BLOCK CIPHERS: OVERVIEW, DES & AES

Applied Cryptography

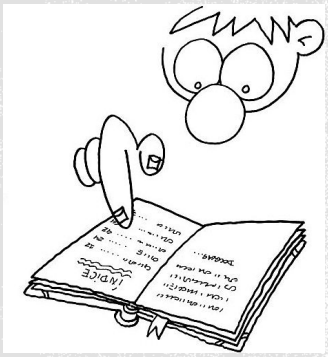
Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

CONTENTS

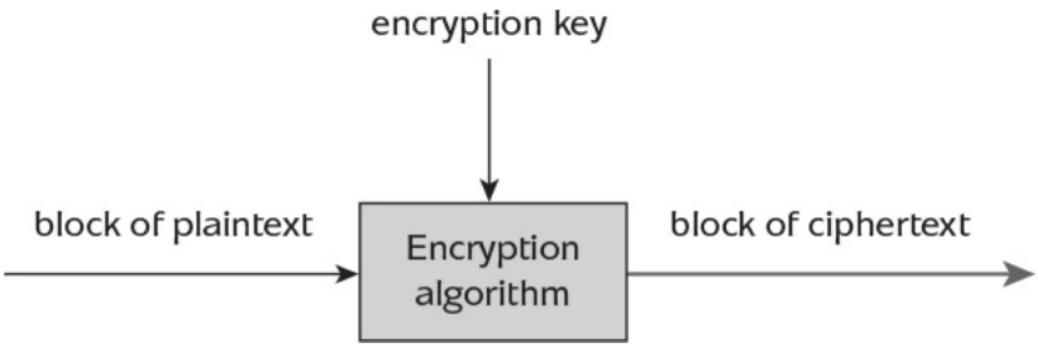
- Block ciphers through the lens of perfect ciphers
- Anatomy of block ciphers
  - DES
    - Feistel function
    - Key scheduling
  - Triple DES
  - AES
- Mode of operations for block ciphers



2 RECALL BLOCK CIPHERS

OVERVIEW

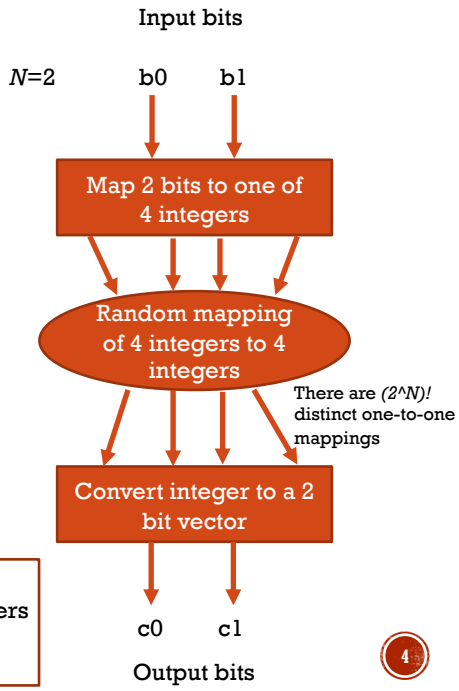
- typical block size = 64 or 128 bits
- while block size is fixed, size of encryption key may vary



BLOCK CIPHERS (1)

- Main idea
  - Replace a block of *N* bits from the plaintext with a block of *N* bits from the ciphertext
  - The relationship between the input blocks and the output block is completely random
  - It must be invertible for decryption to work
  - Thus, it has to be one-to-one, i.e. each input block is mapped to a unique output block
  - Usually, *N=64, 128, 256*

- The first number among the *2^N* can map to any of the *2^N* numbers
- The second number can map to any of the remaining *(2^N) - 1* numbers
- … and so on
- The total number of one-to-one functions is *(2^N)!*



4

BLOCK CIPHERS (2)

- Representation of one member of the family
- For each key, we have a single permutation that is independent of all the others

- A *perfect* block cipher is a **keyed family of random permutations**
- It can be described as follows
  - **Encryption**
    - Daemon checks in the left hand column to see if it has a record of plaintext
    - If not, it asks the random number generator to generate a ciphertext that does not yet appear in the right hand column, and then writes down the plaintext/ciphertext pair in the codebook and returns the ciphertext
    - If it does find a record, it returns the corresponding ciphertext from the right hand column
  - **Decryption**
    - Same procedure as above with columns swapped
- In practice…
  - given a block size *N* and a key size *K*
  - design ways to choose *2^K* permutations uniformly at random from the set of all *(2^N)!* permutations
  - Notice that *2^K << (2^N)!*

Encryption key (codebook)

Plain text	Cipher text
0	?
1	?
…	…
(2^N)-1	?

S. Ranise - Security & Trust (FBK)

5

WHICH ARE THE "RIGHT" WAYS TO DEFINE SUCH FAMILIES OF RANDOM PERMUTATIONS?

6

For this, we need to take another look at perfect ciphers…

PRELIMINARY REMARKS

- To introduce the main concept, we will consider transformations on plaintexts and ciphertexts made up of English letters
- In this context, Vigenere cipher corresponds to Vernam cipher
  - The basic operation for **encryption or decryption in Vernam cipher is** logical xor or, equivalently, **addition modulo 2**
  - If we encode the letters of the English alphabet with the numbers corresponding to the position in which they occur in their natural ordering, Vigenere performs **encryption or decryption by addition modulo 26**
  - As Vernam cipher uses a sequence of bits of equal length of the plaintext for encryption and decryption as key, **Vigenere uses a sequence of letters of equal length of the plaintext** for encryption and decryption **as key**
- We will give an intuition of how these ideas transfer to plaintexts and ciphertexts made up of blocks of bits

UNICITY DISTANCE (1)

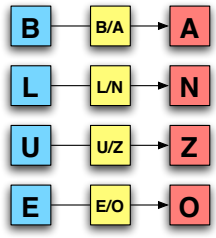
- The **unicity distance** of a cipher encrypting English plaintexts is the **minimum of ciphertext** required for a (computationally unlimited) **attacker** to decrypt a ciphertext uniquely (i.e., to recover the particular key used)

- Example
  - Let WNAIW be the ciphertext obtained by encoding an English word by Vigenere key cipher with a key of length 5
  - Can one determine uniquely the plaintext?
  - It is possible to find two reasonable solutions: RIVER and WATER
  - Many unreasonable ones, e.g., KHDOP, SXOOS, …
  - Not a unique plaintext…

UNICITY DISTANCE (2)

Substitution cipher

- Substitutes one symbol for another
- The **key** is the substitution that can be seen as a **permutation of the letters in the alphabet**



- Example
  - consider the following ciphertext FJKFPO
  - That is encrypted with a **substitution** cipher
- Can an attacker decipher it?
- The answer is 'not really'
- It is possible to find many English words that may be the plaintext generating the ciphertext above:
  - thatis
  - ofyour
  - season
  - thatwe
  - …
- Intuitively, the longer the ciphertext, the fewer possible plaintexts there are that generate the ciphertext
- **QUESTION**: how long does a piece of ciphertext need to be, before it has only one possible decryption?
- **ANSWER**: the minimum length is called the unicity distance

It turns out that the unicity distance depends on the redundancy in the plaintexts…

SUBSTITUTION CIPHERS AND REDUNDANCY

- In English, the substitution cipher has a key that consists of 26 letters since the total number of keys is the number of permutations in which we can arrange 26 letters, i.e. *26!*

- The amount of storage (expressed in number of bits) required for
  - All permutations is log(26!) = 88.28
  - One permutation is log(26) = 4.7

log is intended in base 2

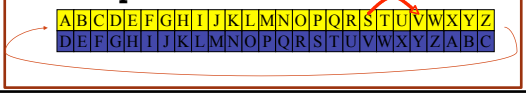
- **Redundancy**
  - The fact that, for instance, the letter 'u' always follows the letter 'q' makes the letter combination 'qu' redundant
  - The redundancy of English has been evaluated to be around 3.2 bits per character

UNICITY DISTANCE (3)

- **Unicity distance** = number of bits required to express the key space divided by the redundancy in bits per character
- In the case of the substitution cipher:
  - Unicity distance = 88.28/3.2 = 27.6 ≈ 28
  - I.e. at least 28 characters are required to be sure a particular decryption is unique

- Different ciphers have different unicity distances
- A higher unicity distance generally indicates a more secure cipher
  - **Example**: the unicity distance of the Caesar cipher is ≈ 2
  - We can conclude that the Caesar cipher is "weaker" than the substitution cipher

Caesar cipher



UNICITY DISTANCE & VIGENERE CIPHER

- **In a system with an infinite length random key**, it is possible to prove that the **unicity distance is infinite**
- This is indeed the case for the Vigenere cipher and this is why in the first example above, it was impossible to identify which one among RIVER and WATER was the plaintext corresponding to the the ciphertext WNAIW
- This is **another way to look at the fact the One Time Pad is a perfect cipher**

WHY UNICITY DISTANCE CAN BE USEFUL FOR BLOCK CIPHERS (1)

- Intuitively, the unicity distance measures the amount of ciphertext required such that there is only one reasonable plaintext
- Consider RC4 as a block cipher as it encrypts data in bytes, i.e. in small blocks of 8 bits
  - With 8 bits, you can encode letters by means of the ASCII code
  - **Imagine** a single ASCII letter as plaintext
    - There are 26 possible plaintexts out of 2^8=256 possible decryptions
    - Any random key, when used to decrypt the ciphertext, has a 26/256 = 10% chances of producing a valid plaintext
    - There is no way to tell the wrong plaintext from the correct plaintext
  - Now **imagine** a 1K e-mail message
    - It is possible to try random keys
    - Eventually a plaintext emerges that looks like an e-mail message: words, phrases, sentences, grammar
  - The unicity distance determines when one can think like the second example instead of the first

Excerpt of ASCII code table

65	101	41	01000001	A
66	102	42	01000010	B
67	103	43	01000011	C
68	104	44	01000100	D
69	105	45	01000101	E
70	106	46	01000110	F
71	107	47	01000111	G
72	110	48	01001000	H
73	111	49	01001001	I
74	112	4A	01001010	J
75	113	4B	01001011	K
76	114	4C	01001100	L
77	115	4D	01001101	M
78	116	4E	01001110	N
79	117	4F	01001111	O
80	120	50	01010000	P
81	121	51	01010001	Q
82	122	52	01010010	R
83	123	53	01010011	S
84	124	54	01010100	T
85	125	55	01010101	U
86	126	56	01010110	V
87	127	57	01010111	W
88	130	58	01011000	X
89	131	59	01011001	Y
90	132	5A	01011010	Z

https://www.ascii-code.com/

WHY UNICITY DISTANCE CAN BE USEFUL FOR BLOCK CIPHERS (2)

- Let us generalize a bit the previous scenario by considering a block cipher with a *N* as the block size and *K* as the key size
- The question becomes then

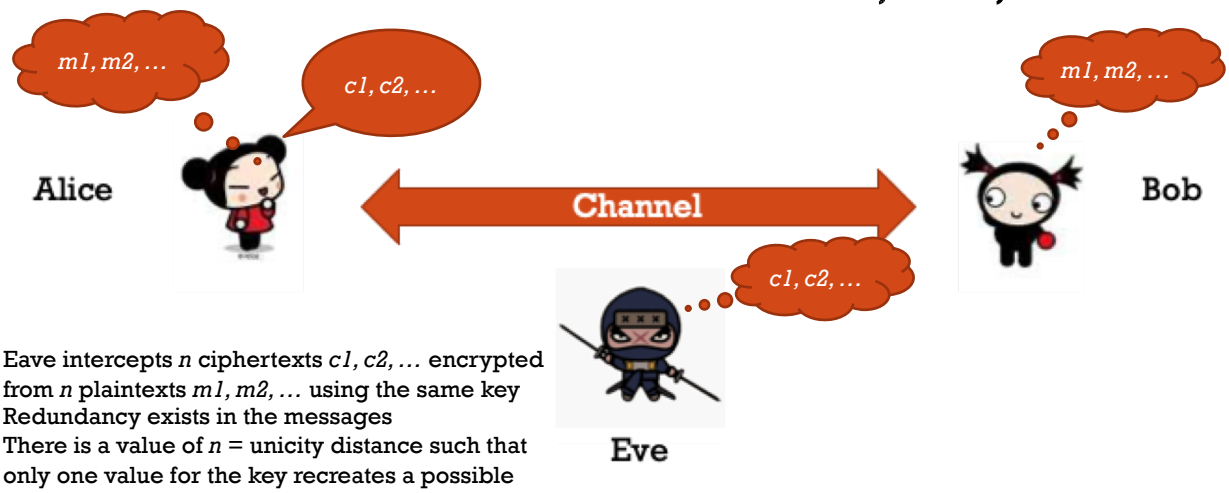
What is the unicity distance under a known plaintext attack*?*

or

Given *t* pairs of plaintexts with corresponding ciphertexts,
which is the value of t, before it is likely that only one value of the key could have encrypted the plaintext *?*

- The answer is the unicity distance, i.e. the number of bits required to express the key space divided by the redundancy in bits per character

