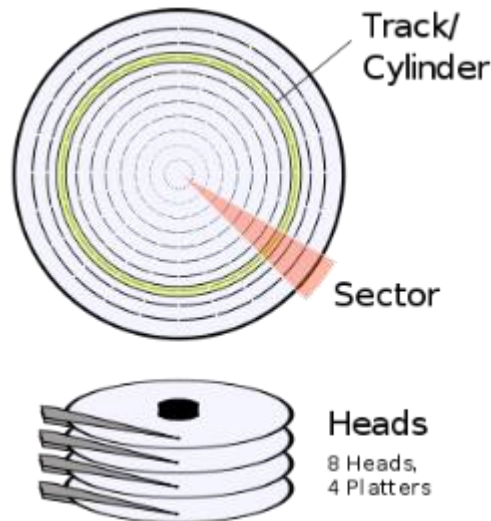
Then $\text{Adv}_{ss}[A, \text{CBC}] = 1$

CBC where attacker can predict the IV is not CPA-secure.

Tweakable Encryption (about AES-XTS)

Disk encryption: no expansion

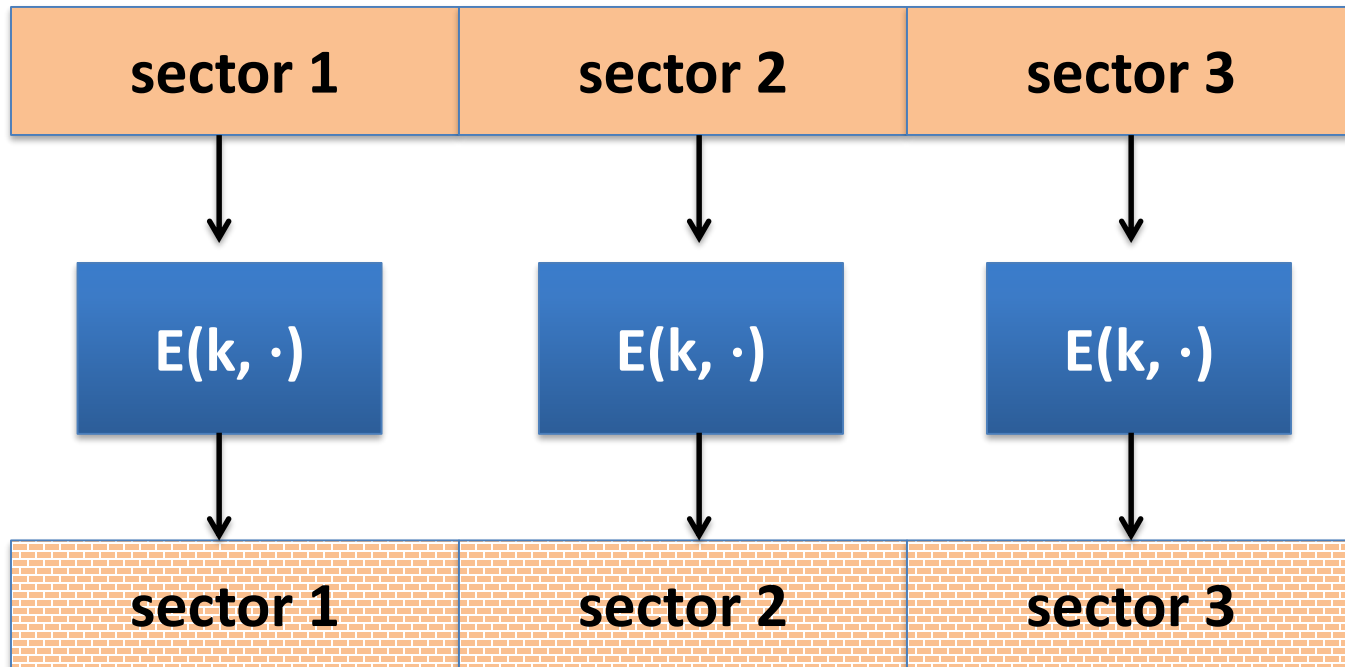
Used for encrypting *fixed*-length data units.



For example, sectors on disk are fixed size (e.g. 4KB).

