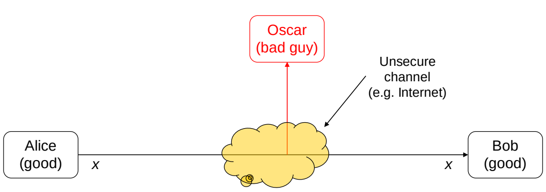
Symmetric Cryptography

Symmetric Cryptography

Alternative names: **private-key**, **single-key** or **secret-key** cryptography.



Problem Statement:

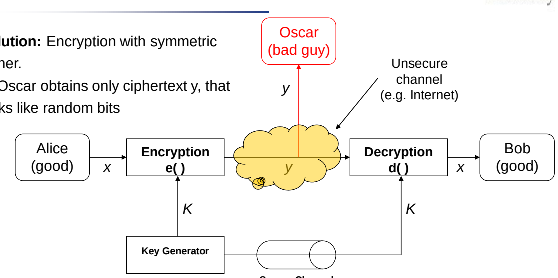
1) Alice and Bob would like to communicate via an unsecure channel (e.g., WLAN or Internet).

2) A malicious third party Oscar (the bad guy) has channel access but should not be able to understand the communication.

Symmetric Cryptography (cont.)

Solution: Encryption with symmetric cipher.

⇒ Oscar obtains only ciphertext y, that looks like random bits



- x is the **plaintext**
- y is the **ciphertext**
- *K* is the **key**
- Set of all keys {*K*1, *K*2, ...,*Kn*} is the **key space**

Symmetric Cryptography (cont.)

- Encryption equation **$y = e_K(x)$**
- Decryption equation **$x = d_K(y)$**

- Encryption and decryption are inverse operations if the same key K is used on both sides:

$$d_K(y) = d_K(e_K(x)) = x$$

- Important: The key must be transmitted via a **secure channel** between Alice and Bob.
- The secure channel can be realized, e.g., by manually installing the key for the Wi-Fi Protected Access (WPA) protocol or a human courier.
- However, the system is only secure if an attacker does not learn the key K!

⇒ **The problem of secure communication is reduced to secure transmission and storage of the key K.**

Brute-Force Attack against Symmetric Cipher

- Treats the cipher as a black box
- Requires (at least) 1 plaintext-ciphertext pair (x_0, y_0)
- Check all possible keys until condition is fulfilled:

$$d_K(y_0) \stackrel{?}{=} x_0$$

- How many keys should we need ?

Key length in bit	Key space	Security life time (assuming brute-force as best possible attack)
64	2^{64}	**Short term** (few days or less)
128	2^{128}	**Long-term** (several decades in the absence of quantum computers)
256	2^{256}	**Long-term** (also resistant against quantum computers – note that QC do not exist at the moment)

Important: An adversary only needs to succeed with **one** attack. Thus, a long key space does not help if other attacks (e.g., social engineering) are possible..

Substitution Cipher

Substitution Cipher

Idea: *replace each plaintext letter by a fixed other letter.*

Plaintext		Ciphertext
A	→	k
B	→	d
C	→	w

for instance, ABBA would be encrypted as kddk

- Example (ciphertext):

iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc hwwhbsqvqbre hwq vhlq

- How secure is the Substitution Cipher? Let‘s look at attacks…

Attacks against Substitution Cipher

Attack: Exhaustive Key Search (Brute-Force Attack)

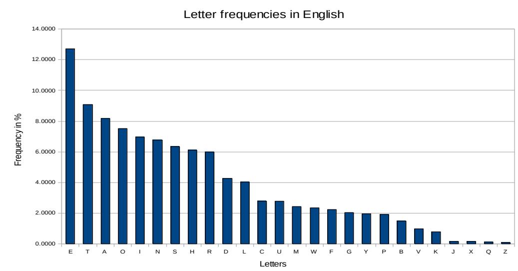
- Simply try every possible subsititution table until an intelligent plaintext appears (note that each substitution table is a key)..
- How many substitution tables (= keys) are there?

  $26 \times 25 \times \ldots \times 3 \times 2 \times 1 = 26! \approx 2^{88}$

  Search through 2^{88} keys is completely infeasible with today's computers! (cf. earlier table on key lengths)
- Q: Can we now conclude that the substitution cipher is secure since a brute-forece attack is not feasible?
- A: No! We have to protect against **all** possible attacks…

Letter Frequency Analysis

- Letters have very different frequencies in the English language
- Moreover: the frequency of plaintext letters is preserved in the ciphertext.



Letter Frequency Attack

- Let's return to our example and identify the most frequent letter:

```
   iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc
hwwhbsqvqbre hwq vhlq
```

- We replace the ciphertext letter q by E and obtain:

```
   iE ifcc vEEr fb rdE vfllcE na rdE cfjwhwz hr bnnb
hcc hwwhbsEvEbre hwE vhlE
```

- By further guessing based on the frequency of the remaining letters we obtain the plaintext:

```
   WE WILL MEET IN THE MIDDLE OF THE LIBRARY AT NOON ALL
ARRANGEMENTS ARE MADE
```


Letter Frequency Attack (cont.)

- In practice, not only frequencies of individual letters can be used for an attack, but also the frequency of letter pairs (i.e., „th" is very common in English), letter triples, etc.
- **Important lesson**: Even though the substitution cipher has a sufficiently large key space of appr. 2^{88}, it can easily be defeated with analytical methods. This is an excellent example that an encryption scheme must withstand all types of attacks.

