UNIVERSITA' DEGLI STUDI DI PARMA
Dipartimento di Ingegneria dell'Informazione

Secret Key (symmetric) Cryptography

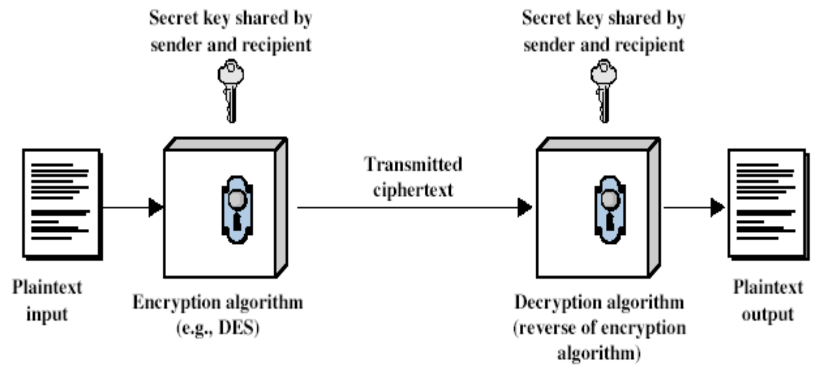
Luca Veltri
(mail.to: luca.veltri@unipr.it)

Course of Network Security, Spring 2014
http://www.tlc.unipr.it/veltri

Symmetric Encryption

- 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**
- Secret key cryptographic systems are designed to take a reasonable-length key (e.g. 64 bits) and generating a one-to-one mapping that "looks like completely **random**", to someone doesn't know the key

Symmetric Cipher Model


Secret key shared by sender and recipient
Secret key shared by sender and recipient
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

Symmetric Cipher Model

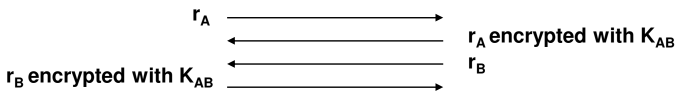
- Two requirements for secure use of symmetric encryption:
  - **a strong encryption algorithm**
  - **a shared secret key known only to sender / receiver**
- plaintext = m
- ciphertext = c = $E_k(m)$
- decrypted plaintext = $D_k(c) = D_k(E_k(m)) = m$
- Assume encryption algorithm is known
- Requires an initial phase where the two parties exchange in secure manner a shared secret key
  - **implies a secure channel (or method) to distribute the key**
- The total number of secret keys for N users is Nx(N-1)/2
  - **in case of pre-shared permanent keys**

Symmetric Cipher Characteristics

- it is generally used for protecting (through encryption) some data stored in a repository or sent to a remote entity
- The robustness of the algorithm is usually proportional to the key length: 40 bit (weak), 128 bit (strong)
- Most common symmetric algorithms:
  - **DES, 3DES, RC2, RC4, IDEA, AES**

Security uses of secret key cryptography

- Transmitting over an insecure channel
  - **the two parties agree on a shared secret key and use secret key cryptography to send messages**
- Secure storage on insecure media
  - **the user uses a secret key to store and retrieve data on an (insecure) media**
- Authentication
  - **strong authentication through a challenge-response mechanism**



- Integrity check
  - **generating a fixed-length cryptographic checksum associated with a message (Message Integrity Check - MIC)**

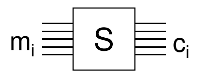
Disadvantages of symmetric cryptography

- Requires a secure method for key exchange
- A lot of keys to be handle when the number of entities grows

$$[\ n(n-1)\ ]/2$$

Classical Substitution Ciphers

- Where letters of plaintext are replaced by other letters or by numbers or symbols
- Or if plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns



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

Substitution Table

- Simple examples of classical substitution ciphers
  - **monoalphabetic substitution (cifrario monoalfabetico)**
  - **monoalphabetic substitution with shift (e.g. Caesar cipher)**
  - **polialphabetic substitution (cifrario polialfabetico)**

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 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(P) = (P + k) mod (26), with k=3

  P = D(C) = (C – k) mod (26) with k=3

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

Cryptanalysis of a Shift Cipher

- only have 26 possible ciphers
  - **A maps to A,B,..Z**
- could simply try each in turn
- a **brute force search** ⟶
- given ciphertext, just try all shifts of letters
- do need to recognize when have plaintext
- eg. break ciphertext "GCUA VQ DTGCM"

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 (jumble) the letters arbitrarily
- Each plaintext letter maps to a different random ciphertext letter
- Example:
  - **Sostitution table:**

abcdefghijklmnopqrstuvwxyz
DKVQFIBJWPESCXHTMYAUOLRGZN

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

- Note: the secret sostitution can be seen as key
  - **26 letters long**

Monoalphabetic Cipher Security

- Now have a total of 26! ≅ 4 x 10^{26} keys
  - **with so many keys, might think is secure**
  - **but would be WRONG!!**
- Problem is language characteristics
  - **letter frequencies**
  - **most common words**
  - **two letters frequencies (e.g. “th” in english)**
  - **etc.**

Polyalphabetic Ciphers

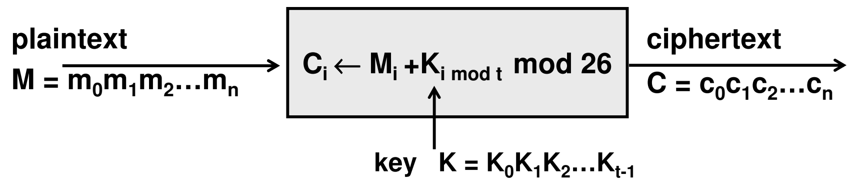
- Another approach to improving security is to use multiple cipher alphabets
- Called **polyalphabetic substitution ciphers**
- Makes cryptanalysis harder with more alphabets to guess and flatter frequency distribution
- Use a key to select which alphabet is used for each letter of the message
- Use each alphabet in turn
- Repeat from start after end of key is reached

Vigenère Cipher

- Simplest polyalphabetic substitution cipher is the **Vigenère Cipher,** 1586
  - **Blaise de Vigenère, 1523-1596**
- Effectively multiple caesar ciphers
- Key is multiple letters long K = k1 k2 ... kd
- ith letter specifies ith alphabet to use
- Use each alphabet in turn
- Repeat from start after *d* letters
- Decryption simply works in reverse

```
(01) 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
(02) 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
(03) 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
(04) 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
(05) 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
(06) 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
(07) 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
(08) 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
(09) 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
(10) 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
(11) 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
(12) 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
(13) 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
(14) 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
(15) 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
(16) 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
(17) 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
(18) 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
(19) 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
(20) 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
(21) 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
(22) 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
(23) 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
(24) 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
(25) 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
(26) 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.)


plaintext
$M = m_0 m_1 m_2 \ldots m_n$
$C_i \leftarrow M_i + K_{i \bmod t} \bmod 26$
ciphertext
$C = c_0 c_1 c_2 \ldots c_n$
key $K = K_0 K_1 K_2 \ldots K_{t-1}$

- Example:

```
key: REBUS

plaintext:       CODIC EMOLT OSICU RO
key:             REBUS REBUS REBUS RE
ciphertext:      TSECU VQPFL FWJWM IS
```

```
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
```

Cryptanalysis of Vigenère Ciphers

- Have multiple ciphertext letters for each plaintext letter
- Hence letter frequencies are obscured
- But not totally lost
- Start with letter frequencies
  - **see if look monoalphabetic or not**
- If not, then need to determine number of alphabets, since then can attack each

One-Time Pad

- If K={k1,k2,k3,..,kn} is as long as the plaintext M,
  - **ciphertext contains no statistical relationship to the plaintext**
  - **for any plaintext & any ciphertext there exists a key mapping one to other**
- If new truly random key K is used for each message
  - **no statistical relationship between distinct ciphertexts**
  - **the cipher will be (unconditionally) secure**
- In case of alphabet {0,1}, it is:
  - **C = M XOR K**
  - **two possible substitutions:**
    - {0,1}→{0,1} that is $c_i = m_i$ XOR k_i, with $k_i=0$
    - {0,1}→{1,0} that is $c_i = m_i$ XOR k_i, with $k_i=1$
  - **Called a One-Time Pad (OTP)**
- Disadvantages: can only use the key once
  - **have problem of safe distribution of key**

Autokey Cipher

- Ideally want a key as long as the message
- Vigenère proposed the **autokey** cipher
- Keyword is prefixed to message and used together as key
- eg. given key *deceptive*