RECALL THE STORY ABOUT ALICE, BOB, EVE

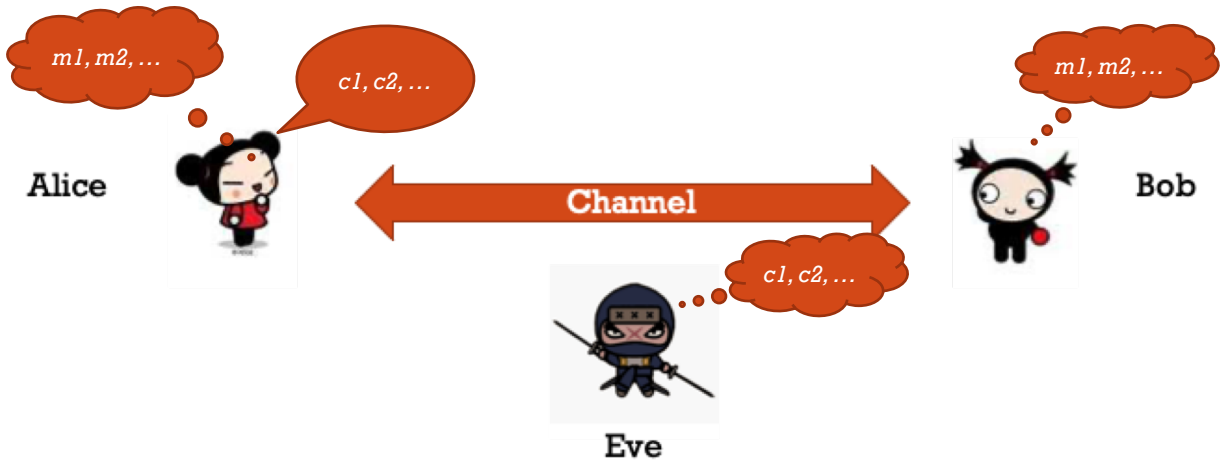


- Eave intercepts *n* ciphertexts *c1, c2, …* encrypted from *n* plaintexts *m1, m2, …* using the same key
- Redundancy exists in the messages
- There is a value of *n* = unicity distance such that only one value for the key recreates a possible plaintext (unless Alice and Bob use the One Time Pad)

A POSSIBLE ATTACK

- **Observation**
  - The same block with the same key always produces the same ciphertext, independently of its position in a sequence
  - I.e. we use simple substitution on the block level
- **Strategy**
  - Eve could, **in principle**, create a table of all plaintext values and their corresponding ciphertexts, **one table for each key**, and use this for cryptanalysis
- As defense, …

DEFENSE METHOD (1)

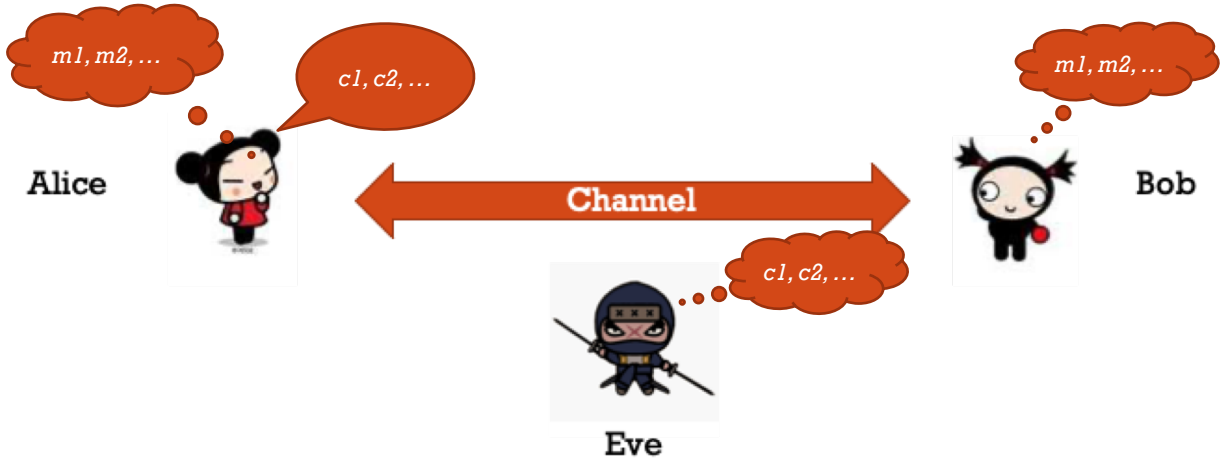


Change key often enough

- Recall that stream ciphers use session keys that are combined with IVs

→ **Unicity distance is not reached**

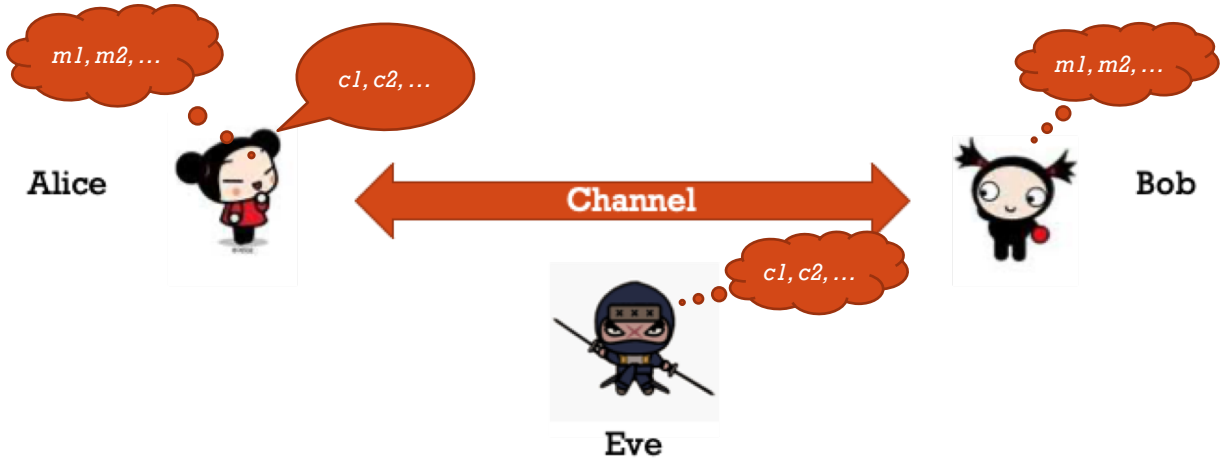
DEFENSE METHOD (2)



Make sure there are too many possible keys for brute force attacks → **Increasing the length of the key increases the unicity distance** (= length of the key / redundancy)

18

DEFENSE METHOD (3)



Encode with larger blocks (than 1, 2, 3, … letters) → See next slide….

19

UNICITY DISTANCE AND LARGE BLOCKS

- The unicity distance is primarily a theoretical measure, useful in relation to perfect ciphers
  - **Definition of unicity distance under Shannon Theorem assumptions**: the minimum amount of ciphertext required to allow a **computationally unlimited adversary** to recover the unique encryption key
- A small unicity distance does not necessarily imply that a block cipher is **insecure in practice**
- **Example**: consider a 64-bit block cipher with a unicity distance of two ciphertext blocks
  - It may still be **computationally infeasible** for an **attacker** (of reasonable but **bounded computing power**) to recover the key, although theoretically there is sufficient information to allow this

TO SUMMARIZE… SHANNON'S DESIGN PRINCIPLES

- **Diffusion**
  - The ciphertext statistics should depend on the plaintext statistics in a manner too complicated to be exploited by the cryptanalyst
  - **Rule of thumb** → Permutations creates diffusion
- **Confusion**
  - Each digit of the plaintext and each digit of the secret key should influence many digits of the ciphertext
  - **Rule of thumb** → Substitutions creates confusion
- **Modern block ciphers** are typically obtained by **mixing substitutions and permutations** to obtain both confusion and diffusion

Diffusion means that if we change a single bit of the plaintext, then about half of the bits in the ciphertext should change

Confusion means that each bit of the ciphertext should depend on several parts of the key

SOME REMARKS

- How can we be sure an attacker will require a large amount of work to break a non-perfect system with every method?
- This is a difficult answer to which no unique or good single answer exists, as far as we know today…
- … in practice, we try to make ciphers secure to all known attacks
  - This is typically the approach used to show the security of symmetric ciphers, both stream and block ciphers
- An alternative, it is to show that breaking the cipher can be reconducted to a computationally difficult problem
  - This is typically the approach used to show the security of public key ciphers

23

ANATOMY OF BLOCK CIPHERS

BLOCK CIPHERS, OVERVIEW (1)

	block size, n	key size, κ	year
DES	64	56	1977
Kasumi	64	128	1999
AES	128	128, 192, 256	2000
Present	64	80, 128	2007

- A block cipher is a **keyed family of pseudorandom permutations**
- For each key, we have a single permutation that is independent of all the others
- Design ways to choose *2^K* permutations uniformly at random from the set of all *(2^N)!* permutations
  - As one can see from the table, it is obvious that block ciphers can only select a tiny fraction of all possible n-bit permutations
- **Goal**
  - for a block cipher to be good, Eve should not be able to recover the key **even using multiple plaintext-ciphertext pairs**

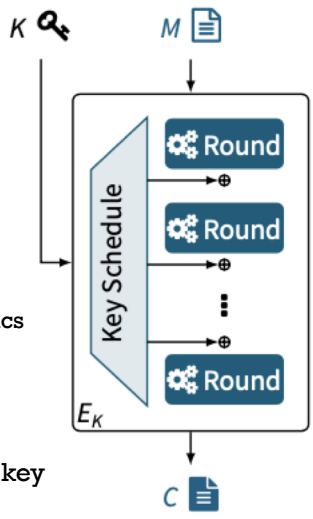
BLOCK CIPHERS, OVERVIEW (2)

Key design principles

- **Diffusion**

  If a plaintext bit changes, several ciphertext bits should change
  - This is a basic demand on a block cipher, and ensures that the statistics used are block statistics
- **Confusion**

  Every bit of the ciphertext should depend on several bits in the key
  - This can be achieved by ensuring that the system is nonlinear



ON DIFFUSION

- Diffusion means that the output bits should depend on the input bits in a very complex way
  - Ideally, if one bit of the plaintext is changed, then the ciphertext should change completely, in an unpredictable or pseudorandom manner
- **Avalanche criterion**
  - Flipping a fixed set of bits should change each output bit with probability one half
- **Strict avalanche criterion**
  - For a randomly chosen input, if one flips the *i*-th bit, then the probability that the *j*-th output bit will change should be one half, for any *i* and *j*

ON CONFUSION

- **Confusion** refers to making the **relationship between the key and the ciphertext as complex and involved as possible**
- **Goal**: make it very hard to find the key even if one has a large number of plaintext-ciphertext pairs produced with the same key
- Each bit of the ciphertext should depend on the entire key, and in different ways on different bits of the key
- Changing one bit of the key should change the ciphertext completely

SUBSTITUTION-PERMUTATION NETWORK

- The simplest way to achieve both diffusion and confusion
  - The plaintext and the key often have a very similar role in producing the output, hence it is the same mechanism that ensures both diffusion and confusion
- It takes a block of the plaintext and the key as inputs, and to produce the ciphertext block applies several alternating "**rounds**" or "**layers**" of
  - substitution boxes (S-boxes) and
  - permutation boxes (P-boxes)
- S-boxes and P-boxes transform (sub-)blocks of input bits into output bits
- It is common for these transformations to be operations that are efficient to perform in hardware, such as xor and bitwise rotation
- The key is introduced in each round, usually in the form of "round keys" derived from it
- Decryption is done by simply reversing the process (using the inverses of the S-boxes and P-boxes and applying the round keys in reverse order)

S-BOX

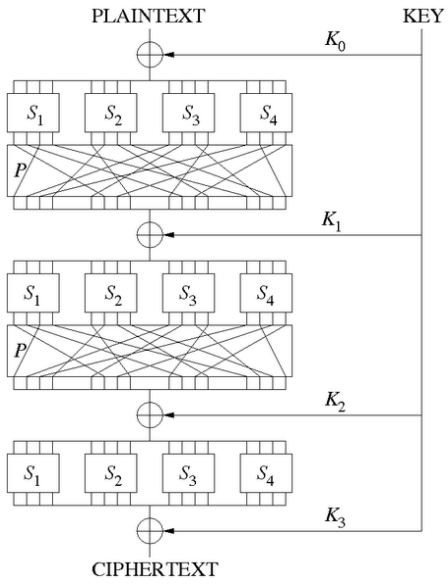
- It substitutes a small block of bits by another block of bits
- This substitution should be one-to-one, to ensure invertibility (hence decryption)
  - There are exceptions as we will see for DES
- In many cases, the length of the output is the same as the length of the input
  - In general, this is not always the case as in the case of DES (Data Encryption Standard)
- An S-box is usually not just a permutation of the bits
- Rather, a good S-box will have the property that changing one input bit will change about half of the output bits (with an avalanche effect)
- It will also have the property that each output bit will depend on every input bit

P-BOX

- It is a **permutation** of all the bits
- It takes the outputs of all the S-boxes of one round, permutes the bits, and feeds them into the S-boxes of the next round
- A good P-box has the property that **the output bits of any S-box are distributed to as many S-box inputs as possible**

Side note

At each round, the **round key** (obtained from the key with some simple operations, for instance, using S-boxes and P-boxes) is combined using some group operation, typically XOR

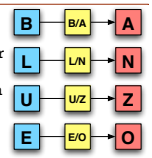


30

SOME REMARKS

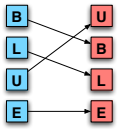
Substitution cipher

- Substitutes one symbol for another
- The **key** is the substitution that can be seen as a **permutation of the letters in the alphabet**



Transposition cipher

- Scrambles the symbols to produce output
- The **key** is the **permutation of symbols as they occur in the text**

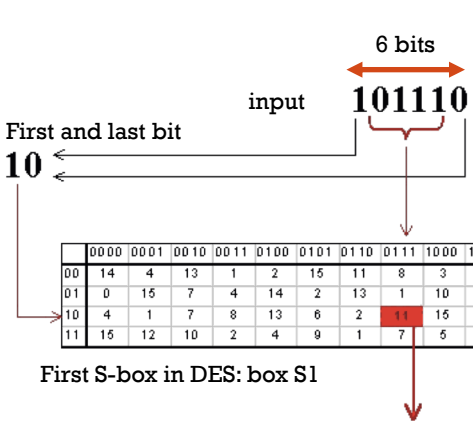


- A single typical S-box or a single P-box alone does not have much cryptographic strength
  - An **S-box** could be thought of as a **substitution cipher**
  - A **P-box** could be thought of as a **transposition cipher**
- A good SP network with several alternating rounds of S- and P-boxes already satisfies Shannon's Diffusion and Confusion
  - **Diffusion**: If one changes one bit of the plaintext, then it is fed into an S-box, whose output will change at several bits, then all these changes are distributed by the P-box among several S-boxes, hence the outputs of all of these S-boxes are again changed at several bits, and so on
    - Doing several rounds, each bit changes several times back and forth, therefore, by the end, the ciphertext has changed completely, in a pseudorandom manner.
  - **Confusion**: changing one bit of the key changes several of the round keys, and every change in every round key diffuses over all the bits, changing the ciphertext in a very complex manner

31

S-BOX: AN EXAMPLE

A s-box is inspired to a substitution cipher for characters as it substitutes bits (the numbers of input and output bit may differ)



A s-box may or may not be invertible. When it is, the number f input and output bit should be the same.