[Truecrypt](#) is the best-known implementation of XTS, and is happily open-source

Disclosure of same information



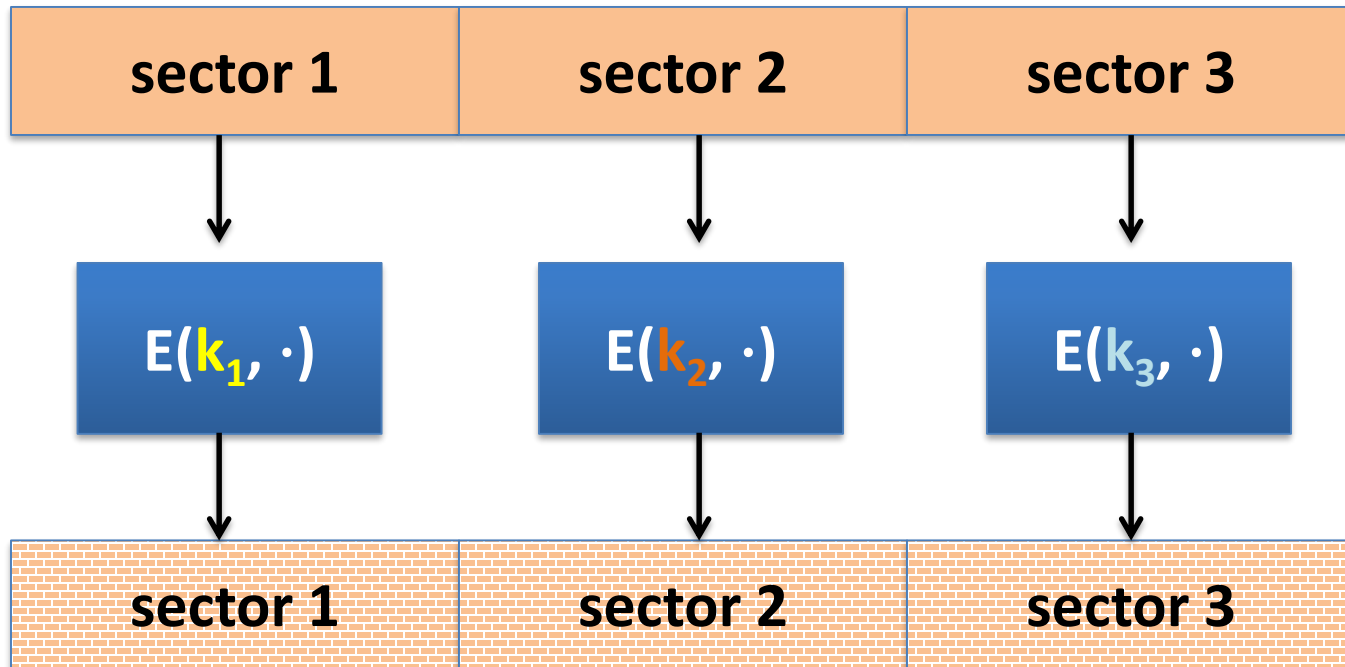
sector 1 and sector 3 may have same content

- Leaks same information as ECB mode
- E.g., finding empty section on your disk

Limitation of existing modes

- Secure CBC can be implemented with **random IVs**, but the device offers no place to store them explicitly
- Sector numbers can be used as IVs. But then the IVs are predictable; attackers can generate plaintexts that cancel them out for CBC and CTR

Avoiding the leakage problem



Managing keys: the trivial construction

$$k_t = F(k, t) \quad , t=1,2,3\dots$$

Tweakable block ciphers

Goal: construct many different encryptions (based on **location of disk** where ciphertext file is to be stored) from **a key** $k \in K$.

Syntax: $E, D : K \times T \times X \rightarrow X$

for every $t \in T$ and $k \leftarrow K$:

$E(k, t, \cdot)$ is an invertible function on X , indistinguishable from random

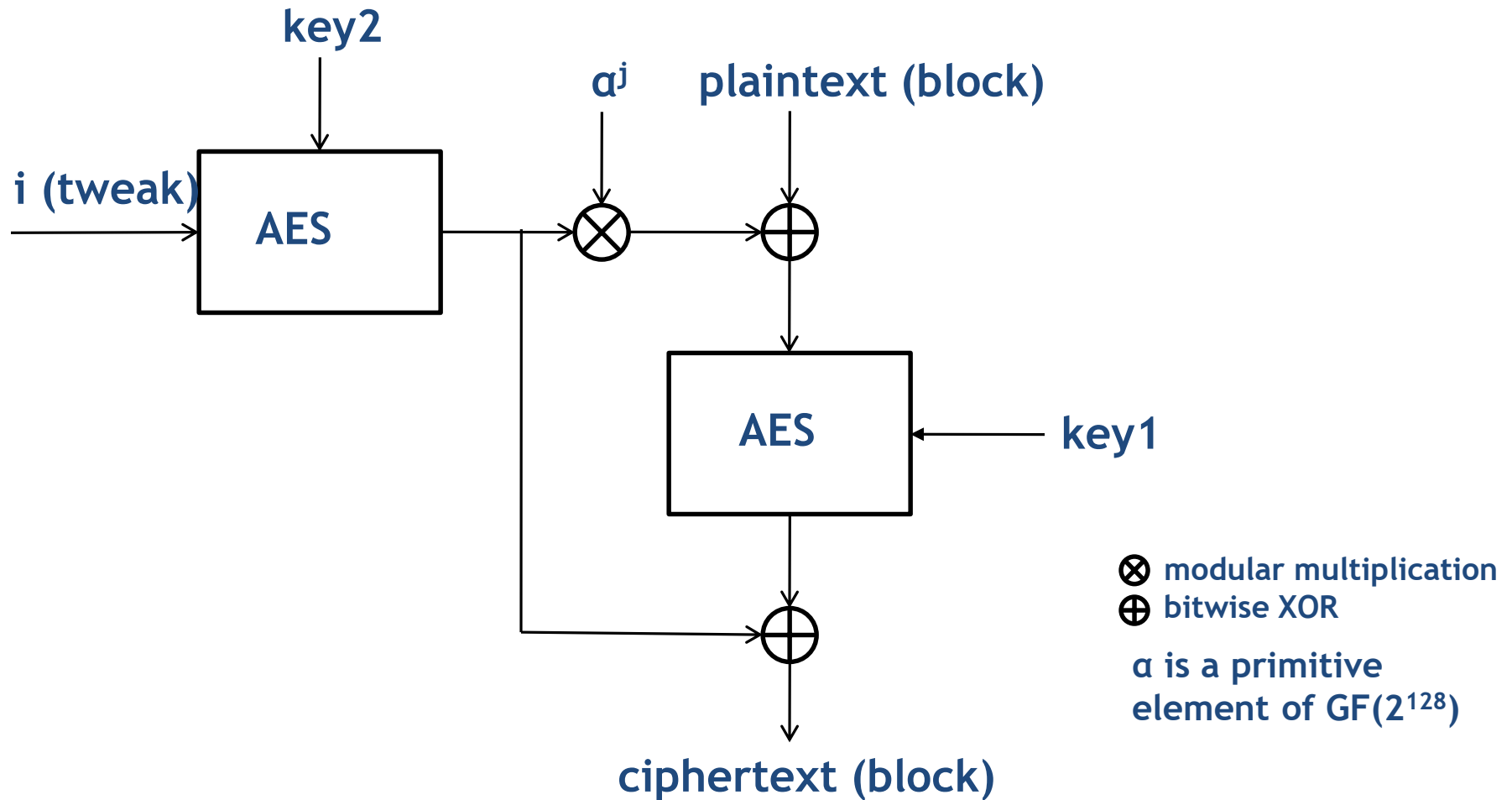
Application: use sector number as the tweak

\Rightarrow every sector gets its own independent encryption

AES-XTS mode

- Encryption of block j is function of:
 - 128 bit keys K_1 and K_2
 - “Tweak” value i (i.e., sector number)
 - Each sector assigned different tweak value consecutively (like counter in CTR mode)
 - Multiplier α^j
 - $\alpha = 000\dots00010$ (that is, α in $\text{GF}(2^{128})$)
 - $\alpha^j = \alpha$ multiplied by itself j times mod $x^{128}+x^7+x^2+x+1$
 - Different for each block j in sector i

AES-XTS mode



i : sector number, j : j_{th} block

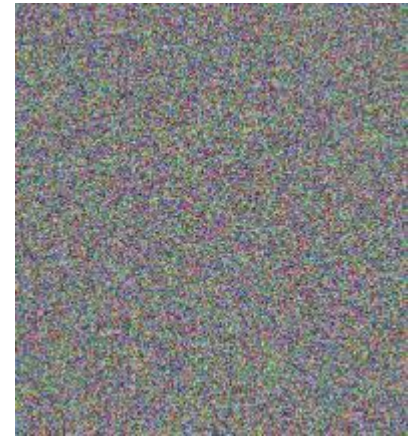
Revisit the previous example



Original



Encrypted with
ECB



Encrypted with
CBC/CTR

CBC and CTR modes *are* secure against
chosen-plaintext attacks

What is the best recommendation?

It depends on the situation.

Overall, **CTR** is the best and most modern way to achieve privacy-only encryption.

It is insecure if a **nonce gets reused** on encryption or decryption.