```
plaintext: wearediscoveredsaveyourself
key:       deceptivewearediscoveredsav
ciphertext:ZICVTWQNGKZEIIGASXSTSLVVWLA
```

- But still have frequency characteristics to attack

Autokey Cipher (cont.)

- Modern autokey ciphers follow the same approach of using either key bytes or plaintext bytes to generate more key bytes
  - **however they use different encryption methods**
- Most modern stream ciphers are based on pseudorandom number generators
  - **the key is used to initialize the generator, and either key bytes or plaintext bytes are fed back into the generator to produce more bytes**
- Some stream ciphers are said to be "self-synchronizing", because the next key byte usually depends only on the previous N bytes of the message
  - **if a byte in the message is lost or corrupted, after N bytes the keystream goes back to normal**

Transposition Ciphers

- Another technique is that used by classical **transposition** or **permutation** ciphers
- These hide the message by rearranging the letter order (blocks fo bits)
- Without altering the actual letters used
- Can recognize these since have the same frequency distribution as the original text

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 secure because of data specific characteristics (e.g. language characteristics)
- Hence consider using several ciphers in succession to make harder:
  - **two substitutions make a more complex substitution**
  - **two transpositions make more complex transposition**
  - **but a substitution followed by a transposition makes a new much harder cipher**
- This is bridge from classical to modern ciphers

Rotor Machines

- Before modern ciphers, rotor machines were most common product cipher
- Were widely used in WW2
  - **German Enigma, Allied Hagelin, Japanese Purple**
- Implemented a very complex, varying substitution cipher
- Used a series of cylinders, each giving one substitution, which rotated and changed after each letter was encrypted
- with 3 cylinders have 26x26x26=26^3=17576 alphabets

Steganography

- An alternative to encryption
- Hides existence of message
  - **using only a subset of letters/words in a longer message marked in some way**
  - **using invisible ink**
  - **hiding in graphic image or sound file**
- Has drawbacks
  - **high overhead to hide relatively few info bits**

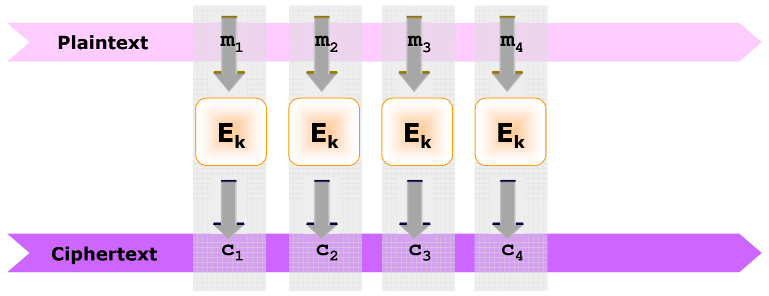
Block and Stream Ciphers

Block and Stream Ciphers

- There are two basic cipher structures
  - **Block ciphers**
    - Block ciphers process messages into blocks, each of which is then en/decrypted
    - Plaintext is treated as a sequence of n-bit blocks of data
    - Ciphertext is same length as plaintext
    - Like a substitution on very big characters (64-bits or more)
    - Can be made to behave as a stream cipher
  - **Stream ciphers**
    - Stream ciphers process messages (Encryption/Decryption) a bit or byte at a time when en/decrypting
    - Often easier to analyze mathematically
- many current ciphers are block ciphers (DES, AES, etc.)

Example of Block vs. Stream Ciphers

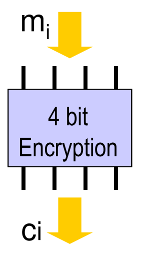
m_i	**plaintext block**
c_i	**ciphertext block**



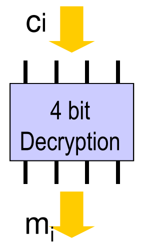
Block Ciphers

- Block ciphers look like an extremely large substitution
- If 64bit blocks are used, 2^{64} possible input values are mapped to 2^{64} output values
- The most general way of encrypting could be to specify completely the mapping table
  - **would need table of 2^{64} entries for a 64-bit block $\rightarrow$ 2^{70} bits!**
  - **it is too long for a key..**
  - **instead create from smaller building blocks**

Example of 4 bits block cipher



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

- How long should the plaintext block be?
  - **having block length too short (say one byte as in monoalphabetic cipher), it could be easier to costruct a decryption table starting from some <plaintext,ciphertext> pairs**
  - **having block length too long, it could be inconvenient due to the increasing of complexity**
- 64bit blocks are often used
  - **it is difficult to obtain all 2^{64} pairs (known plaintext attack)..**

Product ciphers

- Cipher needs to completely obscure statistical properties of original message
  - **a one-time pad does this**
- The concept of product ciphers is due to Claude Shannon
  - **"Communication Theory of Secrecy Systems", 1949**
  - **introduced idea of SP-networks, based on the two primitive cryptographic operations (substitution and permutation)**
  - **these form the basis of modern block ciphers**
- Provide confusion and diffusion of message
  - **confusion**
    - makes relationship between ciphertext and key as complex as possible
  - **diffusion**
    - dissipates statistical structure of plaintext over bulk of ciphertext
    - every single plaintext cipher will influence several ciphertext ciphers

Product ciphers

- Product cipher is a type of block cipher that works by executing in sequence a number of simple transformations such as substitution, permutation, and modular arithmetic
- Usually consist of iterations of several rounds of the same algorithm
  - **while the individual operations are not themselves secure, it is hoped that a sufficiently long chain would imbue the cipher with sufficient confusion and diffusion properties as to make it resistant to cryptanalysis**
- A product cipher that uses only substitutions and permutations is called a SP-network
  - **Feistel ciphers are an important class of product ciphers**
