EECS565 Intro to Computer and Information Security

Elementary Cryptography - 2

Bo Luo
bluo@ku.edu

Elementary Cryptography

Modern Cryptography

Modern Cryptography

- Post-WW-II cryptography
- Secret key cryptography
  - DES
  - AES
- Public key cryptography
  - RSA

Introduction to DES

- Early 70s: non-military crypto research was very unfocused
- 1972: National Bureau of Standards (now NIST) wanted a crypto algorithm which is:
  - secure
  - open
  - efficient
  - useful in diverse applications
- First open solicitation: May 1973
- Second solicitation: August 1974

Introduction to DES

- In response to NBS's second solicitation, IBM submitted *Lucifer*
- DES based on Lucifer
- DES first published in 1975, seeking public comments.
- DES became a federal standard in 1976
- …
- 26 years!
- …
- DES was superseded by AES in 2002

Introduction to DES

- NSA was known to be somewhat involved in the design of DES
  - We sent the S-boxes off to Washington. They came back and were all different.
  - We developed the DES algorithm entirely within IBM using IBMers. The NSA did not dictate a single wire!
- Controversies:
  - Reduced key size
  - Design of S-boxes

Introduction to DES

- DES: Data Encryption Standard
  - Block cipher. 64-bit blocks
  - same algorithm used for encryption and decryption
  - 56-bit keys (effective key length: 56!!)
    - represented as 64-bit
    - but every 8th bit is for parity only
  - symmetric: receiver uses same key to decrypt

Introduction to DES

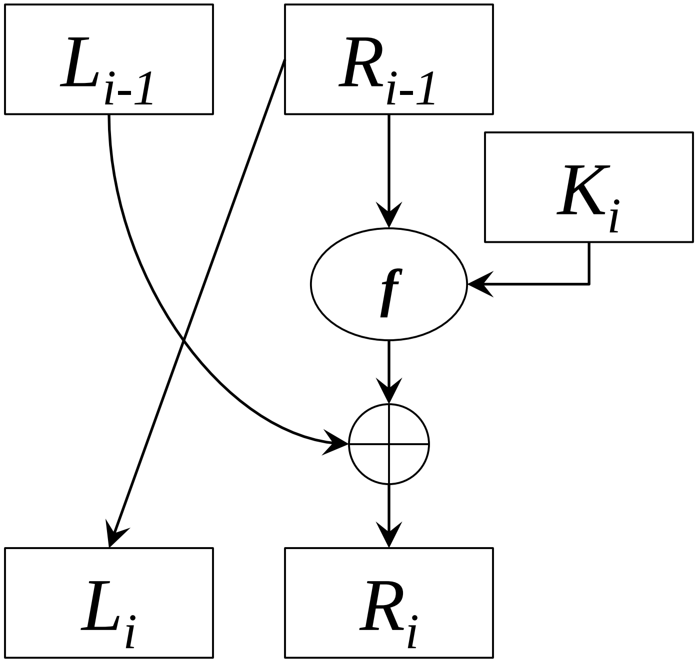
- DES: Data Encryption Standard
- Uses basic techniques of encryption. Provides
  - confusion (substitutions)
  - diffusion (permutations)
- Same process 16 times/block
- Uses standard arithmetic and logical operators
  - efficient hardware implementations

DES

- Break up plaintext into 64-bit blocks
- Each block goes through 16 rounds
  - B_i = block after iteration i
  - L_i = Left half of block after iteration i
    - 32 bits
  - R_i = Right half of block after iteration i
    - 32 bits.

DES

- For each block
  - Initial permutation
  - 16 rounds of substitution and permutation
  - Final permutation
- Each round:
  - $L_i = R_{i-1}$
  - $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$



DES

- Initial Permutation

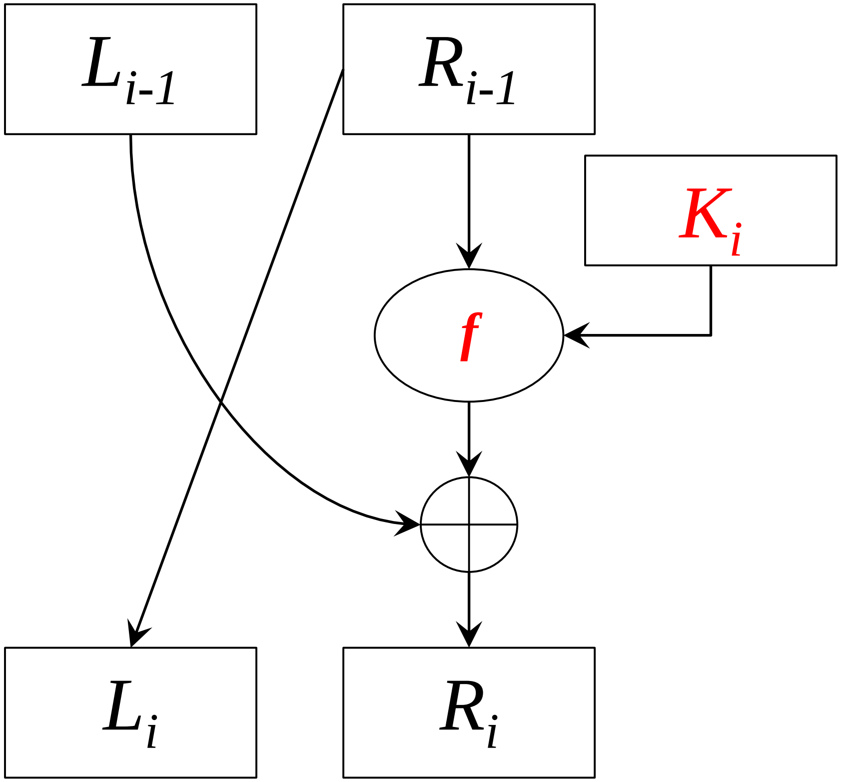
1～64 bit 每行16个

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

- Performed before 16 rounds of encryption
  - Move bit 58 to position 1, move bit 50 to position 2, etc…
- Reversed by Inverse Initial Permutation (a.k.a. final permutation) (after round 16)
- Does not add to security!!
  - This is not a transposition cipher since the algorithm is public!

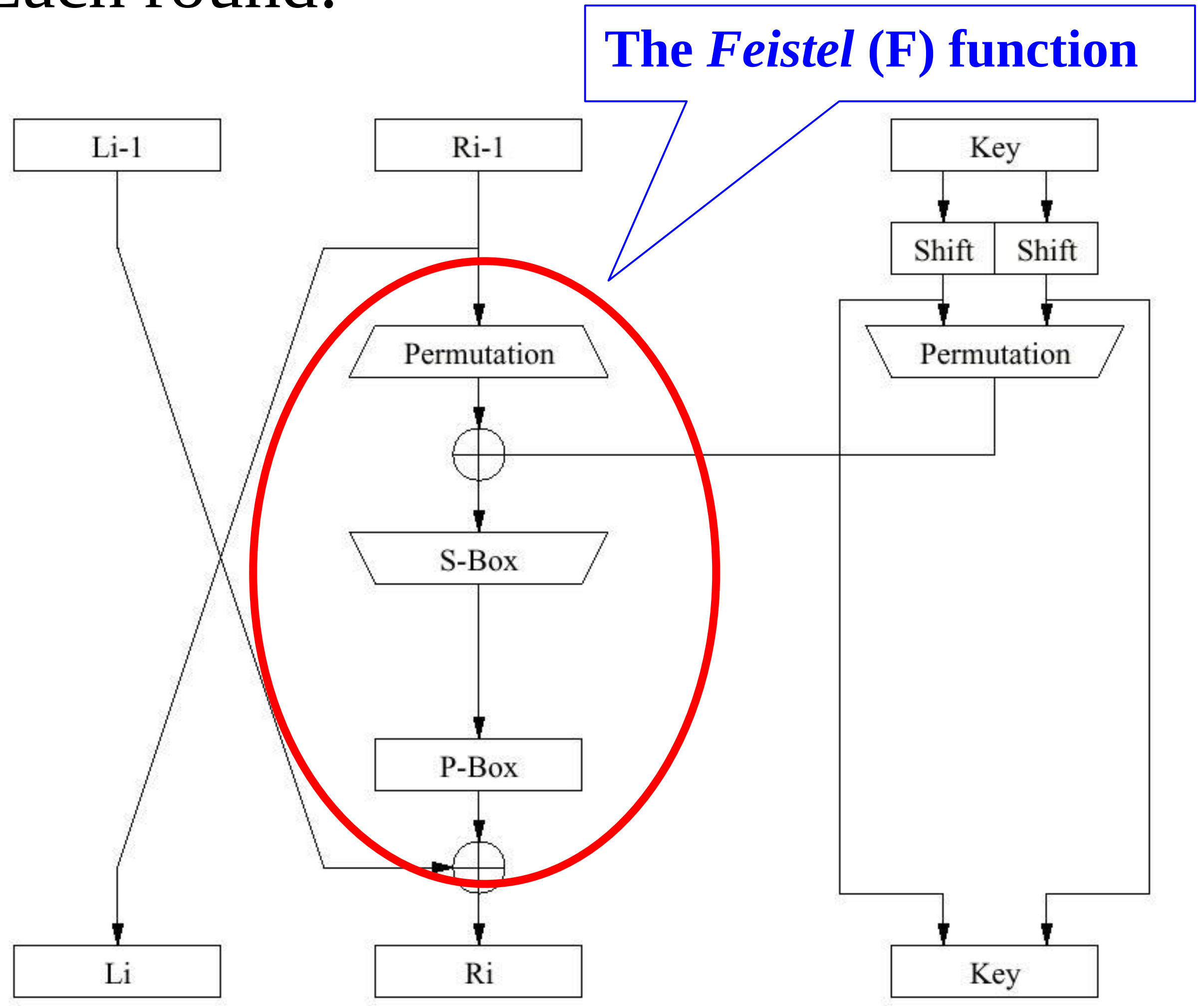
DES

- Each round:
  - $L_i = R_{i-1}$
  - $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$



DES

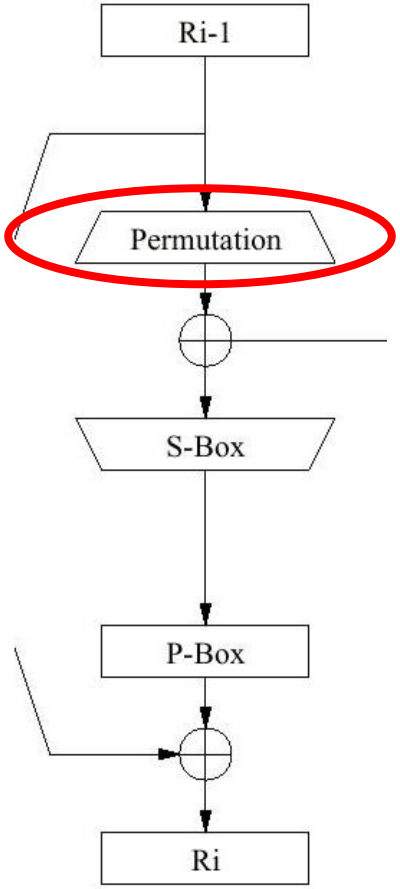
- Each round:



DES

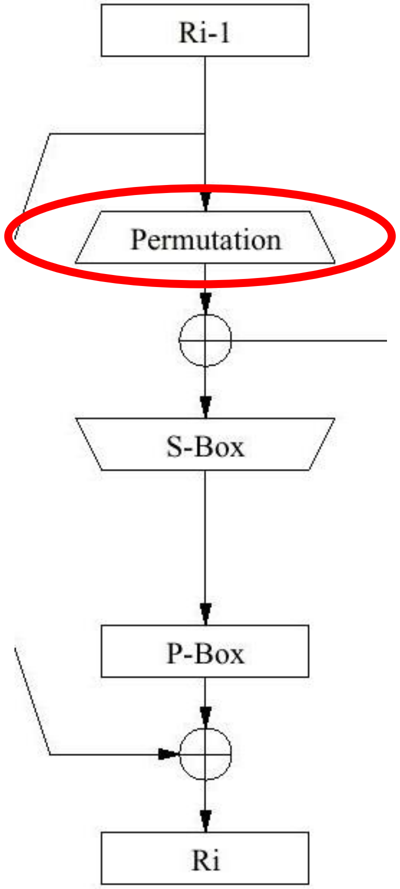
- Expansion permutation
  - *R*: from 32 bits to 48 bits
  - Some bits used twice