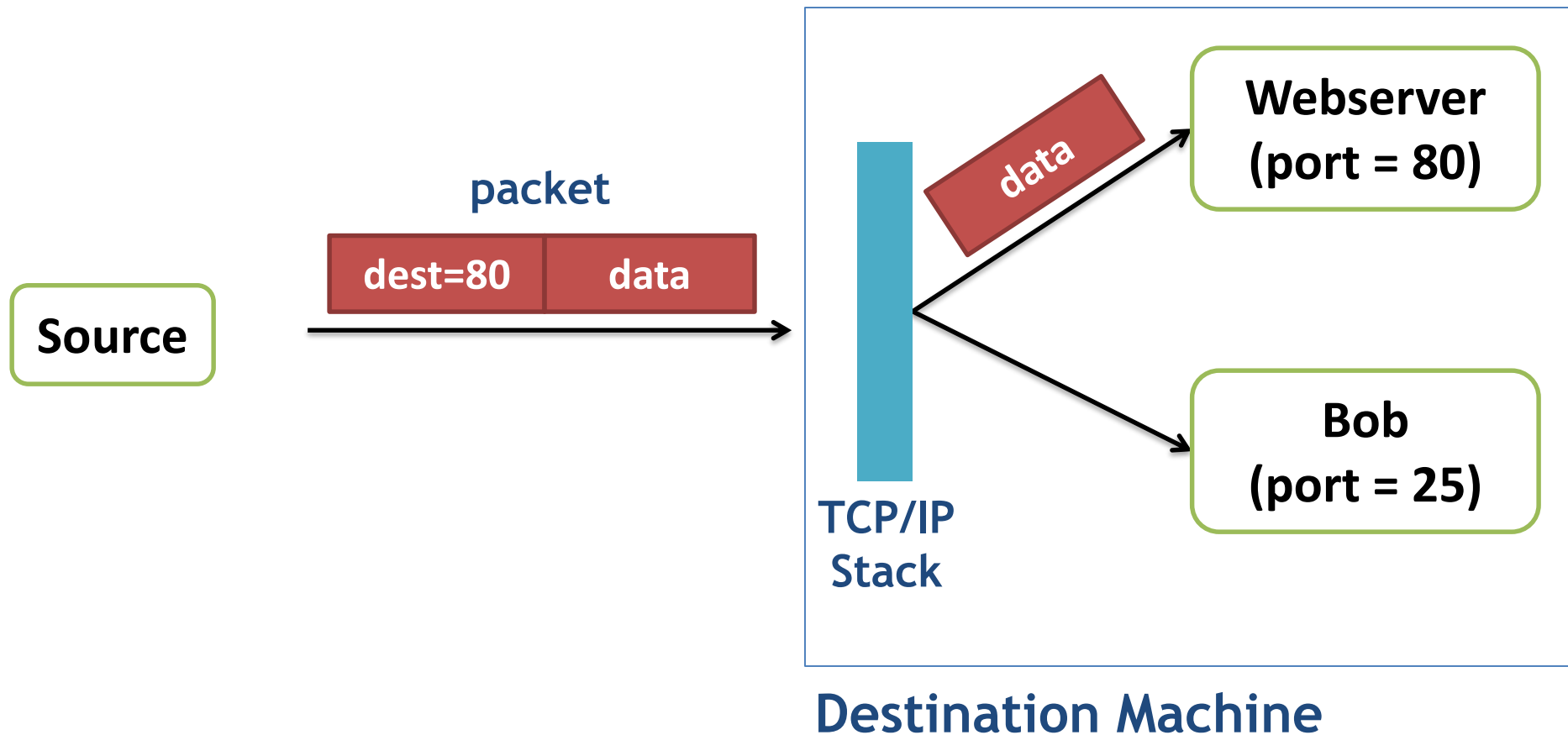
However ...

CBC and CTR modes *are* not secure against chosen-ciphertext attacks.

CPA security cannot guarantee security under **active attacks**.