- The operation must be reversible!

Substitution and Permutation

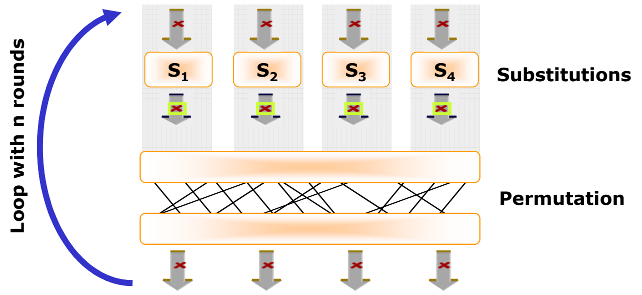
- Substitution
  - **specifies for each of the 2^k possible input values the k-bit output**
  - **this is not practical for 64-bit blocks, but is possible for lower length blocks (e.g. 8bits)**
  - **to specify a substitution on k-bit blocks, $k \cdot 2^k$ bits are required**
- Permutation
  - **specifies for each of the k input bits the corresponding output position**
  - **a permutation is a special case of substitution in which each bit of the output gets its value from exactly one bit of the input**
  - **to specify a permutation on k-bit blocks, $k \cdot \log_2 k$ bits are required**

SP-network

- Substitution-permutation network (SPN)
- A series of linked mathematical operations used in block cipher algorithms
- Consist of S-boxes and P-boxes that transform blocks of input bits into output bits
- A good S-box will have the property that changing one input bit will change about half of the output bits
  - **each output bit should depend on every input bit**
- P-boxes permute or transpose bits
- In addition, at each round the key is combined using some group operation, typically XOR

SP-network

- One possible way to build a secret key algorithm is
  - **to 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***



Design Principles

Block Ciphers are defined in terms of

- **block size**
  - ➢ **increasing size improves security, but slows cipher**
- **key size**
  - ➢ **increasing size improves security, makes exhaustive key searching harder, but may slow cipher**
- **number of rounds**
  - ➢ **increasing number improves security, but slows cipher**
- **subkey generation**
  - ➢ **greater complexity can make analysis harder, but slows cipher**
- **round function**
  - ➢ **greater complexity can make analysis harder, but slows cipher**

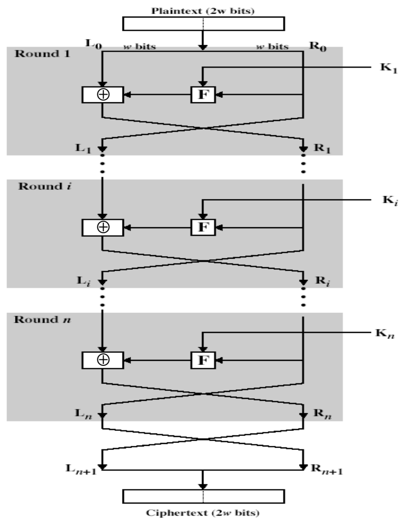
Other considerations

- **fast software en/decryption & ease of analysis**

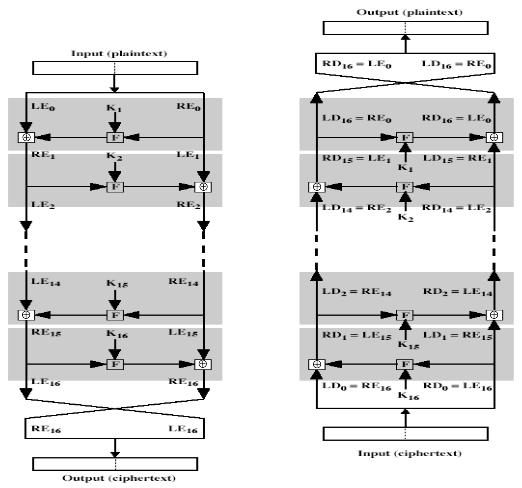
Feistel Cipher Structure

- Horst Feistel devised the **feistel cipher**
  - ➢ **based on concept of invertible product cipher**
- partitions input block into two halves
  - ➢ **process through multiple rounds which**
  - ➢ **perform a substitution on left data half based on round function of right half & subkey**
  - ➢ **then have permutation swapping halves**
- implements Shannon's substitution-permutation network concept

Feistel Cipher Structure



Feistel Cipher Decryption



Data Encryption Standard (DES)

Data Encryption Standard (DES)

- Published in 1977 by National Bureau of Standards (NBS), now NIST (National Institute of Standards and Technology), for use in commercial and unclassified U.S. Government applications
  - **FIPS PUB 46-3 (Federal Information Processing Standards PUBlication)**
  - **U.S. Dept. OF Commerce**
  - **NIST**
- Has widespread use
  - **most widely used block cipher in world**
- Has been considerable controversy over its security

DES History

- 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**
- Redeveloped as a commercial cipher with input from NSA and others
  - **in 1973 NBS issued request for proposals for a national cipher standard**
  - **IBM submitted their revised Lucifer which was accepted as the DES**
- DES has become widely used, e.g. in financial applications
- in 1999 NIST published a new version called *triple DES* (*3DES*) or *TDEA*

Data Encryption Standard (DES)

- DES encrypts 64-bit data using 56-bit key
- It consists of
  - **initial permutation of the 64 bits**
  - **16 identical "rounds" of operation where the data is confused and diffused with the key and the previous round**
  - **A final permutation**
- DES can be efficiently implemented in hardware
- Relatively slow if implemented in software
  - **this was not a documented goal**
  - **..however people have asserted that it was designed with this in mind**

DES Design Controversy

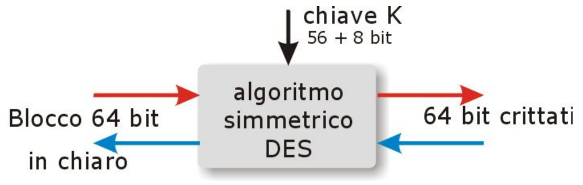
- Although DES standard is public, it was considerable controversy over design
  - **in choice of 56-bit key (vs Lucifer 128-bit)**
  - **and because design criteria were classified**
- Subsequent events and public analysis show design was appropriate

Why 56bits keys?

- yes.. 8x7 bits + 8 bits for parity check..
- however, 8 bits for parity check are too small
  - **64 bits of garbage have 1 in 256 chance to look like a valid key**
- people have suggested that key length has been reduced from 64 to 56 to let DES to be broken (only) by the NSA (some years ago..)

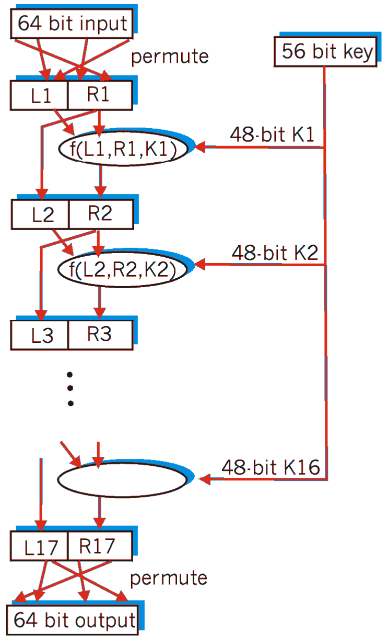
DES

- Uses a 64-bit key that is reduced to 56-bits for parity checking
- Transforms 64-bit blocks of input M to 64-bit blocks of output C
- Same algorithm for encryption and decryption (sub-keys are used in reverse order for decryption)



DES

- Consists of
  - **Key transformation**
    - the 56-bit key is transformed in to 16 48-bit subkeys (one per round)
  - **An initial permutation (P)**
  - **16 rounds of:**
    - the rightmost 32 bits of the input are moved to the left 32 bits of the output
    - then a function f() is run on the left and right halves, and the key
    - the key is shifted for each round
  - **A final permutation (P^{-1})**
- Why permuting??



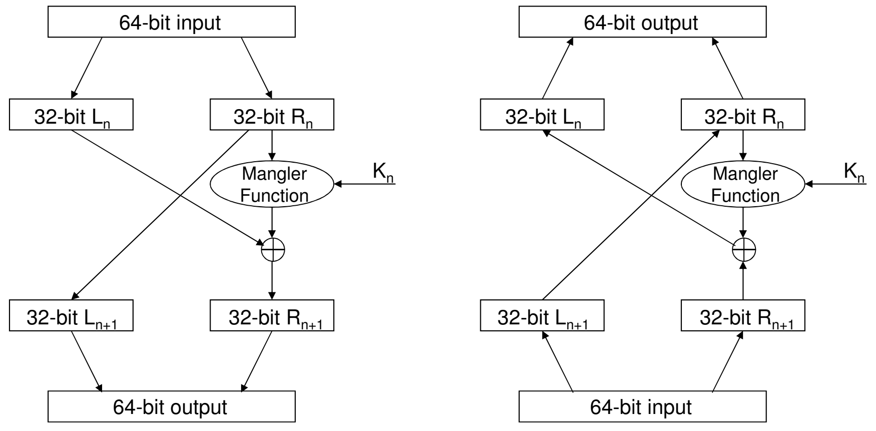
Initial Permutation (IP)

- First step of the data computation
- IP reorders the input data bits
- Even bits to LH half, odd bits to RH half
- Quite regular in structure (easy in HW)

- example:

```
IP(675a6967 5e5a6b5a) = (ffb2194d 004df6fb)
```

DES Round



Encryption ***Decryption***

DES Round

- uses two 32-bit L & R halves
- as for any Feistel cipher can describe as:

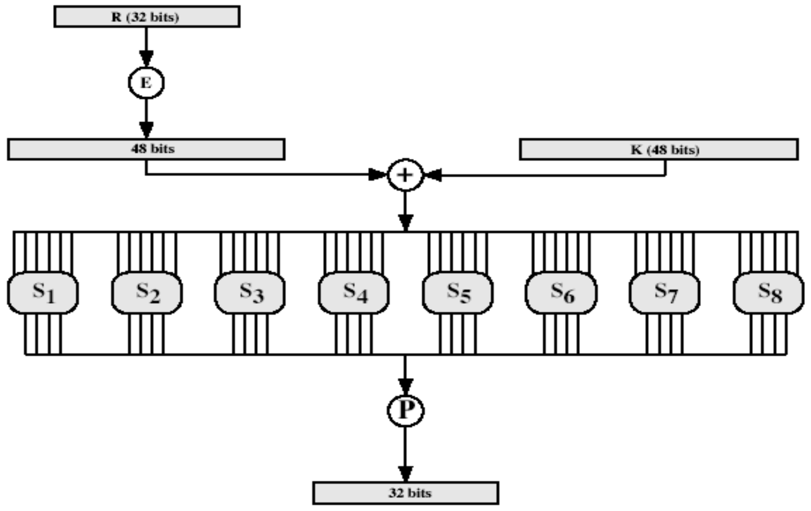
  $\mathbf{L_i = R_{i-1}}$

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

- Mangle Function details
  - **Take 48 bits of the shifted key**
  - **Expand the right 32-bits of the data to 48 bits**
  - **XOR the two together, send it through "S-Box"**
  - **The S-BOX is a predefined substitution table**
  - **The S-BOX produces 32 new bits, which is permuted and XORed with the left half**

- Incredibly, this process is reversible

DES Round: Mangle Function



DES Round: Mangle Function

- Uses 8 (6x4) S-boxes
  - **6 input bytes yields 4 output bytes**
  - **each S-box is actually 4 little 4 bit boxes**
    - outer bits 1 & 6 select one of the 4 little boxes
    - inner bits 2-5 are substituted
    - result is 8 groups of 4 bits (32 bits)
  - **simply a lookup table of 8 x 4 rows and 16 columns**
    - 4 rows for each S-box
    - outer bits 1 & 6 (**row** bits) select one of the 4 rows
    - inner bits 2-5 (**col** bits) are substituted
- Example:
  - **Given bits 110011 as input and S-box 6 from DES**
    - Take first and last bits “11” to choose row 3
    - Take middle four bits “1001” to choose column 9
    - The value from S-box 6 of DES is 14 (“1110”)
    - Substitute “110011” to “1110”
  - **Always count rows and columns from 0 not 1**

DES Round: S-boxes

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Example:

S(18 09 12 3d 11 17 38 39) = 5fd25e03

DES Decryption

- decrypt must unwind steps of data computation
- with Feistel design, do encryption steps again
- using subkeys in reverse order (SK16 … SK1)
- note that IP undoes final FP step of encryption
- 1st round with SK16 undoes 16th encrypt round
- ….
- 16th round with SK1 undoes 1st encrypt round
- then final FP undoes initial encryption IP
- thus recovering original data value

Avalanche Effect (Effetto valanga)

- key desirable property of encryption algorithms
  - **where a change of** one **input or key bit results in changing approx** half **output bits**
- DES exhibits strong avalanche

Strength of DES

- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- Originally complaints that the NSA fixed the S-boxes to provide a backdoor. This has never been found
  - **The S-boxes appear to be strong against even differential cryptanalysis (Which means the NSA knew about DC before 1978. It was first described publicly in 1990)**
- Algorithm has never been "broken"
  - **Successfully attacked by brute force**
- Recent advances have shown is possible to break 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

Triple DES: Why?

- The keyspace of DES is too small
- Clear a replacement for DES was needed
  - **theoretical attacks that can break it**
  - **demonstrated exhaustive key search attacks**
- AES is a new cipher alternative
- Prior to this alternative was to use multiple encryption with DES implementations
- Triple-DES is the chosen form

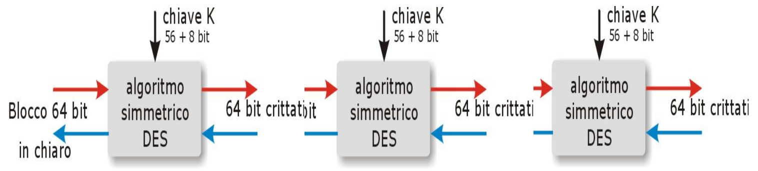
Multiple Encryption

- A possible solution for increase the (computational) security of an encryption algorithm is to use the same algorithm more times
- How many time should be performed? (2,3,4,100..)
- How many keys?
- Both Encryption and Decryption algorithms can be see as encryption functions
  - **what combination of E and D can be chosen? (EEE, ED, etc)**

How many time should be performed? How many keys?

- The more time the block is encrypted the more secure it is
  - **however for computation, no more encryptions than are necessary**
- Encrypting twice with the same key
  - **$c=E_K(E_K(m))$**
  - **no more secure that single encryption with K: exhaustive search requires trying 2^{56} keys**
- Encrypting twise with two keys
  - **$c=E_{K2}(E_{K1}(m))$**
  - **there is an attack (not very practical) that breaks double encryption (EE) in roughly twice the time of a brute-force breaking single E**
    - since $X = E_{K1}[P] = D_{K2}[C]$
    - attack by encrypting P with all keys and store
    - then decrypt C with keys and match X value
    - can show takes $O(2^{56})$ steps
- Triple encryption with two/three keys (e.g. EEE, or EDE)

Triple DES (3-DES)



- **Block size = 64 bit**
- **c= $E_{K3}(D_{K2}(E_{K1}(m)))$**
- **3-DES with three keys: (k_1, k_2, k_3)**
  - key length: 56 + 56 + 56 = 168 bit
- **3-DES with two keys: $(k_1, k_2, k_1)= (k_1, k_2)$**
  - key length: 56+56 = 112 bit
- **Often referred to as EDE (Encrypt Decrypt Encrypt) or TDEA**
- **Used by standard X9.17 e ISO 8732**

Triple-DES with Two-Keys