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1



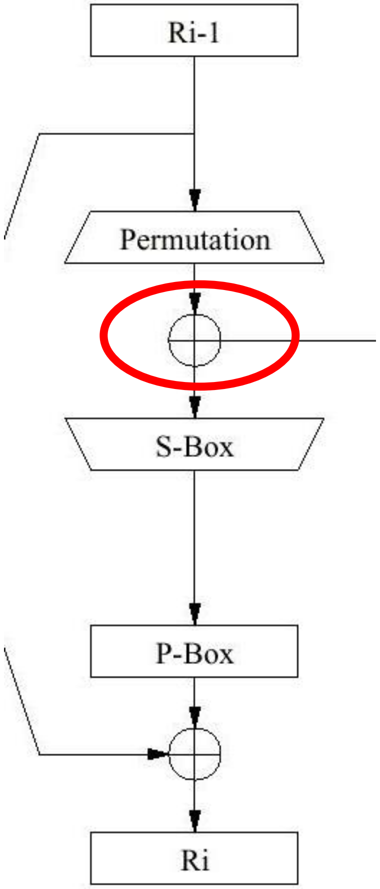
DES

- Expansion permutation
  - *R*: from 32 bits to 48 bits
  - Some bits used twice
    - few bits of plaintext may affect many bits of ciphertext
  - *R* becomes the same length as round-key for XOR



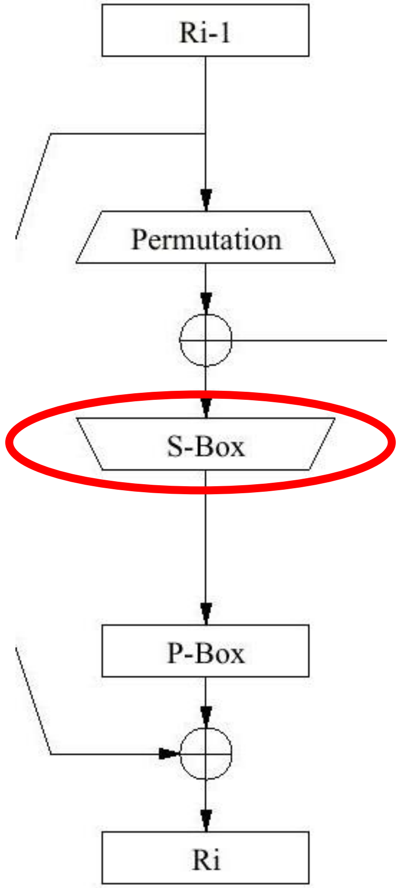
DES

- XOR with key
  - "*Key mixing*"
  - Simple bit-wise XOR with round-*i*-key K_i (48 bits!)
  - How to generate K_i ?
  - Later…



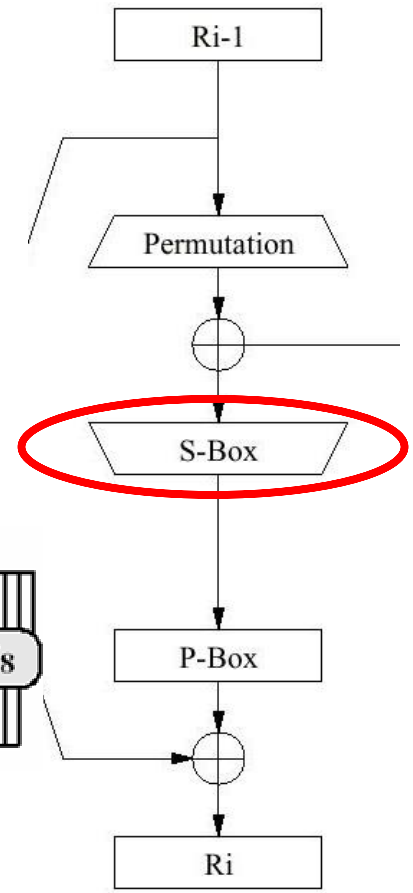
DES

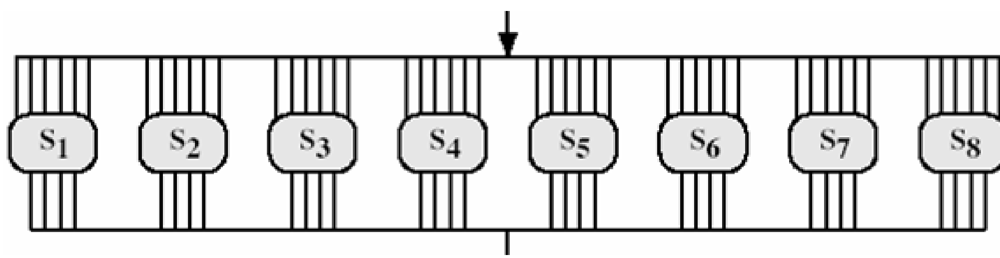
- S-boxes (substitution boxes)
  - *R*: from 48 bits to 32 bits
  - Break *R* into 8 blocks
  - 6 bits/block
  - Block 1 goes through box S_1
  - Block 2 goes through box S_2
  - Block 3 goes through box S_3



DES

- S-boxes
  - *R*: from 48 bits to 32 bits
  - Break *R* into 8 blocks
  - 6 bits/block
  - S-boxes are different





DES

- S-boxes
  - Each box defines a substitution
  - 6-bit input, 4-bit output 输入 输出。。
- S-box 1:

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

  - Look up table: bits 1 and 6 define row, bits 2-5 define column

DES

- S-box 1:

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

  - Look up table: bits 1 and 6 define row, bits 2-5 define column
  - Input: 010011

DES

- S-box 1:

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

  - Look up table: bits 1 and 6 define row, bits 2-5 define column
  - Input: 010011
    - Bits 1,6 → 01 → row 1
    - Bits 2-5 → 1001 → column 9

第一位和最后一位 01 代表 row 1
中间1001 = 9 代表column

DES

- S-box 1:

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

  – Look up table: bits 1 and 6 define row, bits 2-5 define column

  – Input: 010011

    - Bits 1,6 → 01 → row 1
    - Bits 2-5 → 1001 → column 9
    - Output: 6 → 0110

DES

- S-box 1:

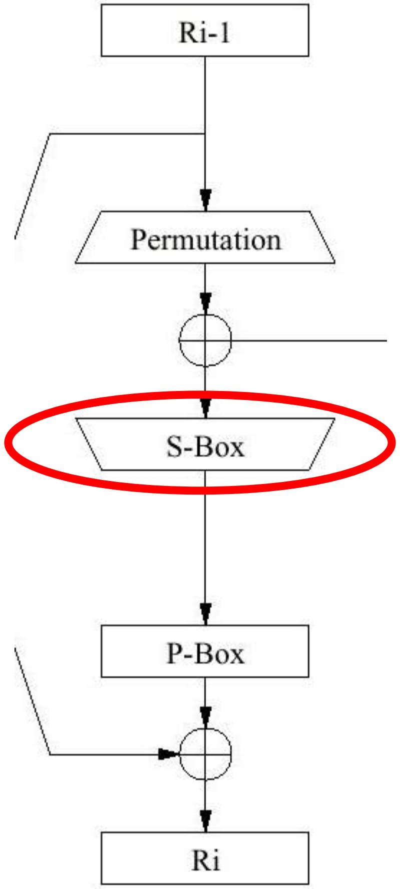
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

  - Look up table: bits 1 and 6 define row, bits 2-5 define column
  - Why use bits 1 and 6 to define row?
  - Each row is a substitution from 0-15 to 0-15, why?

S-box 代表 substitution --> confusion

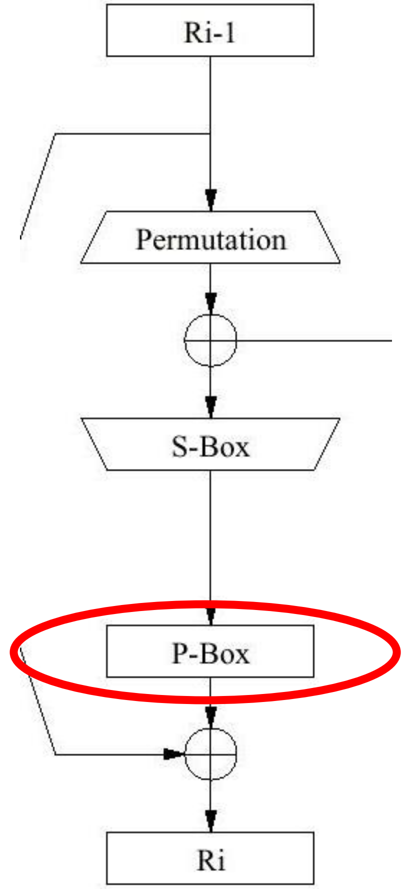
DES

- P-box
  - Input: 32 bits
  - Bits are rearranged according to a fixed permutation.
  - Output: 32 bits
  - Why P-box?
  - Add diffusion
  - Each S-box's output bits are spread across 6 different S-boxes in the next round



DES

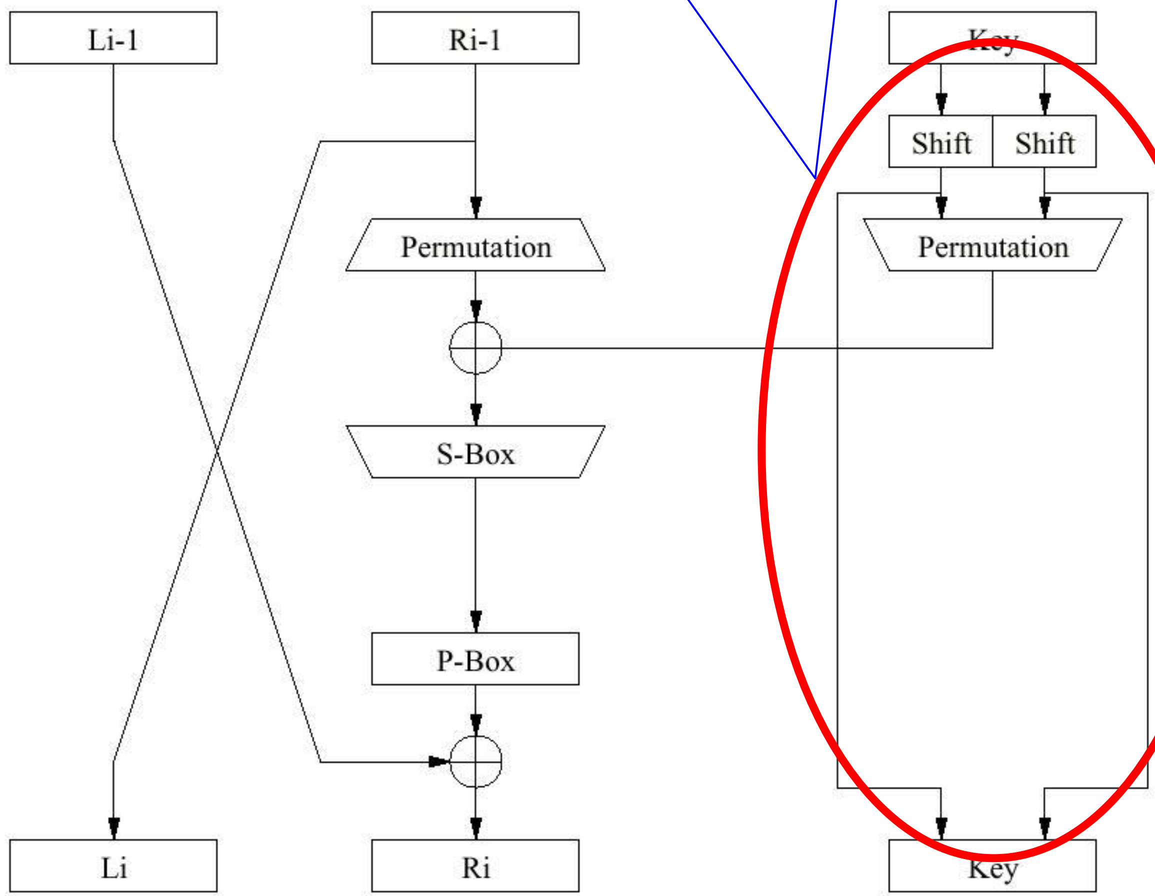
- P-box
  - Input: 32 bits
  - Bits are rearranged according to a fixed permutation.
  - Output: 32 bits
  - Why P-box?
  - Add diffusion
  - Each S-box's output bits are spread across 6 different S-boxes in the next round
  - Change 1 plain text bit: big changes to many blocks after only a few rounds



DES

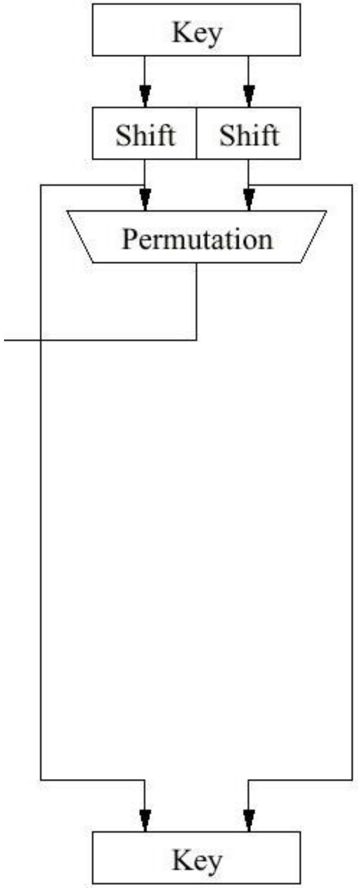
- Each round:

The *key schedule*



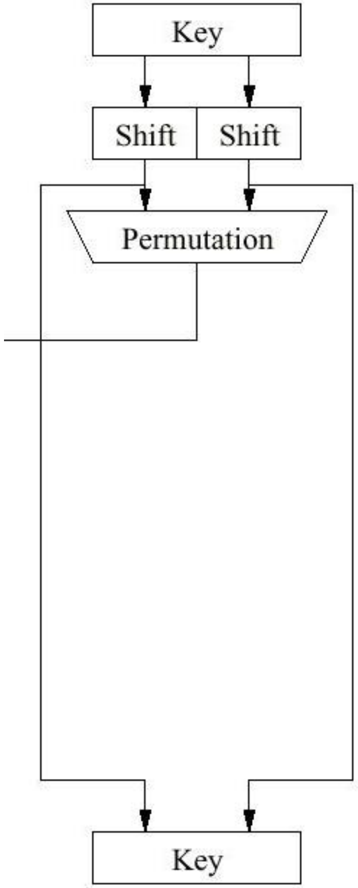
DES

- Key Schedule
  - Key is 56 bits (64 – 8 parity)
- Goes through a permutation before round 1
- Then for each round:
  - divide into two halves
  - circular shift of each half (shift 1 or two bits depending on round)
  - select 48 of the 56 bits



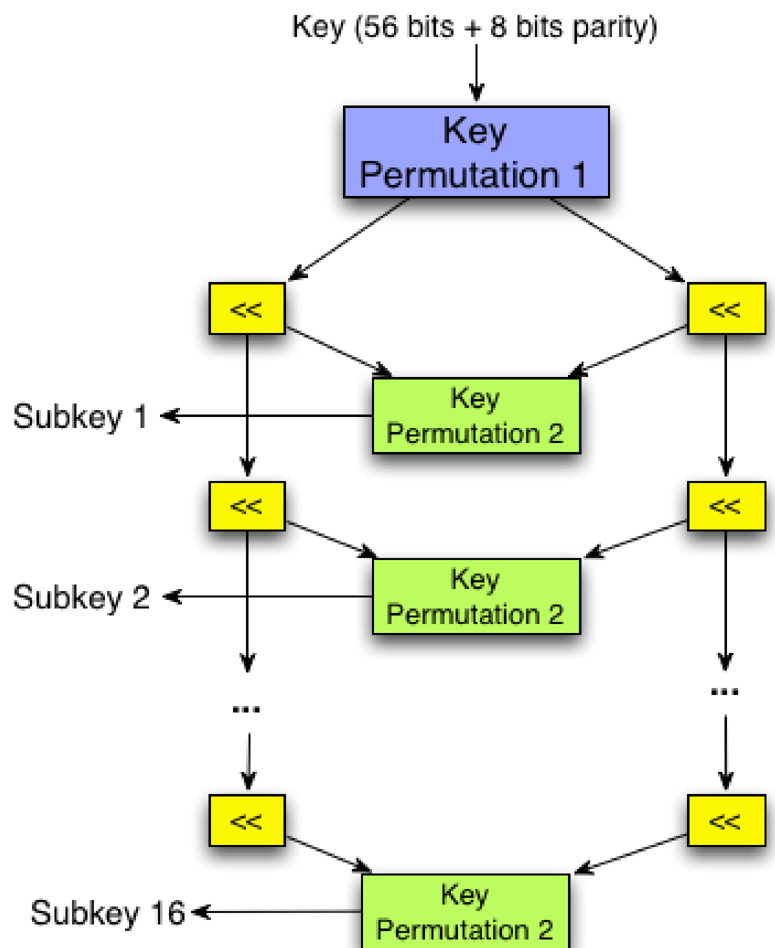
DES

- Key Schedule
  - Key is 56 bits (64 – 8 parity)
- Goes through a permutation before round 1
- Then for each round:
  - divide into two halves
  - circular shift of each half (shift 1 or two bits depending on round)
  - select 48 of the 56 bits



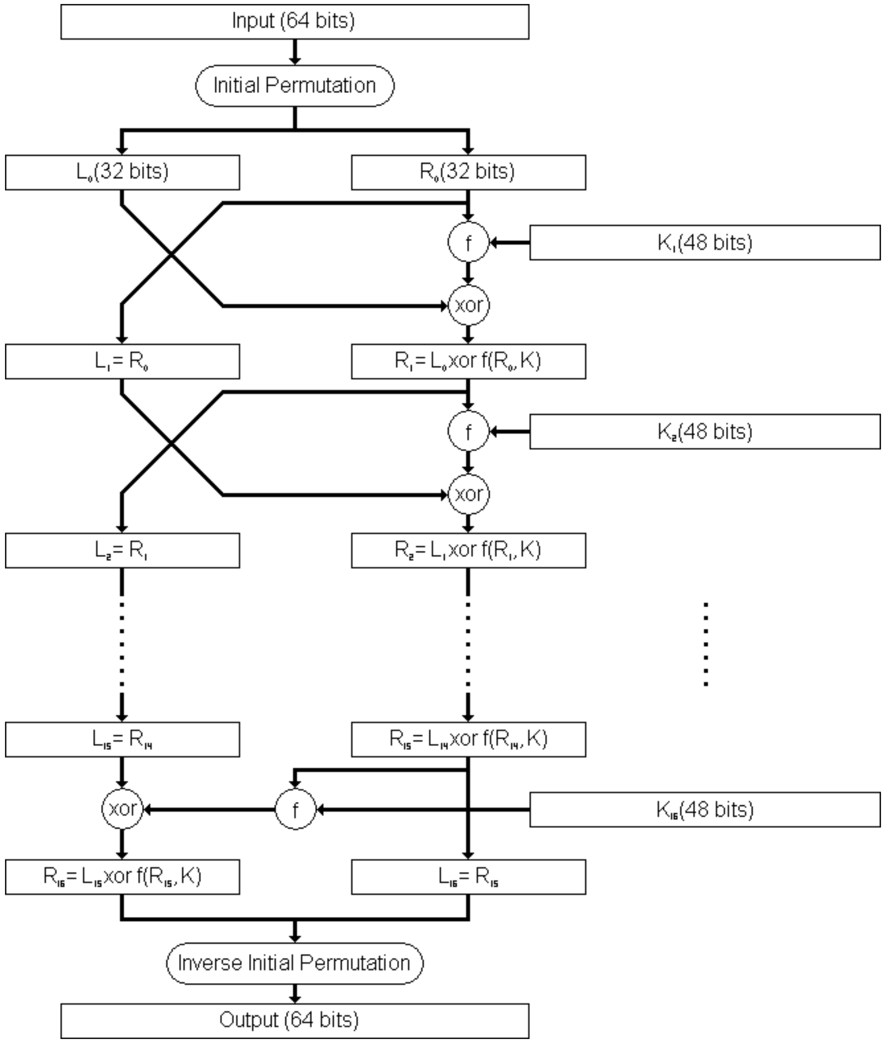
DES

- Key Schedule



DES

- The overall process



DES

- Decryption
  - Same as encryption, but done in reverse
  - E.g. key schedules, etc.

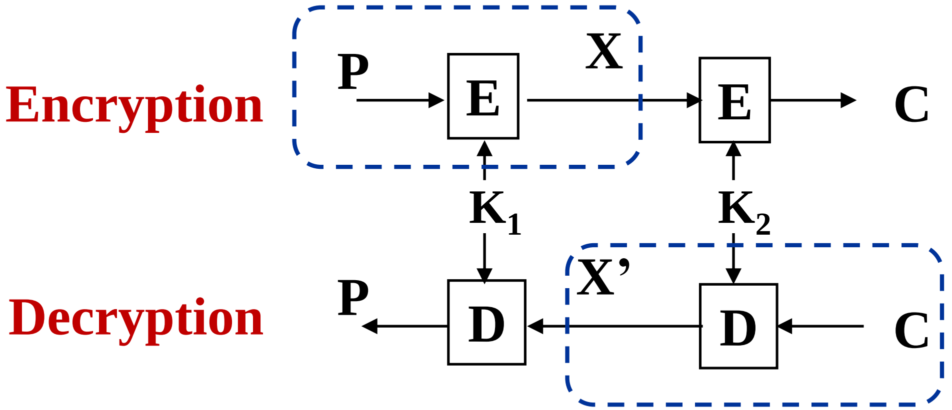
DES

- Strength of DES
  - Key length: 56-bits.
    - Brute force attacks!!
  - DES Challenge: 56-bit-key-encrypted phrase decrypted
    - July 17, 1998, the EFF DES Cracker, which was built for less than $250,000 < 3 days
    - January 19, 1999, Distributed.Net (w/EFF), 22 hours and 15 minutes (over many machines)
    - Now: with commercially available devices: < 1 day
  - We all assume that NSA and agencies like it around the world can crack (recover key) DES in milliseconds

DES

- Multiple Encryption with DES
  - Double DES
  - Encrypt the plaintext twice with two different DES keys
  - Key length increases to 112 bits
  - Not more secure than doing DES
  - Meet-in-the-middle attack

Meet-in-the-middle attack



Observation:

$X = E_{K_1}(P) = D_{K_2}(C)$

- Of course, we don't know K_1 and K_2
  - So we do two parallel exhaustive searches
- For a known pair (P, C),
  - Encrypt P with all 2^{56} possible keys
    - Store the results in a table sorted by the value of X
  - Decrypt C with all 2^{56} possible keys
    - For each result X', check the table for $X = X'$
  - A match reveals a possible combination of keys $< K_1, K_2 >$

DES

- Multiple Encryption with DES
  - Triple DES
  - Encrypt the plaintext three times
  - With two (or three) different DES keys
  - Key length increases to 112 bits (or 168 bits)
  - for each block:
    - encrypt with key 1
    - decrypt with key 2 (this doesn't really decrypt the message!)
    - encrypt with key 1
    - If one key is used, it's equivalent to doing DES once.

AES: Advanced Encryption Standard

- DES cracked, replacement needed
- Triple-DES – slow, has small blocks
- NIST issued call for ciphers in 1997
  - private key symmetric block cipher
  - 128-bit data, 128/192/256-bit keys
  - stronger & faster than Triple-DES
  - provide full specification & design details
  - Secure for next 50-100 years

Advanced Encryption Standard

- NIST have released all submissions & unclassified analyses
  - 15 candidates: 1998
  - 5 finalists: 1999
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin

Advanced Encryption Standard

- Winner: Rijndael
  - Vincent Rijmen and Joan Daemen

> Rijndael. A variant of Square, the chief drawback to this cipher is the difficulty Americans have pronouncing it.
>
> Bruce Schneier

- NIST estimated that a machine that could break a 56-bit DES key in 1 second would take 149 trillion years to crack a 128-bit AES key

AES (Rijndael) Overview

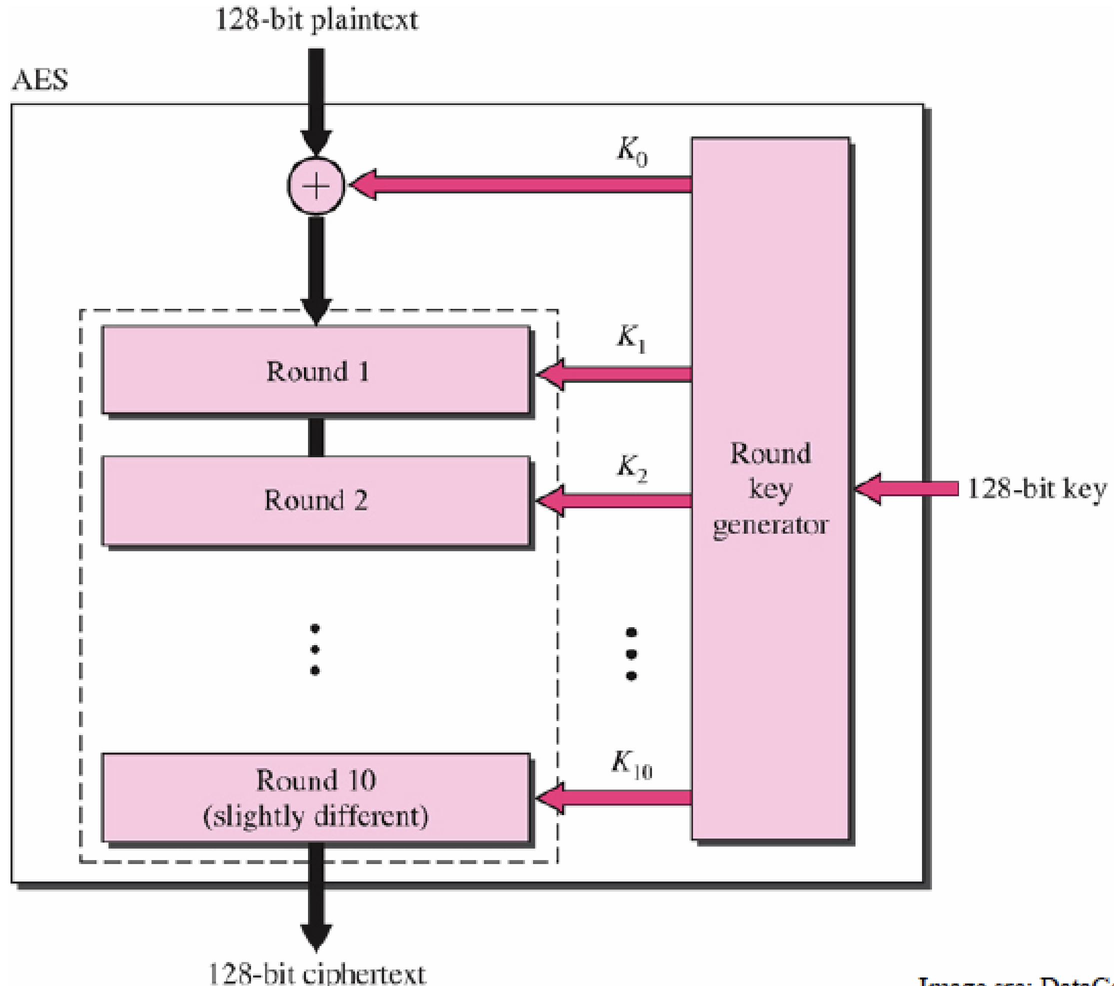


Image src: DataComm

AES (Rijndael) Overview

- Block size: 128 bits
- In each round
  - SubBytes: non-linear byte substitution
  - ShiftRows: circular byte shift in each row
  - MixColumns: add diffusion
  - AddRoundKey

