UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Secret Key (symmetric) Cryptography

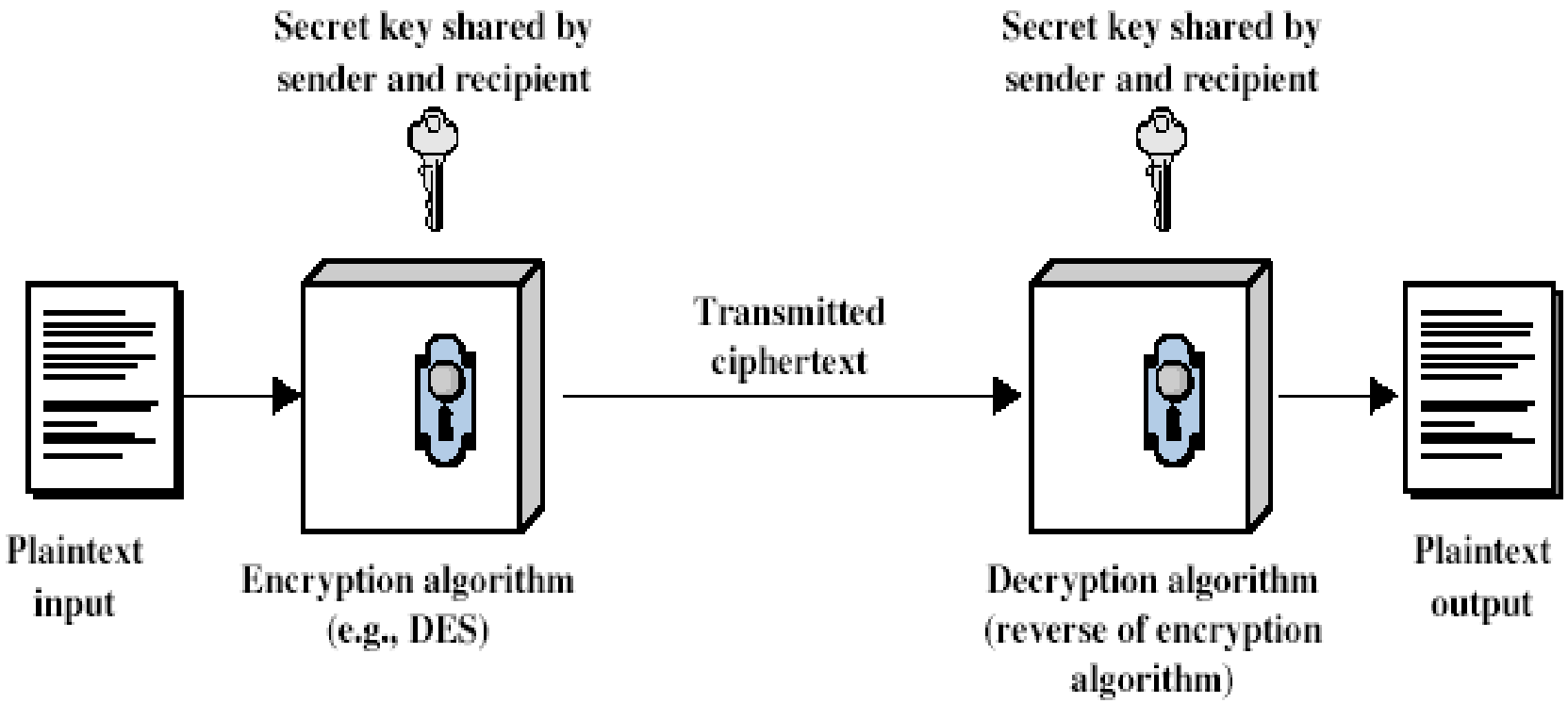
Luca Veltri

(mail.to: luca.veltri@unipr.it)

Symmetric Cryptography

- Or conventional / secret-key / single-key
  - **sender and recipient share a common key**
- All classical encryption algorithms are secret-key
  - **was the only type prior to invention of public-key in 1970's**
- Generally used for protecting (through encryption) some data stored in a repository or sent to a remote entity
- Designed to take a reasonable-length key (e.g. 128 bits) and generating a one-to-one mapping from cleartext to ciphertext that "looks like completely random", to someone doesn't know the key

Symmetric Cipher Model



Symmetric Cipher Model (cont.)

- Plaintext - the original message (m)
- Ciphertext - the encoded message (c)
- Key - info used known only to sender/receiver (k)
- Cipher - algorithm for transforming plaintext to ciphertext

- Two functions:
  - **Encipher (Encryption) - converting plaintext to ciphertext**
    - $c = E(k,m) = E_k(m)$
      - can be either a deterministic or randomized function
  - **Decipher (Decryption) - recovering ciphertext from plaintext**
    - $m = D(k,c) = D_k(c) = D_k(E_k(m))$
      - deterministic

- Common symmetric algorithms:
  - **DES, 3DES, RC4, IDEA, AES**

Symmetric Cipher Model (cont.)

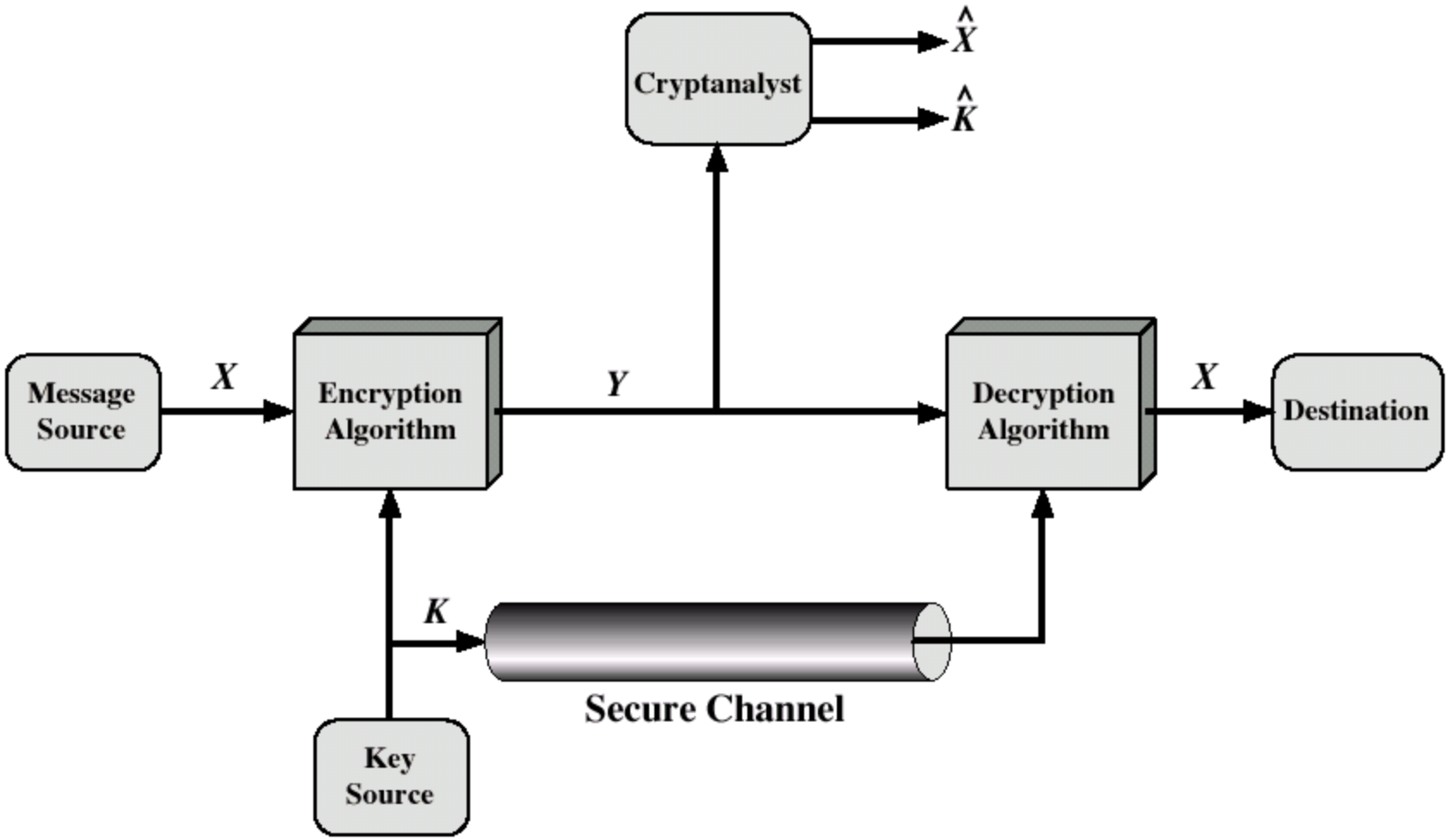
- In theory, the security of a cipher might rest in the secrecy of its restricted algorithm
  - c = E(m), m = D(c)

  however:
  - whenever a user leaves a group, the algorithm must change
  - could be scrutinized by people smarter than you
- In practice, the encryption algorithm is usually not secret
  - **keys are used and the security relies on the secrecy of keys**
    - selected from a large set (a keyspace), e.g., a 256-bit number $\rightarrow 2^{256} \approx 10^{77}$ values!
    - c = E(k,m) = E_k(m), m = D(k,c) = D_k(c)
    - change of authorized participants requires only a change in key
    - the robustness of the algorithm is usually proportional to the key length
      - e.g. 40 bit (weak), 128 bit (strong)
  - **Kerckhoffs' principle: Security should be based on secrecy of the key, not the details of the algorithm**
    - Jean Guillaume Hubert Victor Francois Alexandre Auguste Kerckhoffs von Nieuwenhof, "La Criptographie Militaire", 1883

Symmetric Cipher Model (cont.)

- The two parties must know the algorithm to be used and must share a secret key
  - **requires an initial phase where the two parties exchange in secure manner the shared secret key**
    - implies a secure channel (or method) to distribute the key

Symmetric Cipher Model (cont.)



Threat model

- In general the attacker may try to:
  - **deduce the key used from a specific plain text, to compromise all future and past messages encrypted with that key**
  - **to guess the plain text from the encrypted text**
- Leveraging on:
  - **the knowledge of the encryption algorithm**
  - **some knowledge of the general characteristics of plaintext**
  - **(possibly) some sample pairs of {plaintext, ciphertext}**
- The threat model specifies what “power” the attacker is assumed to have, without placing any restrictions on the adversary’s strategy
- Plausible options for the threat model are:
  - **Ciphertext-only attack**
  - **Known-plaintext attack**
  - **Chosen-plaintext attack**
  - **Chosen-ciphertext attack**

Cryptography Attacks

- There are some general approaches to attacking a conventional encryption scheme:
  - **Brute-force (search) attack**
    - tries every possible encryption/decryption (e.g. by trying all possible keys)
      - it may require the visit of all key space
      - The average number of required attempts is the half of the number of possible keys
    - it requires to be able to recognize when the correct plaintext/ciphertext has been obtained
  - **Cryptographic analysis (cryptoanalysis)**
    - based on the type of the cryptographic algorithm, tries to exploit some characteristic of the algorithm and/or properties of some previous plaintext/ciphertext pairs to deduce a plaintext and/or key
      - does not require to obtain the encryption key for deduce the plaintext
  - **Side-channel attacks**
    - use information from the physical implementation of a cryptosystem

Brute Force Search - Example

Key Size (bits)	Number of Alternative Keys	Time Required at 1 Decryption/μs	Time Required at 10^6 Decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

Cryptoanalysis

- Based on the type of the cryptographic algorithm, tries to exploit some characteristic of the algorithm and/or properties of some previous plaintext/ciphertext pairs
  - **to deduce a plaintext and/or key**
  - **does not require to obtain the encryption key for deduce the plaintext**
- Activity used to:
  - **to test the robustness of the algorithm and of the key by trying possible attacks against it**
  - **to break the code and infer the key from the available ciphertext or decrypt the ciphertext without knowing key**

Side channel attack

- Any attack based on information gained from the physical implementation of a cryptosystem, rather than theoretical weaknesses in the algorithms
  - **e.g. timing information, power consumption**
- General classes of side channel attack include:
  - **Timing attack — attacks based on measuring how much time various computations take to perform**
  - **Power monitoring attack — attacks which make use of varying power consumption by the hardware during computation**
  - **TEMPEST (aka Van Eck or radiation monitoring) attack — attacks based on leaked electromagnetic radiation which can directly provide plaintexts and other information**

Side channel attack (cont.)