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	7	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	10	2	4	9	1	7	5	11	3	14	10	0	6	13

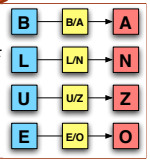
First S-box in DES: box S1

Output 11 that written in base 2 is equal to 1011

4 bits

Substitution cipher

- Substitutes one symbol for another
- The **key** is the substitution that can be seen as a **permutation of the letters in the alphabet**



How to read a S-box table

- First and last bit are interpreted as row index
- Other four bits are interpreted as column index
- Given the 6 input bits, find the cell identified by row and column indexes (extracted as explained above) and read the binary equivalent of the integer in the cell

ON THE DEFINITION OF S-BOXES FOR DES

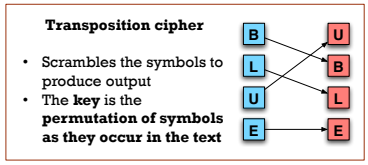
- The definition of S-boxes may be obscure
- This is so because the criteria for their design are not fully disclosed
- Around 1976, NSA disclosed **some** of the properties of the S-boxes in DES:
  - Each row should be a permutation of the integers from 0 to 15
  - No s-box should be an affine or linear mapping of its input
  - Changing one of the input bit at least two out bits should change
  - …
- It is **unknown if more properties were used to design** the S-boxes…

TYPES OF P-BOXES

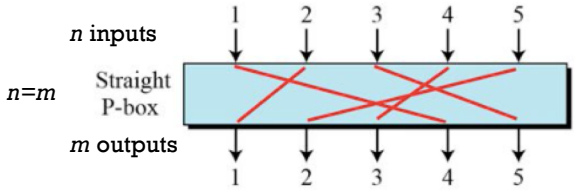
A **p-box** is inspired to the **transposition cipher** for characters as it transposes bits

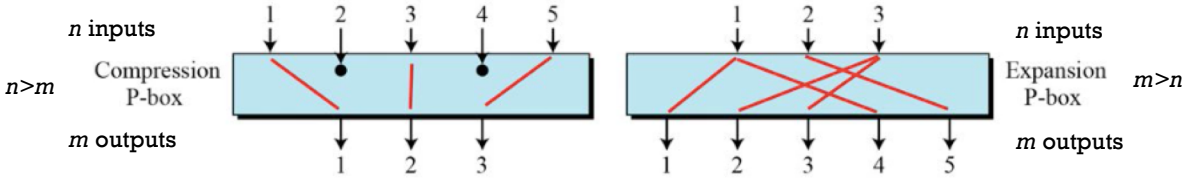
Transposition cipher

- Scrambles the symbols to produce output
- The **key** is the **permutation of symbols as they occur in the text**



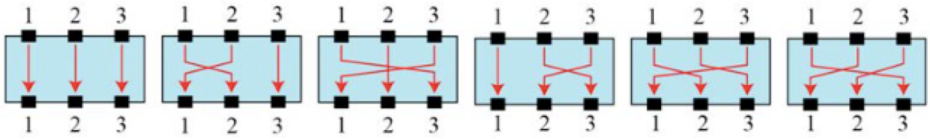
A **straight** p-box is **invertible** whereas neither a compression nor an expansion p-box is so.





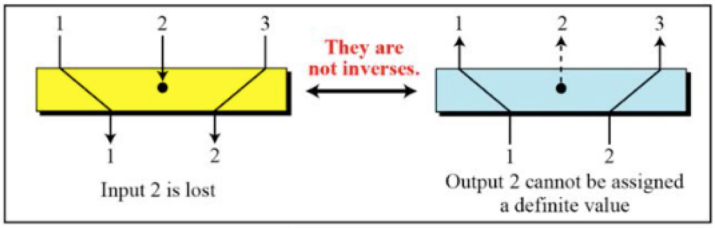
EXAMPLE: 3*3 STRAIGHT P-PBOX

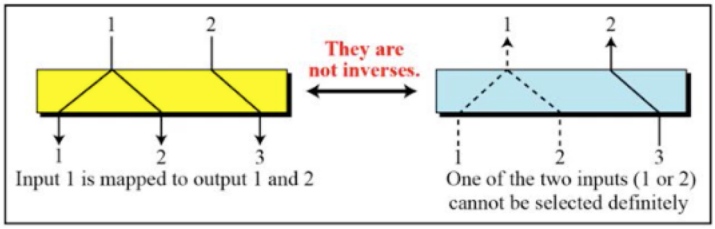
- Although a P-box can use a key to identify one of the *n!*, they are normally **key-less** as the mapping is pre-configured
- Below are the *n!=3!* possible definitions of a P-box of size 3



WHY EXPANSION AND COMPRESSION P-BOXES ARE NOT INVERTIBLE

Compression p-box





Expansion p-box

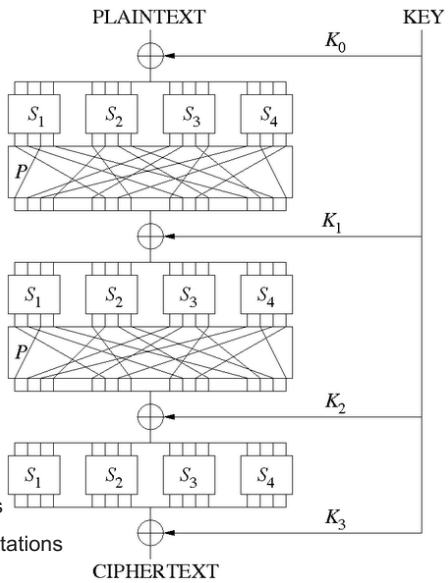
AN EXAMPLE OF A BLOCK CIPHER

- *Si* is an S-box and *P* is a P-box
- 3 rounds or layers and *Kj* is a key derived from the same key (KEY)
- https://en.wikipedia.org/wiki/Block_cipher
- Formalization

$$E_K(P) := E(K,P) : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$$

$$E_K^{-1}(C) := D_K(C) = D(K,C) : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n,$$

  where *k* is the **key size/length** and *n* is the **block size**
- **Typically: *k<n***
  - recall the codebook size problem
- E_K is a permutation (i.e. a bijective mapping) over the set of input blocks
- Each key selects one permutation from the set of *(2^n)*! possible permutations



38

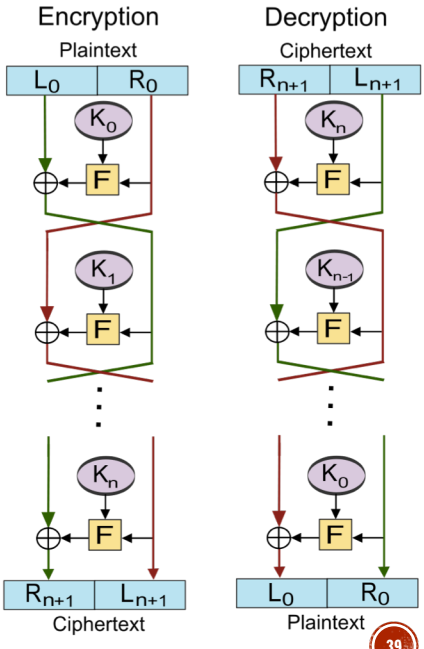
DES

Data Encryption Standard

DES standard original document (withdrawn):
https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf

DATA ENCRYPTION STANDARD (DES)

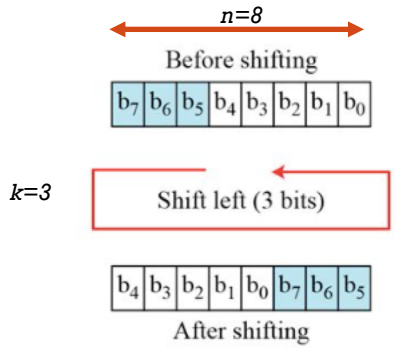
- Based on the **Feistel Structure**
  - Named after the IBM cryptographer Horst Feistel and first implemented in the **Lucifer** cipher by Horst Feistel and Don Coppersmith
- It uses the same basic algorithm for both encryption and decryption
- It consists of multiple rounds of processing of the plaintext, with each round consisting of a **substitution** step **followed by a permutation** step
- In each round
  - the right half of the block, *R*, goes through unchanged
  - the left half, *L*, goes through an operation that depends on R and the encryption key
  - the operation carried out on the left half L is referred to as the **Feistel Function** *F*
  - The **permutation** step consists of **swapping** the modified **L and R**



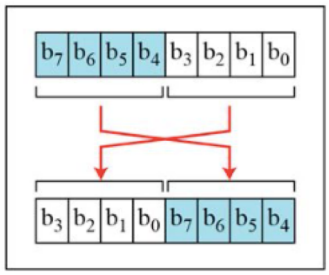
REMARK ON SWAPPING HALVES

- Recall (**circular**) **shift registers**

Note: $k<n$



- **Swap by shifting**: just take $k=n/2$



REMARK ON BITWISE XOR

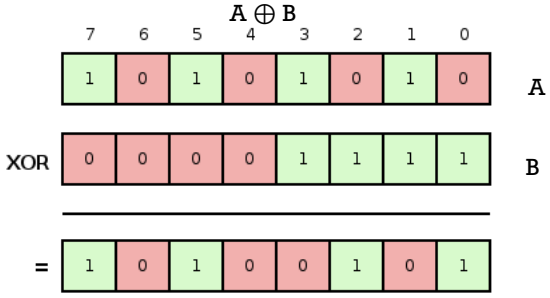
Xor truth table
A + B

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

A and B are bits

Stream ciphers

Bitwise Xor
A ⊕ B



A and B are bytes
(more in general blocks of n-bits)

Block ciphers

MATHEMATICAL DESCRIPTION: ENCRYPTION

- There are 16 rounds numbered from 1 to 15
- 0 identifies the plaintext
- 16 identifies the ciphertext

- Let *LEi* and *REi* denote the output half-blocks at the end of the *i*-th round of processing (*E* stands for encryption)
- Relationship between the output of the *i*-th round and the output of the previous (*i*—1)-th round:

$$LE_i = RE_{i-1}$$
$$RE_i = LE_{i-1} \oplus F(RE_{i-1}, K_i)$$

Feistel function

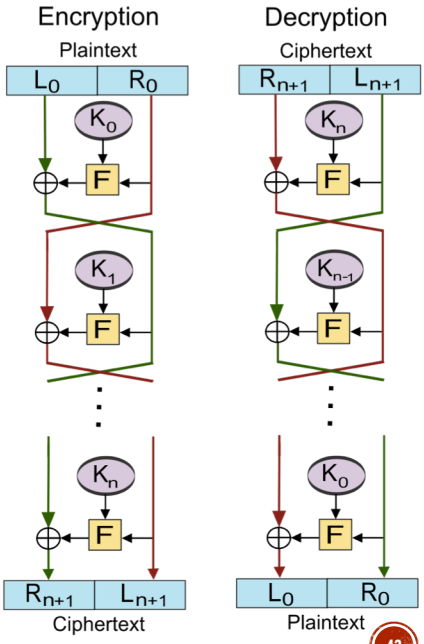
where

- the symbol $\oplus$ denotes xor
- the symbol *F* denotes the operation that "scrambles" *REi-1* of the previous round with the round key *Ki*

- The round key *Ki* is derived from the main encryption key (we will see how below)

ENCRYPTION & DECRYPTION (1)

- Decryption is exactly the same as the encryption with the only difference that the **round keys are used in the reverse order**
- ***Fact.*** The output of each round during decryption is the input to the corresponding round during encryption, except for the left-right switch between the two halves
  - This holds true regardless of the choice of the Feistel function *F*
- ***Proof***
  - Let *LDi* and *RDi* denote the left half and the right half of the output of the *i*-th round
  - $LD_0 = RE_{16}$ $RD_0 = LE_{16}$