Shift (or Caesar) Cipher and Affine Cipher

Shift (or Caesar) Cipher (1)

- Ancient cipher, allegedly used by Julius Caesar
- Replaces each plaintext letter by another one.
- Replacement rule is very simple: Take letter that follows after *k* positions in the alphabet

Shift (or Caesar) Cipher (2)

Needs mapping from letters → numbers:

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

Example for $k = 7$

Plaintext = ATTACK = 0, 19, 19, 0, 2, 10

Ciphertext = haahjr = 7, 0, 0, 7, 9, 17

Note that the letters ”wrap around” at the end of the alphabet, which can be mathematically be expressed as reduction modulo 26, e.g., $19 + 7 = 26 \equiv 0 \bmod 26$

Shift (or Caesar) Cipher (3)

- Elegant mathematical description of the cipher.

Let k, x, y in {0,1, …, 25}
- Encryption: $y = e_k(x) \equiv x + k \bmod 26$
- Decryption: $x = d_k(x) \equiv y - k \bmod 26$

- Q: Is the shift cipher secure?
- A: No! several attacks are possible, including:
  - Exhaustive key search (key space is only 26!)
  - Letter frequency analysis, similar to attack against substitution cipher

Affine Cipher

- Extension of the shift cipher: rather than just adding the key to the plaintext, we also multiply by the key
- We use for this a key consisting of two parts: k = (a, b)

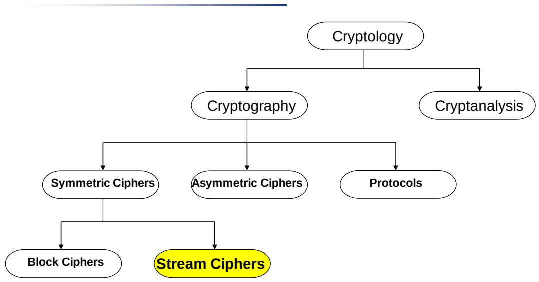
  Let k, x, y in {0,1, …, 25}
  - Encryption: $y = e_k(x) \equiv a\,x + b \bmod 26$
  - Decryption: $x = d_k(x) \equiv a^{-1}(y - b) \bmod 26$
- Since the inverse of *a* is needed for inversion, we can only use values for *a* for which:

  $$\gcd(a, 26) = 1$$

  There are 12 values for *a* that fulfill this condition. $aa^{-1} = 1 \bmod 26$
- From this follows that the key space is only 12 x 26 = 312
- Again, several attacks are possible, including: exhaustive key search and letter frequency analysis, similar to the attack against the substitution cipher

Stream Ciphers

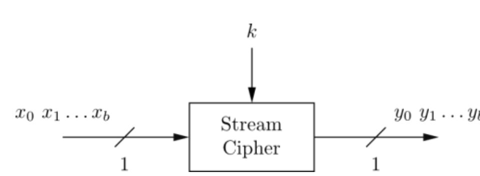
Stream Ciphers in Cryptology



Stream Ciphers were invented in 1917 by Gilbert Vernam

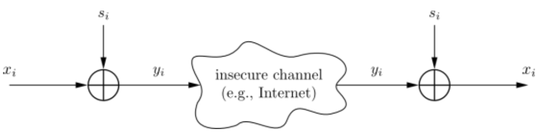
Key Idea of Stream Ciphers

- Encrypt bits individually
- Usually small and fast → common in embedded devices (e.g., A5/1 for GSM phones)



Encryption & Decryption

Plaintext x_i, ciphertext y_i and key stream s_i consist of individual bits



- Encryption and decryption are simple additions modulo 2 (aka XOR)
- Encryption and decryption are the same functions

Encryption: $y_i = e_{si}(x_i) = x_i + s_i \bmod 2,\ x_i, y_i, s_i \in \{0,1\}$
Decryption: $x_i = e_{si}(y_i) = y_i + s_i \bmod 2$

Why is Modulo 2 Addition?

- Modulo 2 addition is equivalent to XOR operation
- For perfectly random key stream s_i, each ciphertext output bit has a 50% chance to be 0 or 1
  → Good statistic property for ciphertext
- Inverting XOR is simple, since it is the same XOR operation

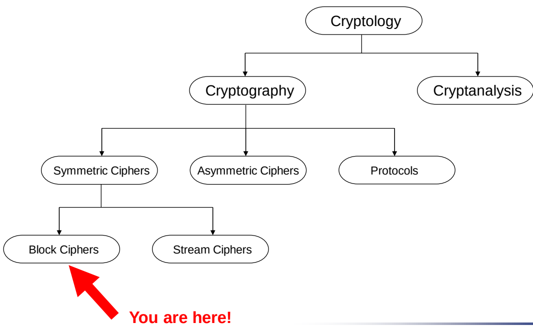
x_i	s_i	y_i
0	0	0
0	1	1
1	0	1
1	1	0

Example

- Encryption of the letter *A* by Alice
- *A* is given in ASCII code as $65_{10} = 1000001_2$.
- Let's assume that the first key stream bits are
  $\Rightarrow z_1, \ldots, z_7 = 0101101$
- Encryption by Alice:
  - plaintext x_i: 1000001 = *A* (ASCII symbol)
  - key stream z_i: 0101101
  - ciphertext y_i: 1101100 = *l* (ASCII symbol)
- Decryption by Bob:
  - ciphertext y_i: 1101100 = *l* (ASCII symbol)
  - key stream z_i: 0101101
  - plaintext x_i: 1000001= *A* (ASCII symbol)

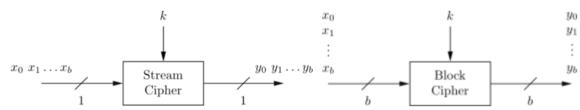
Data Encryption Standard (DES)