- In all cases, that physical effects caused by the operation of a cryptosystem can provide useful extra information about secrets in the system
  - **about the cryptographic key, partial state information, full or partial plaintexts and so forth**
- Side-channel attacks require considerable technical knowledge of the internal operation of the system on which the cryptography is implemented

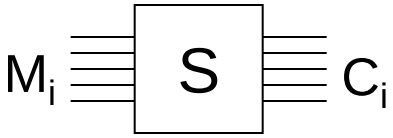
Computational and Unconditional Security

- Unconditional security
  - **an encryption scheme is unconditionally secure if no matter how much computer power is available, the cipher cannot be broken**
    - this property is also called Perfect secrecy
      - for any two messages m1, m2 and any ciphertext c, the probability of obtaining c as the result of the encryption of m1 or m2 is the same
        - » the ciphertext provides insufficient information to determine the corresponding plaintext
    - e.g. the OTP (One Time Pad) chiper
- Computational security
  - **however cryptographic algorithms are often not impossible to attack**
    - e.g. brute force attack
  - **an encryption scheme is computationally secure if given limited computing resources the cipher cannot be broken**
    - e.g. the time required to break the cipher exceeds the useful lifetime of the information
    - depends on the attack complexity and cost
      - processing complexity: a large number of operations required (long time)
      - data complexity: a large number of expected inputs (e.g., ciphertext)
      - storage complexity: a large amount of storage units required
    - requires the estimation of the attacker resources

Classical encryption techniques

Substitution Ciphers

- Substitution is a classical encryption technique
- Symbols (e.g. letters) of plaintext are replaced by other symbols (letters, numbers, same, or other symbols)
- If plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit blocks with ciphertext bit blocks



M	*C*
0000	1101
0001	1001
0010	0111
0011	1000
⋮	⋮
1111	0011

Substitution Table

- Examples of classical substitution ciphers:
  - **monoalphabetic substitution with shift (e.g. Caesar cipher)**
  - **monoalphabetic substitution (monoalphabetic cipher)**
  - **polialphabetic substitution (polialphabetic cipher)**

Caesar Cipher

- Earliest known substitution cipher
  - **by Julius Caesar**
  - **first attested use in military affairs**
  - **it is a monoalphabetic substitution with shift**
- Replaces each letter by 3rd letter on
- Example:

```
meet me after the toga party
PHHW PH DIWHU WKH WRJD SDUWB
```

Caesar Cipher (cont.)

- Can define transformation (substitution) as:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- Mathematically give each letter a number

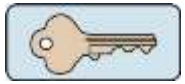
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

- Then have Caesar cipher as:

  $C = E(M) = (M + k) \bmod 26$, with k=3

  $M = D(C) = (C - k) \bmod 26$, with k=3

- If *k* is generic (and secret), we have a *Shift cipher*
  - **k is the key, with $K \in \{0,1,\ldots,25\}$**



- ROT13 is another special case of Shift cipher, with 26-letter alphabet and K=13
  - **in this case, E(X) == D(X), hence only one function ROT13(X)**

Cryptanalysis of a Shift Cipher

- Only have 26 possible ciphers
  - **A maps to A,B,..Z**
  - **If the mapping of one letter is discovered, the entire transformation is found**
- Given ciphertext, could just try all shifts of letters
  - **brute force search**
  - **e.g. break ciphertext "GCUA VQ DTGCM"**
- Do need to recognize when have plaintext

KEY	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	kccr kc ydrcp rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkbdi
20	vnnc vn jocna cqn cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

Monoalphabetic Substitution Ciphers

- Rather than just shifting the alphabet could shuffle (permute) the letters arbitrarily
- Each plaintext letter maps to a different random ciphertext letter
- Example:
  - **Substitution table:**

abcdefghijklmnopqrstuvwxyz
DKVQFIBJWPESCXHTMYAUOLRGZN

  - **`plaintext: ifwewishtoreplaceletters`**
  - **`ciphertext: WIRFRWAJUHYFTSDVFSFUUFYA`**

- Note: the (secret) substitution can be seen as the secret key
- Now have a total of $26! \cong 4 \times 10^{26}$ keys
  - **with so many keys, might think is secure**
    - but would be wrong!
    - cryptanalysis based on text frequency and correlation

Cryptanalysis of Monoalphabetic Cipher

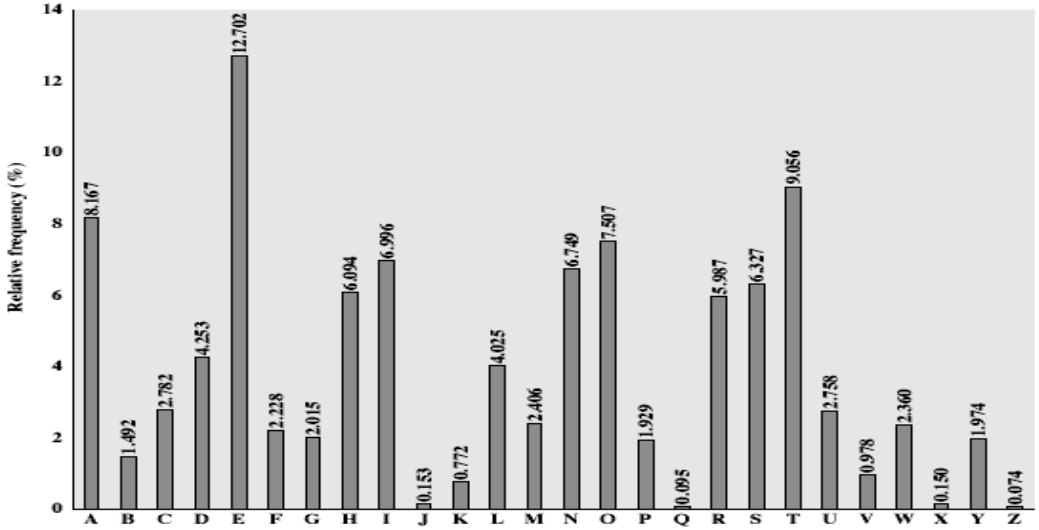
- Main problem with monoalphabetic substitutions is text redundancy
  - **non uniform frequency distributions and correlation**
- In case of human languages:
  - **letter frequencies**
    - in English 'e' is by far the most common letter, then T,R,N,I,O,A,S; other letters are fairly rare (e.g. Z,J,K,Q,X)
  - **two letters frequencies (e.g. “th” in english)**
  - **most common words**
  - **etc.**
- Cryptanalysis:
  - **calculate letter frequencies for ciphertext**
  - **compare counts/plots against known values**
  - **have tables of single, double & triple letter frequencies**
  - **discovered by Arabian scientists in 9th century**

Cryptanalysis Example

- given ciphertext:

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
```

- count relative letter frequencies



- guess *P* & *Z* are *e* and *t*
- guess *ZW* is *th* and hence *ZWP* is *the*
- proceeding with trial and error finally get:

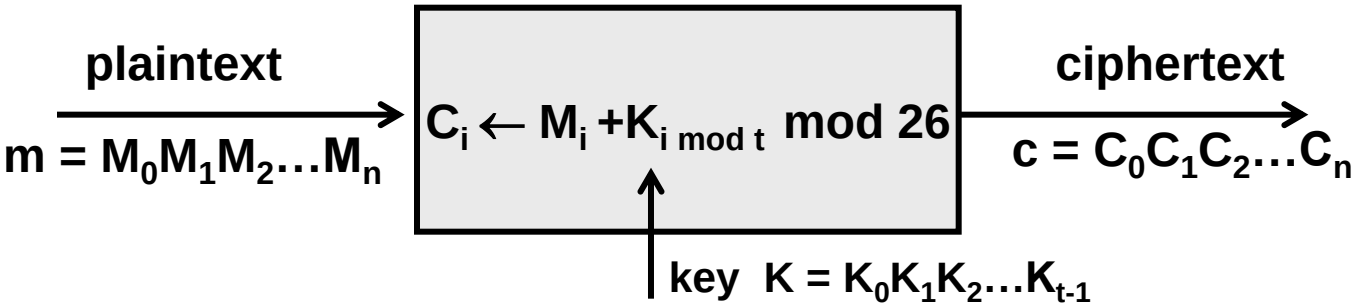
```
it was disclosed yesterday that several informal but
direct contacts have been made with political
representatives of the viet cong in moscow
```

Polyalphabetic Substitution Ciphers

- An approach to improve security is to use multiple cipher alphabets
  - **polyalphabetic substitution ciphers**
  - **uses a set monoalphabetic substitutions**
  - **defines a rule to determine which cipher alphabet (substitution) should be used at each step**
    - normally uses a key to select which substitution is used for each letter of the message
      - repeat from start after end of key is reached

Vigenère Cipher

- **Simplest polyalphabetic substitution cipher is the Vigenère Cipher (Blaise de Vigenère, 1523-1596)**



- Key is multiple letters long
  - i^{th} letter specifies i^{th} alphabet to use
  - use each alphabet in turn
- Repeat from start after *t* letters
- Effectively multiple Caesar ciphers
  - $K = K_0\ K_1\ \ldots\ K_{t-1}$
- Decryption works in reverse

	0	1	2	3	4	5	6	7	8	9	.	.	.	25
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
+ 1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
+ 2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
+ 3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
+ 4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
+ 5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
+ 6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
+ 7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
+ 8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
+ 9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
+10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
+11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
+12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
+13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
+14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
+15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
+16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
+17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
+18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
+19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
+20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
+21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
+22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
+23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
+24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
+25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Vigenère Cipher (cont.)

(00)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
(01)	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
(02)	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
(03)	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
(04)	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
(05)	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
(06)	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
(07)	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
(08)	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
(09)	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
(10)	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
(11)	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
(12)	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
(13)	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
(14)	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
(15)	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
(16)	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
(17)	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
(18)	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
(19)	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
(20)	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
(21)	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
(22)	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
(23)	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
(24)	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
(25)	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

- Example:

  key: REBUS (17,4,1,21,18)

  plaintext: **codic emolt osicu ro**

  key: **REBUS REBUS REBUS RE**

  ciphertext: **TSECU VQPFL FWJWM IS**

Cryptanalysis of Vigenère Ciphers

- Polyalphabetic substitution ciphers make cryptanalysis harder
  - **have multiple ciphertext letters for each plaintext letter**
    - hence letter frequencies are obscured
      - more cipher alphabets to guess and flatter frequency distribution
- But not totally lost
  - **one weakness is the repeating nature of the key**
    - need to determine number of alphabets (key length)
    - start with letter frequencies
    - then can attack each
      - they are like shift ciphers and can be broken individually

One-Time Pad

- One-Time Pad (OTP) cipher
  - **patented by Gilbert Vernam, Bell Telephone Laboratories, in 1919**
  - **first described by Frank Miller (a California banker) in 1882**
- Can be seen as a special case of Vigenère cipher
  - **the key k={K_0,K_1,K_2,..,K_n} is as long as the plaintext *m***
  - **a random key *k* is used for each message**
- Ciphertext contains no statistical relationship to the plaintext
  - **for any plaintext and any ciphertext there exists a key mapping one to other**
- In case of alphabet {0,1}, two possible substitutions:
    - {0,1}→{0,1}, with k_i=0
    - {0,1}→{1,0}, with k_i=1
  - **hence, for each bit: $c_i = m_i$ XOR k_i**
  - **c = m XOR k**

One-Time Pad (cont.)

- Example in binary:

```
m= 01001000011001010110110001101100011011110010110000100000001110100 ...
k= 00010101100101010011010111010110001011101101100101001001011000001 ...
c= 01011101111100000101100110111010010000011111010101101001000101010 ...
```

- Same Example in hexadecimal:

```
m= 48656c6c6f2c2074686973206973206e206578616d706c65
k= 159535d62ed94961c45b5019d2717f0ab74c614549e3c1911d
c= 5df059ba41f56915ac322339bb025f6bd96c043d288eb1fd78
```

One-Time Pad (cont.)

- Claude Shannon (1945) introduced the definition of perfect secrecy and demonstrated that the one-time pad achieves that level of security
  - **the cipher will be unconditionally secure (unbreakable)**
    - no statistical relationship between distinct ciphertexts
    - for any plaintext of equal length to the ciphertext, there is a key that produces that plaintext
- Disadvantages:
  - **can only use the key once**
    - requires a key stream as long as the sum of all messages that has to be encrypted
      - possible problems on distributing and store this long key

Many-Time Pad (MTP)

- If the same key *k* is used to encrypt with OTP two or more different messages *m1, m2*, … some exploitations are possible
  - **for example, given**
    - c1= m1 xor k
    - c2= m2 xor k
  - **it is**
    - c1 xor c2 = m1 xor m2

- It is possible to exploit statistical information from m_i *xor* m_j values

Transposition Ciphers

- Transposition is another classical encryption technique
  - **hides the message by rearranging the letter order (blocks of bits)**
  - **without altering the actual letters used**
  - **performs a sort of permutation**
- Can recognize these since have the same frequency distribution as the original text
- Example
  - **permutation**

Example: Row Transposition Ciphers

- Write letters of message out in rows over a specified number of columns
- then reorder the columns according to some key before reading off the rows