State array

input bytes

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

→

State array

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

→

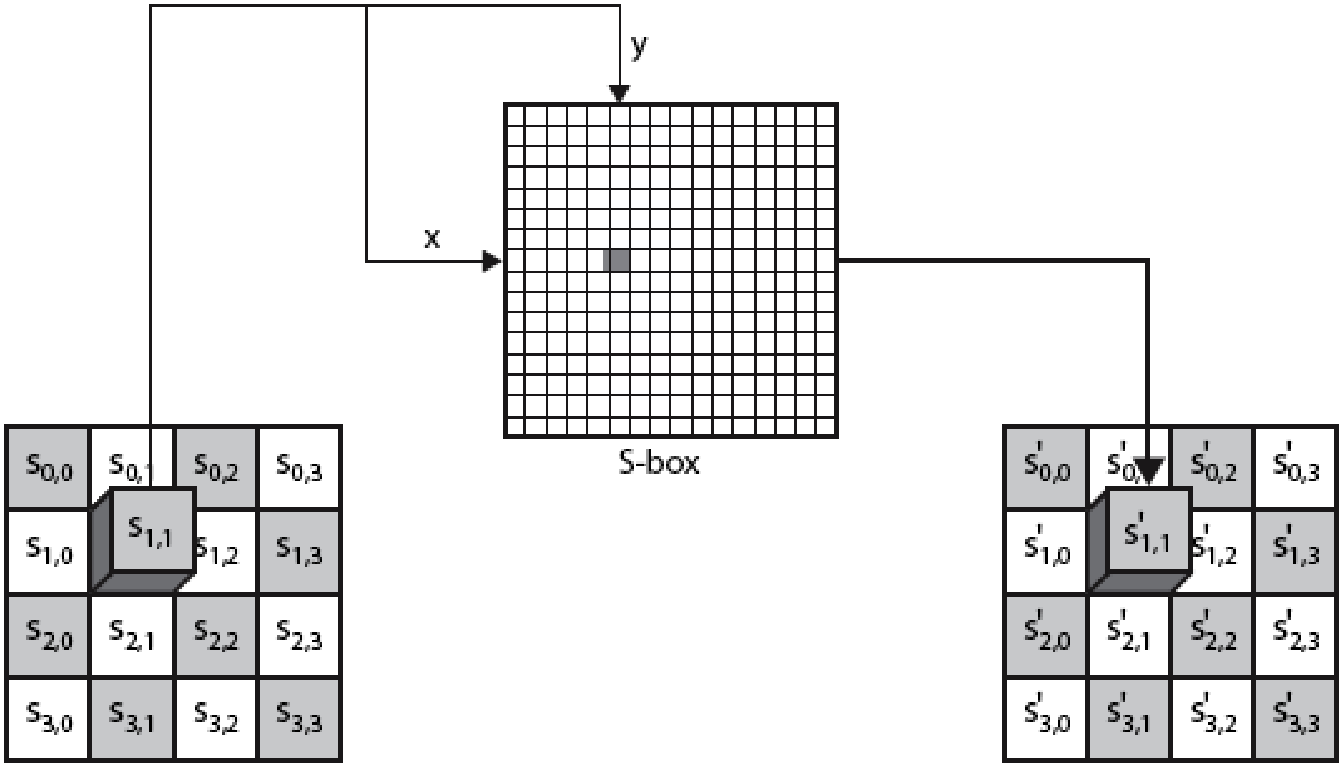
output bytes

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

“State” of machine given by 4x4 array of bytes.

Block size: 128 bits = 16 bytes.

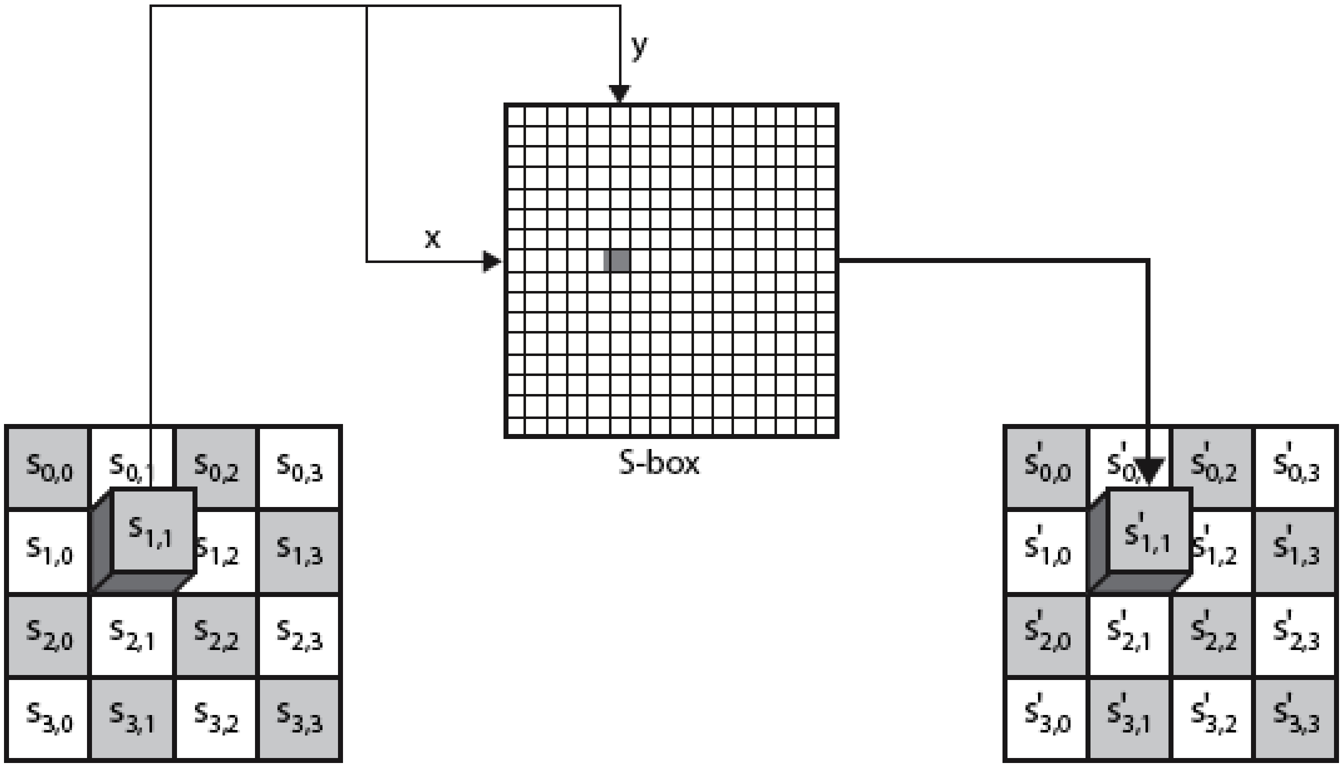
AES: SubBytes



Change each byte of state with corresponding byte from SBOX matrix: SBOX [X,Y]

Non-linear, based on polynomial arithmetic

AES: SubBytes



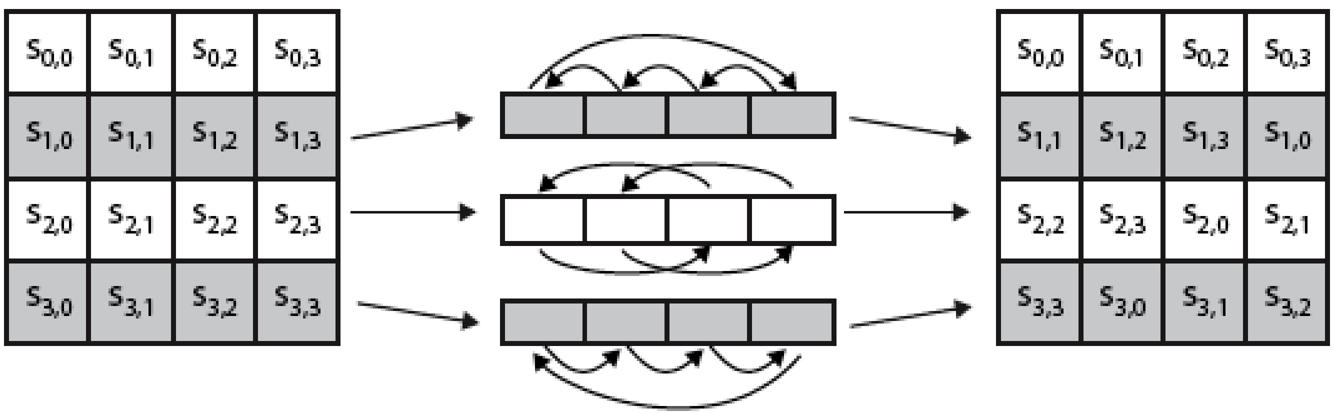
Example:

$S_{1,1}$ = 9A

$S'_{1,1}$ = value at row 9 and column A (10)

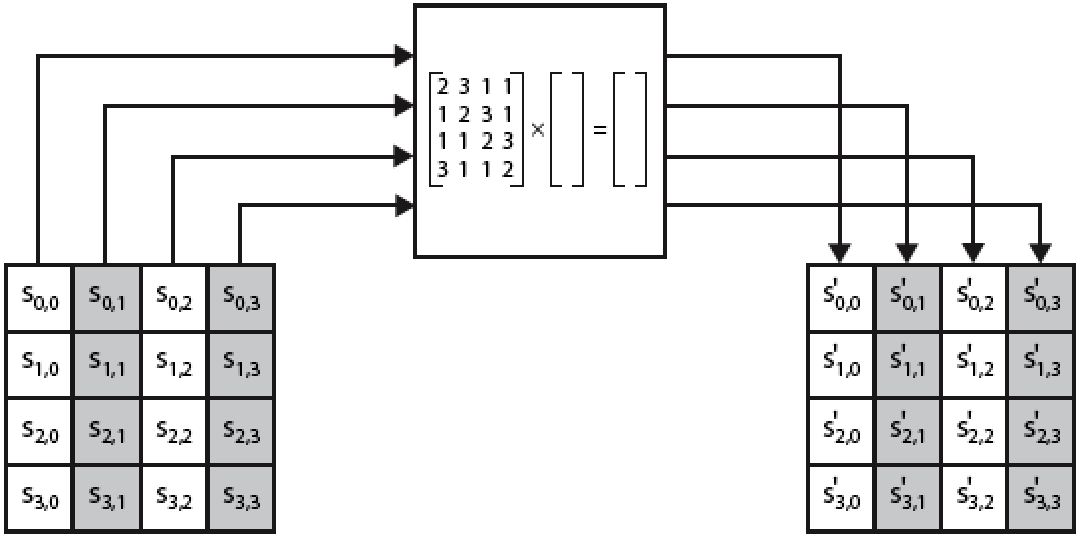
AES: ShiftRows

- 1st row is unchanged
- 2nd row does 1 byte circular shift to left
- 3rd row does 2 byte circular shift to left
- 4th row does 3 byte circular shift to left