DES in the Field of Cryptology



Block Cipher vs. Stream Cipher

- Stream Ciphers
  - Encrypt bits individually
  - Usually small and fast → common in embedded devices
- Block Ciphers:
  - Always encrypt a full block (several bits)
  - Are common for Internet applications



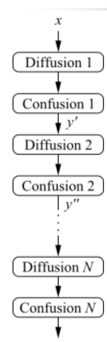
Block Cipher: Confusion & Diffusion

Claude Shannon: There are two **primitive operations** with which strong encryption algorithms can be built:

- **Confusion:** An encryption operation where the **relationship between key and ciphertext is obscured**.
  Today, a common element for achieving confusion is **substitution**, which is found in both DES and AES.
- **Diffusion:** An encryption operation where the **influence of one plaintext symbol is spread over many ciphertext symbols** with the goal of hiding statistical properties of the plaintext.
  A simple diffusion element is the **bit permutation**, which is frequently used within DES.

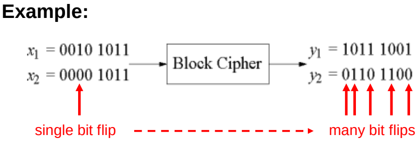
Both operations by themselves cannot provide security. The idea is to concatenate confusion and diffusion elements to build so called *product ciphers.*

Product Ciphers



- Most of today's block ciphers are *product ciphers* as they consist of rounds which are applied repeatedly to the data.
- Can reach excellent diffusion: **changing of one bit of plaintext results** *on average* in the **change of half the output bits**.

Example:

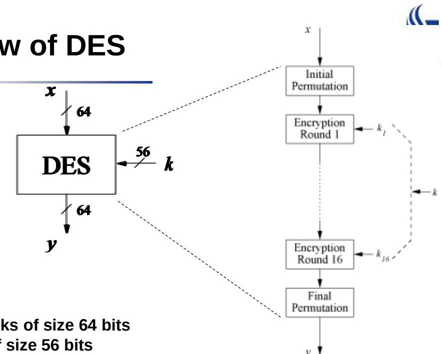


DES Facts

- Data Encryption Standard (DES) encrypts **blocks of size 64 bit**.
- Developed by **IBM** based on the cipher *Lucifer* under influence of the *National Security Agency* (NSA), the design criteria for DES have not been published
- **Standardized 1977** by the **National Bureau of Standards** (NBS) today called *National Institute of Standards and Technology* (NIST)
- Most popular **block cipher** for most of the last 30 years.
- By far best studied symmetric algorithm.
- Nowadays considered insecure due to the small **key length of 56 bit.**
- **But: 3DES yields very secure cipher**, still widely used today.
- Replaced by the *Advanced Encryption Standard* (**AES**) in 2000

29

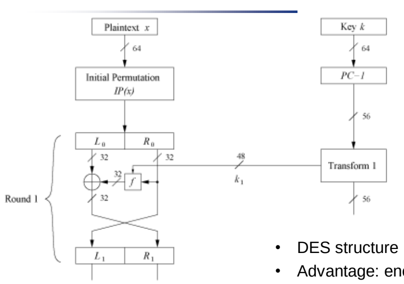
Overview of DES



- Encrypts **blocks of size 64 bits**
- Uses a **key of size 56 bits**
- **Symmetric cipher**: uses same key for encryption and decryption
- Uses **16 rounds** which all perform the identical operation
- **Different subkey** in each round derived from main key

30

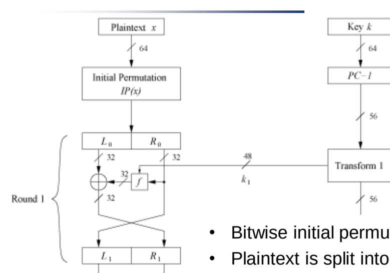
DES Feistel Network (1)



- DES structure is a *Feistel network*
- Advantage: encryption and decryption differ only in keyschedule

31

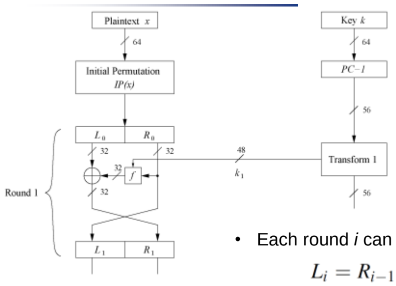
DES Feistel Network (2)



- Bitwise initial permutation, then 16 rounds
- Plaintext is split into 32-bit halves L_i and R_i
- R_i is fed into the function f, the output of which is then XORed with L_i
- Left and right half are swapped

32

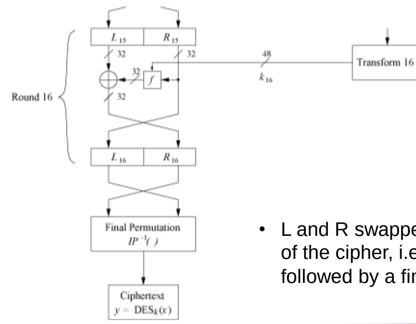
DES Feistel Network (3)



- Each round i can be expressed as:

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

DES Feistel Network (4)



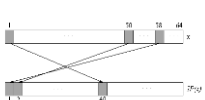
- L and R swapped again at the end of the cipher, i.e., after round 16 followed by a final permutation

Initial and Final Permutation

- Bitwise Permutations.
- Inverse operations.
- Described by tables *IP* and *IP*$^{-1}$.