ENCRYPTION & DECRYPTION (2)

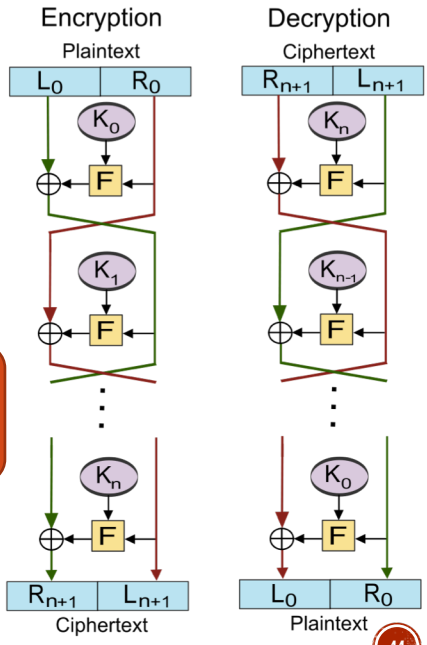
The result is independent of the definition of the Feistel function *F*

- ***Proof (continued)***
  - Let *LDi* and *RDi* denote the left half and the right half of the output of the *i*-th round during Decryption
  - $LD_0 = RE_{16}$ $RD_0 = LE_{16}$
  - Then

$$\begin{aligned} LD_1 &= RD_0 \\ &= LE_{16} \\ &= RE_{15} \end{aligned}$$

The output of the **first round of decryption** is the same as the input to the **last stage of the encryption round**

$$\begin{aligned} RD_1 &= LD_0 \oplus F(RD_0, K_{16}) \\ &= RE_{16} \oplus F(LE_{16}, K_{16}) \\ &= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16}) \\ &= LE_{15} \end{aligned}$$



MORE ON DES

- Adopted by NIST in 1977
- Based on the Lucifer cipher developed earlier by IBM for Lloyd's of London for cash transfer
- DES uses the Feistel cipher structure with 16 rounds of processing
- DES uses a **56-bit encryption key**
  - The key size was apparently dictated by the memory and processing constraints imposed by a single-chip implementation of the algorithm for DES
  - The key itself is specified with 8 bytes, but one bit of each byte is used as a parity check

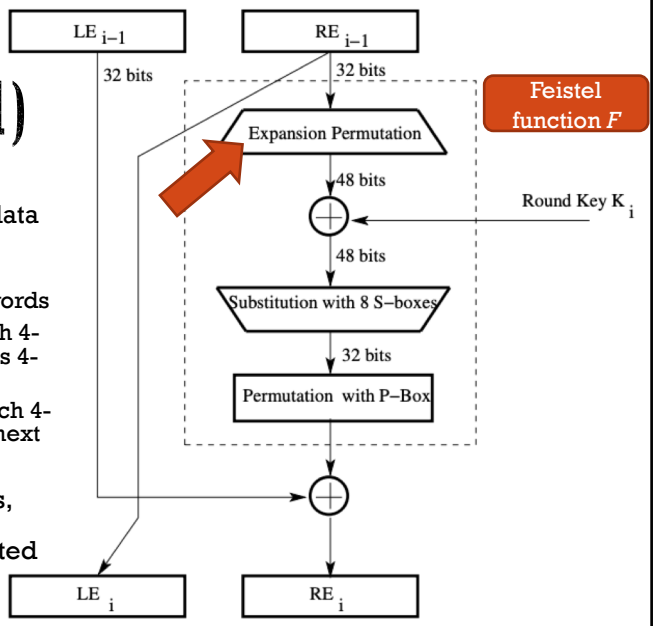
Parity checking (by example)
- if the original data is 1010001, there are three 1s
- when **even** parity checking is used, a parity bit with value 1 is added to the data's left side to make the number of 1s even; transmitted data becomes 11010001
- when **odd** parity checking is used, then parity bit value is zero; transmitted data is 01010001
- In case data is transmitted incorrectly, the parity bit value becomes incorrect; thus, indicating error has occurred during transmission

46

THE FEISTEL FUNCTION

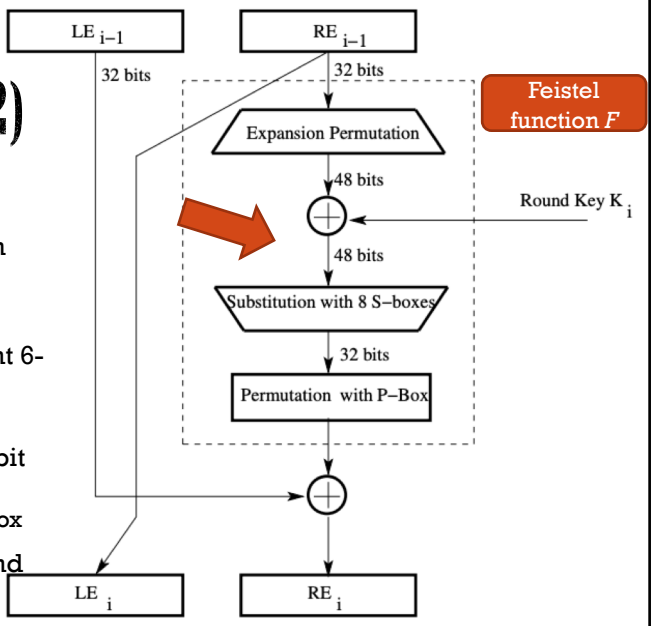
FEISTEL FUNCTION (1)

- The 32-bit right half of the 64-bit input data block is expanded into a 48-bit block
  - **Expansion permutation step**
  1. divide the 32-bit block into eight 4-bit words
  2. attach an additional bit on the left to each 4-bit word that is the last bit of the previous 4-bit word
  3. attach an additional bit to the right of each 4-bit word that is the beginning bit of the next 4-bit word
- The 56-bit key is divided into two halves, each half shifted separately, and the combined 56-bit key permuted/contracted to yield a 48-bit round key
  - Details will be given later



FEISTEL FUNCTION (2)

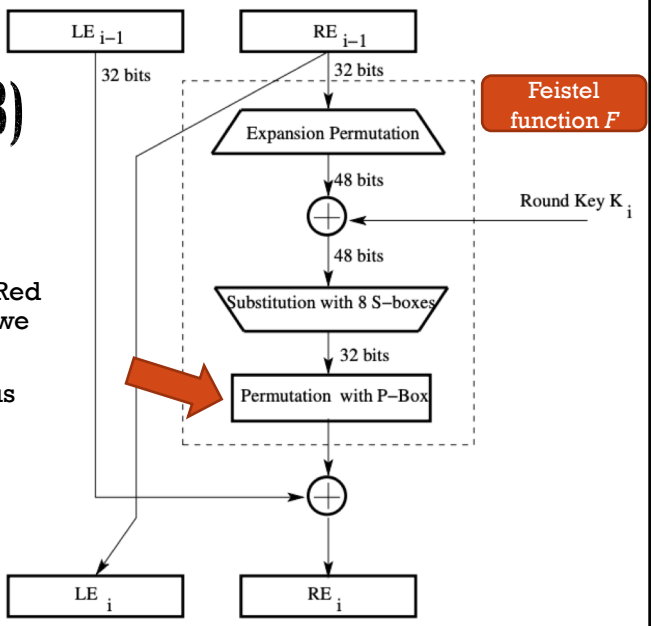
- The 48 bits of the expanded output produced by the Expansion permutation step are xor-ed with the round key
  - This is called **key mixing**
- The output produced is broken into eight 6-bit words
- Each six-bit word goes through a substitution step; its replacement is a 4-bit word
  - The substitution is carried out with an S-box
- So after all the substitutions, we again end up with a 32-bit word

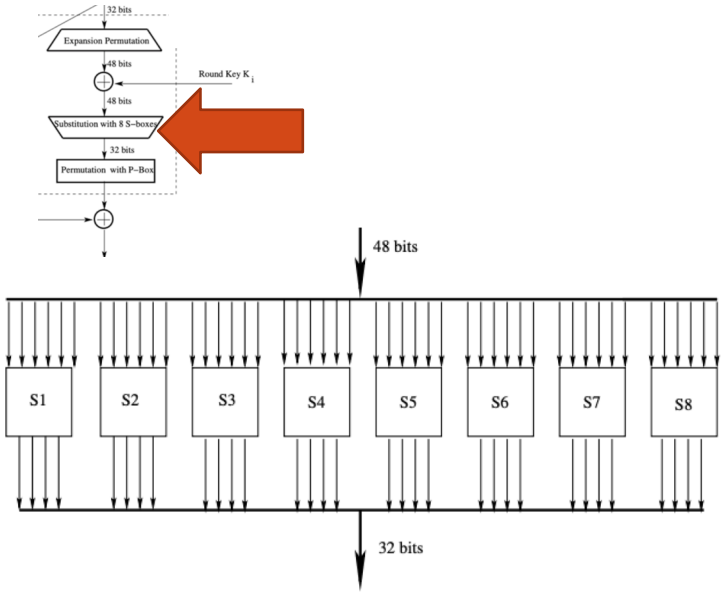


FEISTEL FUNCTION (3)

- The 32-bits of the previous step then go through a P-box based permutation
- What comes out of the P-box is then XORed with the left half of the 64-bit block that we started out with
- The output of this XOR operation gives us the right half block for the next round

- The goal of the substitution step implemented by the S-box is to introduce diffusion in the generation of the output from the input
- The strategy used for creating the different round keys from the main key is meant to introduce confusion into the encryption process (we will see how below)



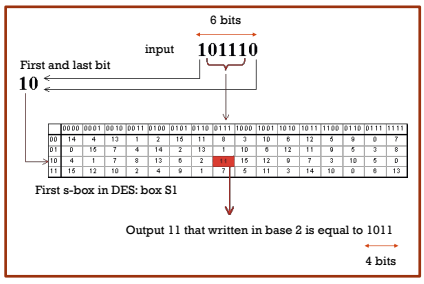


FEISTEL: S-BOX (1)

- The 48-bit input word is divided into eight 6-bit words
- Each 6-bit word fed into a separate S-box
- Each S-box produces a 4-bit output
- The 8 S-boxes together generate a 32-bit output
- Recall that the input 48 bits are obtained by xor-ing the output from the expansion permutation step with the round key

FEISTEL: S-BOX (2)

- Each of the eight S-boxes consists of a 4 × 16 table lookup for an output 4-bit word
- The first and the last bit of the 6-bit input word are decoded into one of 4 rows and the middle 4 bits decoded into one of 16 columns for the table lookup
- The goal of the substitution carried out by an S-box is to **enhance diffusion**
- The expansion-permutation step expands a **32-bit block into a 48-bit block** by **attaching a bit at the beginning and a bit at the end of each 4-bit sub-block**
  - the 2 bits for these attachments belong to the adjacent blocks
- Note that the row lookup for each of the eight S-boxes becomes a function of the input bits for the previous S-box and the next S-box



DIGRESSION ON DIFFERENTIAL ATTACKS (1)

Recall that in a chosen plaintext attack the attacker must be able to obtain ciphertexts for some set of plaintexts of their choosing

- The S-boxes were tuned to enhance the resistance of DES to **differential attacks**
  - It is an instance of a **chosen plaintext attack**
- Differential cryptanalysis of block ciphers consists of presenting to the encryption algorithm pairs of plaintext bit patterns with known differences between them and examining the differences between the corresponding ciphertexts
- Typically the notion of difference between two plaintexts or ciphertexts is the XOR of the bits performed position-wise
- Let X1 and X2 be two plaintexts related by a constant difference $\Delta X = X1 + X2$ where + is the bitwise xor operator
- The attacker computes the difference $\Delta Y = Y1 + Y2$ of the ciphertexts Y1 and Y2 corresponding to the plaintexts X1 and X2, respectively

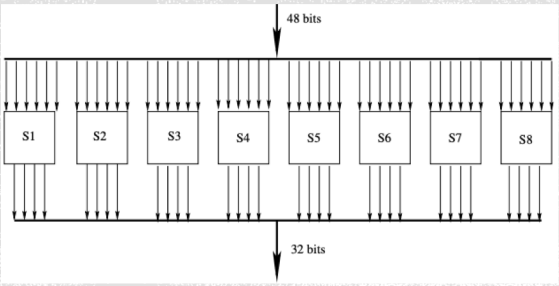
DIGRESSION ON DIFFERENTIAL ATTACKS (2)

- In an ideally randomizing block cipher, the probability of ΔY being a particular value for a given ΔX is $1/(2^n)$ for an n-bit block cipher
- The probability of ΔY taking on different values for a given ΔX can be shown to be
  - **independent of the encryption key** (because of the properties of the XOR operator)
  - but **strongly dependent on the S-box tables**
- By feeding into a cipher several pairs of plaintext blocks with known ΔX and observing the corresponding ΔY, it is possible to discover parts of the round keys
- **Block ciphers should be shown resistant to this class of attacks**
  - DES is surprisingly robust to differential attacks
  - It is believed that such an attack was known when designing DES…
- If you want to have a deeper look at this kind of attacks, you may consider to read the following tutorial introduction to (linear and) differential attacks:

  http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf

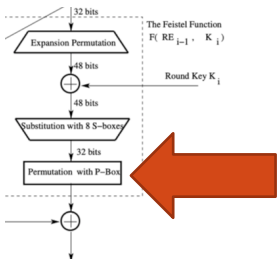
S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	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