AES: MixColumns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column



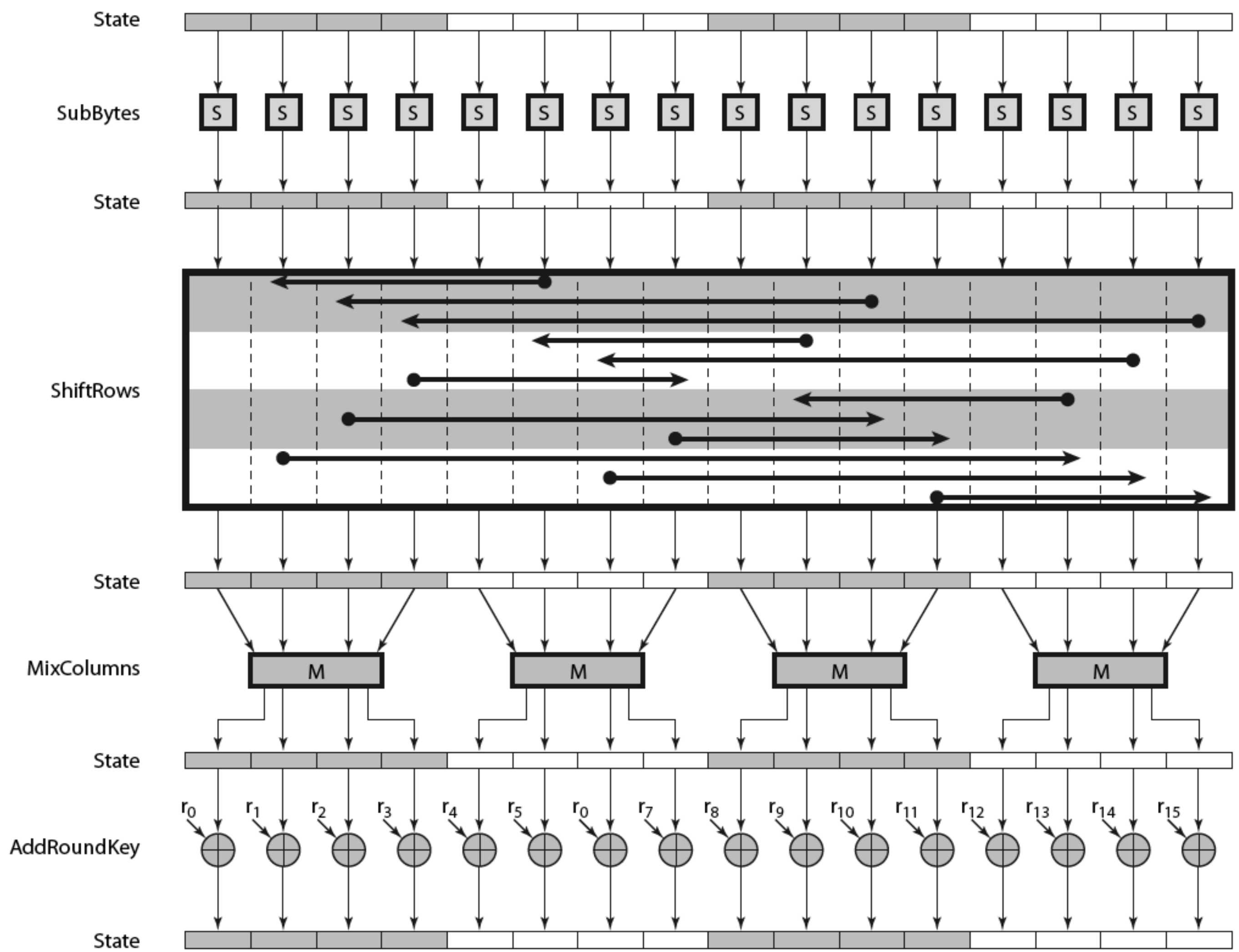
$$S'_{0,0}=2S_{0,0}+3S_{1,0}+1S_{2,0}+1S_{3,0}$$

AES: AddRoundKey

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)

$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \oplus \begin{bmatrix} w_i & w_{i+1} & w_{i+2} & w_{i+3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

AES: the complete round



Secret Key Cryptography

- DES
- AES
  - Secure (at least for now)
  - Efficient
  - Applicable in a wide range of applications

How to encrypt large messages?

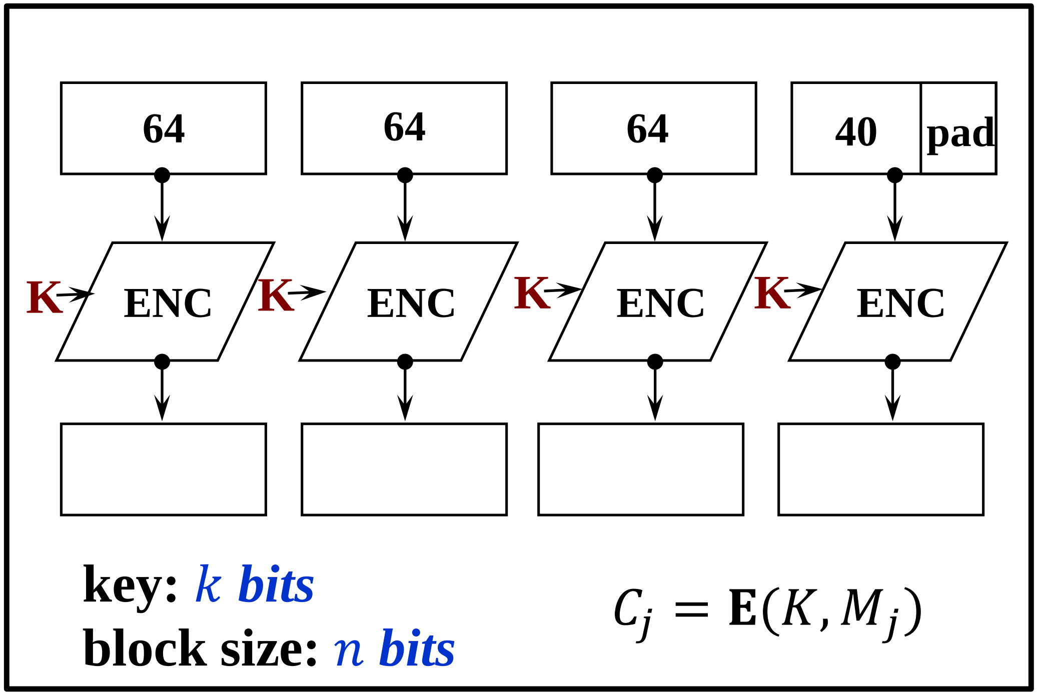
- Use block ciphers
  - Divide a large message into blocks
  - Pad the last block if it is short
    - Use known non-data values and a number to indicate the padding size or the message size
- How to decide the keys for a sequence of data blocks
  - Use the same key
    - A same block in the plaintext results in a same block in the ciphertext
  - Use different keys
    - How to generate and securely transmit a large number of keys?

Modes of Operations

- 4 main modes of operations
  - Electronic Codebook (ECB)
  - Cipher Block Chaining (CBC)
  - Output Feedback (OFB)
  - Cipher Feedback (CFB)
  - Counter mode (CTR)
- Consider properties
  - Robustness to repetitions in message
  - Efficiency
  - Error propagation

Electronic Codebook (ECB)

- Each block is encoded independently using the same key

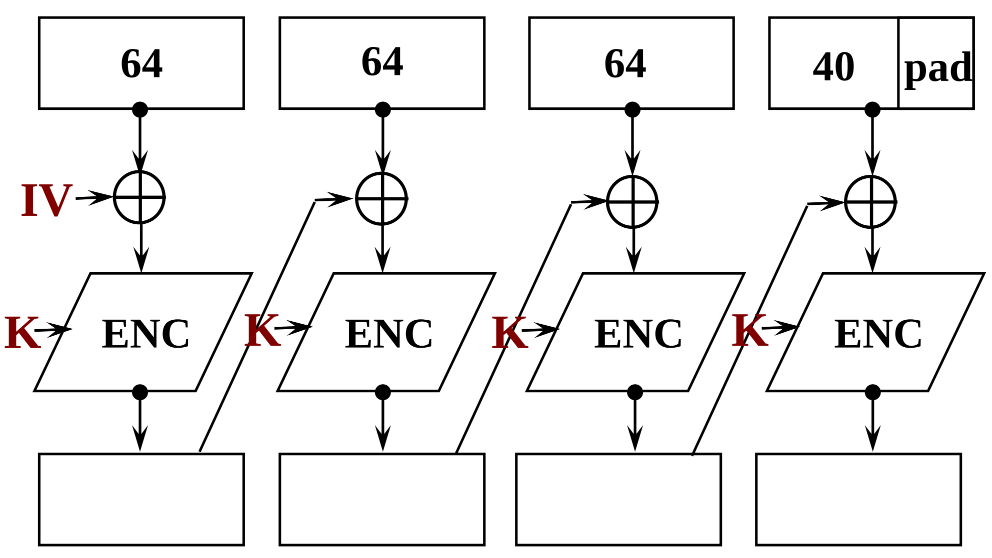


Electronic Codebook (ECB)

- Each block is encoded independently using the same key
  - **Deterministic**
    - Repeated data blocks in plaintext will reveal a pattern
    - E.g., tcp headers, mail headers, etc., long strings of 0's.
  - **No chaining dependency**
    - Reordered ciphertext → reordered plaintext
  - **No error propagation**
    - Error in C_i only results in error in the corresponding P_i
  - Used in secure transmission of a single value
    - Not recommend for encrypting more than 1 data block with the same key

Cipher Block Chaining (CBC)

- Each block is XOR'ed with the preceding ciphertext block



key: *k bits*

block size: *n bits*

$$C_1 = \mathbf{E}(K, M_1 \oplus IV)$$

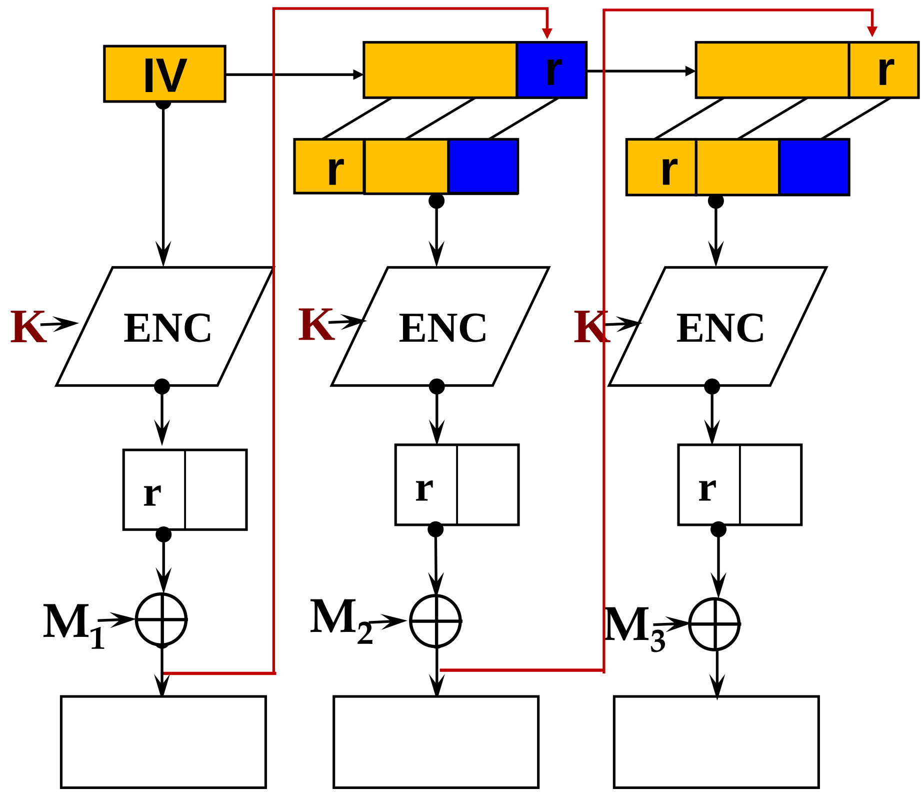
$$C_j = \mathbf{E}(K, M_j \oplus C_{j-1})$$

53

Cipher Block Chaining (CBC)

- Each block is XOR'ed with the preceding ciphertext block
  - **Randomized**
    - Repeated data blocks are encrypted differently
    - Secure if IV is random
  - **Chaining dependent**
    - Reorder affects decryption
  - **Error propagates**
    - Error in 1 ciphertext block propagates to 2 blocks in **decryption**, but no further
  - Used in secure transmission, authentication

Cipher Feedback (CFB)



key: k ***bits*** **block size:** n ***bits***

message segment size: r ***bits***

$$C_1 = M_1 \oplus \mathbf{Left}_r[\mathbf{E}(K, IV)]$$

$$C_j = M_j \oplus \mathbf{Left}_r[\mathbf{E}(K, I_{j-1})]$$

$$I_j = \mathbf{Right}_{n-r}[I_{j-1}] || C_{j-1}$$

Cipher Feedback (CFB)

- Block encryption
  - *n*-bit IV
  - Use the same key to get *n*-bit output
- Leftmost *r* bits of the output
  - Is XORed with a *r*-bit message segment
  - Is fed back to the shift register
- Shift register
  - Shifts left *r* bits
  - Fills the rightmost bits with *r*-bit ciphertext