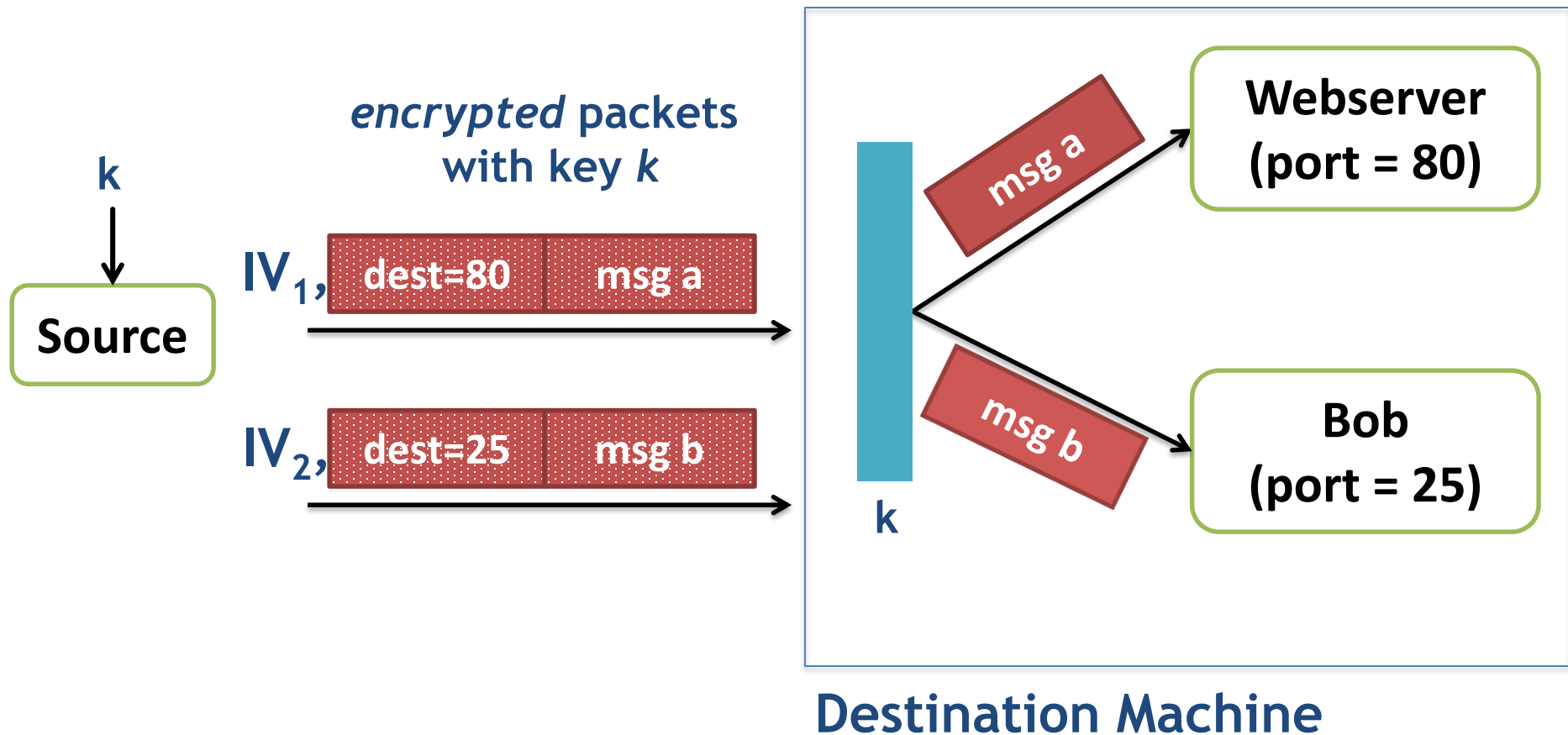
Example tampering attack

TCP/IP (highly abstracted)