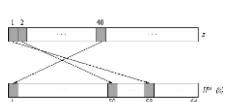
Initial Permutation

IP							
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



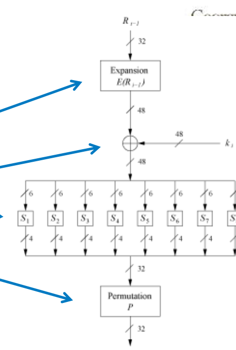
Final Permutation

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



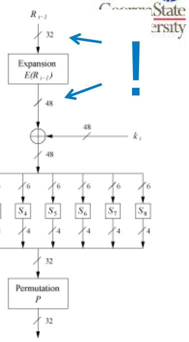
The f-Function

- **main operation of DES**
- *f*-Function inputs: R_{i-1} and round key k_i
- **4 Steps**:
  1. Expansion *E*
  2. XOR with round key
  3. S-box substitution
  4. Permutation

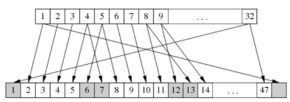


The Expansion Function E

1. Expansion *E*
   – main purpose: increases diffusion



E					
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

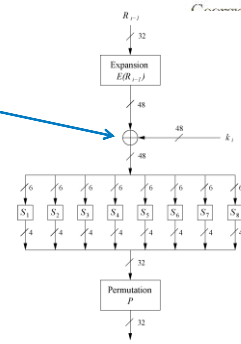


Add Round Key

2. XOR Round Key

– Bitwise XOR of the round key and the output of the expansion function *E*

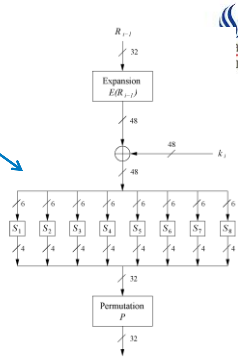
– Round keys are derived from the main key in the DES keyschedule (in a few slides)

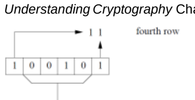


The DES S-Boxes

3. S-Box substitution

- Eight substitution tables.
- 6 bits of input, 4 bits of output.
- Non-linear and resistant to differential cryptanalysis.
- Crucial element for DES security!
- Find all S-Box tables and S-Box design criteria in *Understanding Cryptography* Chapter 3.



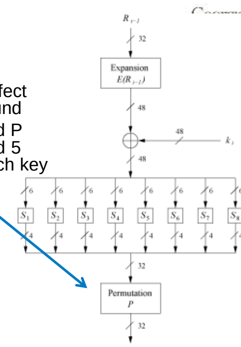


S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

The Permutation P

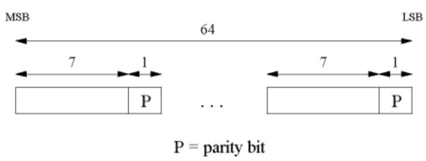
4. Permutation P
   – Bitwise permutation.
   – Introduces diffusion.
   – Output bits of one S-Box effect several S-Boxes in next round
   – Diffusion by E, S-Boxes and P guarantees that after Round 5 every bit is a function of each key bit and each plaintext bit.

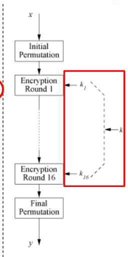
P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25



Key Schedule (1)

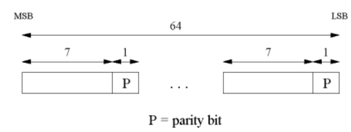
- Derives 16 round keys (or *subkeys*) k_i of 48 bits each from the original 56 bit key.
- The input key size of the DES is 64 bit: **56 bit key** and 8 bit parity:





Key Schedule (2)

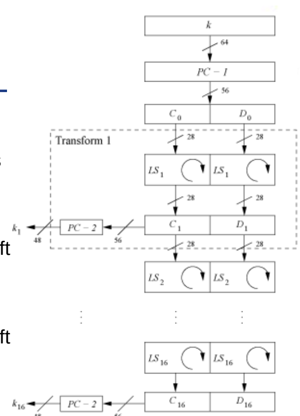
- **Parity bits are removed** in a first **permuted choice** *PC-1* (note that the bits 8, 16, 24, 32, 40, 48, 56 and 64 are not used at all)



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

Key Schedule (3)

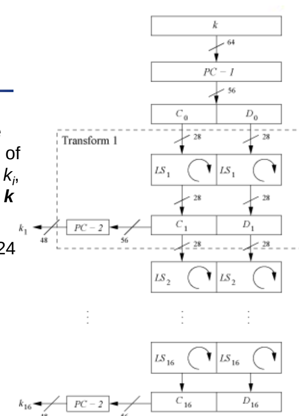
- Split key into two 28-bit halves C_0 and D_0
- In **rounds i = 1, 2, 9 ,16,** the two halves are each rotated left by **one bit**.
- In **all other rounds** where the two halves are each rotated left by **two bits**.