CFB Properties

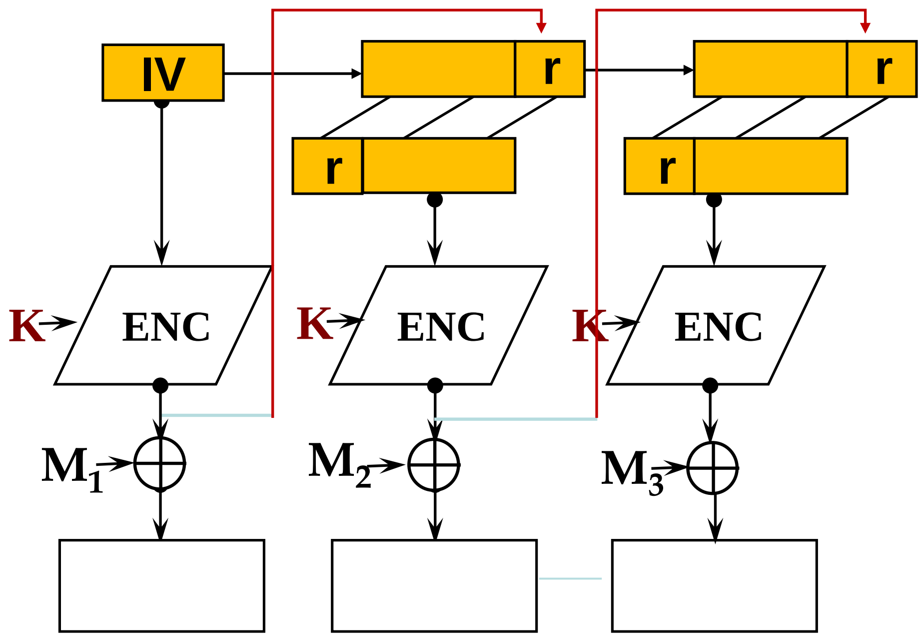
- **Randomized**
  - Repeated data blocks are XORed with different bitstrings
  - Secure if IV is random
- **Chaining dependent**
  - Reorder affects decryption
- **Error propagates**
  - Error in 1 ciphertext block propagates to several blocks in decryption
- Generally used in stream-oriented transmission, authentication

Cipher Feedback (CFB)

- Covert DES into a stream cipher
  - No need to pad the message: ciphertext is of the same length as the plaintext
  - Can operate in real-time
  - Output of the block encryption is used as subkeys of the stream cipher
  - Preceding ciphertext segment forms part of the input of the block encryption
  - A same key is used in the block encryption

Output Feedback (OFB)

- Leftmost r bits of the encryption output is fed back to the shift register



key: k ***bits*** **block size:** n ***bits***

message segment size: r ***bits***

$$C_1 = M_1 \oplus \mathbf{Left}_r[\mathbf{E}(K, IV)]$$

$$C_j = M_j \oplus \mathbf{Left}_r[\mathbf{E}(K, I_{j-1})]$$

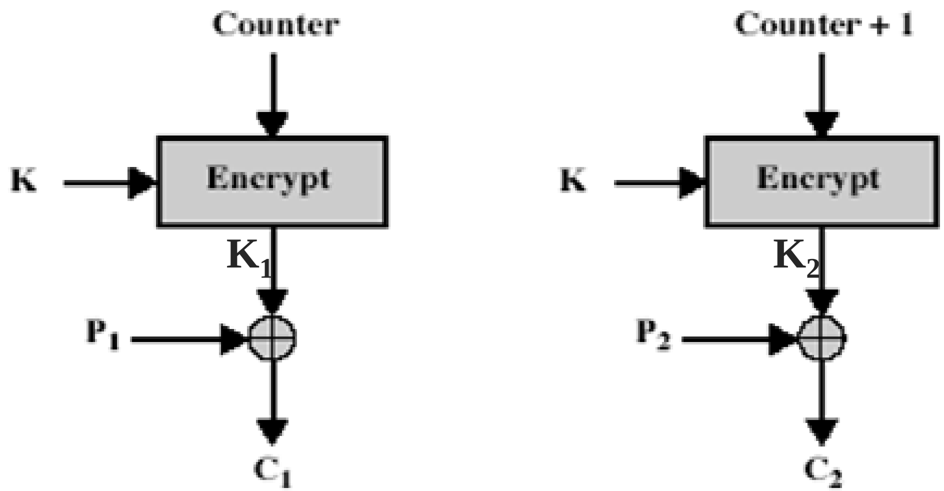
$$I_j = \mathbf{Right}_{n-r}[I_{j-1}] || \mathbf{Left}_r[\mathbf{E}(K, I_{j-1})]$$

OFB Properties

- **Randomized**
  - Repeated data blocks encrypted with different keys
  - Secure if IV is random
- **Chaining independent**
  - Reorder does not affect decryption
  - Key stream is plaintext-independent: allow pre-computing of pseudo-random stream
  - **Throughput:** r varies
- **No error propagation**
  - Preceding ciphertext is not involved in later encryption
- Used in stream-oriented transmission over noisy channel (satellite communication)

Counter Mode (CTR)

- Use a counter equal to the plaintext block size to construct key streams
  - No chaining, pre-computing the key, very efficient
  - Used to encrypt high-speed data, or to generate random bitstreams (PRNG)



key: ***k bits***

block size: ***n bits***

$$counter_1 = IV$$
$$counter_j = IV + j - 1$$
$$C_j = P_j \oplus \mathbf{E}(K, counter_j)$$

Secret Key Cryptography

- Two difficult problems associated with the secret-key cryptosystem:

- How to provide non-repudiation?
  - Need to uniquely identify an entity

Secret Key Cryptography

- Two difficult problems associated with the secret-key cryptosystem:

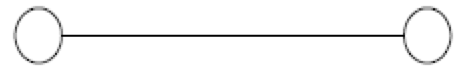
- How to securely distribute secret keys?
  - Which key to use? How to obtain the key securely?
  - Pre-load keys are used in many applications, e.g., at sensor nodes
  - However, risk exists if keys are stolen
  - We need to pre-load many keys…

Secret Key Cryptography

- How many keys do we need?
- For *n* people

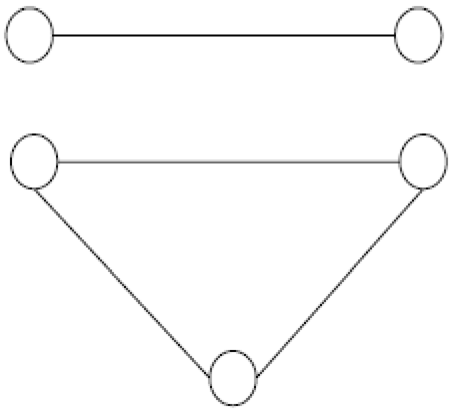
Secret Key Cryptography

- How many keys do we need?
- For *n* people
  - 2 people: 1 key



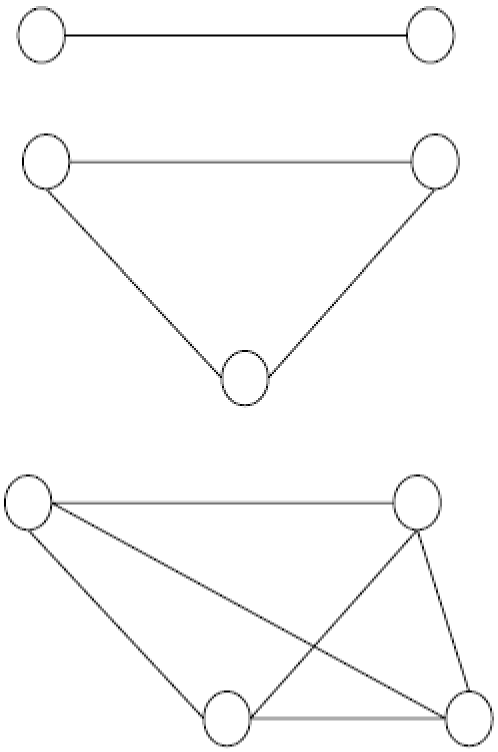
Secret Key Cryptography

- How many keys do we need?
- For *n* people
  - 2 people: 1 key
  - 3 people: 3 keys



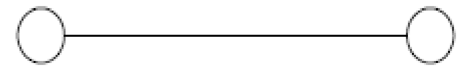
Secret Key Cryptography

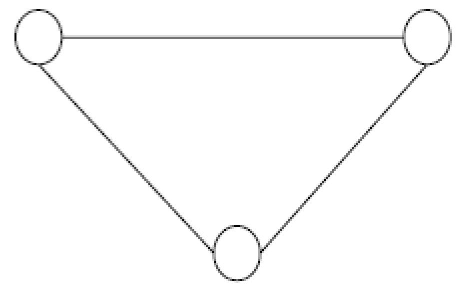
- How many keys do we need?
- For *n* people
  - 2 people: 1 key
  - 3 people: 3 keys
  - 4 people: 6 keys

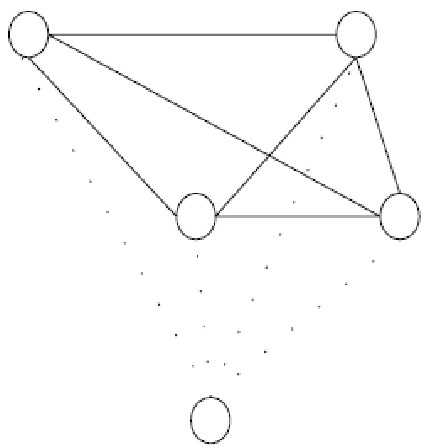


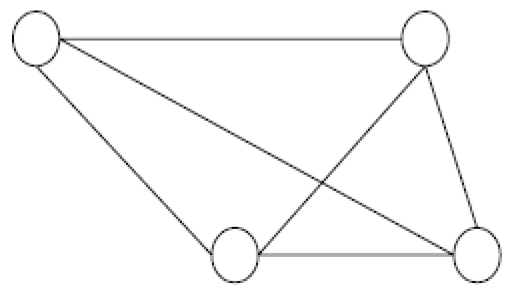
Secret Key Cryptography

- How many keys do we need?
- For *n* people
  - 2 people: 1 key
  - 3 people: 3 keys
  - 4 people: 6 keys
  - 5 people: 10 keys









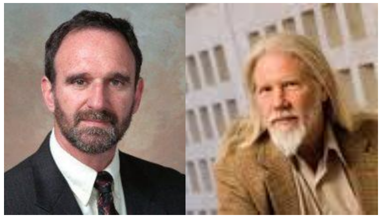
Secret Key Cryptography

- How many keys do we need?
- For *n* people
  - n people: n(n-1)/2 keys
    - $O(n^2)$
    - We don't like anything more than O(n)…
  - Can we ask all *n* people to share the same key?
  - Do we have a better way to generate and distribute keys?

Key Distribution/Agreement

- Key Distribution
  - The process to assign and transfer keys to a participant
- Key Agreement
  - The process whereby two (or more) parties negotiate a key
  - As part of communication: SKIP (Simple Key-Management for Internet Protocol)
- Typically, key distribution/agreement occurs in conjunction with or after authentications.

Diffie-Hellman Key Agreement



- Diffie and Hellman: important breakthrough in 1976,
  - **Started the modern age of cryptography**
  - Enable negotiation of a secret over an *insecure* media
  - **Idea**: participants exchange intractable puzzles that can be solved easily with additional information.
- Mathematics are very deep
  - Working in multiplicative group **G**
  - Use the hardness of computing discrete logarithms in finite field to guarantee secure

Diffie-Hellman Protocol

- For two participants Alice and Bob:
- **Setup**: both agree on a large prime p and a generator $g \in Z_p$
  - Both are public information
  - E.g., $p = 13, g = 4$
- **Step 1**: Each principal picks a private value x_a, x_b $(< p - 1)$, and generates a new value $y_a = g^{x_a} \bmod p$, and $y_b = g^{x_b} \bmod p$
- **Step 2**: Exchange y, and generate the secret shared key $z_a = {y_b}^{x_a} \bmod p$, and $z_b = {y_a}^{x_b} \bmod p$
- **Step 3**: The agreed session key is: $g^{x_a x_b} \bmod p$
- Provide good confidentiality against eavesdropping, however …

Attacks on Diffie-Hellman

- This is a key agreement, not authentication
  - You don't know anything about who you have exchanged keys with
  - Insecure against active attacks, e.g., *man-in-the-middle*
    - Alice and Bob think they are talking directly to each other
    - Mallory is actually performing two separate exchanges