- Use 2 keys K1 and K2 with E-D-E sequence
  - **$C = E_{K1}(D_{K2}(E_{K1}(M)))$**
- A key space of 2^{112} possible keys
- Encrypt & decrypt are equivalent in security: there is no advantage to using decryption for the second stage
- If K1=K2 we have backwards compatibility with DES
  - **$E_{k1}(D_{k1}(E_{k1}(M))) = E_{k1}(M)$**
- No current known practical attacks

Triple-DES with Three-Keys

- Triple-DES with Three-Keys
  - **$C = E_{K3}(D_{K2}(E_{K1}(M)))$**
- If K1=K2=K3 we have backwards compatibility with DES
  - **$E_{k1}(D_{k1}(E_{k1}(M))) = E_{k1}(M)$**
- Has been adopted by some Internet applications, eg PGP, S/MIME

Some other ciphers

- IDEA (International Data Encryption Algorithm) [1990]
- SAFER (Secure And Fast Encryption Routine)
  SAFER K-64 [1994], SAFER K-128 [1995]
- RC5 [1995]

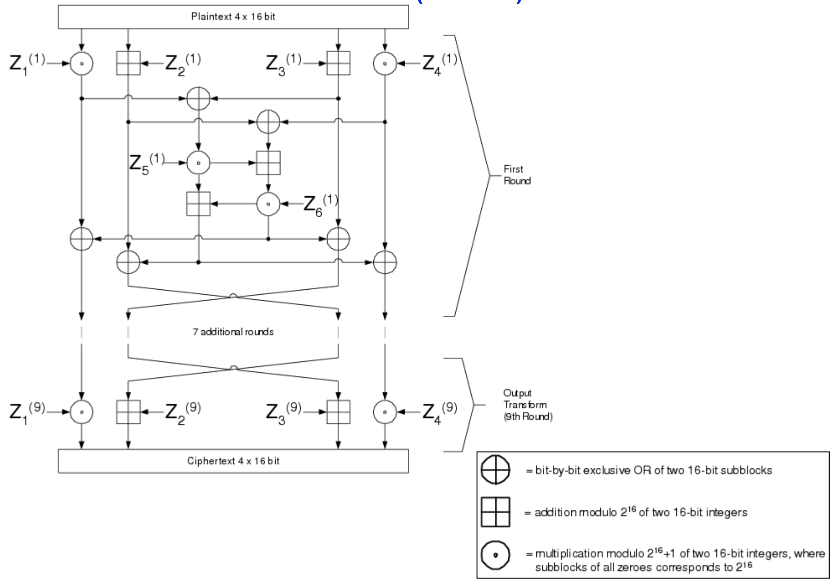
cifrario	bit chiave	bit testo
IDEA	128	64
SAFER K-64	64	64
SAFER K-128	128	64
RC5	<256 byte	32,64,128

- Madryga, NewDES, FEAL, REDOC, LOKI, Khufu, Knafre, RC2, MMB, GOST, Blowfish, …
- AES

IDEA

- International Data Encryption Algorithm
  - **Used in PGP (Pretty Good Privacy)**
  - **Similar to DES, SP-network, substitution are replaced by algebraic operations**
- Works on 64-bit input blocks
  - **Taken as 4 16-bit blocks**
- Operates in rounds (17 steps)
  - **8 identical rounds (composed by two steps) followed by another transformation**
- Uses a 128-bit key
  - **six 16-bit subkeys for each round, plus four keys for the last transformation**
  - **total of 6x8+4=52 16-bit subkeys**
- Decryption uses same algorithm, but different subkey generation

IDEA (cont.)


Plaintext 4 x 16 bit
$Z_1^{(1)}$
$Z_2^{(1)}$
$Z_3^{(1)}$
$Z_4^{(1)}$
$Z_5^{(1)}$
$Z_6^{(1)}$
First Round
7 additional rounds
$Z_1^{(9)}$
$Z_2^{(9)}$
$Z_3^{(9)}$
$Z_4^{(9)}$
Output Transform (9th Round)
Ciphertext 4 x 16 bit
= bit-by-bit exclusive OR of two 16-bit subblocks
= addition modulo 2^{16} of two 16-bit integers
= multiplication modulo $2^{16}+1$ of two 16-bit integers, where subblocks of all zeroes corresponds to 2^{16}

IDEA (cont.)

- Subkeys generation
  - **First, the 128-bit key is partitioned in 8 16-bit blocks (used as the first 8 subkeys)**
  - **The 128-bit key is then cyclically shifted to the left by 25 positions, and partitioned in 8 16-bit blocks as above**
  - **The operation is repeated until 52 subkeys are generated**

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 2 Oct 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)
  - **effective as a standard May 26, 2002**
    - adopted as an encryption standard by the U.S. government
    - currently, one of the most popular algorithms used in symmetric key cryptography

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
- Unlike DES, Rijndael is a substitution-permutation network, not a Feistel network
- Fast in both software and hardware
  - **relatively easy to implement**
  - **requires little memory**

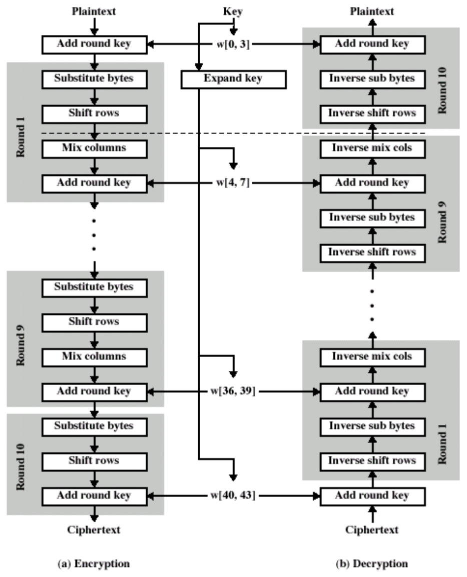
AES Description

- 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 have additional columns in the state**
- Most AES calculations are done in a special finite field

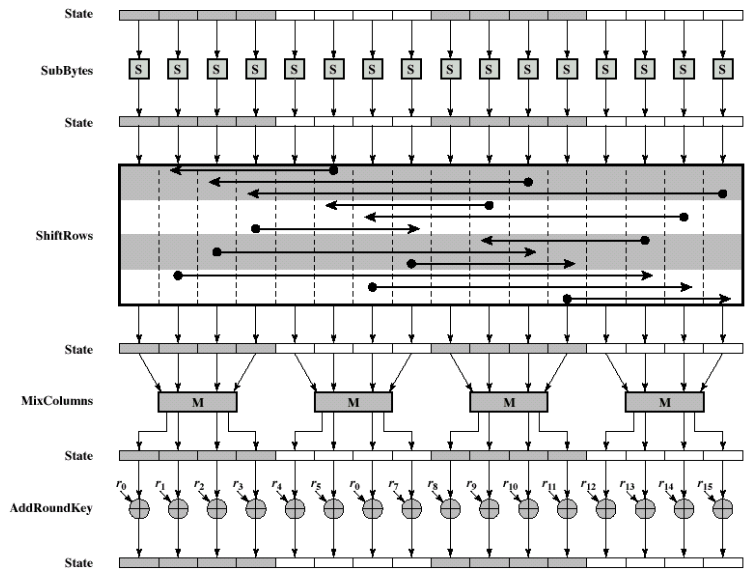
AES cipher

- Algorithm
  - **KeyExpansion using Rijndael's key schedule**
  - **Initial Round**
    - AddRoundKey
  - **9 Rounds**
    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 cipher

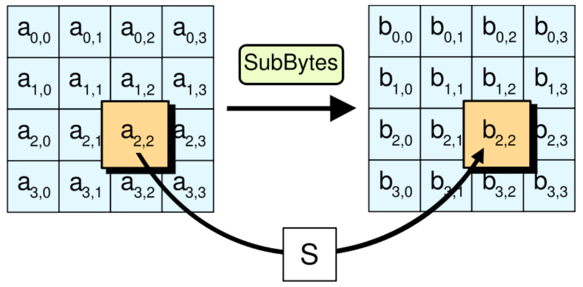


AES single round



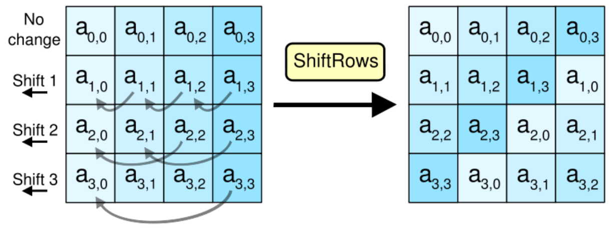
AES SubBytes step

- Each byte in the state is replaced using an 8-bit substitution box, the Rijndael S-box ($b_{ij} = S(a_{ij})$)
  - **this operation provides the non-linearity in the cipher**
  - **the S-box is derived from the multiplicative inverse over GF(2^8), known to have good non-linearity properties**
  - **S-box can be represented by a table of 256 8-bit values**


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

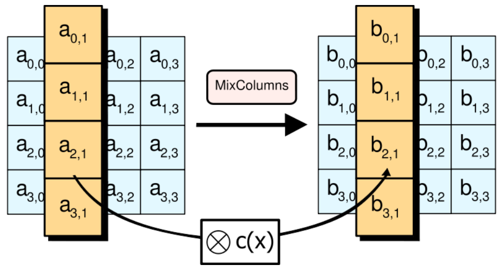
AES ShiftRows step

- ShiftRows step operates on the rows of the state
  - **it cyclically shifts the bytes in each row by a certain offset**
  - **for AES, the first row is left unchanged**
  - **each byte of the second row is shifted one to the left**
  - **similarly, the third and fourth rows are shifted by offsets of two and three respectively**


No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

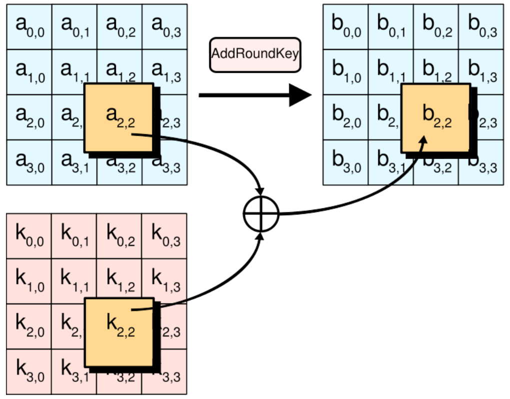
AES MixColumns step

- Four bytes of each column of the state are combined using an invertible linear transformation
  - **together with ShiftRows, MixColumns provides diffusion in the cipher**