S-BOXES: SUBSTITUTION TABLES FOR DES



FEISTEL: P-BOX

- This table should be read as follows
  - the 0th output bit will be the 15th bit of the input
  - the 1st output bit the 6th bit of the input
  - and so on for all of the 32 bits of the output that are obtained from the 32 bits of the input
- Each row of the table specifies how to select the input bits for the output byte corresponding to the row
  - Consider the second output byte
  - The first entry in the second row means that the 0th bit of the second output byte (or, equivalently, the 8th bit of the output) will be the 0th bit of the 32-bit input
  - …



P-Box Permutation								
15	6	19	20	28	11	27	16	1st byte
0	14	22	25	4	17	30	9	2nd byte
1	7	23	13	31	26	2	8	4th byte
18	12	29	5	21	10	3	24	5th byte

56

KEY SCHEDULING

DES KEY SCHEDULING (1)

- The 56-bit encryption key is represented by 8 bytes, with the least significant bit of each byte used as a parity bit
- The relevant 56 bits are subject to a permutation before any round keys are generated
  - This is called **Key Permutation 1** (see right)
- The bit indexing is based on using the range 0-63 for addressing the bit positions in an 8-byte bit pattern in which the last bit of each byte is used as a parity bit.
  - Each row has only 7 positions: the positions corresponding to the parity bit are not included above, i.e. position indexes 7, 15, 23, 31, 39, 47, 55, 63 are not shown
- The table specifies that
  - the 0th bit of the output will be the 56th bit of the input (in a 64 bit representation of the 56-bit encryption key)
  - the 1st bit of the output the 48th bit of the input,
  - … and so on, until we have for the 55th bit of the output the 3rd bit of the input

Initial permutation

Key Permutation 1						
56	48	40	32	24	16	8
0	57	49	41	33	25	17
9	1	58	50	42	34	26
18	10	2	59	51	43	35
62	54	46	38	30	22	14
6	61	53	45	37	29	21
13	5	60	52	44	36	28
20	12	4	27	19	11	3

DES: KEY SCHEDULING (2)

- At the beginning of each round
  - we divide the 56 relevant key bits into two 28 bit halves and
  - circularly shift to the left each half by one or two bits, depending on the round, according to the table on the right

Round number	Number of left shift
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

DES: KEY SCHEDULING (3)

The key permutation with the one-bit or two-bit rotation of the two key halves prior to each round (previous slide) aims to ensure that each bit of the original encryption key is used in roughly 14 of the 16 rounds

- For generating the round key, we glue together the two halves and apply a 56 bit to 48 bit contracting permutation (this is referred to as **Permutation Choice 2**) to the joined bit pattern
  - The resulting 48 bits constitute the round key
- **Key permutation 2**
  - The bit addressing now spans the 0 through 55 index values for the 56 bit key. Out of this index range, the permutation shown above retains only 48 bits for the round key. Since there are only six rows and there are 8 positions in each row, the output will consist of 48 bits.
  - As for the permutation tables above, what is shown on the right is not a table, in the sense that the rows and the columns do not carry any special and separate meanings
    - The permutation order for the bits is given by reading the entries shown from the upper left corner to the lower right corner

Key Permutation 2							
13	16	10	23	0	4	2	27
14	5	20	9	22	18	11	3
25	7	15	6	26	19	12	1
40	51	30	36	46	54	29	39
50	44	32	47	43	48	38	55
33	52	45	41	49	35	28	31

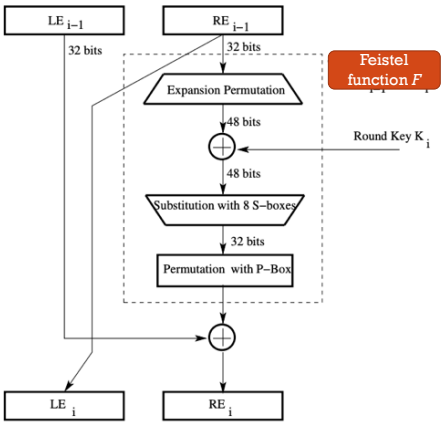
60

REMARKS

SOME REMARKS ON DES (1)

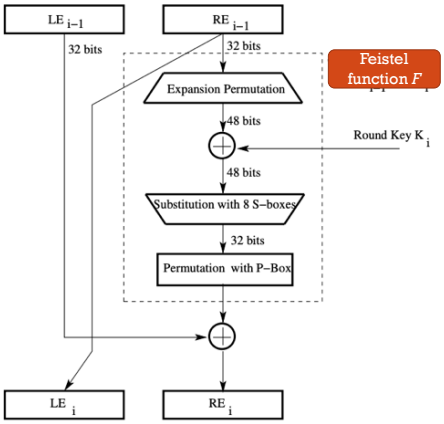
- The substitution step is very effective in supporting **diffusion**
  - If one **changes just one bit of the 64-bit input** data block, on the average it **propagates out to affect 34 bits of the ciphertext block**
- The manner in which the round keys are generated from the encryption key is also very effective in supporting **confusion**
  - If one **changes just one bit of the encryption key**, on the average that **affects 35 bits of the ciphertext**

Avalanche effect



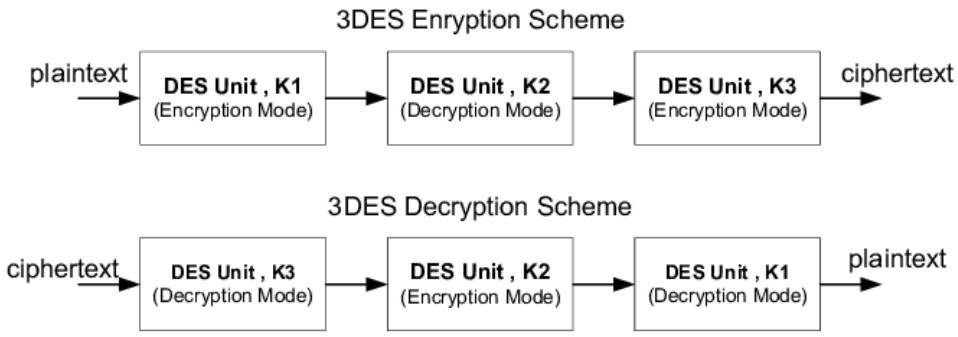
SOME REMARKS ON DES (2)

- The 56-bit encryption key means a **key space** of size 2^(56) ≈ 7.2 × 10^(16)
- In a brute-force attack, a machine able to **process 1,000 keys per microsecond** (i.e. 10^(-6) second) would need roughly **13 months** to break the code
  - Estimate based on **average case** reasoning: to discover the key, one would need to try **half the keys**
- A **parallel-processing machine trying 1 million keys simultaneously would need only about 10 hours**
- The EFF took three days on a machine with a special architecture to break the code
  - https://web.archive.org/web/20170507231657/https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html



BREAKING DES

- DES was broken in 1999
- NIST issued a new directive that year that required organizations to use **Triple DES**, i.e. **3 consecutive applications of DES**
- The attack to DES prompted NIST to initiate the development of new standards for data encryption
  - The result is AES that we will see later



TRIPLE DES

64

TRIPLE DES: OVERVIEW

- Triple DES continues to enjoy wide usage in commercial applications even today
  - In the electronic payment industry, standards continue to use Triple DES
  - **Example**: EMV (Europay, Mastercard, and Visa)
    - Standard for smart payment cards and for payment terminals and automated teller machines which can accept them
- To understand Triple DES, one needs to understand why it is not sufficient to consider only Two DES or Double DES…

65

TWO DES AND MEET-IN-THE-MIDDLE (1)

- A naive approach to increase the strength of a block cipher with short keys is to use two keys *k1* and *k2* instead of one and encrypt the block twice with them
- The hope is that the scheme will provide the same security of using a key of length *k1+k2* when the two keys have lengths *k1* and *k2* by abuse of notation
- This is not the case because of the **meet-in-the-middle** attack
  - Instead of performing *2^(k1+k2)* guesses…
  - Let $C = ENC_{k_2}(ENC_{k_1}(P))$
    $P = DEC_{k_1}(DEC_{k_2}(C))$
  - Decrypt the ciphertext *C* with *k2* to derive the following equalities:

$$C = ENC_{k_2}(ENC_{k_1}(P)) \quad \text{[apply decryption with k2 to both sides]}$$
$$DEC_{k_2}(C) = DEC_{k_2}(ENC_{k_2}[ENC_{k_1}(P)]) \quad \text{[simplify right hand side]}$$
$$DEC_{k_2}(C) = ENC_{k_1}(P)$$

TWO DES AND MEET-IN-THE-MIDDLE (2)

- Now, considering the two sides of the equality $DEC_{k_2}(C) = ENC_{k_1}(P)$, the attacker can compute
  - $DEC_{k2}(C)$ for all possible values of k_2
  - $ENC_{k1}(P)$ for all values of k_1 and

  for a total of $2^{k1} + 2^{k2}$ or 2^{k1+1} if k_1 and k_2 are the same size operations
- If the result from any of the $ENC_{k1}(P)$ operations matches a result from the $DEC_{k2}(C)$ operations, the pair of k_1 and k_2 is possibly the correct key, called a **candidate key**
- The attacker can determine which candidate key is correct by testing it with a second test-set of plaintext and ciphertext
- Contrast the complexity $2^{k1} + 2^{k2}$ of this attack with that of the brute force method requiring $2^{k1+k2} = 2^{k1} * 2^{k2}$

TRIPLE DES

- To avoid meet-in-the-middle attacks…
- Triple DES uses a **key bundle** comprising **3 DES keys *(K1, K2, K3)***, each one of 56 bits length
- **Encryption** works as follows

$$E_{K3}(D_{K2}(E_{K1}(\text{plaintext})))$$

  - i.e. perform encryption with *K1*, followed by a decryption with *K2* and then encryption with *K3*

In each case the middle operation is the reverse of the first and last

- **Decryption** is the inverse, namely

$$D_{K1}(E_{K2}(D_{K3}(\text{ciphertext})))$$

  - i.e. perform decryption with *K3*, followed by an encryption with *K2*, and then decryption with *K1*
- **Each triple encryption works on one block** of 64 bits of data

TRIPLE DES: CHOOSING KEYS

- Care should be put in selecting the keys in key bundle
- There are 3 possible options
  1. **All three keys are independent**
     - Strongest option, still vulnerable to meet-in-the-middle but requires *2^(2*56)* operations
  2. ***K1* and *K2* are independent but *K3=K1***
     - Similar to double DES and vulnerable to the same attack with equal complexity, **deprecated**
  3. **All three keys are equal**
     - For backward compatibility with DES, **forbidden**
- Additionally, some specific values for keys are forbidden
- With these restrictions, Triple DES has been **reapproved** with keying options 1 and 2 only although it is current best practice to use only option 1 as keys should be generated by using random generators

70

OVERVIEW OF AES

AES MAIN FEATURES

- **Block length: 128 bits**
- 3 different key lengths: 128, 192, or 256 bits
  - In the following, for simplicity, we assume the key length to be 128 bits
  - **For other key lengths, the main difference is the key scheduling algorithm**
- Encryption consists of
  - 10 rounds for 128-bit keys
  - 12 rounds for 192-bit keys
  - 14 rounds for 256-bit keys
- Except for the last round, all other rounds are identical

- Notified by NIST as a standard in 2001
- AES is a slight variation of the Rijndael cipher invented by two Belgian cryptographers Daemen and Rijmen
- In 1999, Rijndael was shortlisted with other 4 ciphers to become the successors of DES:
  - MARS from IBM
  - RC6 from RSA Security
  - Serpent by Anderson Biham, and Knudsen
  - Twofish by a team led by Schneier

AES ROUND: ENCRYPTION

Organization of an AES block of 128 bits or equivalently 16 keys (**keys are stored by column**)

State array of AES

key0	key4	key8	key12
key1	key5	key9	key13
key2	key6	key10	key14
key3	key7	key11	key15

Word of 4 keys

Each round includes

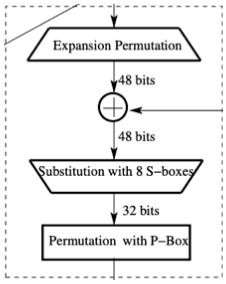
1. one single-key based substitution step
2. a row-wise permutation step
3. a column-wise mixing step
4. the addition of the round key

- The order of these steps is different for encryption and decryption
- **Each round takes an input state array and returns an output state array**
- The output state array produced by the last round is rearranged into a 128-bit output block

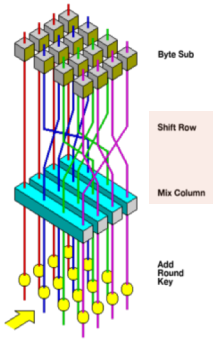
Unlike DES, decryption differs substantially from encryption although similar transformations are used

AES VS DES (1)

The Feistel function, after key mixing, first applies a substitution and then a permutation



- DES is based on the Feistel (substitution-permutation) network
  - DES involves substitutions and permutations
  - Permutations are based on the Feistel notion of dividing the input block into two halves, processing each half separately, and then swapping the two halves
- **AES uses a substitution-permutation network in a more general sense**
  - Each round in AES involves key-level substitutions followed by word-level permutations
- **Like DES**, AES is an iterated block cipher in which plaintext is subject to multiple rounds with each round applying the same overall transformation function to the incoming block