```
Key:        4 3 1 2 5 6 7
Plaintext:  a t t a c k p
            o s t p o n e
            d u n t i l t
            w o a m x y z
Ciphertext: TTNAAPTMTSUOAODWCOIXKNLYPETZ
```

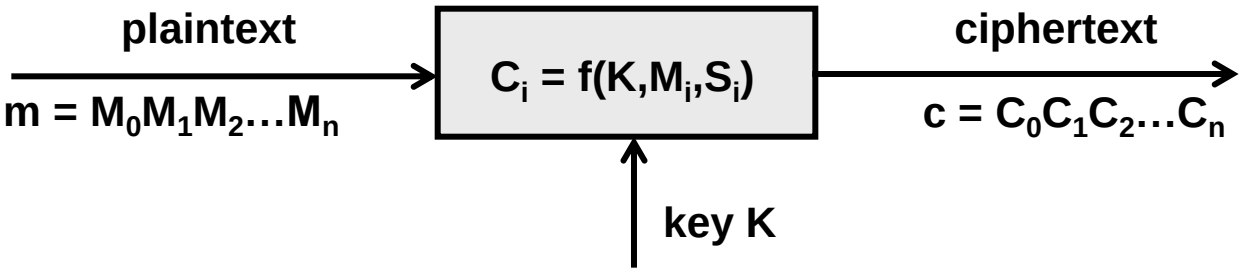
Product Ciphers

- Ciphers using substitutions or transpositions may be not sufficiently secure
- Hence consider using several ciphers in succession to make harder:
  - **two substitutions make a more complex substitution**
  - **two transpositions make a more complex transposition**
  - **a substitution followed by a transposition make a new much harder cipher**
- This is bridge from classical to modern ciphers

Stream and Block Ciphers

Stream ciphers

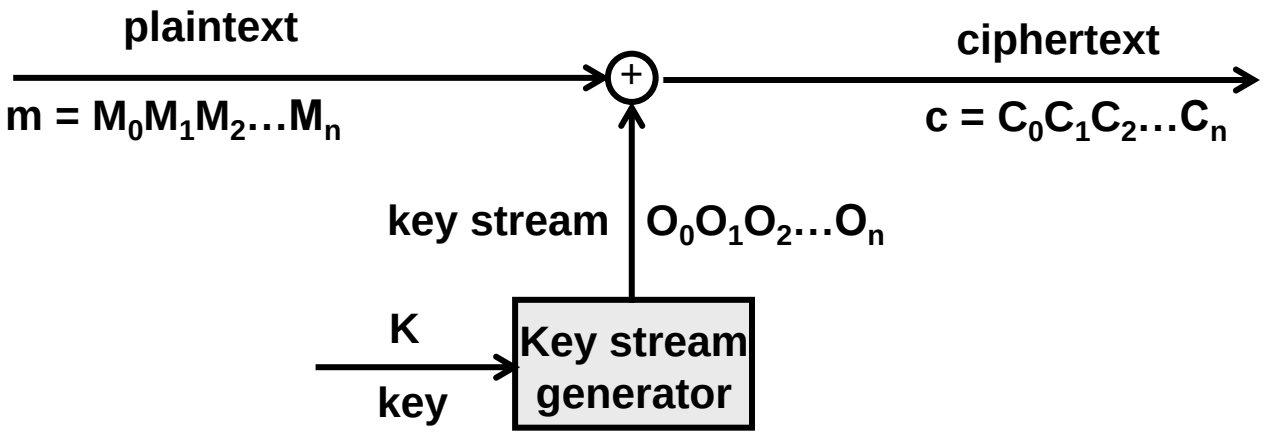
- There are two basic cipher structures
  - **Stream ciphers**
  - **Block ciphers**
- Stream ciphers process messages (Encryption/Decryption) a bit or byte at a time when en/decrypting



- The output unit C_i may be function of the current input M_i, an internal state S_i, and the secret key K
  - **the internal state S_i may be a function of previous M_j and/or C_j, with $j<i$**

Stream ciphers (cont.)

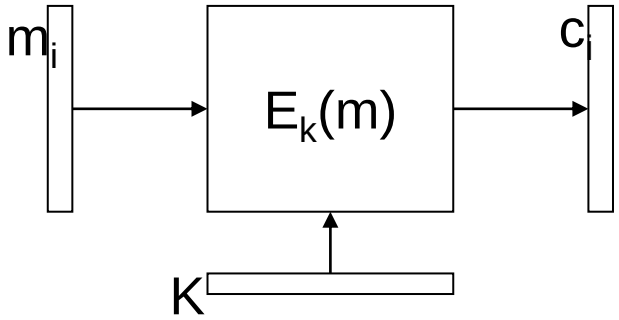
- A very common scheme for stream ciphers (autokey ciphers) is:



- Want a key as long as the message (OTP)
- Example, ChaCha20 stream cipher
- Most stream ciphers are based on pseudorandom number generators (PRNG)
  - **the key is used to initialize the generator, and either key bytes or plaintext bytes are fed back into the generator to produce the byte stream**

Block ciphers

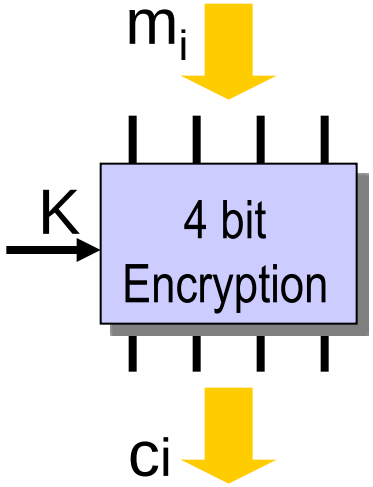
- Block ciphers process messages into blocks, each of which is then en/decrypted
  - **plaintext and ciphertext are treated as a sequence of n-bit blocks of data**
  - **ciphertext is same length as plaintext**
  - **ciphertext depends on plaintext and a key value**



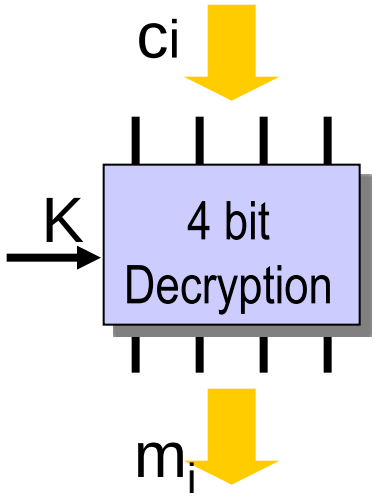
- Larger messages must be processed into blocks, each of which is then en/decrypted
  - **the resulted cipher can be made to behave as a stream cipher**
  - **e.g. CBC, OFB, CFB, and CTR modes**
- Many current ciphers are block ciphers (DES, IDEA, AES, etc.)

Block ciphers (cont.)

- Same input blocks encrypted with the same key are transformed into the same output block
- Example: block size = 4 bits



Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111



Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

Block ciphers (cont.)

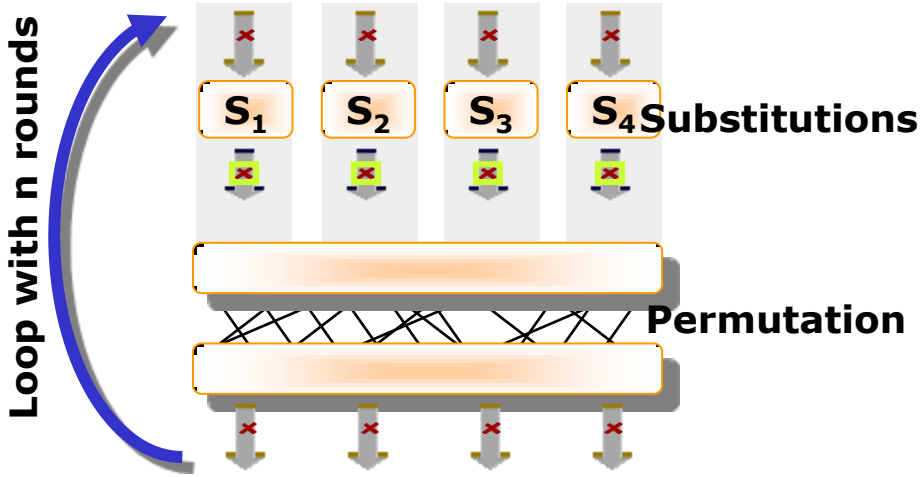
- Like a substitution on (very big) set of possible inputs
  - **If the block size is *n* bits, 2^n possible input values are mapped to 2^n output values**
    - like a *n*-bit substitution
  - **A way for encrypting could be to specify completely the mapping table (substitution table)**
    - permutation of 2^n *n*-bit inputs
    - there are $2^n!$ different possible transformations
  - **If n=64, would need table of 2^{64} entries storing 64-bit blocks**
    - 2^{64} 2^6 bit = 2^{70} bit = 2^{67}B = 2^{37}TB ≈ 10^{11}TB
    - it is too long
- Instead, a block cipher is created from smaller building blocks and a secret key
  - **if *n* is the cipher size and *k* is the key length, there are a total of 2^k possible transformations, rather than $2^n!$**

Block cipher (cont.)

- How long should the plaintext block be?
  - **having block size too small**
    - in case of known-plaintext attack, an opponent may try to collect $\{M_i, C_i\}$ pairs and construct a decryption table
      - n-bit block cipher requires 2^n pairs
      - It is not required to find the key
    - In case of ciphertext–only attack, if a sequence of M_i have some properties (recognizable sequences of plaintext), it is possible to cryptanalyze the sequence of C_i
      - e.g. exploitation of language redundancy
  - **having block size too long, it could be inconvenient due to the increasing of complexity**
- 64-bit or 128-bit blocks are often used
  - **it is difficult to obtain all 2^{64} pairs (known-plaintext attack)**

Substitution–permutation network

- SP-network is a product cipher that uses only substitutions and permutations
- One possible way to build a block cipher based on SP-network is
  - **break the input into managed-sized chunks (say 8 bits),**
  - **do a substitution on each small chunk,**
  - **and then take the output of all the substitutions and run them through a permuter (big as the input)**
  - **the process is repeated, so that each bit winds up as input to each substitution**
  - **each time is called *round***



DES (Data Encryption Standard)

- Block cipher based on concept of invertible product cipher
  - **NIST (National Institute of Standards and Technology) standard**
    - FIPS PUB 46-3 (Federal Information Processing Standards PUBlication), 1977
  - **64bit block cipher with 56bit key**
    - it is a "Feistel cipher" (use Feistel network)
      - partitions input block into two halves
  - **Widely used in the past, now replaced by AES**
- Based on an algorithm known as *Lucifer cipher* (1971)
  - **by an IBM team led by Horst Feistel**
    - used 64-bit data blocks with 128-bit key
- Efficiently implemented in hardware, relatively slow if implemented in software

DES (in)security

- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- Originally complaints that the NSA fixed the internal S-boxes to provide a backdoor; this has never been found
- There are some theoretical attacks that break DES; however they are unfeasible to mount in practice
- **It is now possible to attack by brute force**
  - **in 1997 on Internet in a few months**
  - **in 1998 on dedicated HW ($250K) in a few days (Electronic Frontier Foundation)**