Key Schedule (4)

- In each round i permuted choice ***PC-2*** selects a permuted subset of 48 bits of C_i and D_i as round key k_i, i.e. **each k_i is a permutation of k**
- Select 24 bits from left half and 24 bits from right

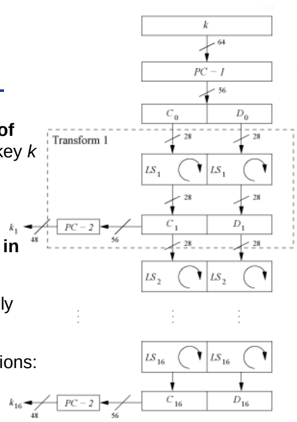
$PC-2$							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32



Key Schedule (5)

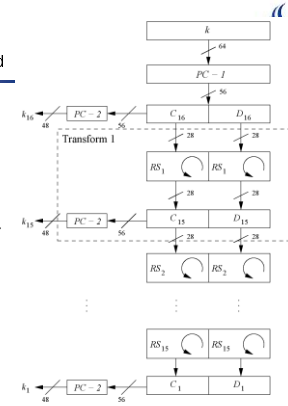
- Each round key is a **selection of 48 permuted bits** of the input key *k*
- Key schedule realizes **16 permutations** systematically
- Each of the 56 key bits is **used in different round**
- Each bit is used in approximately **14 of the 16 round** keys
- **Note:** The total number of rotations: 4 x 1 + 12 x 2 = 28

→ $D_0 = D_{16}$ and $C_0 = C_{16}$



Decryption

- In **Feistel ciphers** only the keyschedule has to be modified for decryption.
- Generate the same 16 round keys in reverse order. (for a detailed discussion on why this works see *Understanding Crptography* Chapter 3)
- **Reversed key schedule:** As $D_0=D_{16}$ and $C_0=C_{16}$ the first round key can be generated by applying *PC-2* right after *PC-1* (no rotation here!). All other rotations of *C* and *D* can be reversed to reproduce the other round keys resulting in:
  - No rotation in round 1.
  - One bit rotation **to the right** in rounds 2, 9 and 16.
  - Two bit rotations **to the right** in all other rounds.



Security of DES

- **After proposal of DES two major criticisms arose:**
  1. Key space is too small (2^{56} keys)
  2. S-box design criteria have been kept secret: Are there any hidden analytical attacks (*backdoors*), only known to the NSA?
- **Analytical Attacks:** DES is highly resistent to both *differential* and *linear cryptanalysis*, which have been published years later than the DES. This means IBM and NSA had been aware of these attacks for 15 years! So far there is no known analytical attack which breaks DES in realistic scenarios.
- **Exhaustive key search:** For a given pair of plaintext-ciphertext (*x*, *y*) test all 2^{56} keys until the condition $DES_k^{-1}(x)=y$ is fulfilled.
  ⇒ Relatively easy given today's computer technology!

Advanced Encryption Standard (AES)

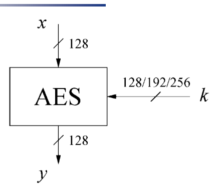
Some Basic Facts

- DES is not secure
- AES is the most widely used symmetric cipher today
- The algorithm for AES was chosen by the US National Institute of Standards and Technology (NIST) in a multi-year selection process
- The requirements for all AES candidate submissions were:
  - Block cipher with **128-bit block size**
  - Three supported key lengths: 128, 192 and 256 bit
  - Security relative to other submitted algorithms
  - Efficiency in software and hardware

Chronology of the AES Selection

- The need for a new block cipher announced by NIST in January, 1997
- 15 candidates algorithms accepted in August, 1998
- 5 finalists announced in August, 1999:
  - *Mars* – IBM Corporation
  - *RC6* – *RSA* Laboratories
  - *Rijndael* – J. Daemen & V. Rijmen
  - *Serpent* – Eli Biham et al.
  - *Twofish* – B. Schneier et al.
- In October 2000, Rijndael was chosen as the AES
- AES was formally approved as a US federal standard in November 2001

AES: Overview (1)

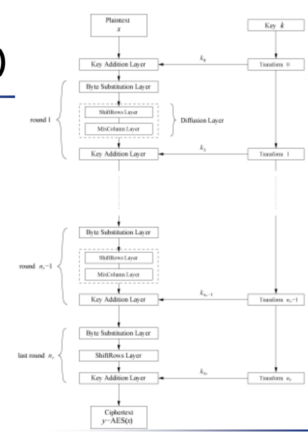


The number of rounds depends on the chosen key length:

Key length (bits)	Number of rounds
128	10
192	12
256	14

AES: Overview (2)

- Iterated cipher with 10/12/14 rounds
- The whole block is encrpted in each round
- Each round consists of **"Layers"**



Internal Structure of AES (1)

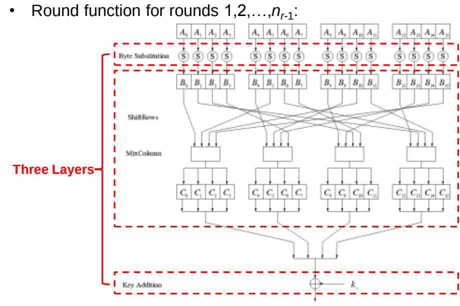
- AES is a byte-oriented cipher (1 byte = 8 bits)
- The state A (i.e., the 128-bit data path) can be arranged in a 4x4 matrix:

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

with $A_0, \ldots, A_{15}$ denoting the 16-byte input of AES

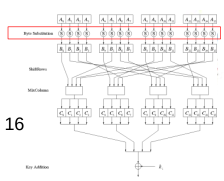
Internal Structure of AES (2)

- Round function for rounds $1,2,\ldots,n_{r-1}$:



- **Note:** In the last round, the MixColumn tansformation layer is omitted

Byte Substitution Layer



- The Byte Substitution layer consists of 16 **S-Boxes** with the following properties: the S-Boxes are
  - **Identical:** ByteSub(A_i) = B_i
  - Only **nonlinear** elements of AES, i.e., ByteSub(A_i) + ByteSub(A_j) ≠ ByteSub($A_i + A_j$), for $i,j = 0,\ldots,15$
  - **bijective**, i.e., there exists a one-to-one mapping of input and output bytes
    $\Rightarrow$ S-Box can be uniquely reversed
- In software implementations, the S-Box is usually realized as a lookup table