Elementary Cryptography

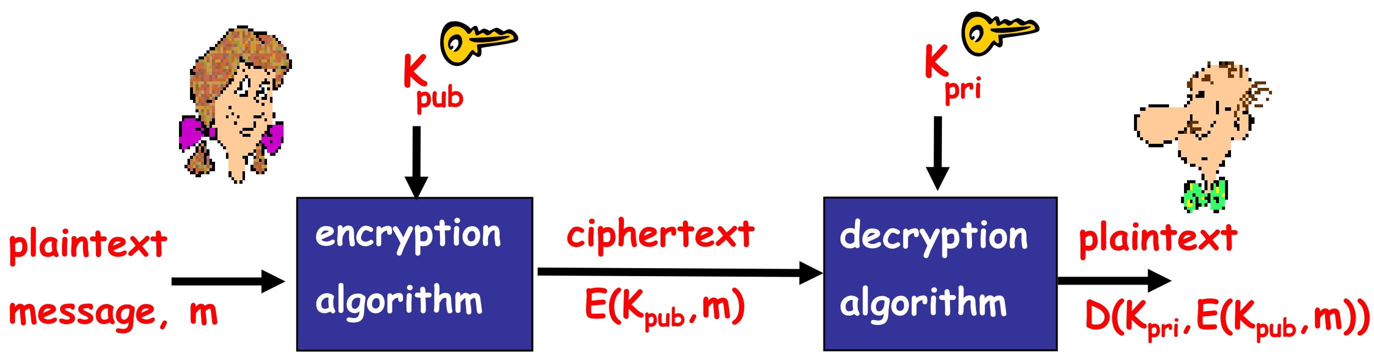
Public Key Cryptography

Asymmetric Encryption

- Public Key Cryptography
  - a.k.a. asymmetric encryption
  - Encryption and decryption with different keys
  - Bob has a pair of public and private keys
  - Bob's public key is known by Alice
    - Alice uses Bob’s public key to encrypt the message

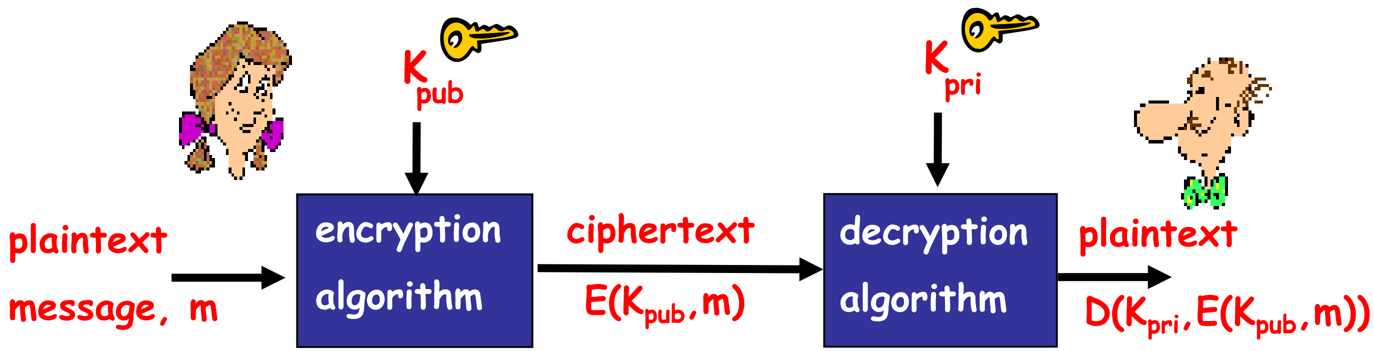
$$C = E(K_{pub}, P)$$

$$P = D(K_{pri}, C) = D(K_{pri}, E(Kpub, P))$$



Asymmetric Encryption

- Public Key Cryptography
  - Public key: anyone can know
  - Private key: only known to the owner
- The keys are inverses of each other:
  - Anything encrypted with your public key can only be decrypted with your private key; it cannot be decrypted by your public key!
  - Anything encrypted with your private key can only be decrypted with your public key; it cannot be decrypted with your private key!!

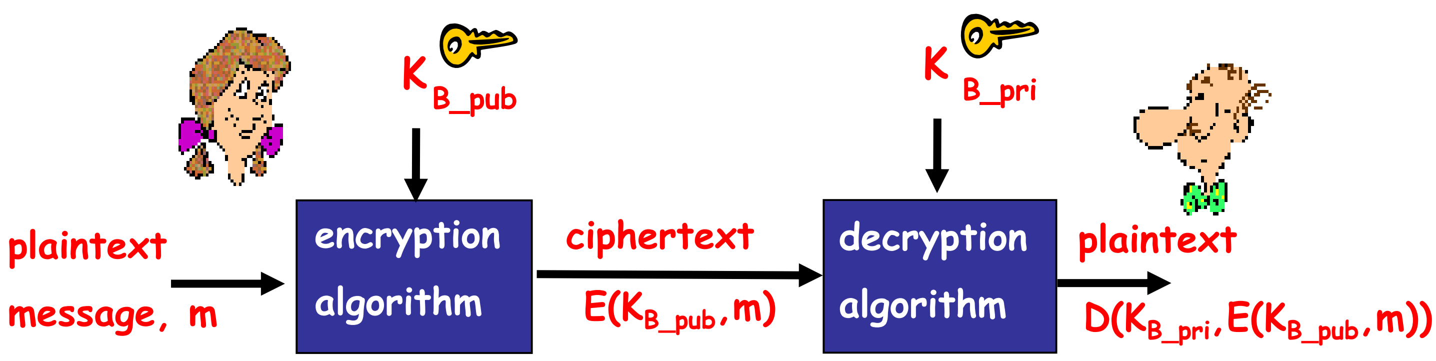


Asymmetric Encryption

- What can we do with it?
  - Encryption: keep your data secret
  - Authentication: you are who you say you are
  - Integrity: the message has not been changed

Asymmetric Encryption

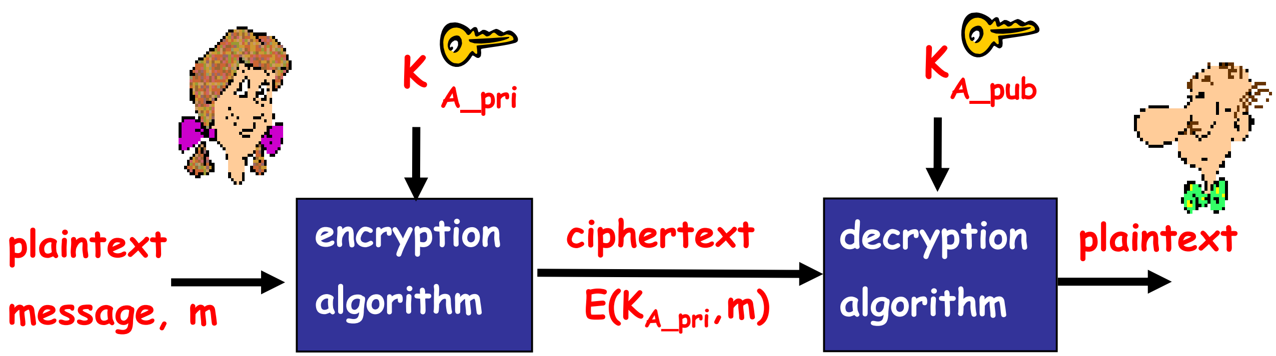
- Encryption: keep your data secret
- Alice wants to send a message to Bob
  - Only Bob should be able to read it
  - Alice encrypts the message with Bob's public key.
  - Bob decrypts the ciphertext with his private key.



Asymmetric Encryption

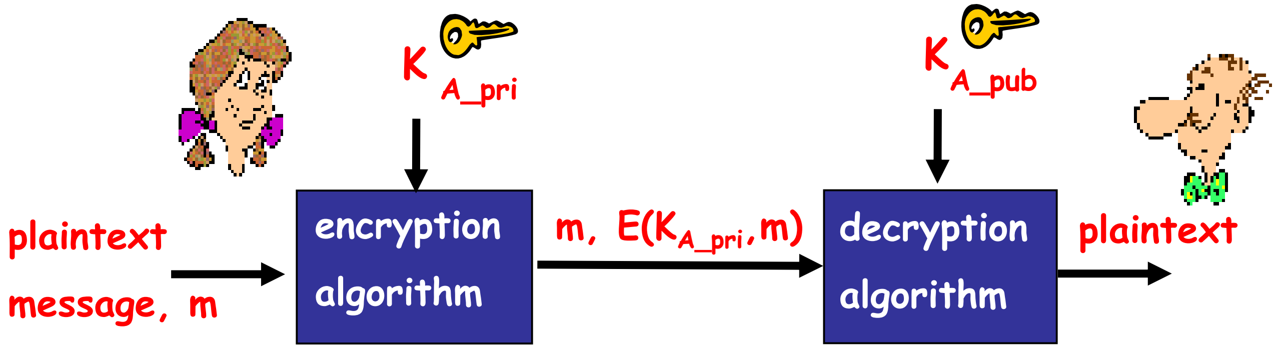
- Authentication: you are who you say you are
- Digital signatures
  - Should work like handwritten signatures: verify the sender of the document
  - Alice sends a message to Bob
  - How can Alice prove that she is the real sender?
  - Alice sends the message encrypted with her private key
  - Bob decrypts with Alice's public key.

如何知道发送者的身份：本人加密信息用私钥，和其他人解密用公钥



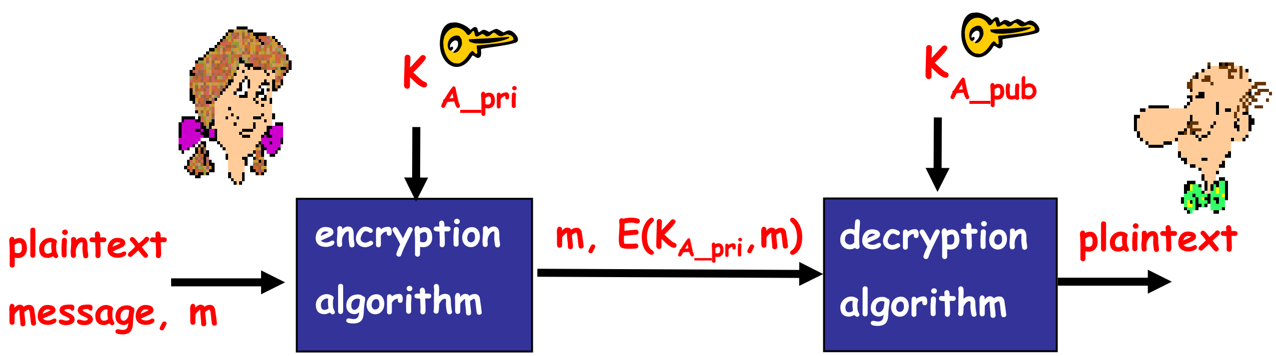
Asymmetric Encryption

- Authentication: you are who you say you are
- Digital signatures
  - Could also send two copies:
  - One clear
  - One encrypted with Alice's private key
  - Why?



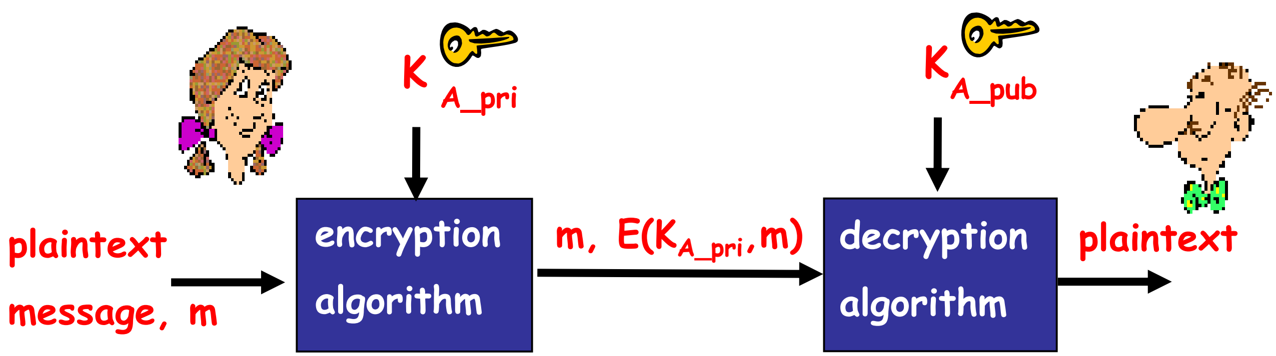
Asymmetric Encryption

- Authentication: you are who you say you are
- Digital signatures
  - Why?
    - You can still read the message without decryption.



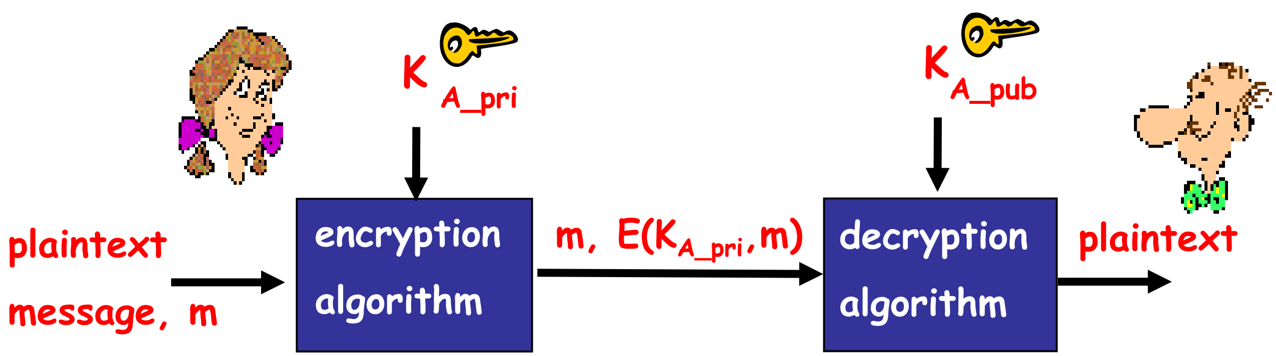
Asymmetric Encryption

- Authentication: you are who you say you are
- Digital signatures
  - Why?
    - You can still read the message without decryption.
  - Problem with it?