  - **in 1999 above (EFF) combined in 22hrs!**
  - **In 2006 with FPGA based parallel machine of the Universities of Bochum and Kiel (Germany) at $10,000 HW cost, in to 6.4 days**
  - **reasonable for a small business to buy**
- Still must be able to recognize plaintext, in case of ciphertext-only attacks

DES weak keys

- We call weak keys, cipher keys that make a cipher behave in some undesirable way
  - **usually represent a very small fraction of the overall keyspace**
- With DES there are some known key values such that

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

  - **e.g.:**
    - 0x0101010101010101
    - 0xFEFEFEFEFEFEFEFE

- There are also some pair of keys, called semi-weak keys, such that

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

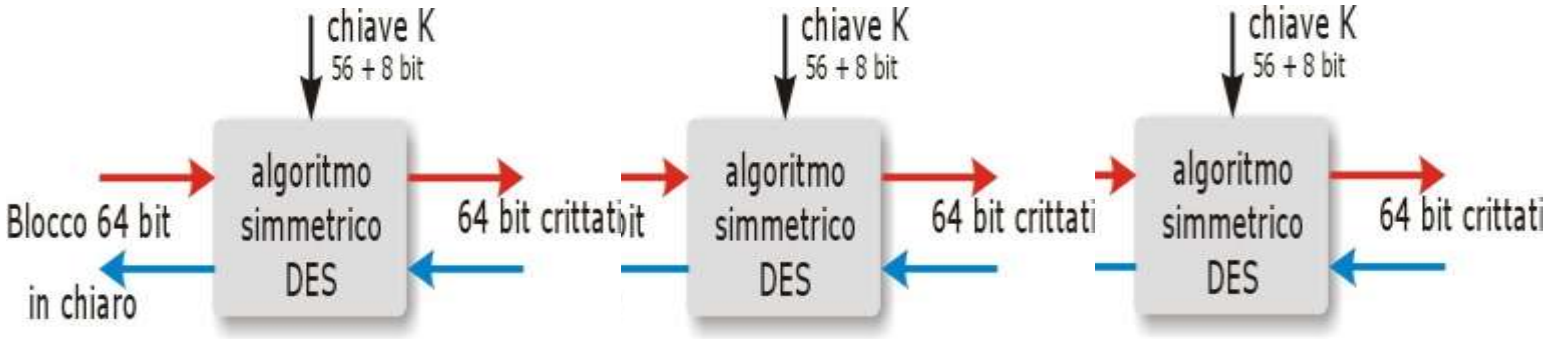
Multiple Encryption

- A possible solution for increasing the (computational) security of an encryption algorithm is to use the same algorithm more times with different keys
  - **$c=E_{Kn}(\ldots(E_{K2}(E_{K1}(m)))$**
  - **$m=D_{K1}(D_{K2}(\ldots (D_{Kn}(c)))$**
  - **K=K1||K2||K3||…||Kn, with length = n |K1|**
- How many time should be performed? (2,3,4, ..)
  - **the more time the block is encrypted the more secure it is**
    - however for computation, no more encryptions than are necessary
- How many keys?
  - **the used keys, together, can be seen as the overall secret key**
- In most cases, both Encryption and Decryption algorithms can be see as encryption functions
  - **what combination of E and D can be chosen? (EEE, ED, etc)**

Multiple Encryption (cont.)

- Encrypting twice with the same key
  - **$c=E_K(E_K(m))$**
  - **no more secure that single encryption with K with length *n*: exhaustive search requires trying 2^n keys**
- Encrypting twice with two keys
  - **$c=E_{K2}(E_{K1}(m))$**
  - **there is an attack (not very practical), known as meet-in-the-middle, that breaks double encryption (EE) in roughly twice the time of a brute-force breaking single E**
    - since $X = E_{K1}(M) = D_{K2}(C)$
    - attack by encrypting *M* with all keys and store
    - then decrypt *C* with keys and match *X* value
    - can show takes $O(2^{n+1})$ encryptions and $O(2^n)$ memory
- Triple encryption with two/three keys (e.g. EEE, or EDE)
  - **meet-in-the-middle attack is not possible**

Triple DES (3-DES)



- **$c = E_{K3}(D_{K2}(E_{K1}(m)))$**
- **3-DES is also referred to as EDE (Encrypt Decrypt Encrypt) or TDEA**
- **Options:**
  - **3-DES with three different 56-bit keys: (k_1, k_2, k_3)**
    - key length: 56 + 56 + 56 = 168 bit (TDEA option 1)
  - **3-DES with two different 56-bit keys: (k_1, k_2, k_1)= (k_1, k_2)**
    - key length: 56+56 = 112 bit (TDEA option 2)
  - **3-DES with one 56-bit key: (k_1, k_1, k_1)= k_1**
    - equivalent to DES: $c = E_{K1}(D_{K1}(E_{K1}(m))) = E_{K1}(m)$

Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES)

- Block cipher designed to replace DES
  - **organized by National Institute of Standards and Technology (NIST)**
  - **NIST standard on November 26, 2001**
    - FIPS PUB 197 (FIPS 197)
  - **chosen from five candidate algorithms**
    - reviewed by US government (NSA), industry and academia
    - required a four-year process to pick the algorithm
    - winning algorithm chosen Oct 2, 2000
  - **also known as Rijndael block cipher**
    - original name of the algorithm submitted to AES selection process
    - developed by Joan Daemen and Vincent Rijmen (Belgium)
  - **currently, one of the most popular algorithms used in symmetric key cryptography**
    - also adopted as an encryption standard by the U.S. government

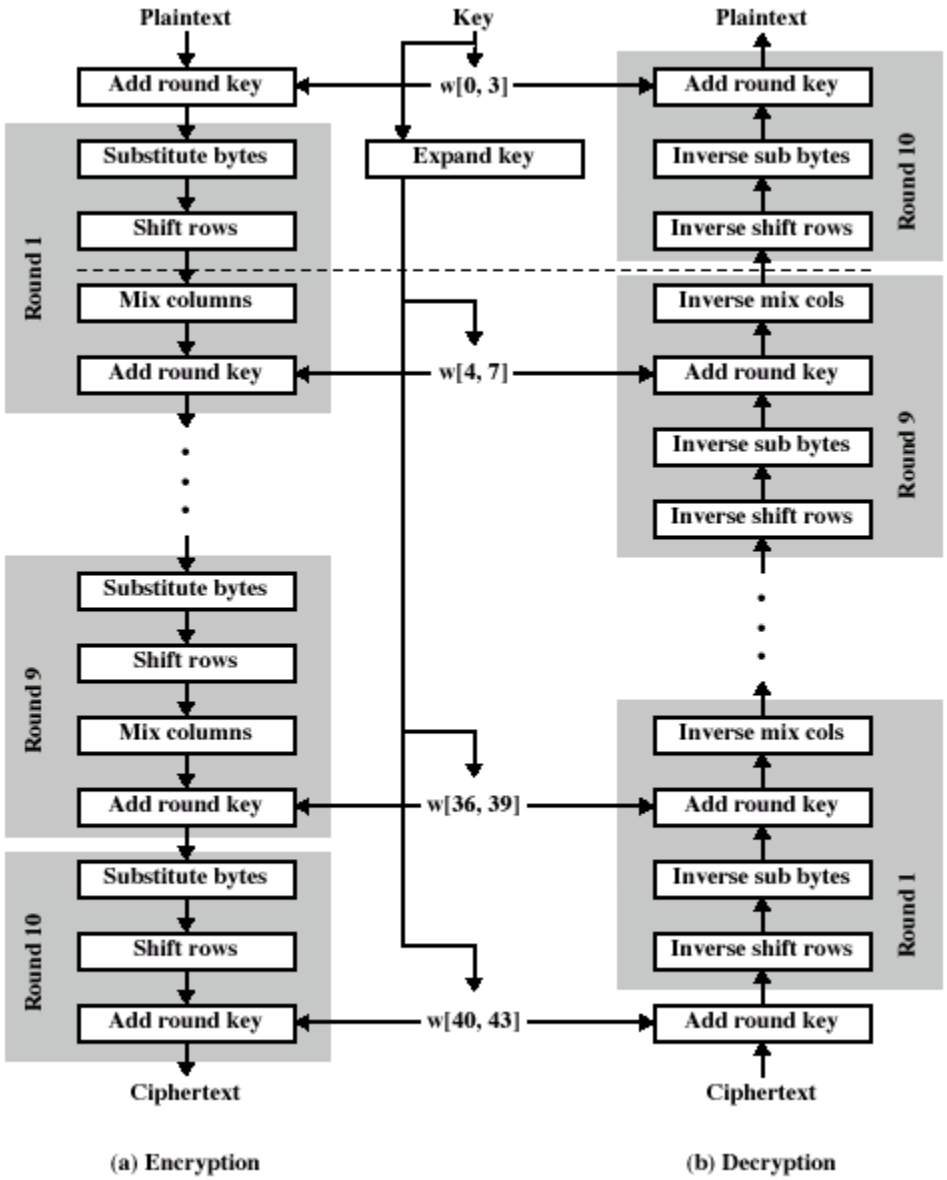
Advanced Encryption Standard (AES) (cont.)

- AES is not precisely the original Rijndael
  - **the original Rijndael algorithm supports a larger range of block and key sizes**
    - Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits
- AES has fixed block size of 128 bits and a key size of 128, 192, or 256 bits
  - **i.e. block size: 16 bytes, key size: 16, 24, or 32 bytes**
- Rijndael is a substitution-permutation network
  - **not a Feistel network like DES**
- Fast in both software and hardware
  - **relatively easy to implement**
  - **requires little memory**

AES Description

- Due to the fixed block size of 128 bits, AES operates on a 4×4 array of bytes, termed the state
  - **versions of Rijndael with a larger block size had additional columns in the state**
- AES has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys
- Most AES calculations are done in a special finite field

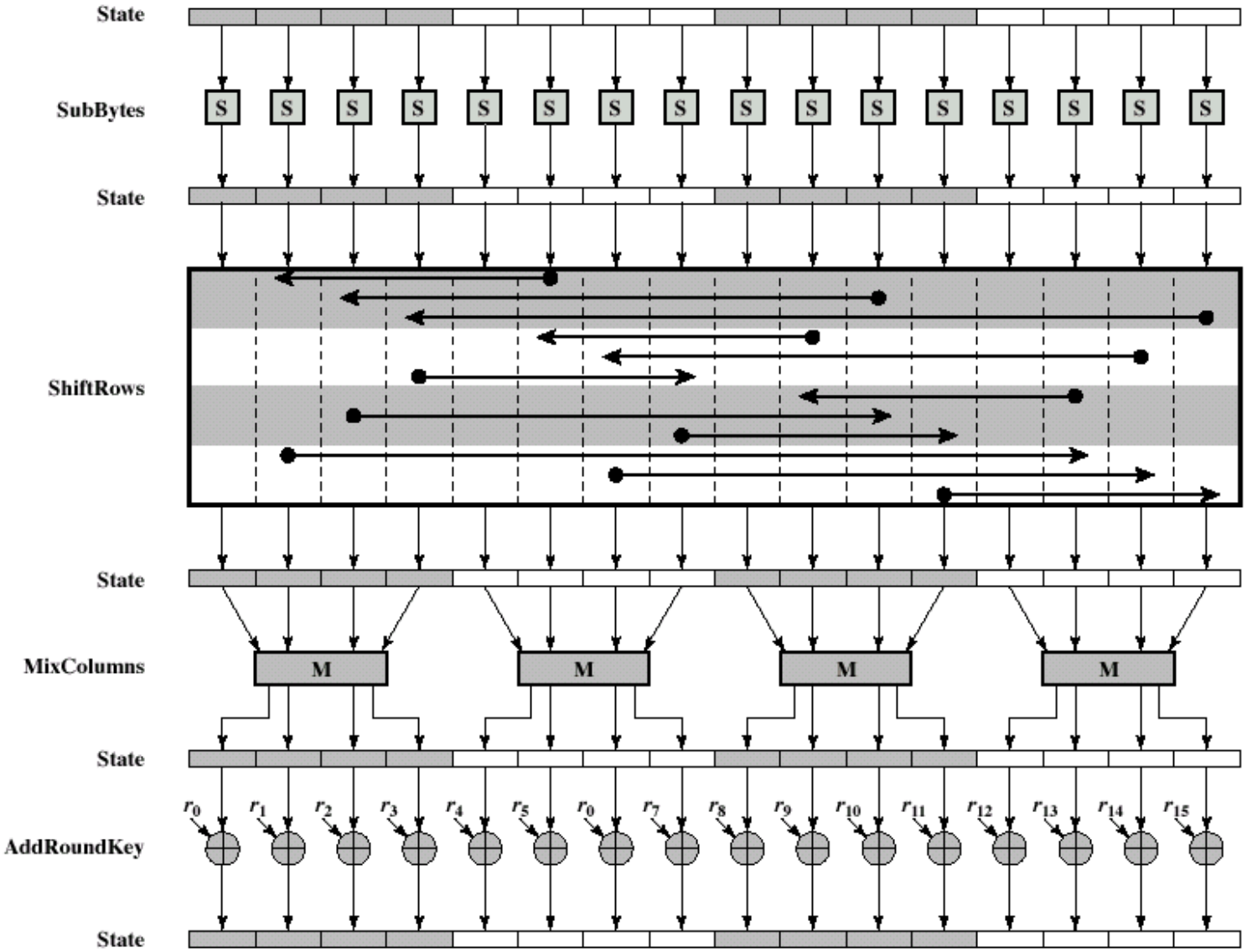
AES cipher


Plaintext
Add round key
Substitute bytes
Shift rows
Mix columns
Add round key
Round 1
Substitute bytes
Shift rows
Mix columns
Add round key
Round 9
Substitute bytes
Shift rows
Add round key
Round 10
Ciphertext
(a) Encryption
Key
w[0, 3]
Expand key
w[4, 7]
w[36, 39]
w[40, 43]
Plaintext
Add round key
Inverse sub bytes
Inverse shift rows
Round 10
Inverse mix cols
Add round key
Inverse sub bytes
Inverse shift rows
Round 9
Inverse mix cols
Add round key
Inverse sub bytes
Inverse shift rows
Round 1
Add round key
Ciphertext
(b) Decryption

AES cipher

- Algorithm
  - **KeyExpansion using Rijndael's key schedule**
  - **Initial Round**
    - AddRoundKey
  - **(*N*-1) Rounds (9 for 128bit key, 11 for 224bit key, 13 for 256bit key)**
    1. SubBytes — a non-linear substitution step where each byte is replaced with another according to a lookup table
    2. ShiftRows — a transposition step where each row of the state is shifted cyclically a certain number of steps