Example: S-Boxes

- Realize S-Box as a 256-by-8 bit lookup table
- Substitution values are in hexadecimal form for input byte (xy)

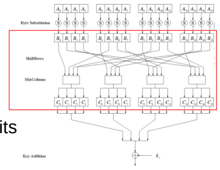
x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Hexadecimal

For example,
$A_i = (11000010)_2$
$= (C2)_{hex}$

$S(A_i) = S((C2)_{hex})$
$= (25)_{hex}$
$= (00100101)_2$
$= B_i$

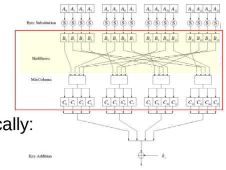
Diffusion Layer



- Provides diffusion over all input state bits
- Consists of two sublayers:
  - **ShiftRows Sublayer**: Permutation of the data on a byte level
  - **MixColumn Sublayer**: Matrix operation which combines ("mixes") blocks of four bytes
- Performs a linear operation on state matrices *A*, *B*, i.e.,

  DIFF(*A*) + DIFF(*B*) = DIFF(*A* + *B*)

Diffusion Layer: ShiftRows Sublayer



Rows of the state matrix are shifted cyclically:

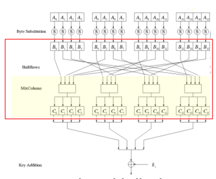
Input matrix

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

Output matrix

B_0	B_4	B_8	B_{12}	no shift
B_5	B_9	B_{13}	B_1	← one position left shift
B_{10}	B_{14}	B_2	B_6	← two positions left shift
B_{15}	B_3	B_7	B_{11}	← three positions left shift

Diffusion Layer: MixColumn Sublayer



- Linear transformation which mixes each column of the state matrix
- Each 4-byte column is considered as a vector and multiplied by a fixed 4x4 matrix, e.g.,

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

  where 01, 02 and 03 are given in hexadecimal notation
- All arithmetic is done in the Galois field $GF(2^8)$, containing 256 elements

Key Schedule (1)

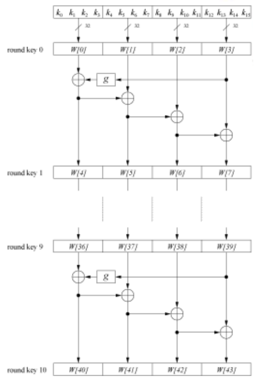
- Subkeys are derived recursively from the original 128/192/256-bit input key
- Each round has 1 subkey, plus 1 subkey at the beginning of AES

Key length (bits)	Number of rounds	Number of subkeys
128	10	11
192	12	13
256	14	15

- Key whitening: Subkey is used both at the input and output of AES
  ⇒ # subkeys = # rounds + 1
- There are different key schedules for the different key sizes

Key Schedule (2)

- Word-oriented: 1 word = 32 bits
- The length of k_i is 1 byte (i.e., 8 bits)
- Each k_i is derived recursively based on k_{i-1}
- 11 subkeys are stored inthe key expansion array *W[0]…W[3], W[4]…W[7], … , W[40]…W[43]*
- Subkey of round 0, *W[0]…W[3],* is the original AES key



Example: Key schedule for 128-bit key AES

61

Key Schedule (3)

- Leftmost word of a subkey at round i, W[4i]:

  W[4i] = W[4(i-1)] ⊕ g(W[4i - 1])

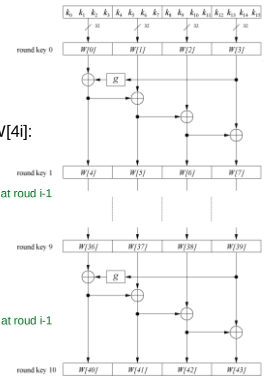
  leftmost word of subkey at roud i-1 rightmost word of subkey at roud i-1

  – g() is a nonlineaer function
  – i = 1, 2, 3, …, 10
- Remaining three words of a subkey:

  W[4i+j] = W[4i + j - 1] ⊕ W[4(i-1) + j]

  the (j-1)th of subkey at roud i the j-th word of subkey at roud i-1

  – i = 1, 2, 3, …, 10
  – j = 1, 2, 3



Example: Key schedule for 128-bit key AES

62

Key Schedule (4)

- Function *g()* rotates its four input bytes and performs a bytewise S-Box substitution

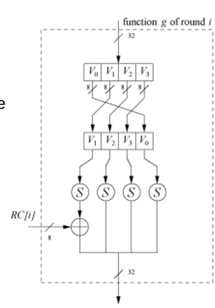
⇒ Add nonlinearity to key schedule

- The round coefficient *RC* is only added to the leftmost byte and varies from round to round

⇒ Remove symmetry in AES

$RC[1] = x^0 = (00000001)_2$
$RC[2] = x^1 = (00000010)_2$
$RC[3] = x^2 = (00000100)_2$
…
$RC[10] = x^9 = (00110110)_2$

– x^i represents an element in a Galois field



63

- ***What we have learnt about block cipher by now?***

  DES, 3DES, and AES encrypt a block of data

- ***How to utilize a block cipher?***
- ***How to encrypt long plaintexts with a block cipher?***