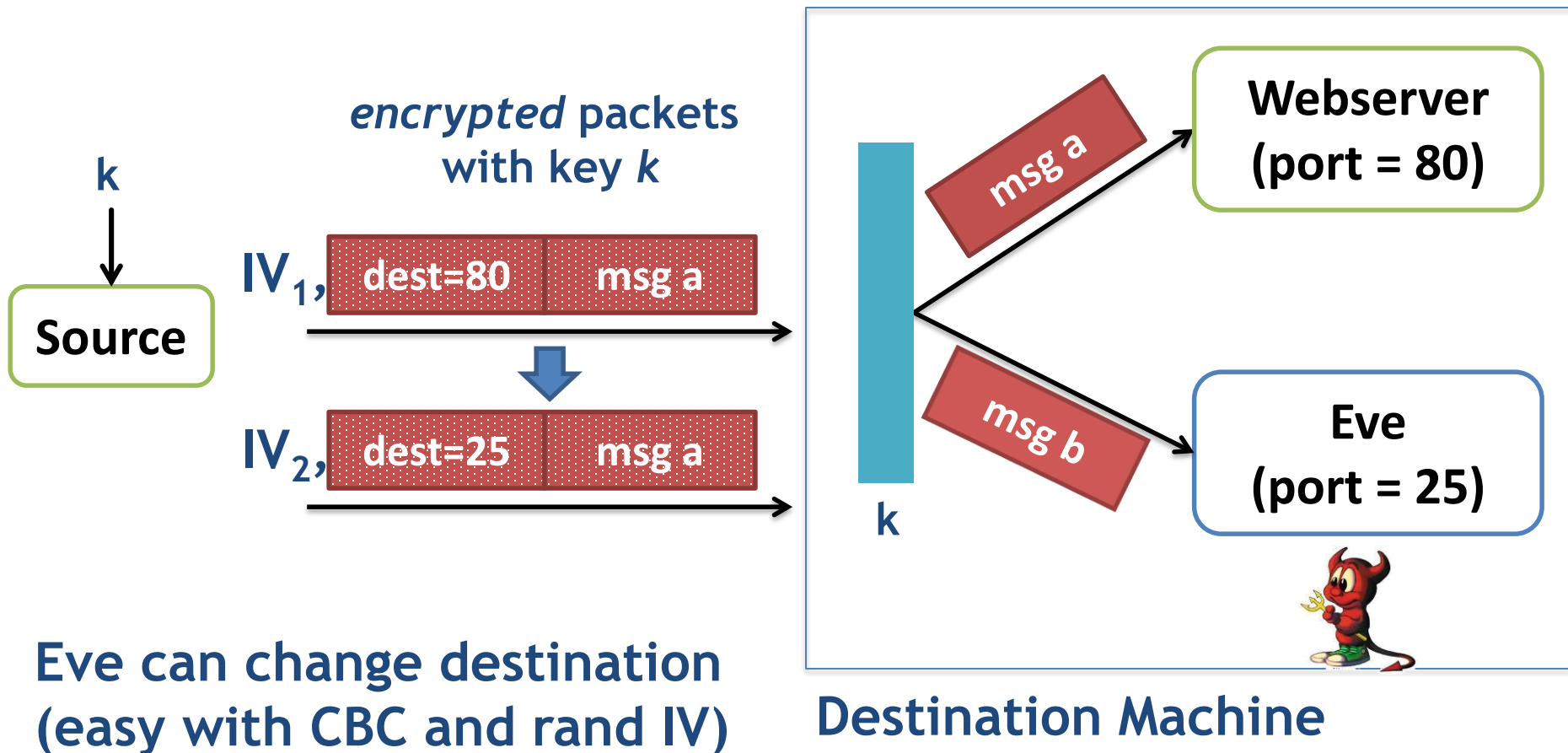
Example tampering attack

Encrypted with CBC and random IV



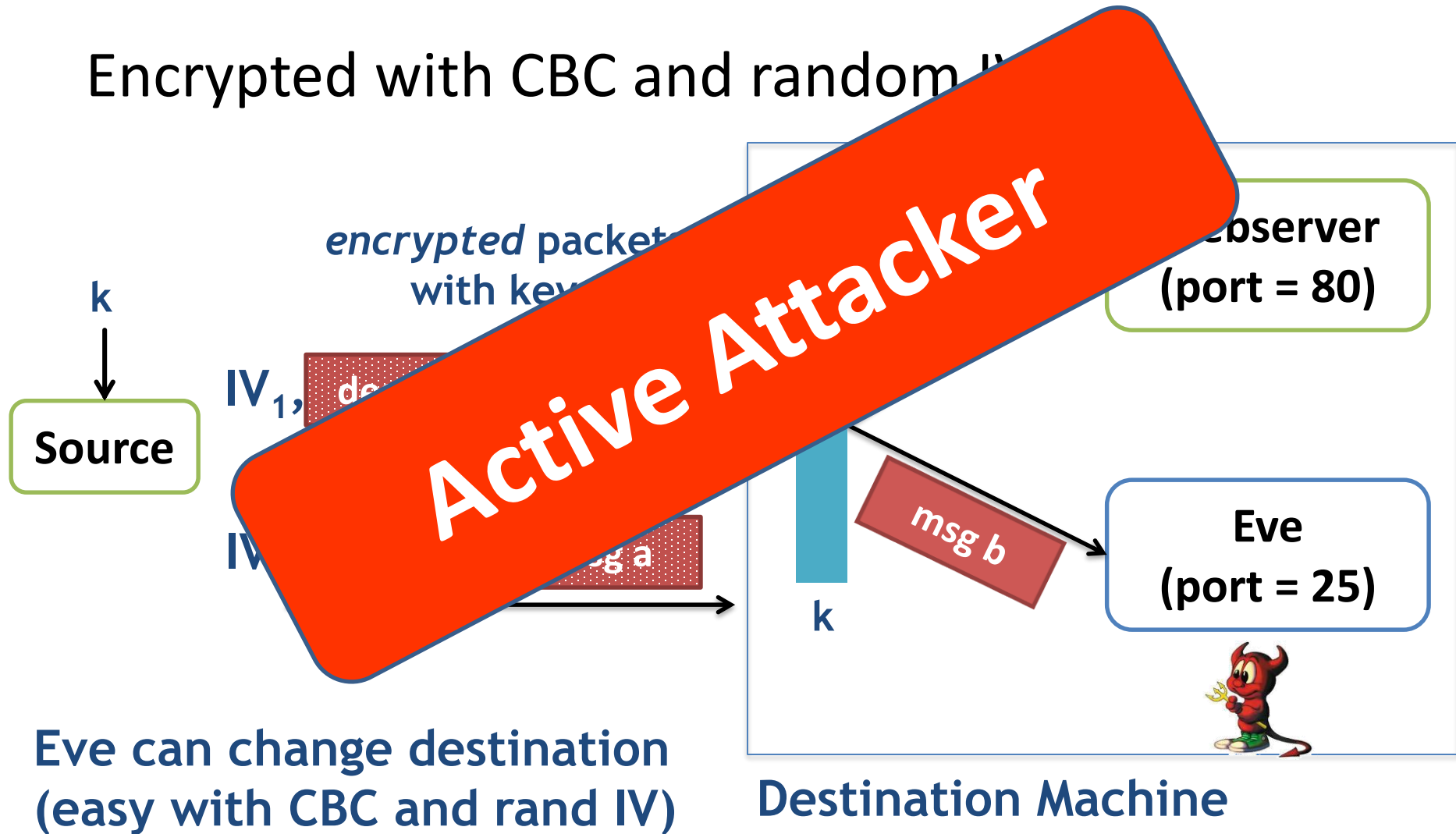
Example tampering attack

Encrypted with CBC and random IV



Example tampering attack

Encrypted with CBC and random IV



How?