In AES round, substitution is applied first followed by two permutations (row and column wise) and then key mixing is done

AES VS DES (2)

- **Unlike DES**, AES is an example of **key-alternating** block ciphers
  - Each round first applies a diffusion-achieving transformation operation to the entire incoming block, which is then followed by the **application of the round key to the entire block**
  - DES is based on the Feistel structure in which, for each round, one-half of the block passes through unchanged and the other half goes through a transformation that depends on the S-boxes and the round key
  - Key alternating ciphers lend themselves well to theoretical analysis of the security of the ciphers
- The substitution step in DES requires bit-level access to the block coming into a round whereas all operations in AES are purely key-level (which is good for fast software implementations)
  - DES is a bit-oriented cipher
  - AES is a key-oriented cipher

ON THE SECURITY OF AES

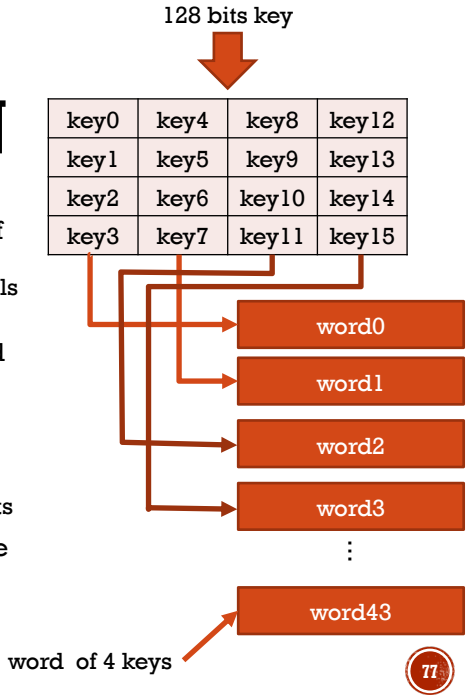
- **Since its introduction in 2001, all of the threats against the cipher remain theoretical**, i.e. their time complexity is way beyond what any computer system will be able to handle for a long time to come
- For the 128-bit key AES, the worst-case time complexity for a brute-force attack would be 2^(128) which is beyond any practical implementation
- A meet-in-the-middle attack is marginally better with 2^(126) which is also beyond any practical implementation
  - Bogdanov, Khovratovich, Rechberger. *Biclique Cryptanalysis of the Full AES*. 2011
- Many other attacks have been attempted based on algebraic techniques…
- For full details on the AES design, the standard is available at https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

76

STRUCTURE

PREVIEW OF KEY EXPANSION

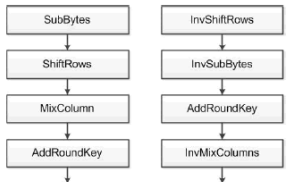
- The 128 bits key is arranged in the form of an array of 4 * 4 keys:
  - Similarly to the input block, the first word from the key fills the first column of the array, and so on
- The four column words of the key array are expanded into a schedule of 44 words
  - Details will be given later
- Each round consumes four words from the key schedule
  - Recall that AES has 10 rounds with a key length of 128 bits
- The first 4 words are used for adding to the input state array before any round can begin
- The remaining 40 words are used for the 10 rounds



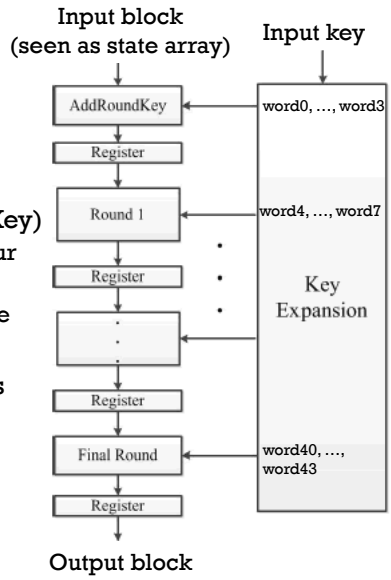
ABSTRACT ARCHITECTURE

- Recall that key expansion generates 44 keys
- The first 4 are mixed with the input blocks (AddRoundKey)
  - **Encryption**: the input state array is xor-ed with the first four words of the key schedule.
  - **Decryption**: same as above except that the ciphertext state array is xor-ed with the last four words of the key schedule
- The remaining 40 are used in each one of the 10 rounds
  - 4 at a time
- Structure of each round

Similar transformations are adopted in each round for encryption and decryption but they are not identical (need to consider inverses and also in different order) as it was the case for DES



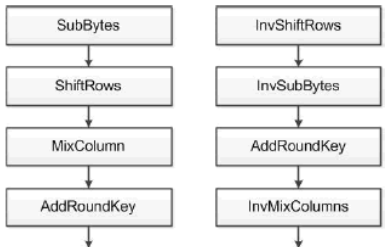
Encryption Decryption



78

ON THE STRUCTURE OF A ROUND

- Encryption steps:
  1. Substitute bytes
  2. Shift rows
  3. Mix columns
  4. Add round key
     - It consists of xor-ing the output of the previous 3 steps with 4 words from the key expansion/schedule



Encryption Decryption

- Decryption steps
  1. Inverse shift rows
  2. Inverse substitute bytes
  3. Add round key
  4. Inverse mix columns

- Note the differences between the order in which substitution and shifting operations are carried out in a decryption round vis-a-vis the order in which similar operations are carried out in an encryption round.
- The **final round for encryption** does not involve the "Mix columns" step
- The **final round for decryption** does not involve the "Inverse mix columns" step

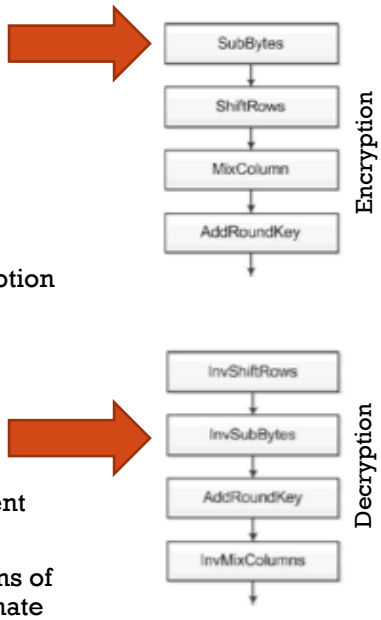
79

80

ROUND STEPS

ROUND STEP 1

- SubBytes for **byte-by-byte substitution** during encryption
- InvSubBytes for the corresponding substitution during decryption



- It consists of using a **16 × 16 lookup table** to find a replacement byte for a given byte in the input state array
- The entries in the lookup table are created by using the notions of multiplicative inverses in *GF(2^8)* and bit scrambling to eliminate the bit-level correlations inside each byte

81

ROUND STEP 1: INTRODUCTION

- To find the substitute byte for a given input byte, **divide the input byte into two 4-bit patterns**, each yielding an integer value between 0 and 15
  - It is possible to represent these integers by their hex values 0 through F
- One of the hex values is used as a **row index** and the other as a **column index** for selecting one cell in a **16 × 16 lookup table**

Dec.	Hex.	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
Dec.	Hex.	Binary
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

CONSTRUCTING THE 16×16 LOOKUP TABLE (1)

- Fill each cell of the 16 × 16 table with the byte obtained by joining together its row index and the column index
  - Both the row and column indexes of this table range from hexadecimal 0 to hexadecimal F
- Here is a part of the table

	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	10	11	12	13	14	15	16	17	18	19

…

CONSTRUCTING THE 16×16 LOOKUP TABLE (2)

- Then, **replace the value in each cell by its multiplicative inverse in *GF(2^8)* based on the irreducible polynomial *x^8 + x^4 + x^3 + x + 1***
  - The hex value 00 is replaced by itself since it has no multiplicative inverse
- Represent a byte stored in a cell of the table by *b7b6b5b4b3b2b1b0* where *b7* is the MSB and *b0* the LSB
  - **Example**: the byte stored in the cell (9, 5) of the lookup table is the multiplicative inverse of 95, i.e. 8A
  - The polynomial representation of 95 (bit pattern: 10010101) is x^7 + x^4 + x^2 + 1
  - The polynomial representation of 8A (bit pattern: 10001010) is x^7 + x^3 + x
  - Exercise: show that the product of the two polynomials modulo the irreducible polynomial x^8 + x^4 + x^3 + x + 1 is indeed 1

CONSTRUCTING THE 16×16 LOOKUP TABLE (3)

- **Bit scrambling**: apply the following transformation to each bit *bi* of the byte stored in a cell of the lookup table

$$b'_i = b_i \otimes b_{(i+4) \bmod 8} \otimes b_{(i+5) \bmod 8} \otimes b_{(i+6) \bmod 8} \otimes b_{(i+7) \bmod 8} \otimes c_i$$

where *ci* is the *i*-th bit of a specially designated byte *c* whose hex value is 63, i.e. the bit pattern is *c7c6c5c4c3c2c1c0 ≡ 01100011*

- The transformation can also be described by using matrix multiplication (xor):

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

CONSTRAINTS ON THE 16×16 LOOKUP TABLE

Requirements

1. The mapping defined by the lookup table must be invertible
   - the byte-to-byte mapping induced by the 16 × 16 table must be one-one, i.e., for each input byte, there must be a unique output byte
   - To each output byte there must correspond only one input byte
2. No input byte should map to itself, since a byte mapping to itself would weaken the cipher
   - Taking multiplicative inverses in the construction of the table does give us unique entries in the table for each input byte except for 00 (for which no multiplicative inverse exists)

Role of the constant byte *c*

- If it were not for the byte *c*, the bit scrambling step would leave the input byte 00 unchanged
- With the mapping considered above, the 00 input byte is mapped to *c* i.e. 63 and, at the same time, all other bytes are mapped one-to-one

THE LOOKUP TABLE IS THE S-BOX

- The 16 × 16 table created as described above is the S-Box
- The S-Box is the same for all the bytes in the state array
- The steps that go into constructing the 16 × 16 lookup table are reversed for the decryption table:
  - first apply the **reverse of the bit-scrambling operation** to each byte
  - then take the multiplicative inverse in *GF(2^8)*
- **Bit scrambling for decryption**: carry out the following bit-level transformation in each cell of the table

$$b'_i = b_{(i+2) \bmod 8} \otimes b_{(i+5) \bmod 8} \otimes b_{(i+7) \bmod 8} \otimes d_i$$

where *di* is the *i*-th bit of a specially designated byte *d* whose hex is 05 or, equivalently, the bit vector *d7d6d5d4d3d2d1ddc0 = 00000101*

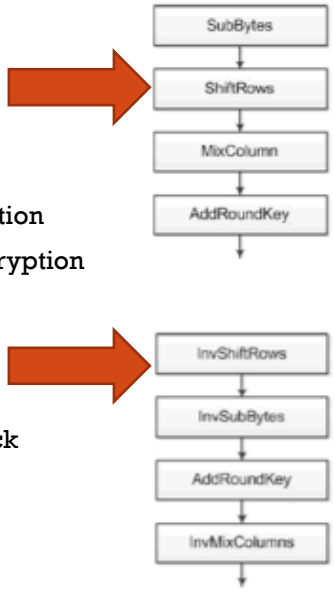
ALTERNATIVE VIEW OF STEP 1

- Let *Xin* be a byte of the state array for which we want to define a substitute byte *Xout*, i.e. *Xout = f(Xin)*
- For encryption, the function *f()* involves **two nonlinear operation**
  1. Find the multiplicative inverse *X′ = Xin^(−1)* in *GF(2^8)*
  2. Scramble the bits of X′ by xor-ing X′ with
     - four different circularly rotated versions of itself and
     - a special constant byte c = (hex) 63
  - This can be expressed as follows: $Xout = A \cdot X' + c$

$$\underset{A}{\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}} \underset{X'}{\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}} \underset{\otimes}{\text{xor}} \underset{c}{\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}}$$

- For decryption, the situation is similar except for the fact that one needs to first apply the bit scrambling operation to the byte and then find its multiplicative inverse in *GF(2^8)*

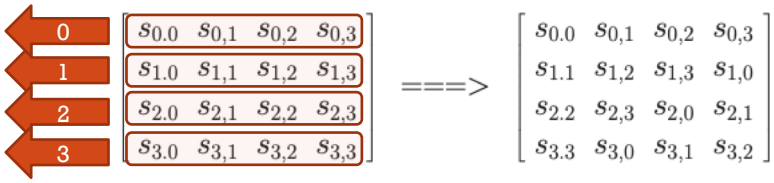
ROUND STEP 2



- ShiftRows for shifting the rows of the state array during encryption
- InvShiftRows for the corresponding transformation during decryption
- The goal is to scramble the byte order inside each 128-bit block

ROUND STEP 2: DETAILS

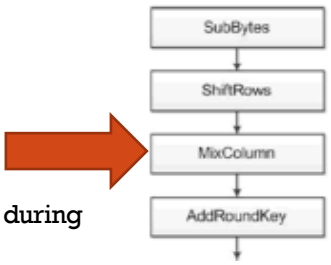
- ShiftRows transformation consists of
  1. not shifting the first row of the state array at all
  2. circularly shifting the second row by one byte to the left
  3. circularly shifting the third row by two bytes to the left
  4. circularly shifting the last row by three bytes to the left