64

Block Ciphers

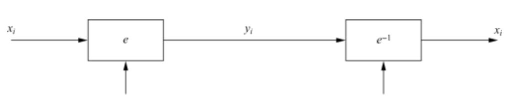
- A block cipher is much more than just an encryption algorithm, it can be used ...
  - to build different types of block-based encryption schemes
  - to realize stream ciphers
  - to construct hash functions
  - to make message authentication codes
  - to build key establishment protocols
  - to make a pseudo-random number generator
  - ...
- The security of block ciphers also can be increased by
  - key whitening
  - multiple encryption

Encryption with Block Ciphers

- There are several ways of encrypting long plaintexts, e.g., an e-mail or a computer file, with a block cipher ("modes of operation")
  - Electronic Code Book mode (ECB)
  - Cipher Block Chaining mode (CBC)
  - Output Feedback mode (OFB)
  - Cipher Feedback mode (CFB)
  - Counter mode (CTR)
  - Galois Counter Mode (GCM)
- Goal: in addition to confidentiality, they provide authenticity and integrity:
  - Is the message really coming from the original sender? (authenticity)
  - Was the ciphertext altered during transmission? (integrity)

Electronic Code Book Mode (ECB)

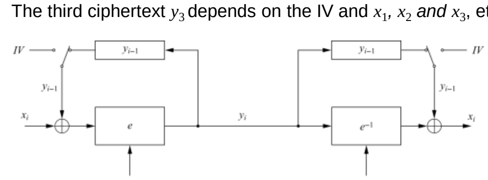
- $e_k(x_i)$: the encryption of a b-bit plaintext block x_i with key k
- $e_k^{-1}(y_i)$: the decryption of b-bit ciphertext block y_i with key k
- Messages which exceed b bits are partitioned into b-bit blocks
- **Each Block is encrypted separately**



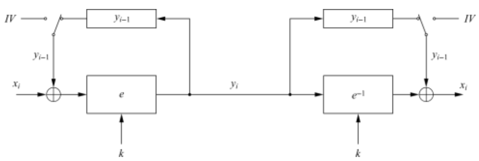
Encryption: $y_i = e_k(x_i),\ i \geq 1$
Decryption: $x_i = e_k^{-1}(y_i) = e_k^{-1}(e_k(x_i)),\ i \geq 1$

Cipher Block Chaining Mode (CBC)

- For the first plaintext block x_1 there is no previous ciphertext
  - An IV is added to the first plaintext to make each CBC encryption nondeterministic
  - The first ciphertext y_1 depends on plaintext x_1 and the IV
- The second ciphertext y_2 depends on the IV, x_1 *and* x_2
- The third ciphertext y_3 depends on the IV and x_1, x_2 *and* x_3, etc.



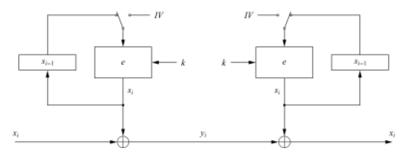
Cipher Block Chaining Mode (CBC)



***Encryption* (*first block*)**: $y_1 = e_k(x_1 \oplus \mathrm{IV})$
***Encryption* (*general block*)**: $y_i = e_k(x_i \oplus y_{i-1}),\ i \geq 2$
***Decryption* (*first block*)**: $x_1 = e_k^{-1}(y_1) \oplus \mathrm{IV}$
***Decryption* (*general block*)** : $x_i = e_k^{-1}(y_i) \oplus y_{i-1},\ i \geq 2$

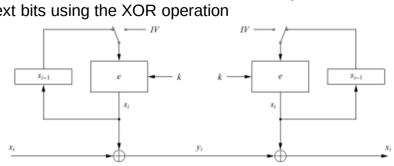
Output Feedback Mode (OFB)

- It is used to build a *synchronous* **stream cipher** from a block cipher
- Key stream is not generated bitwise but instead in a **blockwise** fashion
- Output of the cipher gives us key stream bits S_i with which we can encrypt plaintext bits using the XOR operation



Output Feedback Mode (OFB)

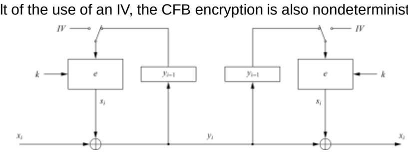
- It is used to build a **synchronous stream cipher** from a block cipher
- Key stream is not generated bitwise but instead in a **blockwise** fashion
- Output of the cipher gives us key stream bits S_i with which we can encrypt plaintext bits using the XOR operation



***Encryption* (*first block*)**: $s_1 = e_k(\mathrm{IV})$ *and* $y_1 = s_1 \oplus x_1$
***Encryption* (*general block*)**: $s_i = e_k(s_{i-1})$ *and* $y_i = s_i \oplus x_i,\quad i \geq 2$
***Decryption* (*first block*)**: $s_1 = e_k(\mathrm{IV})$ *and* $x_1 = s_1 \oplus y_1$
***Decryption* (*general block*)** : $s_i = e_k(s_{i-1})$ *and* $x_i = s_i \oplus y_i,\quad i \geq 2$

Cipher Feedback Mode (CFB)

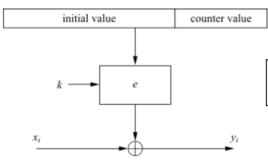
- It uses a block cipher as a building block for an **asynchronous stream cipher** (similar to the OFB mode), more accurate name: "Ciphertext Feedback Mode"
- Key stream S_i is generated in a blockwise fashion and is also a function of ciphertext
- As a result of the use of an IV, the CFB encryption is also nondeterministic