    3. MixColumns — a mixing operation which operates on the columns of the state, combining the four bytes in each column
    4. AddRoundKey — each byte of the state is combined with the round key derived from the cipher key using a key schedule
  - **Final Round (no MixColumns)**
    1. SubBytes
    2. ShiftRows
    3. AddRoundKey

AES single round



Building a stream cipher using a block cipher (Block cipher modes)

Encrypting large messages

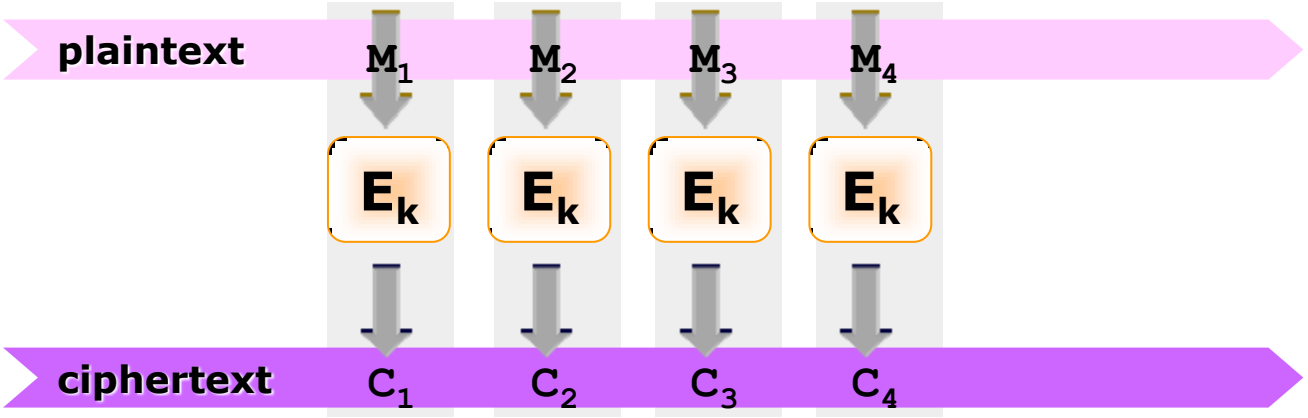
- Block ciphers encrypt fixed size blocks
  - **eg. AES has fixed block size of 128 bits**
- Usually we have arbitrary amount of information to encrypt (longer than the block size)
  - **need way to use in practice**
- Use of a block cipher for creating a stream cipher
  - **five modes have been defined for TDEA in: *NIST Special Publication 800-38A* (2001)**
    - Electronic Code Book (ECB)
    - Cipher Block Chaining (CBC)
    - (k-bit) Output Feedback (OFB)
    - (k-bit) Cipher Feedback (CFB)
    - Counter Mode (CTR)
  - **these schemes are applicable to any block cipher**
  - **other modes are also possible**

Padding

- If the encryption mode requires that the length of the whole message has to be a multiple of a given block size, a Padding operation has to be performed first
- Examples of padding algorithms:
  - **Bit padding (e.g. ISO/IEC 7816-4)**
    - symbol '1' (bit) is added, and then as many '0' bits as required are added
  - **ANSI X.923**
    - the last byte defines the number of padding bytes; the remaining bytes are filled with zeros
      - eg. [b1 b2 b3 00 00 00 00 05] (3 data bytes, then 5 bytes pad+count)
  - **RFC 5652 Cryptographic Message Syntax / PKCS#7 / PKCS#5**
    - the value of each added byte is the number of bytes that are added
      - eg. [b1 b2 b3 05 05 05 05 05]

Electronic Codebook (ECB) Mode

- Consist of doing the obvious thing, and it is usually the worst method
- The message is broken into n blocks of *b* with padding for the last one
  - **b = block size of the cipher**
  - **$n_* b$ = message size including padding**
- Each block is independently encrypted with the secret key K
  - **$C_i = E_K (M_i)$**
  - **$M_i = D_K (C_i)$**



- Each block is a value which is substituted, like a codebook
- The cipher text has the same size of the plaintext (excluding padding)

Advantages and Limitations of ECB

- ECB is very simple and does not introduce extra operation (except for padding)
- There are a number of problems that arise and that don't show up in the single block case
  - **repetitions in message may show in ciphertext if aligned with message block**
    - if a message contains 2 identical blocks, the corresponding cipher blocks are identical; it can be a problem
      - in some cases it can be possible to guess a portion of the message
  - **by comparing two ciphertexts it is possible to discover similarities in the plaintexts**
    - no avalanche effect
  - **(partially) knowing the plaintext, is possible to rearrange the ciphertext blocks in order to obtain a new (known) plaintext**
- As result, ECB is rarely used
  - **main use is sending a few blocks of data (e.g. a secret key)**

Advantages and Limitations of ECB (cont.)

- Example of how repetitions may affect the security of ECB:

Image before ECB Encryption

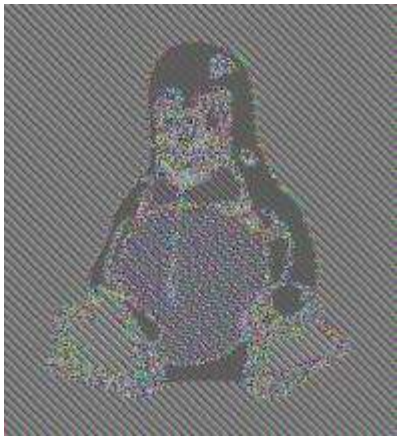


Image after ECB Encryption

from https://commons.wikimedia.org

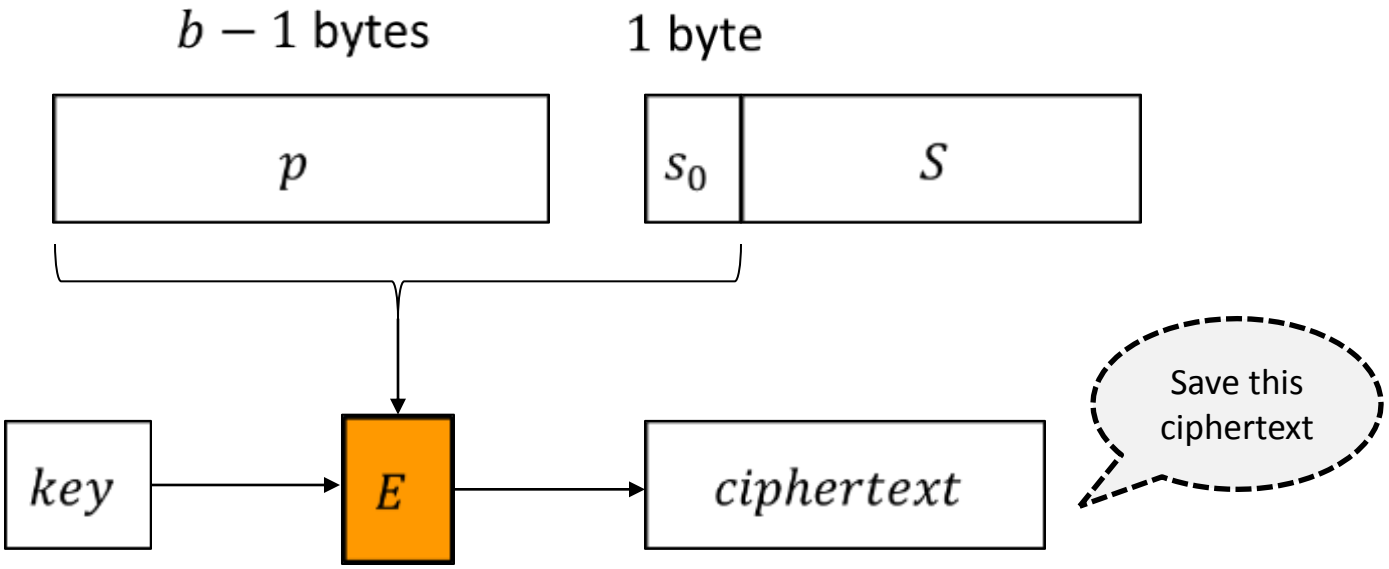
Stream Ciphers – Encryption Oracle

- We call an encryption oracle a service that, given a plaintext message m, returns the corresponding ciphertext c using always the same key

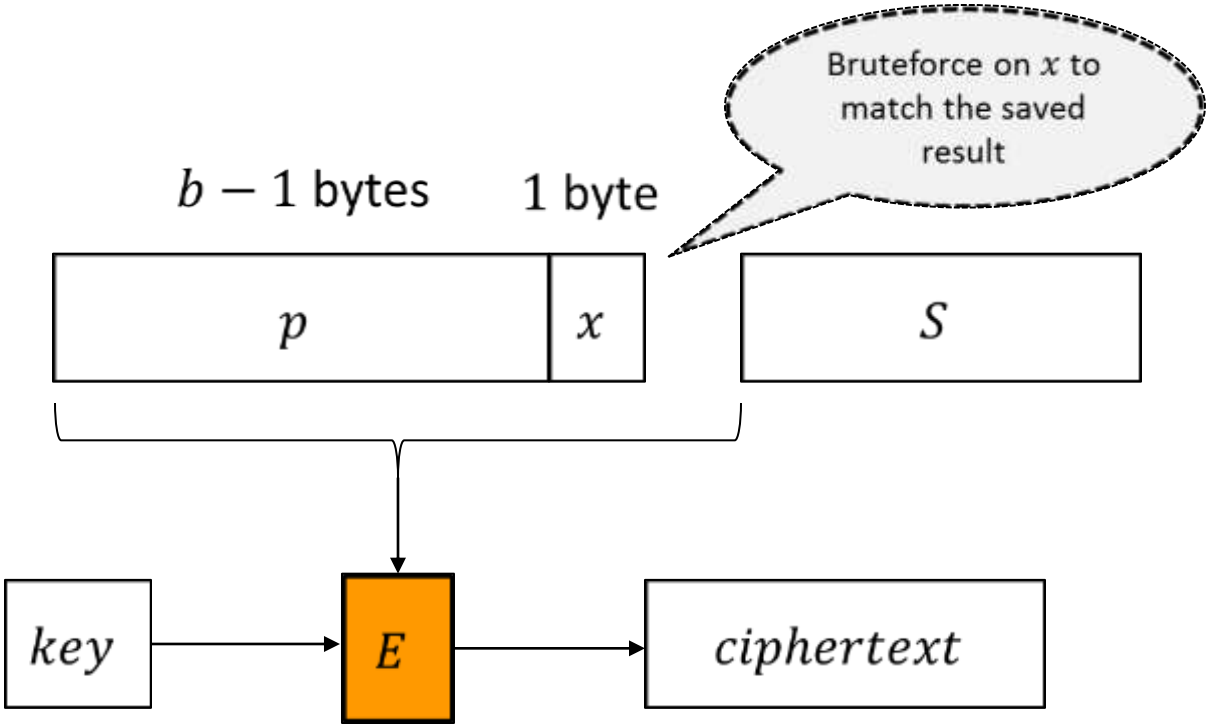
ECB Oracle Attack

- We show that, if misimplemented, ECB can be completely broken
- Scenario: an oracle that returns $c = ECB(key, m||s)$, where:
  - **m is a chosen plaintext**
  - **s is a secret string**
- In this scenario we can recover s
- Strategy:
  - **We send a message that is 1 byte shorter than the block size and we save the result**
  - **We bruteforce the last byte until we find the same ciphertext**
  - **We proceed like this, bruteforcing one byte at a time**

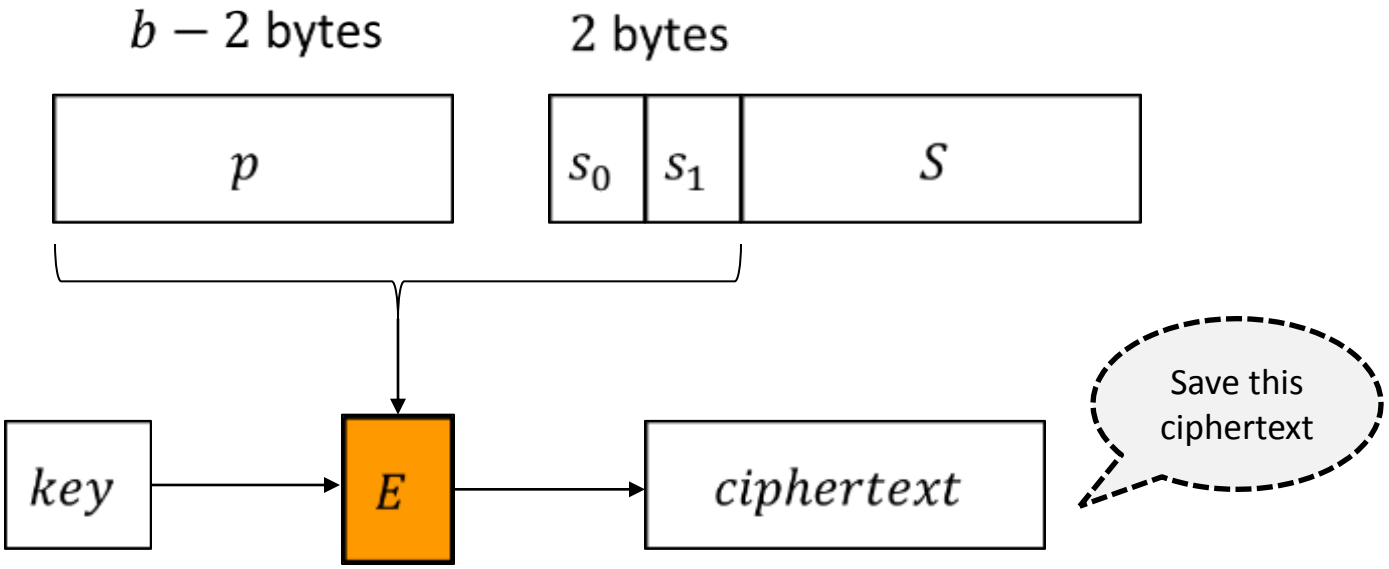
ECB Oracle Attack – step 1



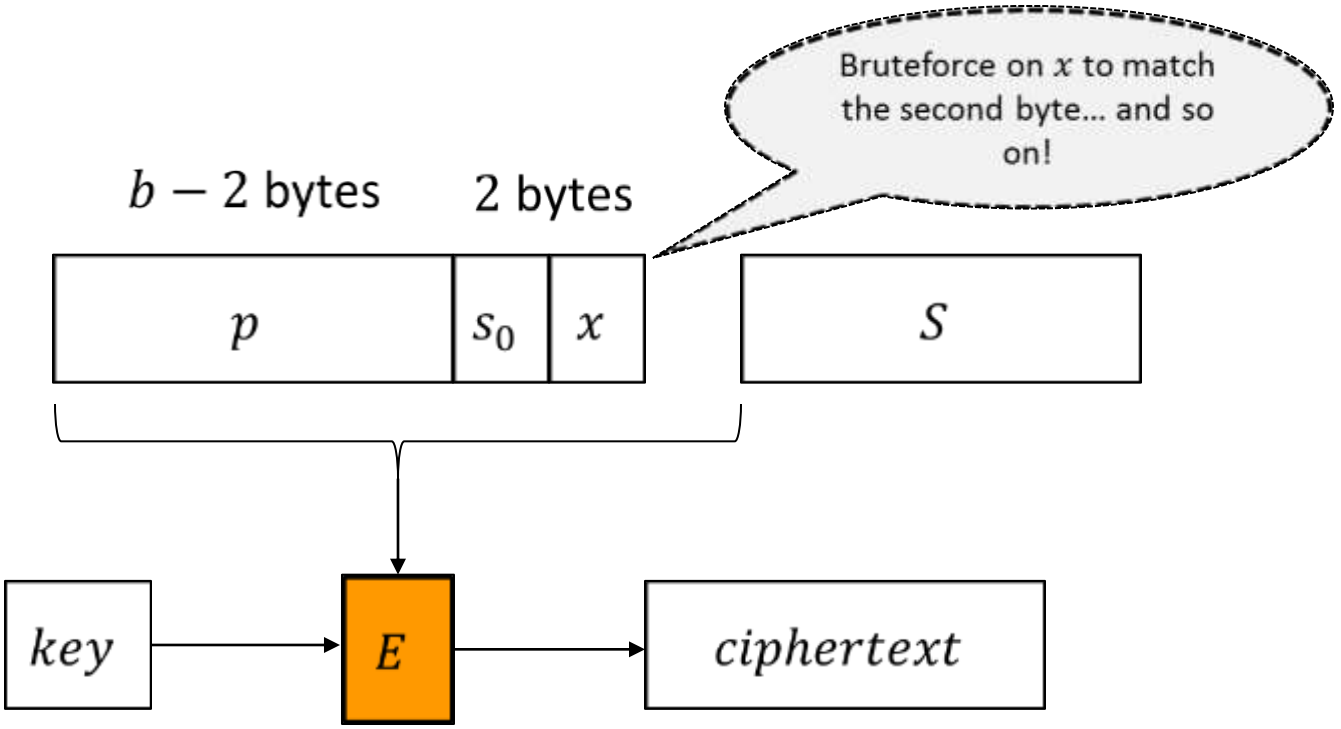
ECB Oracle Attack – step 2



ECB Oracle Attack – step 3



ECB Oracle Attack – step 4

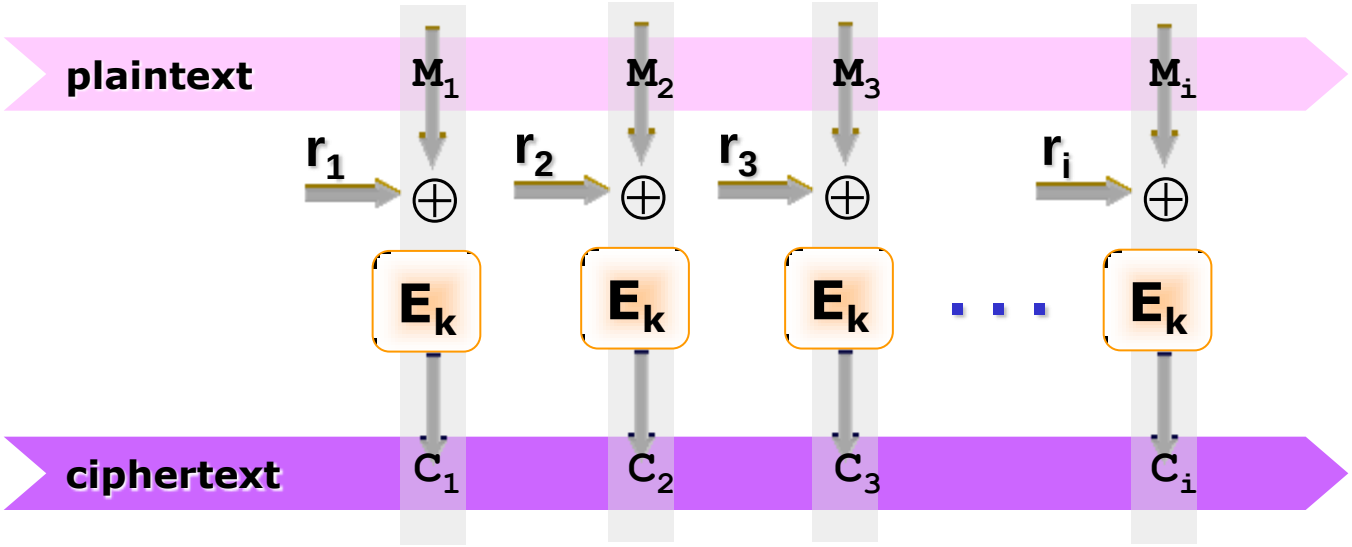


ECB Oracle Attack – Performance

- With AES-128 we have that:
  - **Bruteforcing the key takes $2^{128} = 256^{16}$ tries**
  - **ECB Oracle takes only $256 * 16$ tries!**

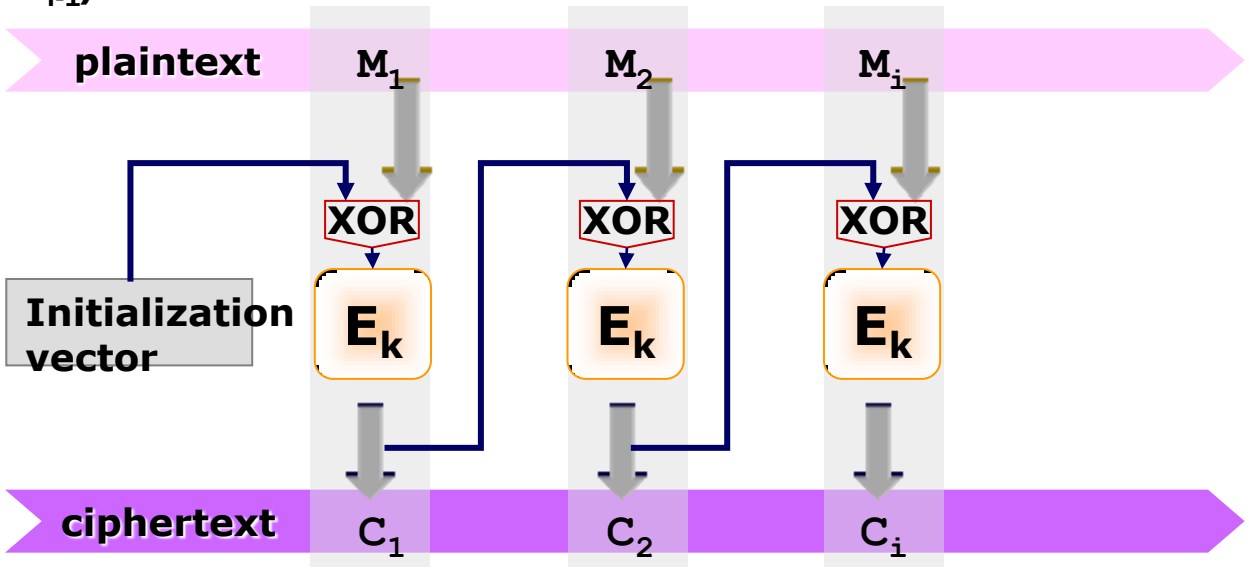
Cipher Block Chaining (CBC) Mode

- CBC objective: if the same block repeats in the plain text, it will not cause repeats of ciphertext
  - **avoiding some problems in ECB**
- How: adds a feedback mechanism to the cipher
- Result: plaintext is more difficult to manipulate
- Basic idea:



Cipher Block Chaining (CBC) Mode

- Plaintext patterns are concealed by XORing a block of *m* with the previous block of *c*
- Requires an IV (Initialization vector) of random data to encrypt the first block
  - $C_i = E_k(M_i \text{ XOR } C_{i-1})$
  - $C_0 = IV$



- Decryption is simple because ⊕ is reversible (A=B⊕C ⇔ A⊕C=B):
  - $M_i = D_k(C_i) \text{ XOR } C_{i-1}$

Advantages and Limitations of CBC

- CBC has the same performance of ECB, except for the cost of generating and transmitting the IV (and the cost of one $\oplus$)
- Each ciphertext block depends on **all** previous message blocks
  - **thus a change in the message affects all ciphertext blocks after the change as well as the original block**
    - forward avalanche effect
  - **a change in IV changes all bits of the ciphertext**