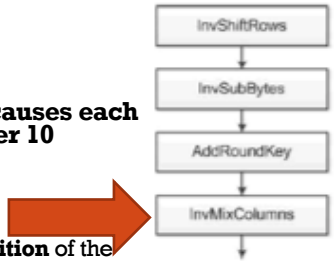
- Recall that the input block is written column-wise
  - The first four bytes of the input block fill the first column of the state array, the next four bytes the second column, etc
- Shifting the rows as indicated scrambles up the byte order of the input block
- For decryption, the corresponding step shifts the rows in exactly the opposite direction

ROUND STEP 3

Notice that we consider the 3rd step for encryption and the 4th step for decryption



- MixColumns for mixing up of the bytes in each column separately during encryption
- InvMixColumns for the corresponding transformation during decryption



- The goal is to further scramble up the 128-bit input block
- **The combination of the shift-rows and the mix-column steps causes each bit of the ciphertext to depend on every bit of the plaintext after 10 rounds of processing**
  - Recall the avalanche effect
  - In DES, one bit of plaintext affected roughly 31 bits of ciphertext
  - In AES, the goal is that each bit of the plaintext will affect every **bit position** of the ciphertext block of 128 bits

ROUND STEP 3: DETAILS

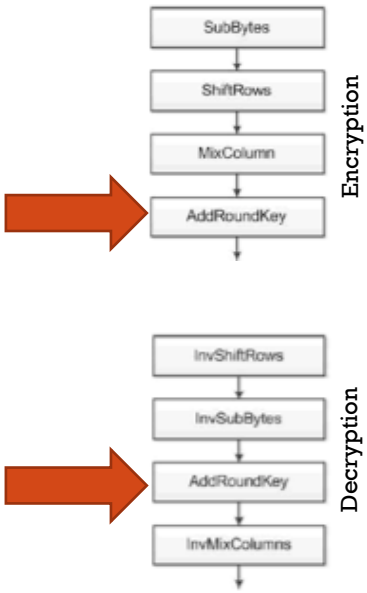
- This step replaces each byte of a column by a function of all the bytes in the same column: **each byte in a column is replaced by two times that byte, plus three times the next byte, plus the byte that comes next, plus the byte that follows**
  - The multiplications and the additions are meant to be carried out in *GF(2^8)*
  - For instance, 'two times' means multiplication by bit pattern 000000010 and 'three times' by bit pattern00000011,
  - The words **'next' and 'follow' refer to bytes in the same column, and their meaning is circular**, in the sense that the byte that is next to the one in the last row is the one in the first row
- The transformation can be compactly represented as follows:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

ROUND STEP 4

Notice that we consider the 4th step for encryption and the 3rd step for decryption

- AddRoundKey for adding the round key to the output of the previous step during encryption
- AddRoundKey or InvAddRoundKey for inverse add round key transformation during decryption



KEY EXPANSION: INTRODUCTION

- Each round has its own round key that is derived from the original 128-bit encryption key as described in the following
- One of the four steps of each round, for both encryption and decryption, involves the xor-ing of the round key with the state array
- The AES Key Expansion algorithm is used to derive the 128-bit round key for each round from the original 128-bit encryption key
- The logic of the key expansion algorithm is designed to ensure that if one changes one bit of the encryption key, it should affect the round keys for several rounds

KEY EXPANSION ALGORITHM (1)

- In the same manner as the 128-bit input block is arranged in the form of a state array, the algorithm first arranges the 16 bytes of the encryption key in the form of a 4 × 4 array of bytes
- The first four bytes of the encryption key constitute the word w0
- The next four bytes the word w1
- The next four bytes the word w2
- The last four bytes the word w3

$$\begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}$$

$$\Downarrow$$

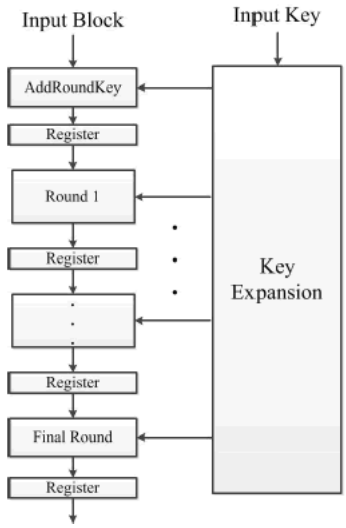
$$\begin{bmatrix} w_0 & w_1 & w_2 & w_3 \end{bmatrix}$$

KEY EXPANSION ALGORITHM (2)

- The algorithm subsequently expands the words [w0, w1, w2, w3] into a 44-word key schedule:

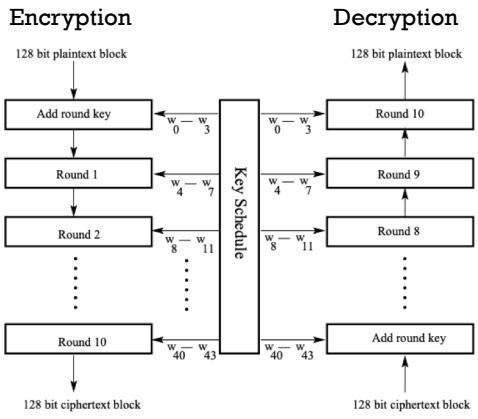
  w0, w1, w2, w3, ..., w43

- The first four words w0, w1, w2, w3 are bitwise xor-ed with the input block before the round-based processing begins
- The remaining 40 words w4, …, w43 of the key schedule are used 4 words at a time in each of the 10 rounds



KEY EXPANSION ALGORITHM (3)

- The description in the previous slide is for encryption
- However, **the same holds for decryption except for the fact that the order of the words in the key schedule is reversed**
  - The last four words of the key schedule are bitwise xor-ed with the 128-bit ciphertext block before any round-based processing begins
  - Subsequently, each of the four words in the remaining 40 words of the key schedule are used in each of the ten rounds of processing



KEY EXPANSION ALGORITHM (4)

- How does the Key Expansion Algorithm expand four words w0, w1, w2, w3 into the 44 words w0, w1, w2, w3, w4, w5, ..., w43 ?
- High-level answer
  - The key expansion takes place on a 4-word to 4-word basis, in the sense that each grouping of 4 words decides what the next grouping of 4 words will be
- Detailed answer…

KEY EXPANSION ALGORITHM (5)

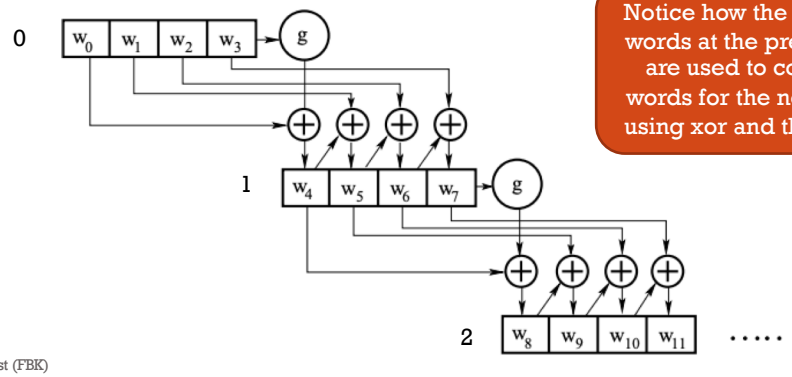
- The **core of the key expansion algorithm** is a procedure for generating the four words of the round key for a given round from the corresponding four words of the round key for the previous round
- Assume we have four words of the round key for the i-th round

$$w(i)\ w(i+1)\ w(i+2)\ w(i+3)$$

  - Notice that it must be that *i* should be divided by 4 as the four words constitute the key for the *(i/4)*-th round
  - Examples
    - w4, w5, w6, w7 form the key for round 1
    - w8, w9, w10, w11 form the key for round 2
    - …
- We are going to describe how to determine the words $w(i+4)\ w(i+5)\ w(i+6)\ w(i+7)$ from $w(i)\ w(i+1)\ w(i+2)\ w(i+3)$

KEY EXPANSION ALGORITHM (6)

- The transformation can be graphically depicted as follows for rounds 0, 1, 2,…



Notice how the values of the words at the previous round are used to compute the words for the next round by using xor and the function g

KEY EXPANSION ALGORITHM (7)

Function defined on top of the lookup tables of the SubBytes transformation

- Mathematically, the transformation is expressed as follows:

$$
\begin{aligned}
w_{i+4} &= w_i \otimes g(w_{i+3}) \\
w_{i+5} &= w_{i+4} \otimes w_{i+1} \\
w_{i+6} &= w_{i+5} \otimes w_{i+2} \\
w_{i+7} &= w_{i+6} \otimes w_{i+3}
\end{aligned}
$$

	Round i	Round i+1
g(.) →	w(i)	→ w(i+4)
	w(i+1)	→ w(i+5)
	w(i+2)	→ w(i+6)
	w(i+3)	→ w(i+7)

- Except for the first word in a new 4-word grouping, each word is obtained by xor-ing the previous word and the corresponding word in the previous 4-word grouping
- The first word of the new 4-word grouping is obtained by xor-ing the first word of the last grouping with what is returned by applying a function *g()* to the last word of the previous 4-word grouping

KEY EXPANSION ALGORITHM (8)

The addition of the round constants is for the purpose of destroying any symmetries that may have been introduced by the other steps in the key expansion algorithm

- The function *g()* consists of the following three steps:
  1. Perform a one-byte left circular rotation on the argument 4-byte word
  2. Perform a byte substitution for each byte of the word returned by the previous step by using the same 16 × 16 lookup table as used in the SubBytes step of the encryption rounds
  3. xor the bytes obtained from the previous step with a round constant
     - The round constant is a word whose three rightmost bytes are always zero
     - xor-ing with the round constant amounts to xor-ing with just its leftmost byte
- The round constant for the *i*-th round is *Rcon[i]* and is defined as follows
  - *Rcon[i] = (RC[i], 00, 00, 00)*

    where RC is such that *RC[1] = 01* and *RC[j] = 02 × RC[j − 1]*
  - Observe that the multiplication by 02 can be interpreted as a multiplication by x of the polynomial corresponding to *RC[j − 1]*

A NOTE ON KEYS

- The nice property of the **128-bit key** is that one can think of the **key expansion being in one-to-one correspondence with the rounds**
- **This is no longer the case with 192-bit k**eys and one has to think of key expansion as something that is not in synch from round-based processing of the input block
- The key expansion algorithm ensures that AES has no weak keys
- A weak key reduces the security of a cipher in a predictable manner
  - Example
    - DES is known to have weak keys (e.g., alternating ones and zeros) that are those that produce identical round keys for each of the 16 rounds
    - This causes all the round keys to become identical, which, in turn, implies the encryption to become self-inverting, i.e. plain text encrypted and then encrypted again will lead back to the same plain text

SECURITY OF AES

- Cryptographers are constantly probing AES for weaknesses
- This is essential, because if it was not being thoroughly tested by academics, then criminals or nation states could eventually find a way to crack it without the rest of the world knowing
- So far, **researchers have only uncovered theoretical breaks and side channel attacks**

SOME ATTACKS (1)

- In 2009, a series of **related-key** attacks were discovered
  - These involve observing how a cipher operates under different keys
  - These attacks are only possible against protocols that are not implemented properly
  - More information at http://www.cs.haifa.ac.il/~orrd/RK-Attacks.pdf
- In 2009, there was a **known-key distinguishing** attack against an **eight round version** of AES-128
  - This attack uses a key that is already known in order to figure out the inherent structure of the cipher
  - As this attack was only against an eight round version, there is not much to worry about for everyday users of AES-128
  - More information at https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=31551

SOME ATTACKS (2)

- There have been several other theoretical attacks, but under current technology they would still take billions of years to crack
- This means that **AES is essentially unbreakable at the moment**
- Despite this, AES can still be vulnerable **if it is not implemented properly**, especially with respect to a **side-channel attack**
- Brute force attacks on keys are out of reach for the moment especially if we consider AES-256
  - The size of the key space is 2^(256) i.e. around 1.15*10^(77)

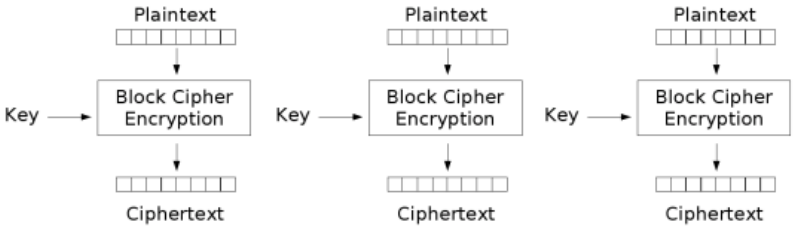
107

MODES OF OPERATIONS FOR BLOCK CIPHERS

https://csrc.nist.gov/publications/detail/sp/800-38a/final

SECURE USE OF BLOCK CIPHERS?