***Encryption* (*first block*)**: $y_1 = e_k(\mathrm{IV}) \oplus x_1$
***Encryption* (*general block*)**: $y_i = e_k(y_{i-1}) \oplus x_i,\quad i \geq 2$
***Decryption* (*first block*)**: $x_1 = e_k(\mathrm{IV}) \oplus y_1$
***Decryption* (*general block*)** : $x_i = e_k(y_{i-1}) \oplus y_i,\quad i \geq 2$

Counter Mode (CTR)

- It uses a block cipher as a **stream cipher** (like the OFB and CFB)
- The key stream is computed in a blockwise fashion
- The input to the block cipher is a counter which assumes a different value every time the block cipher computes a new key stream block
- Unlike CFB and OFB modes, the CTR mode can be parallelized since the 2nd encryption can begin before the 1st one has finished
  - Desirable for high-speed implementations, e.g., in network routers

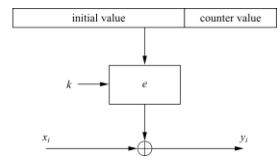


Encryption: $y_i = e_k(\mathrm{IV} \,||\, \mathrm{CTR}_i) \oplus x_i, \quad i \geq 1$
Decryption : $x_i = e_k(\mathrm{IV} \,||\, \mathrm{CTR}_i) \oplus y_i, \quad i \geq 1$

Example: CTR

Assume the following implementation:

- Block cipher algorithm: AES (with 128 bits)
- Initiation Vector (IV): 96 bits
- CTR: 128 – 96 = 32 bits
- Number of blocks for encryption with the same IV: 2^{32}
- Plaintext size: 16 * 2^{32} = 2^{36} bytes



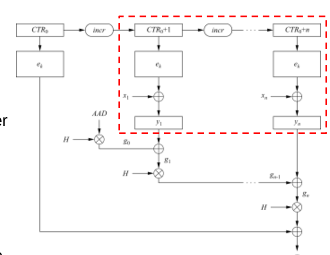
Galois Counter Mode (GCM) (1)

- It also computes a ***message authentication code (MAC)***, i.e., a cryptographic checksum is computed for a message
- By making use of GCM, two additional services are provided:
  - **Message Authentication**
    - the receiver can make sure that the message was really created by the original sender
  - **Message Integrity**
    - the receiver can make sure that nobody tampered with the ciphertext during transmission

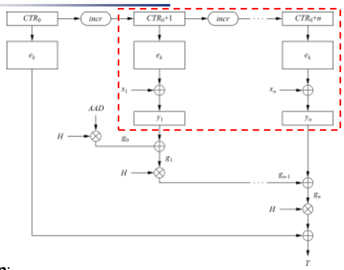
Galois Counter Mode (GCM) (2)

For encryption (counter mode)

- An initial counter is derived from an IV and a serial number
- The initial counter value is incremented then encrypted and XORed with the first plaintext block
- For subsequent plaintexts, the counter is incremented and then encrypted



Galois Counter Mode (GCM) (3)



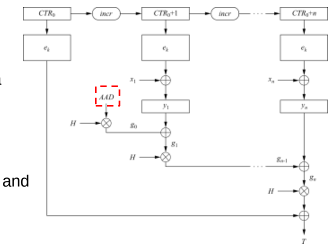
Encryption:

1. Derive a counter value CTR_0 from the IV and compute $\mathrm{CTR}_1 = \mathrm{CTR}_0 + 1$
2. Compute ciphertext: $y_i = e_k(\mathrm{CTR}_i) \oplus x_i, \quad i \geq 1$

Galois Counter Mode (GCM) (4)

For authentication

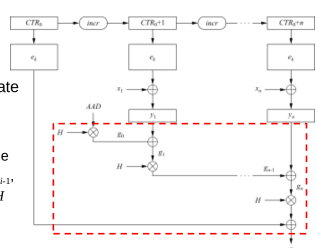
- Protects authenticity of the plaintexts and the string additional authenticated data (AAD)
- AAD is in clear (without encryption)
- AAD may include addresses and parameters in a network protocols



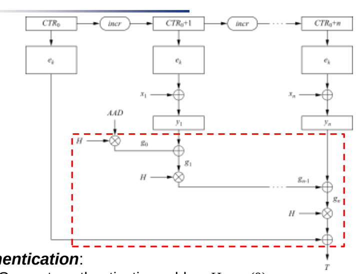
Galois Counter Mode (GCM) (5)

For authentication

- A chained Galois field multiplication is performed
- For every plaintext an intermediate authentication g_i is derived
  - g_i is computed as the XOR of the current ciphertext and the last g_{i-1}, and multiplied by the constant H
  - H is generated by encryption of the zero input with the block cipher
- All multiplications are in the 128-bit Galois field GF(2^{128})



Galois Counter Mode (GCM) (6)



Authentication:

1. Generate authentication subkey $H = e_k(0)$
2. Compute $g_0 = AAD \times H$ (Galois field multiplication)
3. Compute $g_i = (g_{i-1} \oplus y_i) \times H, 1 \leq i \leq n$ (Galois field multiplication)
4. Final authentication tag: $T = (g_n \times H) \oplus e_k(\mathrm{CTR}_0)$

Galois Counter Mode (GCM) (7)

- Sender send the packet [$(y_1, y_2, \ldots, y_n)$, T, AAD]
- Receiver
  - Decrypts $(y_1, y_2, \ldots, y_n)$ by also using the counter mode
  - Checks the authenticity of data by computing an authentication tag T' with $(y_1, y_2, \ldots, y_n)$ and AAD, which is exactly the same as the method used by sender
    - If T' = T, it was not manipulated