  - **each column is treated as a polynomial over GF(2^8) and is then multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x) = 3x^3+x^2+x+2$**
  - **can also be viewed as a multiplication by a particular MDS (Maximum Distance Separable) matrix in Rijndael's finite field**


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$\otimes c(x)$

AES AddRoundKey step

- The subkey is combined with the state
  - **for each round, a subkey with same size as the state is derived from the main key using Rijndael's key schedule**
  - **the subkey is added to the state using bitwise XOR**


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,}$ $a_{2,2}$ $_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
AddRoundKey
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,}$ $b_{2,2}$ $_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,}$ $k_{2,2}$ $_{2,3}$
$k_{3,0}$ $k_{3,1}$ $_{3,2}$ $_{3,3}$

AES Security

- Brute force attack
  - **AES key sizes**
    - $2^{128} = 3.4 \times 10^{38}$ possible 128-bit keys
    - $2^{192} = 6.2 \times 10^{57}$ possible 192-bit keys
    - $2^{256} = 1.1 \times 10^{77}$ possible 256-bit keys
- Comparing to DES, If you could crack a DES key in one second (i.e., try 2^{56} keys per second), it would take 149 trillion years to crack a 128-bit AES key by brute force at the same speed
  - **the universe is believed to be less than 20 billion years old**
  - **but, things change..**

AES Security (cont.)

- In June 2003, the US Government announced that AES may be used also for classified information
  - **This marks the first time that the public has had access to a cipher approved by NSA for encryption of TOP SECRET information**
- As of 2006, the only successful attacks against AES have been side channel attacks
  - **AES has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. By 2006, the best known attacks were on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys**

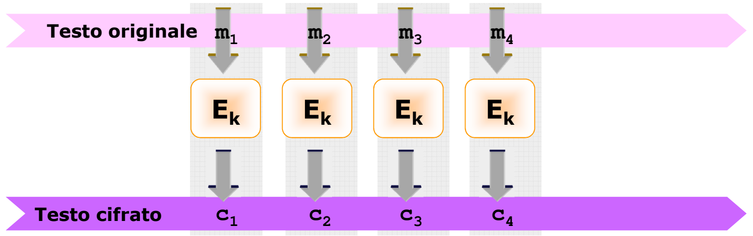
Encrypting Large Messages

Encrypting large messages

- Block ciphers encrypt fixed size blocks
  - **eg. DES encrypts 64-bit blocks, with 56-bit key**
- Need way to use in practice, given usually have arbitrary amount of information to encrypt (longer than 64 bits..)
- Four were defined for DES in ANSI standard ANSI X3.106-1983 Modes of Use
- These schemes are applicable for DES, 3DES, IDEA, AES, etc
- Modes:
  - **Electronic Code Book (ECB)**
  - **Cipher Block Chaining (CBC)**
  - **(k-bit) Output Feedback (OFB)**
  - **(k-bit) Cipher Feedback (CFB)**

Electronic Codebook (ECB)

- Consist of doing the obvious thing, and it is usually the worst method
- The message is broken into 64-bit blocks, with padding for the last one
- Each block is independently encrypted with the secret key K
  - **$C_i = E_K(M_i)$**
  - **each block is a value which is substituted, like a codebook**



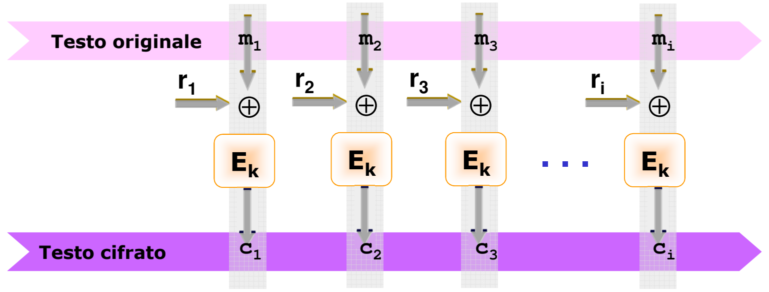
- At end of message, handle possible last short block
  - **by padding either with known non-data value (eg nulls)**
  - **or pad last block with count of pad size, eg. [b1 b2 b3 0 0 0 0 5] (3 data bytes, then 5 bytes pad+count)**

Advantages and Limitations of ECB

- The cipher text has the same size of the plaintext (excluding the 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 64-bit blocks, the corresponding cipher blocs are identical; it can be a problem
      - in some cases it can be possible to guess a portion of the message
      - in some cases it can be possible to alter the message
  - **knowing the plaintext, is possible to rearrange the ciphertext blocks is order to obtain a new (known) plaintext**
- As result, ECB is rarely used
  - **main use is sending a few blocks of data**

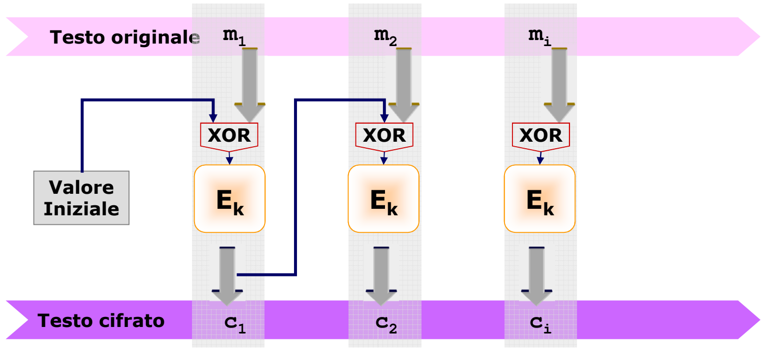
Cipher Block Chaining (CBC)

- CBC avoids some problems in ECB
- If the same block repeats in the plain text, it will not cause repeats of ciphertext
- Adds a feedback mechanism to the cipher
- Plaintext is more difficult to manipulate
- Basic idea:



Cipher Block Chaining (CBC)

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



Advantages and Limitations of CBC

- Decryption is simple because ⊕ is reversible
- CBC has the same performance of ECB, except for the cost of generating and transmitting the IV
- Each ciphertext block depends on **all** message blocks
  - **thus a change in the message affects all ciphertext blocks after the change as well as the original block**
- It can be used the value of 0 as IV, however it can lead to some problems
  - **e.g. if a message is transmitted weekly, it is possible to guess if changes occurred**
  - **Moreover, IV!=0 prevents attackers for supplying chosen plaintext**
- If IV is sent in the clear, an attacker can change bits of the first block, and change IV to compensate