- It can be used a constant value as IV (e.g. 0), however it can lead to some problems
  - **if a message is transmitted periodically, it is possible to guess if changes occurred**
  - **if two ciphertexts have the same initial blocks it means that the corresponding plaintext blocks are equal**

Advantages and Limitations of CBC (cont.)

- The IV is not required to be a secret, and should not contain a secret
  - **it could be obtained by a chosen-ciphertext attack by asking to decrypt two same blocks C1=C2, obtaining M1 and M2:**

    M1 xor M2 xor C1 = (D(C1) xor IV) xor (D(C2) xor C1) xor C1 =

    = D(C1) xor (D(C2) xor IV = D(C1) xor (D(C1) xor IV = IV

CBC Threats: Modifing ciphertext blocks

- Using CBC does not eliminate the problem of someone modifying the message in transit
- If IV is sent in the clear, an attacker can change bits of the first plaintext block (working on the ciphertext and IV) by simply changing the IV
  - **changing bit *h* of IV has predictable effect to bit *h* of M_1**
  - **hence either IV must be a fixed value or it must be sent encrypted in ECB mode before rest of message**
    - e.g. by changing IV $\rightarrow$ IV' $\equiv$ IV$\oplus$X, when decrypting we have:
      $M_1' = D_K(C_1) \oplus IV' = (M_1 \oplus IV) \oplus IV' = M_1 \oplus IV \oplus IV \oplus X = M_1 \oplus X$
- If the attacker changes the ciphertext block C_i, C_i gets $\oplus$'d with the decrypted C_{i+1} to yield M_{i+1}
  - **since M_{i+1} is obtained as: $M_{i+1} = D_K(C_{i+1}) \oplus C_i$**
  - **changing bit *h* of C_i has predictable effect to bit *h* of decrypted M_{i+1}**
  - **however the attacker cannot know the new decrypted M_i' (a new random block, as side effect)**
    - e.g. by changing $C_i \rightarrow C_i' \equiv C_i \oplus X$, when decrypting we have:
      $M_i' = D_K(C_i') \oplus C_{i-1} = D_K(C_i \oplus X) \oplus C_{i-1}$ = unpredictable without knowing *K*
      $M_{i+1}' = D_K(C_{i+1}) \oplus C_i' = (M_{i+1} \oplus C_i) \oplus C_i' = M_i \oplus X$

CBC Threats: Rearranging ciphertext blocks

- Knowing the plain text, the corresponding ciphertext and IV, it is possible to rearrange the C_1, C_2, C_3, .. (building blocks), in such a way to obtain a new known M'_1, M'_2, M'_3 ..
  - **e.g. suppose an intruder rearranges the plaintext $c=C_1,C_2,C_3,C_4$:**
    - if the modified ciphertext $c'=C_1,C_3,C_3,C_4$, then when decrypting:
      $M_1' = D_K(C_1') \oplus IV' = D_K(C_1) \oplus IV = M_1$
      $M_2' = D_K(C_2') \oplus C_1' = D_K(C_3) \oplus C_1 = (M_3 \oplus C_2) \oplus C_1$
      $M_3' = D_K(C_3') \oplus C_2' = D_K(C_3) \oplus C_3 = (M_3 \oplus C_2) \oplus C_3$
      $M_4' = D_K(C_4') \oplus C_3' = D_K(C_4) \oplus C_3 = M_4$
    - if the modified ciphertext $c'=C_1,C_3,C_2,C_4$, then when decrypting:
      $M_1' = D_K(C_1') \oplus IV' = D_K(C_1) \oplus IV = M_1$
      $M_2' = D_K(C_2') \oplus C_1' = D_K(C_3) \oplus C_1 = (M_3 \oplus C_2) \oplus C_1$
      $M_3' = D_K(C_3') \oplus C_2' = D_K(C_2) \oplus C_3 = (M_2 \oplus C_1) \oplus C_3$
      $M_4' = D_K(C_4') \oplus C_3' = D_K(C_4) \oplus C_2 = (M_4 \oplus C_3) \oplus C_2$
- These threads can be combated by adding a MIC (message integrity check) code, or a strong checksum to the plaintext before encrypt
  - **use of 32 bit checksum doesn't completely solve the problem, since 1 in 2^{32} chance that the checksum will work**
    - after 2^{31} attempts, the probably of obtaining a correct checksum is ~50%

Padding Oracle

- In some cases an attacker can modify a ciphertext and checking whether an unpadding error occurs when decrypting the ciphertext
  - **in general it is always possible to decrypt a tampered ciphertext, however an error may occur when unpadding**
    - due to malformed padding value
- We call Padding Oracle a server receives a ciphertext and tells whether the padding is correct

CBC Padding Oracle Attack

- Outline of the attack (for 1 block ciphertext C):
  - **Create a random block R**
  - **Append the target block obtaining $R||C$**
  - **Discover the padding length using the oracle**
  - **Decrypt one byte at a time exploiting it**

CBC Padding Oracle Attack

- Step 1: look for a "correct padding" message
  - **Try to decrypt $R||C$**
  - **With high probability, you will get "wrong padding"**
  - **Keep changing the last byte of R in order to get "correct padding" (this is the same as bitflipping!)**
  - **Now you know that the decryption of $R||C$ ends in $0x01$ or $0x02\ 0x02$ or $0x03\ 0x03\ 0x03$ or ...**

CBC Padding Oracle Attack

- Step 2: find the length of the padding
  - **Let R now be the block that gives "correct padding"**
  - **Change randomly the first byte of R: if it still gives correct padding, the padding length is $b-1$ or less**
  - **Change randomly the second byte of R: if it still gives correct padding, the padding length is $b-2$ or less, and so on**
  - **If you reach an "incorrect padding" on the k^{th} byte, you found the padding length!**

CBC Padding Oracle Attack

- Step 3: decrypt the padding bytes
  - **Now we discovered (at least) one byte of the plaintext**
  - **In reality, we discovered n bytes, where n is the padding length**
  - **In order to get them, just XOR the corresponding bytes of R with the padding bytes**

CBC Padding Oracle Attack

- Step 4: decrypt subsequent bytes
  - **To get one more byte, we need to "increase the padding"**
  - **To do it, XOR the padding bytes with $n \oplus (n+1)$ (this just increase them by 1)**
  - **Repeat from step 1 using the first non-padding byte instead of the last one!**

Output Feedback (OFB) Mode

- Acts like a pseudorandom number generator
  - **more precisely as a stream cipher based on XOR**