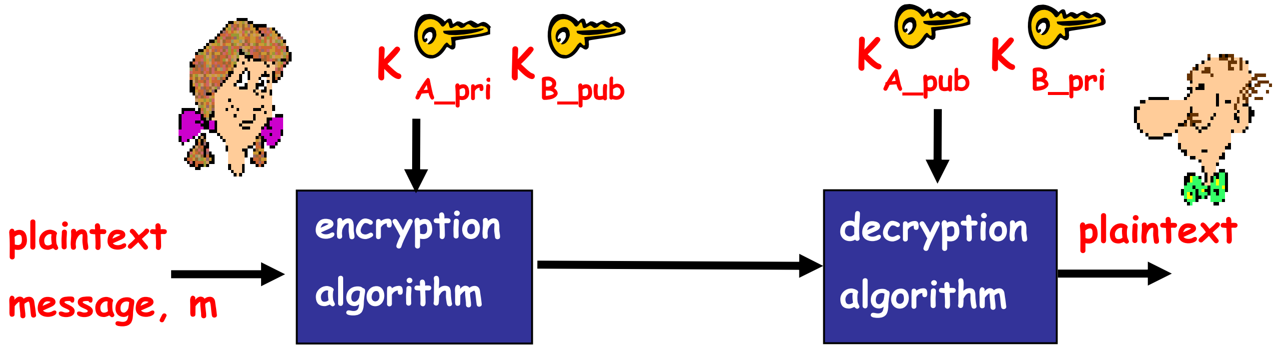
Asymmetric Encryption

- Authentication: you are who you say you are
- Digital signatures
  - Why?
    - You can still read the message without decryption.
  - Problem with it?
    - The size of the message is doubled.
  - Solution? No need to encrypt the entire message!
    - Just a "digest" of the message.



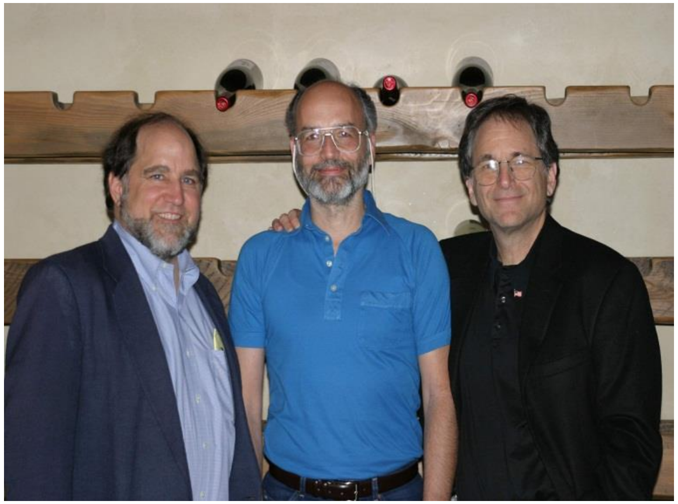
Asymmetric Encryption

- How to provide both confidentiality and and authenticity?
  - Alice both signs and encrypts the message
  - Could either:
  - $E_{A\_pri}(E_{B\_pub}(M))$
  - or
  - $E_{B\_pub}(E_{A\_pri}(M))$



RSA 三个人名字的首字母

- Diffie-Hellman key exchange (1976)
- Rivest (MIT), Shamir (Weizmann Institute), and Adleman (USC) published RSA asymmetric encryption scheme in 1978
  - 2002 Turing Award



- British Government Communications Headquarters
  - James Ellis proposed “non-secret encryption” in 1970 (made public by in 1997)
  - Clifford Cocks proposed basic ideas as RSA in 1973
  - Malcolm Williamson developed key distribution scheme similar to Diffie-Hellman key exchange in 1974

RSA

- RSA
  - Key generation
  - Encryption
  - Decryption

RSA

- RSA Key generation
  - Select two large primes p and q; (p != q)
  - Calculate $n = pq$
  - Calculate $\varphi(n) = (p-1)(q-1)$
    - Euler's totient function.
  - Select a random integer e, $1<e<\varphi(n)$, and e is relatively prime to $\varphi(n)$: $\gcd(e, \varphi(n))=1$
  - Compute d, $1<d<\varphi(n)$, and $d \equiv e^{-1} \bmod \varphi(n)$
    - $de \equiv 1 \bmod \varphi(n)$

Public key: $<e, n>$

Private key: $<d, n>$

Note: p, q, and $\varphi(n)$ should be thrown away!

RSA

- RSA Key generation
  - Calculate $\varphi(n) = (p - 1)(q - 1)$
    - Euler's totient function.
  - $\varphi(n)$: number of positive integers less than or equal to n that are relatively prime to n.
  - When $n=pq$, and both p and q are prime numbers: $\varphi(n)=(p-1)(q-1)$
  - When n is large, it's hard to compute $\varphi(n)$ for an arbitrary n.
    - No easier than factoring n

RSA

- RSA Encryption
  - Given: message m, $0<m<n$, public key $<e, n>$
  - Compute $c = m^e \bmod n$

- RSA Decryption
  - Given: ciphertext c, and private key $<d, n>$
  - Compute $m = c^d \bmod n$
  - Actually: $c^d \bmod n = m \bmod n$

RSA

- Why RSA works?
- Decryption: $c^d = (m^e)^d = m^{ed}$
- In key generation:

  $d \equiv e^{-1} \bmod \varphi(n)$ ➔ $de \equiv 1 \bmod \varphi(n)$ ➔ $de = k\varphi(n)+1$
- Hence,

$$c^d = m^{ed} = m^{k\varphi(n)+1} = m*(m^{\varphi(n)})^k$$

- Euler's theorem (the Fermat–Euler theorem or Euler's totient theorem):

$$m^{\varphi(n)} \equiv 1 \bmod n$$

- Therefore

$$m*(m^{\varphi(n)})^k \equiv m*(1)^k \equiv m \quad (\text{mod n})$$

RSA

- Select two “large” primes p and q; (p != q)
  - p = 17, q = 13

RSA

- Select two "large" primes p and q; (p != q)
  - p = 17, q = 13
- Calculate $n = pq$
  - n = 221

RSA

- Select two "large" primes p and q; (p != q)
  - p = 17, q = 13
- Calculate $n = pq$
  - n = 221
- Calculate $\varphi(n) = (p-1)(q-1)$
  - 16*12 = 192

RSA

- Select two "large" primes p and q; (p != q)
  - p = 17, q = 13
- Calculate $n = pq$
  - n = 221
- Calculate $\varphi(n) = (p-1)(q-1)$
  - 16*12 = 192
- Select a random integer ***e***, $1<e<\varphi(n)$, and e is relatively prime to $\varphi(n)$: gcd(e, φ(n))=1
  - Choose 11

gcd: greatest common divisor: 最大公因素

RSA

- Select two "large" primes p and q; (p != q)
  - p = 17, q = 13
- Calculate $n = pq$
  - n = 221
- Calculate $\varphi(n) = (p-1)(q-1)$
  - 16*12 = 192
- Select a random integer $\boldsymbol{e}$, $1<e<\varphi(n)$, and e is relatively prime to $\varphi(n)$: gcd(e, φ(n))=1
  - Choose 11
- Compute $\boldsymbol{d}$, $1<d<\varphi(n)$, and $d \equiv e^{-1} \bmod \varphi(n)$
  - Pick 35: 35*11 = 385 = 2*192+1

d = (2*192+1)/11 = 35

RSA

- Select two "large" primes p and q; (p != q)
  - p = 17, q = 13
- Calculate $n = pq$
  - n = 221
- Calculate $\varphi(n) = (p-1)(q-1)$
  - 16*12 = 192
- Select a random integer $\boldsymbol{e}$, $1<e<\varphi(n)$, and e is relatively prime to $\varphi(n)$: gcd(e, φ(n))=1
  - Choose 11
- Compute $\boldsymbol{d}$, $1<d<\varphi(n)$, and $d \equiv e^{-1} \bmod \varphi(n)$
  - Pick 35: 35*11 = 385 = 2*192+1

Public key: <*e*, *n*> = <11, 221>

Private key: <*d*, *n*> = <35, 221>

RSA

- Encrypt: “MAIL”
  - MAIL = {12, 0, 8, 11}
    - Message needs to be converted to numeric type
  - Public key: $<e, n> = <11, 221>$
  - $c = m^e \bmod n = m^{11} \bmod 221$
  - $c = \{142, 0, 70, 97\}$
- Decrypt {142, 0, 70, 97}
  - $m = c^d \bmod n = c^{35} \bmod 221$

RSA

- Cryptanalysis
- RSA is thought to be secure because:
  - to find d (inverse of e mod $\varphi(n)$)
    - need to know $\varphi(n)$
  - given n it's very difficult to find $\varphi(n)$
    - thought to be no easier than factoring n
    - Quantum computers and Shor’s algorithm?
- *Note*: when p and q are 100 decimal digits
  - n is about 200 decimal digits
  - millions of years of computer time needed to factor

RSA

- Cryptanalysis
  - Textbook RSA vs. Public-Key Cryptography Standards (PKCS)
  - 2003: Timing attack on OpenSSL
    - Problem with protocol, not really RSA itself.
  - 2012: "*Ron is wrong, Whit is right*"
    - A paper by Arjen Lenstra, James Hughes, *et al*
    - "Public keys are shared among unrelated parties"
    - 12,720 out of 6.4 million certificates "offer no security".
    - Again, problem with RSA implementation.

RSA

- Problems with RSA
  - Key distribution still a problem
    - Proving to whom a key belongs
    - How does Bob know if the public key really belongs to Alice?
    - How do you know if you are using the public key of Chase Bank, not “Cheat Bank”?
    - Mallory could hand you a public key and claim it Alice’s…

RSA

- Problems with RSA
  - Key distribution still a problem
  - Slow
    - Look at the operations!
    - keys must be much longer than symmetric keys to provide the same degree of security
    - RSA – size of message to be encrypted is limited by *n*.

RSA

- Hybrid scheme (public + session key)
  - Public key crypto is slow
  - Symmetric key is fast but key distribution problem
  - solution:
    - Create a symmetric key called *session key*
    - Encrypt the data with the *session key*
    - Encrypt the *session key* with the receiver's *public key*

Elementary Cryptography

Data Integrity

Message Authentication Codes

- Small block appended to message for authentication

  $MAC = C_K(M)$
  – C mac function
  – K shared secret key
  – M message

- The block is called
  – *cryptographic checksum* or
  – *Message Authentication Code* (MAC)
- MACs verify
  – that the message came from A
  – that message has not been altered

Cryptographic hash functions

- Hash functions take a message as input
  - Message may be of any length
  - Output is string/number of fixed length
- Sometimes called *message digest* functions
- Hash result: called *digest* or *fingerprint*
- Cryptographic hashes ≠ function used in hash tables
  - Cryptographic features!

Cryptographic hash functions

- Cryptographic hashes ≠ function used in hash tables
- Cryptographic hashes are one way:
  - Given *M*, it's easy to compute *H(M)*
  - Given H(M), should be very difficult to produce M
  - or any M' where H(M') = H(M)
    - "Collision"
  - Implies uniform distribution of hash values
- Example cryptographic hashes:
  - MD5 – 128 bits
  - SHA1 – 160 bits

Cryptographic hash functions

- Hash Functions for Authentication
  - Can we use a hash function as an authenticator?
    - Just send M + H(M)
    - No: Bad guy will send M' + H(M')
    - Again: the hash function (crypto algorithm) is public
  - Try this:
    - Send: $M + E_{A\_pri}(H(M))$
    - Or: H(secret + M)

Cryptographic hash functions

- Hash Functions for Authentication
  - Can we use a hash function as an authenticator?
    - Just send M + H(M)
    - No: Bad guy will send M' + H(M')
    - Again: the hash function (crypto algorithm) is public
  - Try this:
    - Send: $M + E_{A\_pri}(H(M))$
    - Or: H(secret + M)

    - Can I use Bob’s public key?
    - Send: $M + E_{B\_pub}(H(M))$

Cryptographic hash functions

Not in the test

- Collision Resistance
  - Hash: many-to-one, never collision free
  - “Birthday paradox”
  - After a number of attempts, adversary may find a collision
  - How many attempts?
  - Weak Collision Resistance: given H(m), it’s difficult to find m’ such that H(m’) = H(m).
  - Strong Collision Resistance: computationally infeasible to find m1, m2 such that H(m1) = H(m2).