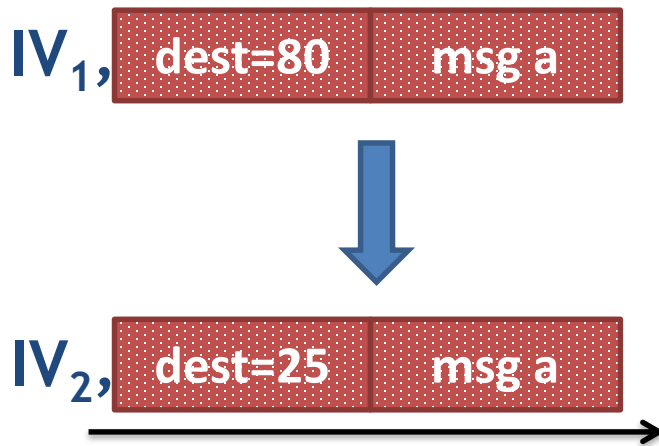
CBC encryption:

$$D(k, c[0]) \oplus IV_1 = \text{"dest=80"}$$

Attack:

$$IV_2 = IV_1 \oplus \underbrace{000\dots80 \oplus 000\dots0025}$$

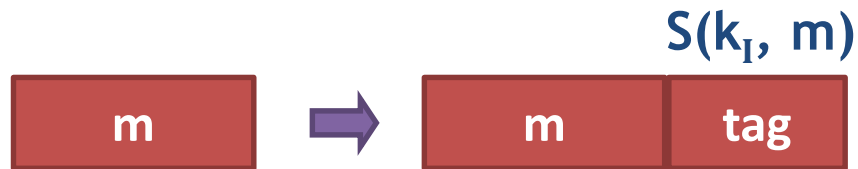
xor out "80" and xor
in "25"



Eve

How can we solve this ...