- The message is encrypted by ⊕ing it with the pseudorandom stream generated by the OFB
  - **message is treated as a stream of bits**
- How it works:
  - **A pseudorandom number O_0 is generated (named IV as in CBC)**
  - **O_0 is encrypted (using secret key K) obtaining O_1**
  - **from O_1 is obtained O_2 and so on, as many block are needed**

    $O_i = E_K(O_{i-1})$

    $O_0 = IV$
  - **the pseudorandom stream (like a one-time pad) is independent of message and can be computed in advance**
  - **the one-time pad is simply ⊕'d with the message**

    $C_i = M_i$ XOR O_i
- Example of uses: stream encryption over noisy channels

Output Feedback (OFB) Mode

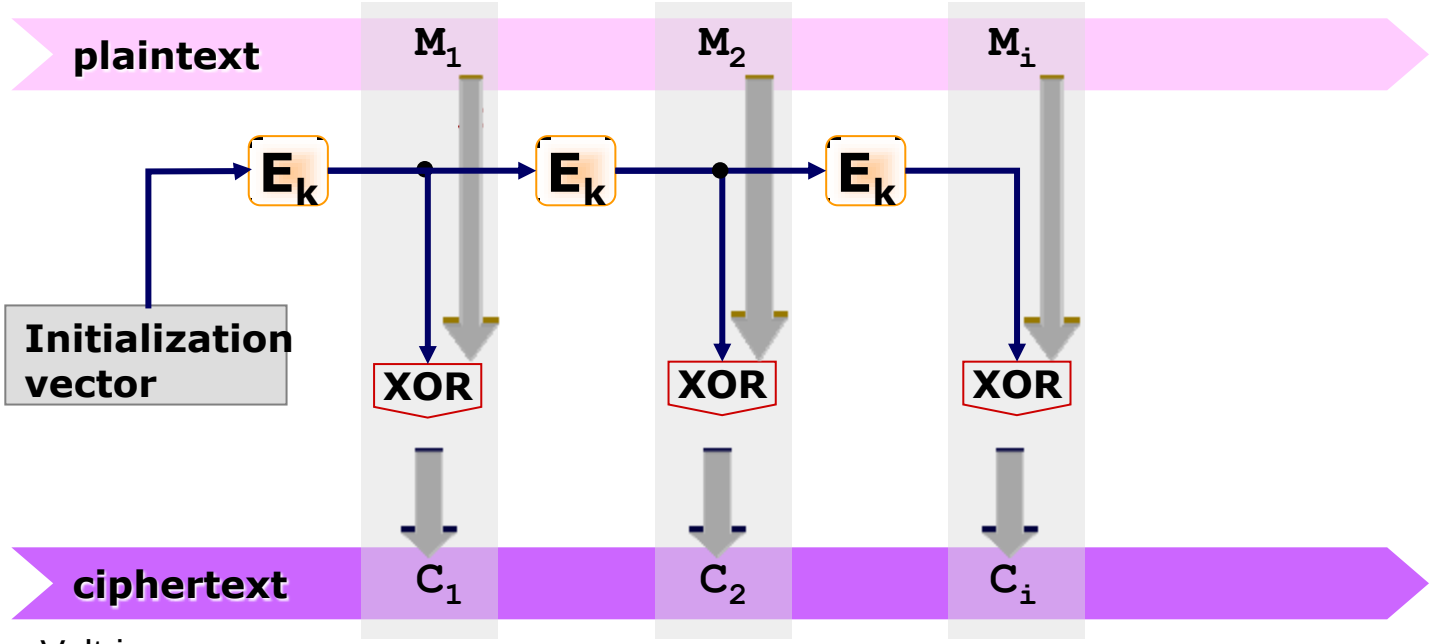
- OFB in short:
  - **a long pseudorandom string is generated (one-time pad)**

    $O_i = E_K(O_{i-1})$

    $O_0 = IV$
  - **the one-time pad is ⊕'d with the message**

    $C_i = M_i$ XOR O_i

    $M_i = C_i$ XOR O_i



Advantages and Limitations of OFB

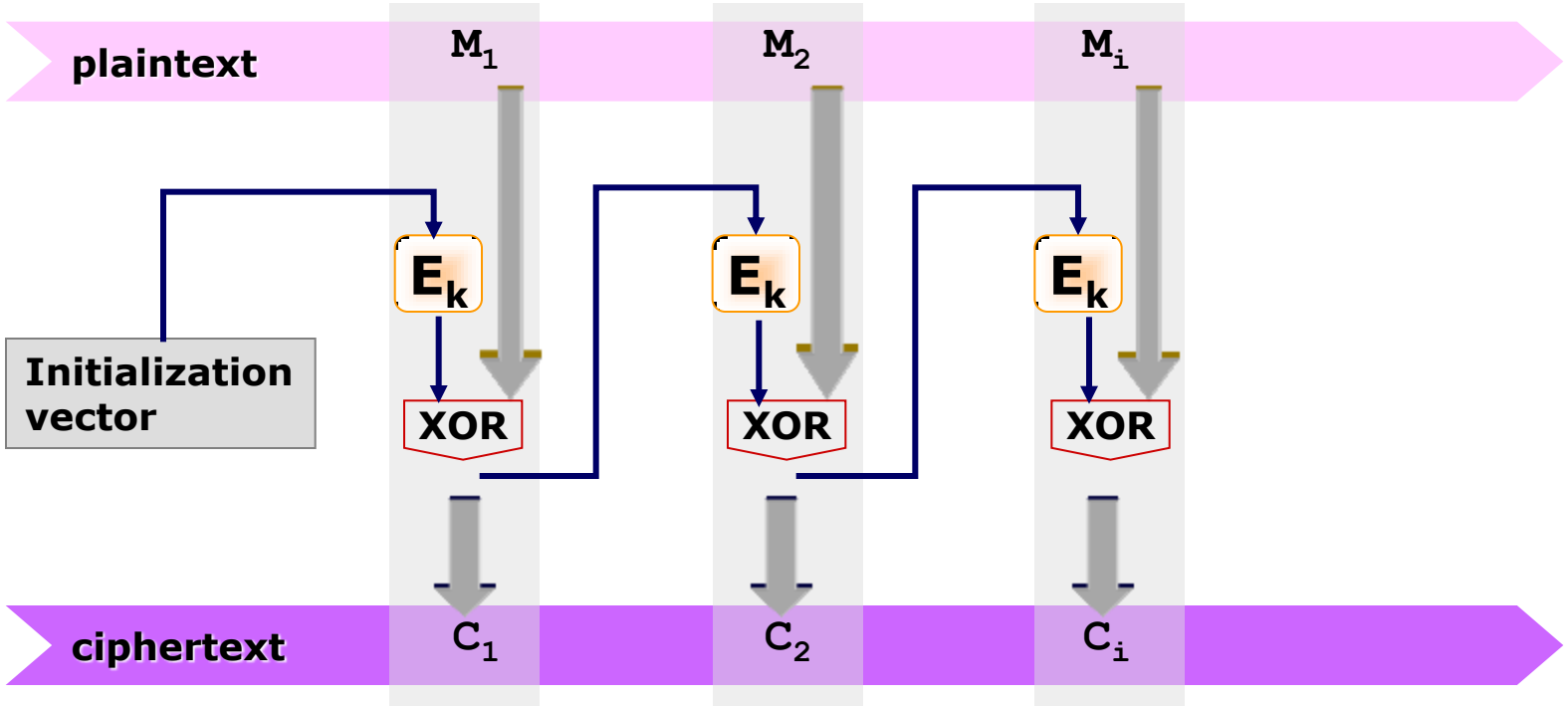
- Advantages of OFB:
  - **one-time pad can be generated in advances**
  - **if a message arrives in arbitrary-sized chucks, the associated ciphertext can immediately be transmitted**
  - **possibility to encrypt variable-length messages: no need of padding**
  - **if some bits of the ciphertext get garbled, only those bits of plaintext get garbled (no error propagation)**
- Disadvantages of OFB:
  - **if the plaintext is known by a bad guy, he can modify the ciphertext forcing the plaintext into anything he wants**
  - **hence must never reuse the same sequence (key+IV)**
  - **sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs**

Cipher Feedback (CFB) Mode

- Similar to OFB
- The *b* bits shifted in to the encryption module are the *b* bits of the ciphertext from the previous block

$C_0 = IV$

$C_i = M_i \text{ XOR } E_K(C_{i-1})$



Advantages and Limitations of CFB

- The block cipher is used in encryption mode at both ends
  - **like in OFB**
  - **however:**
    - the one-time pad cannot be generated entirely in advance
      - needs to stall while do block encryption after every *b* bits
- Errors in cipher text propagate to the next block
  - **like in CBC**
- The lost of a portion of the ciphertext can be resumed if it is multiple of s-bit used by the CFB
  - **it is possible to have *s*-bit CFB with *s* different from *b* (i.e. the $E_k(\cdot)$ size), e.g. 8 bits**
    - with OFB or CBC if octects (bytes) are lost in transmission or extra octects are added, the rest of transmission is garbled
    - with 8-bit CFB as long as an error is an integral number of octects, things will be resynchronized

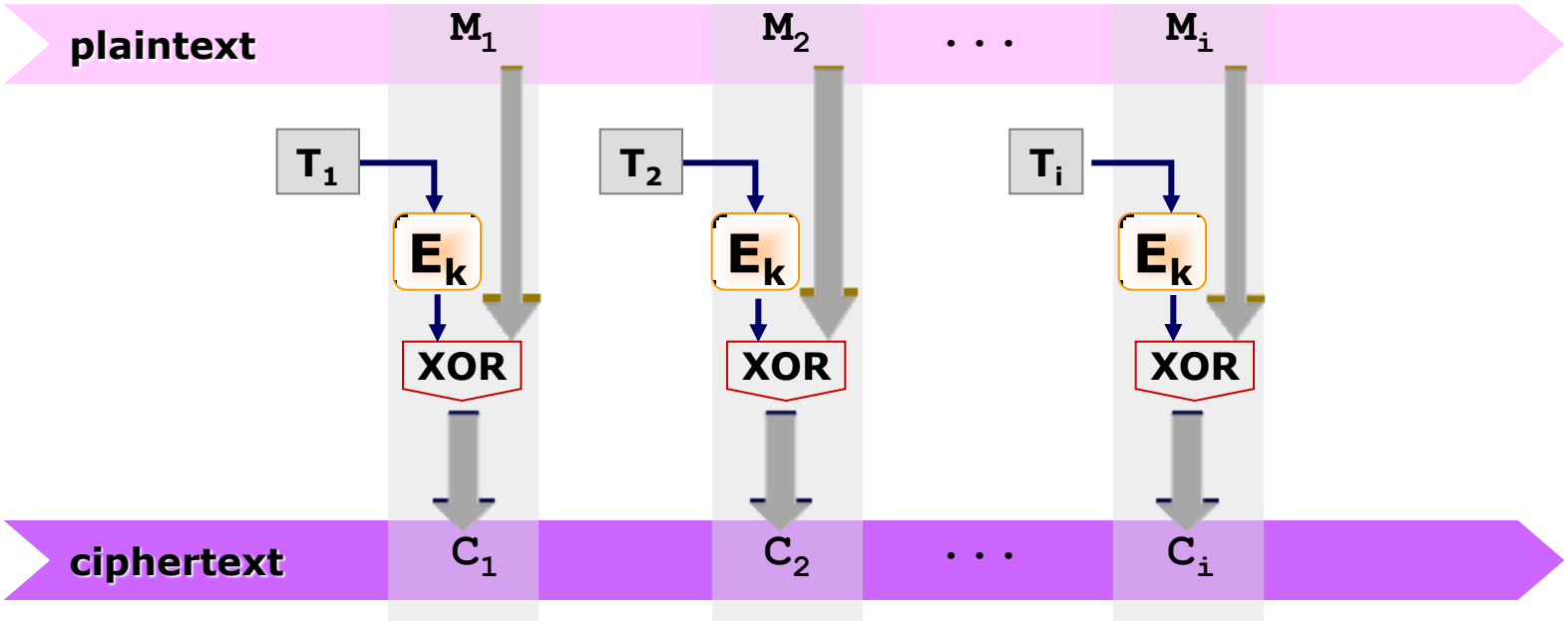
Counter (CTR) Mode

- Similar to the OFB
- The CTR output blocks are generated by encrypting a set of input blocks T_i called counters

  $O_i = E_K(T_i)$

  $C_i = M_i \text{ XOR } O_i$

  - **the sequence of counters $T_1, T_2, .., T_i$ must have the property that each block in the sequence is different from every other block**

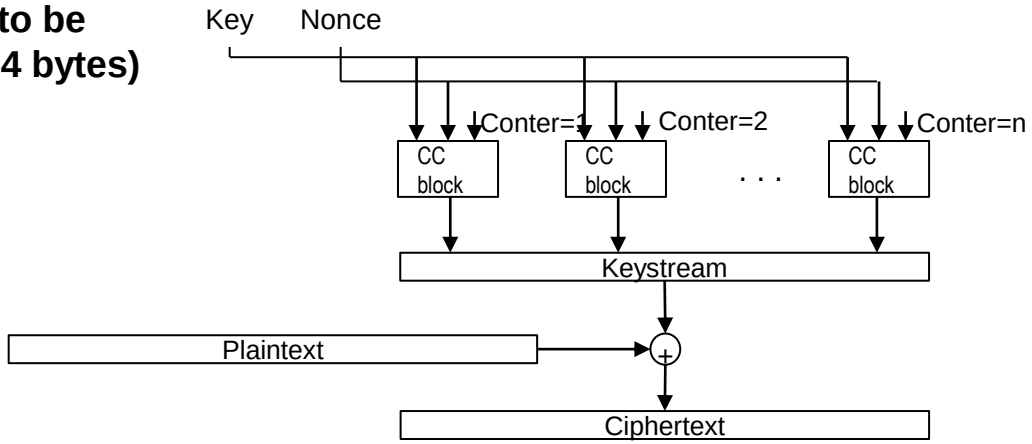


Counter (CTR) Mode (cont.)

- Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^n, where n is the block size)
- Advantages:
  - **the forward cipher functions can be applied to the counters prior to the availability of the plaintext or ciphertext data**
    - like OFB
  - **the cipher functions can be performed in parallel**
  - **the plaintext block that corresponds to any particular ciphertext block can be recovered independently from the other plaintext blocks if the corresponding counter block can be determined**

ChaCha20

- Stream cipher proposed by Daniel J. Bernstein (in 2008)
  - **refinement of Salsa20 stream cipher (2005, same author)**
- Encryption/decryption performed by XOR of a keystream with the plaintext/ciphertext
  - **Uses a 256-bit key, 32-bit initial counter, 96-bit nonce (IV)**
  - **Keystream is the concatenation of 512-bit keystream blocks generated by a block function**
  - **ChaCha20 block function takes as input the key, current counter, and nonce values, and gives as output a 512-bit keystream block**
    - internally works in rounds (20) on a internal state
    - the initial state is formed by the concatenation of 128-bit constant, key, counter, and nonce
  - **No requirement for the plaintext to be an integral multiple of 512 bits (64 bytes)**



Symmetric encryption/decryption in Python

- Example of ECB encryption (using PyCryptodome module):

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

plaintext= b"hello"
key= bytes.fromhex("11223344556677881122334455667788")
print("text:",plaintext.decode())
print("m:",plaintext.hex())
print("k:",key.hex())

cipher= AES.new(key,AES.MODE_ECB)
ciphertext= cipher.encrypt(pad(plaintext,16))
print("c:",ciphertext.hex())

plaintext= cipher.decrypt(ciphertext)
print("m:",plaintext.hex())
print("text:",unpad(plaintext,16).decode())
```

Symmetric encryption/decryption in Python

- Example of CBC encryption:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

plaintext= b"This is a long message that requires several AES blocks."
print("text:",plaintext.decode())
plaintext= pad(plaintext,16);
key= bytes.fromhex("11223344556677881122334455667788")
print("text:",plaintext.decode())
print("m:",plaintext.hex())
print("k:",key.hex())
iv= bytes.fromhex("00000000000000000000000000000000")
print("iv:",iv.hex())
cipher= AES.new(key,AES.MODE_CBC,iv)
ciphertext= cipher.encrypt(plaintext)
print("c:",ciphertext.hex())
cipher= AES.new(key,AES.MODE_CBC,iv)
plaintext= cipher.decrypt(ciphertext)
print("m:",plaintext.hex())
plaintext= unpad(plaintext,16)
print("text:",plaintext.decode())
```