  - **hence either IV must be a fixed value or it must be sent encrypted in ECB mode before rest of message**

CBC Threat 1 - Modifing ciphertext blocks

- Using CBC does not eliminate the problem of someone modifying the message in transit
- If the attacker changes the ciphertext block c_n, c_n gets ⊕'d with the decrypted c_{n+1} to yield m_{n+1}
  - **changing bit h of c_n has predictable effect to bit h of m_{n+1}**
  - **the attacker cannot know the new m_n (a new random 64-bit value, as side effect)**
- This thread can be combated by adding a MIC, or simply a CRC to the plaintext before encrypt

CBC Threat 2 - 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 m'_1, m'_2, m'_3 ..
- A 32 bit CRC can help but not solve the problem (1 in 2^{32} chance that the CRC will work; if the attack consist only in modifying the message, it is possible to try several combination)

Output Feedback (OFB)

- Acts like a pseudorandom number generator
- 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**

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

    $O_0 = IV$
  - **feedback is independent of message and can be computed in advance**
  - **so, a long pseudorandom string is generated (one-time pad)**
  - **the one-time pad is simply ⊕'d with the message**

    $C_i = M_i \text{ XOR } O_i$
- Uses: stream encryption over noisy channels

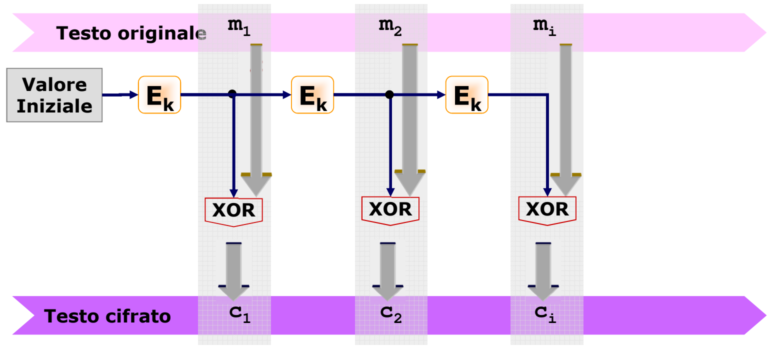
Output Feedback (OFB)

- 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**

    $C_i = M_i \text{ XOR } O_i$



Advantages and Limitations of OFB

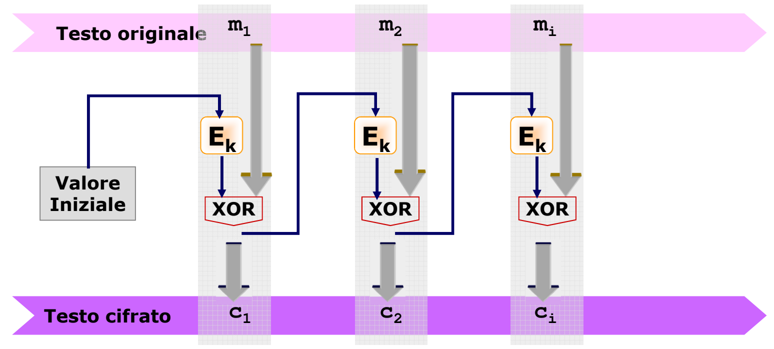
- Advantages of OFB:
  - **one-time pad can be generated in advances**
  - **if a message arrives in arbitrary-sized chucks, the associated ciphertext can immediately be transmitted**
  - **possibility to encrypt variable-length messages: no need of padding**
  - **if some bits of the ciphertext get 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)

- Very similar to OFB
- The k-bit shifted in to the encryption module are the k-bit of the ciphertext from the previous block

  $O_0 = IV$

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



Advantages and Limitations of CFB

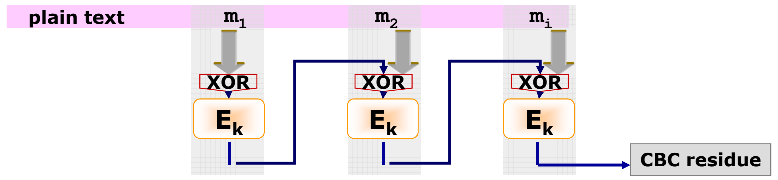
- The block cipher is used in **encryption** mode at **both** ends
- however:
  - **the one-time pad cannot be generated in advance**
  - **needs to stall while do block encryption after every n-bits**
- Errors propagate for several blocks after the error
- however:
  - **it is possible to have k-bit CFB with k different bits from $E_k[\cdot]$ size (64bit for DES), e.g. 8bit**
    - With OFB or CBC if character are lost in transmission or extra character are added, the rest of trasmission is garbled
    - With 8-bit CFB as long as an error is an integral number of bytes, things will be resynchronized
      - a disadvantage is that $E_k[\cdot]$ operation is required each byte

Integrity protection

- CBC, CFB and OFB, when properly used, offer good protection against an eavesdropper deciphering a message
- None of these offers good protection against an eavesdropper (who already knows the plaintext or not) modifying it undetected
  - **note that any random string of bits will decrypt into something**
- In many contexts message are not secret but integrity must be assured
- If both privacy and integrity have to be guaranteed, a sufficient-long checksum (used as MIC) can be generated and encrypted together with the message
  - **send: $E_k(m||mic(m))$**

Generating MIC through CBC

- If privacy and integrity have to be guaranteed separately, or in case only integrity protection has to be provided, a cryptographic checksum, known as MIC (Message Integrity Code), has to be generated as function of the message m and a key k
  - **calculate: mic=mic(k,m)**
  - **send: m||mic**
- A way of generating such MIC is to compute the CBC but send only the last block (named CBC residue) along with the plaintext



Ensuring Privacy and Integrity

- To assure privacy it is possible to CBC-encrypt the message
- To assure only integrity is appropriate to transmit unencrypted data plus a CBC residue
- To assure both privacy and integrity is possible to send CBC-encrypted message together with a CBC-residue, computed with two different keys
  - **related keys can be used**
    - it could be sufficient to switch just one bit
      - e.g. kerberos V5 ⊕s the key with 0xF0F0F0F0F0F0F0F0
      - this has the property of preserving key parity and never transforming non-weak key into a weak-key
  - **double complexity (respect to encrypt only)**
    - other mic functions are computationally less expensive