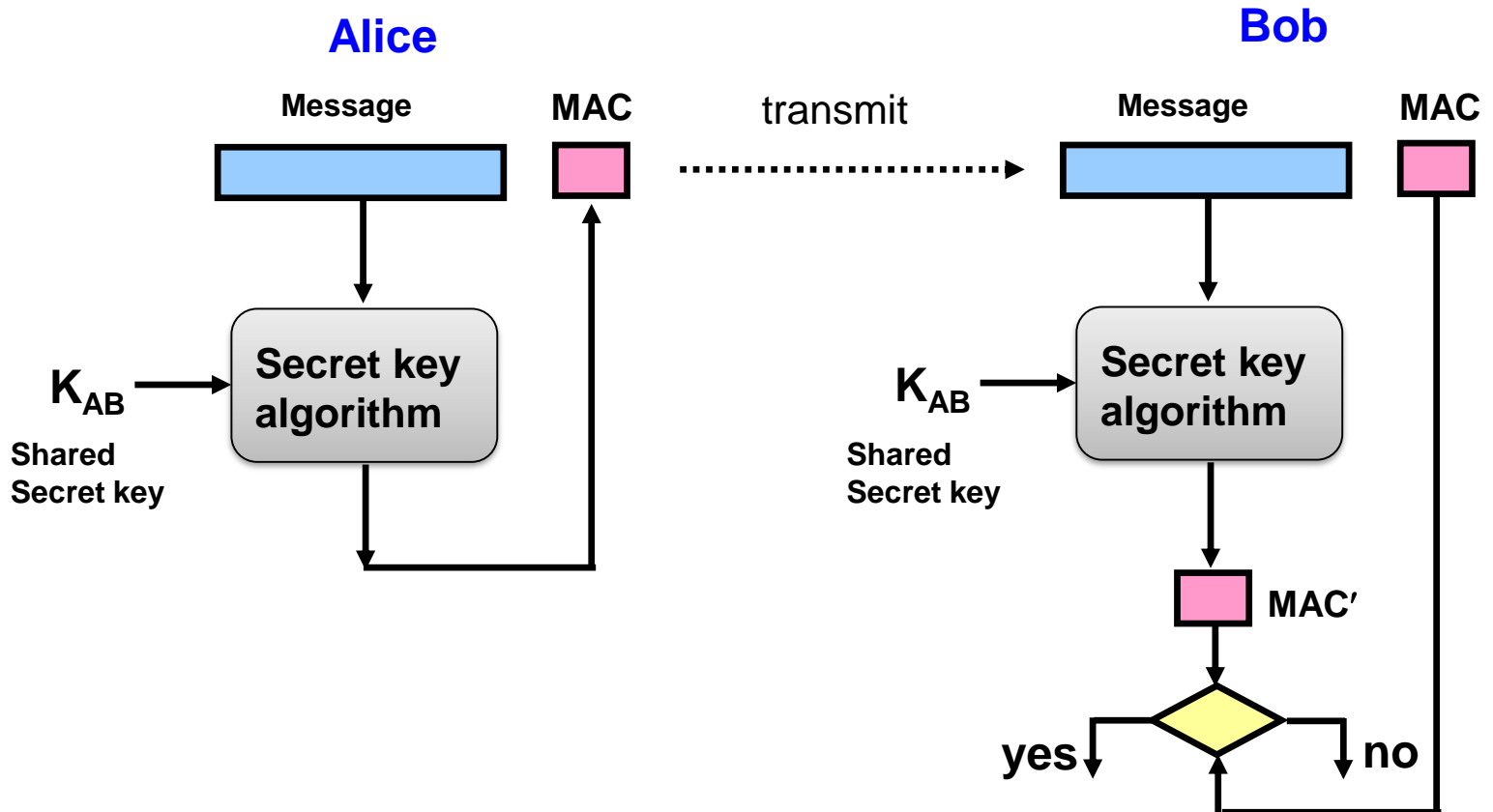
Message authentication is needed.



How can we integrate this with encryption?

Message Authentication Code (MAC)

- A message authentication code (MAC) is a key-dependent message digest function
 - $\text{MAC}(M,k) = h$



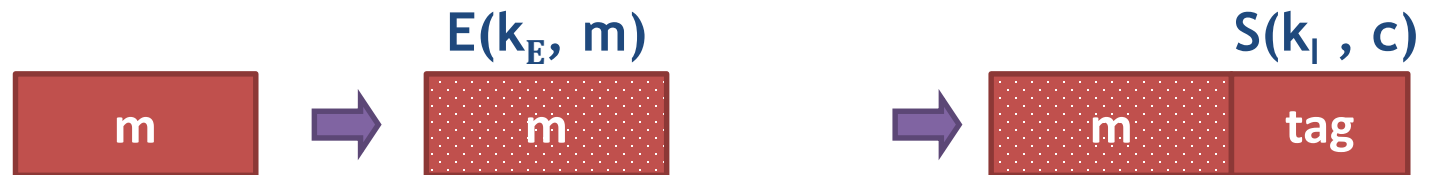
Motivating Question: Which is best?

Encryption Key = k_E ; MAC key = k_I

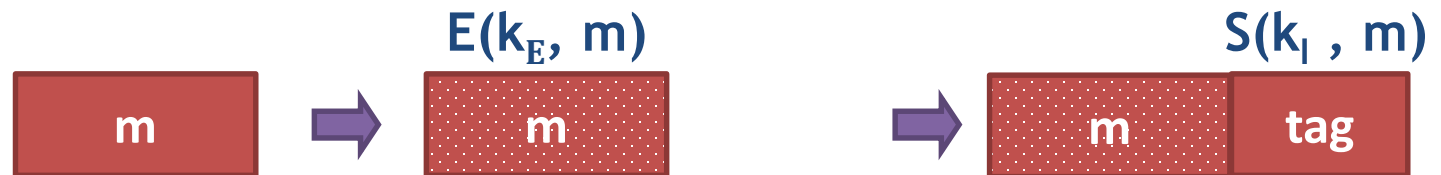
Option 1: SSL (MAC-then-encrypt)



Option 2: IPsec (Encrypt-then-MAC)



Option 3: SSH (Encrypt-and-MAC)



Theorems

Let (E,D) be a CPA secure cipher and (S,V) a MAC secure against existential forgery. Then:

1. Encrypt-then-MAC always provides authenticated encryption
2. MAC-then-encrypt may be insecure against CCA attacks
 - however, when (E,D) is rand-CTR or rand-CBC mode, MAC-then-encrypt provides authenticated encryption

Questions?