- These observations apply to all block ciphers including AES
- Just because a block cipher has been demonstrated to be strong as it is the case of AES, this does not imply that it will be sufficiently secure if it is used to transmit long (i.e. many times longer than the block size) messages
- The interaction between the block-size based periodicity of such ciphers and **any repetitive structures in the plaintext** may still leave too many clues in the ciphertext that compromise its security
- There are **5 different modes** in which any block cipher can be used
  - The first (called ECB) is for using a block cipher as it is, i.e. by scanning a long document one block at a time and enciphering it independently of the blocks seen before or the blocks to be seen next
  - This is not suitable for long messages
  - The remaining four modes, that are variations on the first, are actually used in real-world applications for the encryption of long messages

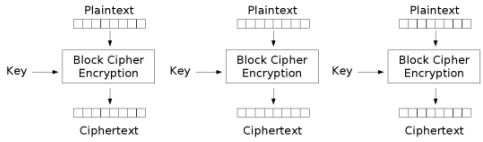


Electronic Codebook (ECB) mode encryption

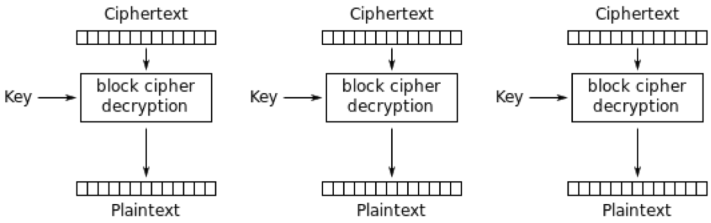
1. ELECTRONIC CODE BOOK MODE (ECB)

- When a block cipher is used in ECB mode, each block of plaintext is coded independently
- This makes it **not very secure for long segments of plaintext**, especially plaintext containing repetitive information
- Primarily used for secure transmission of **short pieces of information**, e.g., encryption keys

ECB DECRYTION



Electronic Codebook (ECB) mode encryption



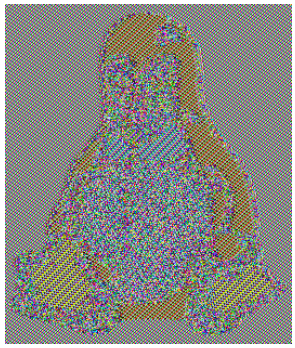
Electronic Codebook (ECB) mode decryption

MAIN PROBLEM WITH ECB MODE (1)

- When each block of a plaintext file is encrypted independently of the other blocks, the "**structure**" of the information **in the ciphertext file can hold important clues to what is in the plaintext file**
- *Example*

Linux penguin mascot known as "Tux"

AES with ECB

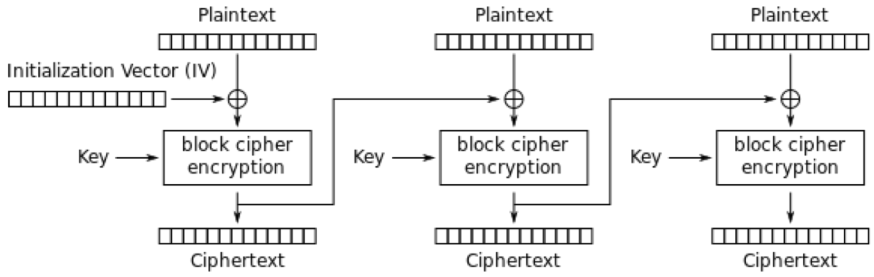


MAIN PROBLEM WITH ECB MODE (2)

- Given a key, a *block cipher* always encrypts the same contents the same way
- At first, this does not seem to be a problem because the output is still encrypted but it reveals information
- In the previous picture, we can see that two different blocks in the encrypted message are in fact the same, even if we do not know their contents
- The penguin picture emphasizes this effect as cartoon images tend to have regions where all the colours are the same, and hence, will encrypt the same when using a block cipher
- In the previous picture, large regions of the cartoon are either completely white or completely black
- As a result, the image is still recognizable as a penguin

REMARKS ON ECB MODE

- ECB mode block encryption can leave too many clues in the ciphertext for an attacker
- As a consequence, the **ECB mode is good only for short messages or messages without too much repetitive structure**
- Another shortcoming of ECB is that **the length of the plaintext message must be integral multiple of the block size**
- When that condition is not met, the plaintext message must be padded appropriately.

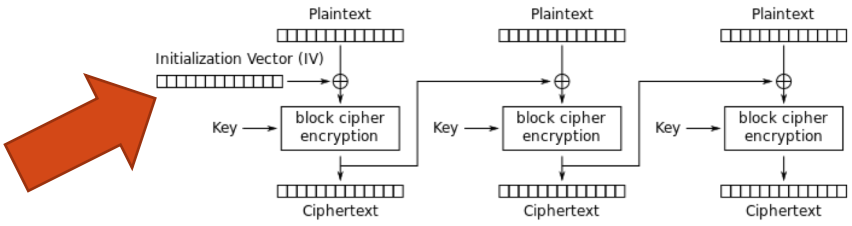


Cipher Block Chaining (CBC) mode encryption

2. CIPHER BLOCK CHAINING MODE (CBC)

- To overcome the security problems of the ECB mode, the input to the encryption algorithm consists of the xor of the plaintext block and the ciphertext produced from the previous plaintext block
- This makes it difficult to identify patterns in the ciphertext that may correspond to the known structure of the plaintext

CBC MODE

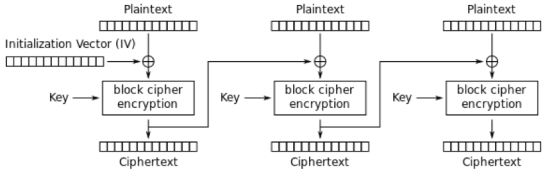


Cipher Block Chaining (CBC) mode encryption

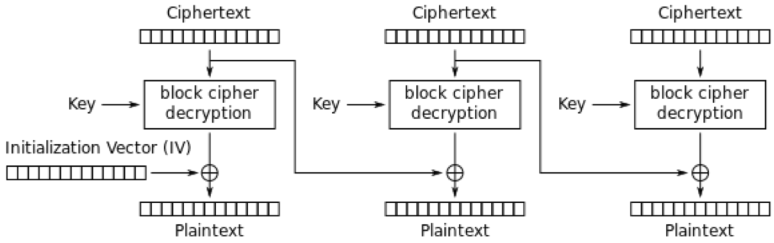
- The chaining scheme shown in the figure needs an **Initialization Vector** (IV) for the first invocation of the encryption algorithm
- The IV is sent separately as a short message using the ECB mode

- With CBC, the **ciphertext block for any given plaintext block becomes a function of all the previous ciphertext blocks**
  - This **is similar to stream ciphers except for the fact that it is the ciphertext (and not the plaintext) of the previous block that is xor-ed with plaintext of the current block**
  - Below, we will see another mode that is much closer to stream ciphers…

CBC MODE DECRYPTION



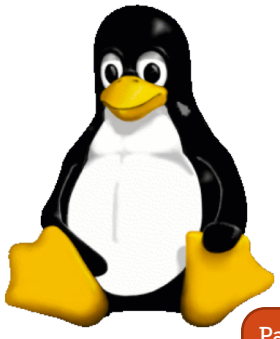
Cipher Block Chaining (CBC) mode encryption



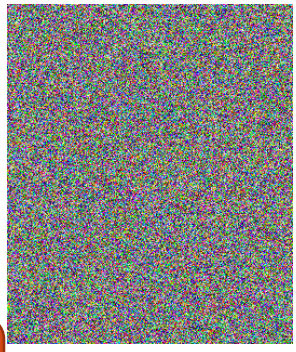
Cipher Block Chaining (CBC) mode decryption

BACK TO THE PENGUIN ICON

Small modifications in previous blocks end up in very different ciphertext… cancelling any structure!

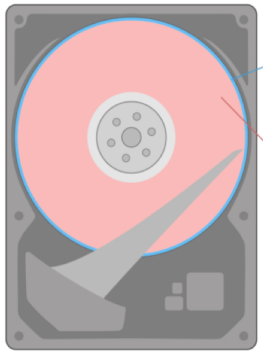


AES with CBC



Pay attention to the value of the IV, if it is predictable (e.g., constant) then the effectiveness of the chaining idea is lessened

REMARK ON CBC

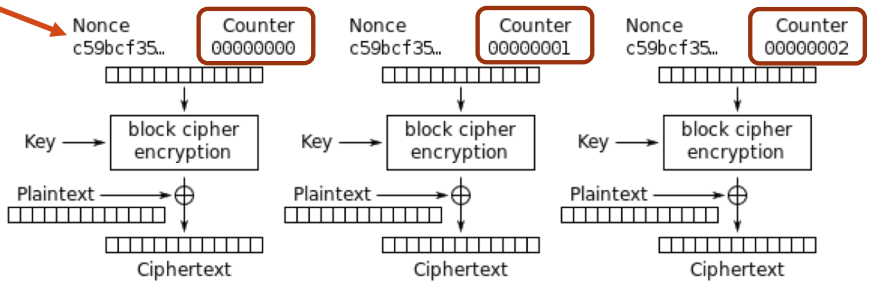


File Metadata (Stored in File System)
- Name
- Folder Path
- Owner
- Size
- Tags / EXIF Metadata
- Location on Disk
- Date Created/Modified
- Author
- Permissions
- Recycled Y/N
- Deleted Y/N
- Etc

File Data (Actual Contents of File Stored on Disk)
- Block 1
- Block 2
- Block 3
- Block 4
- Block 5
- Block 6
- Block 7
- Block 8
- Block 9
- .....
- .....
- Block N

- The idea of chaining becomes much less appealing in certain use case scenarios, e.g., when considering encryption disk drives that have "random access" and need to randomly change data in the middle of the drive
- This is where other modes might be more interesting such as the one in the next slides…

Nonce plays a similar role to the IV in previous mode



Counter (CTR) mode encryption

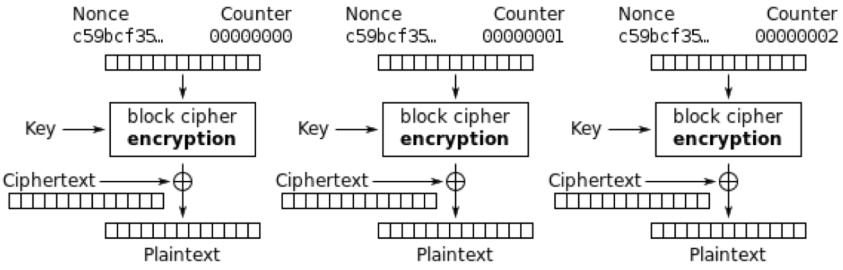
3. COUNTER MODE (CTR)

The CTR mode is similar to the Output FeedBack (OFB) mode that we will see below

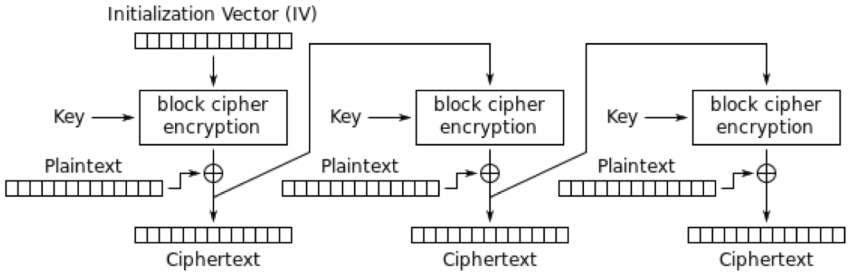
- In the use case scenario considered above: every block on the disk is assigned a different number so that you can seek to that location on the disk and decrypt just that block without having to preliminarily decode all the blocks that came before it

CTR DECRYPTION

- Notice that only the encryption algorithm is used in both encryption and decryption
- This can be an important implementation-level detail for those block ciphers for which the encryption and the decryption algorithms are significantly different such as AES



Counter (CTR) mode decryption



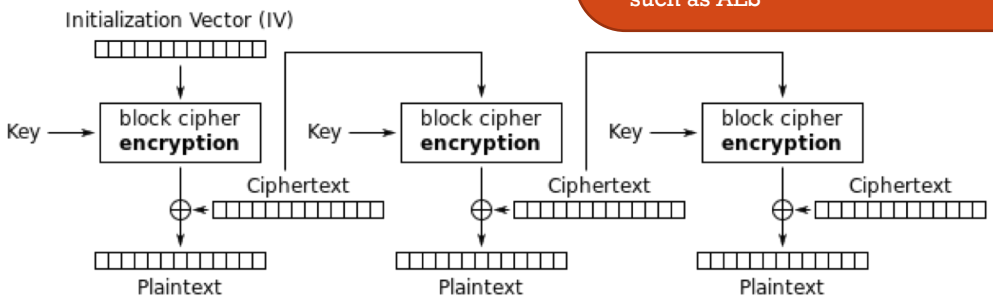
Cipher Feedback (CFB) mode encryption

4. CIPHER FEEDBACK MODE (CFB)

- The schema is very similar to a stream cipher

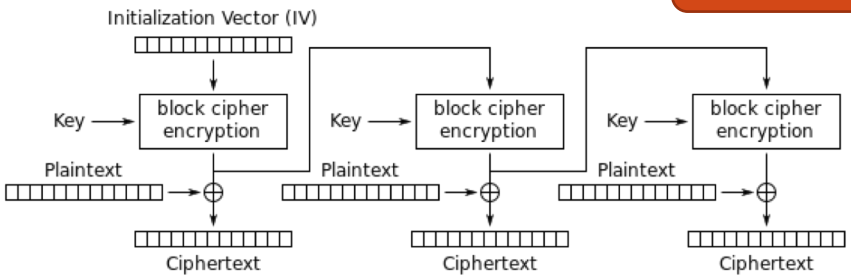
CFB MODE DECRYPTION

- Notice that only the encryption algorithm is used in both encryption and decryption
- This can be an important implementation-level detail for those block ciphers for which the encryption and the decryption algorithms are significantly different such as AES



Cipher Feedback (CFB) mode decryption

It turns out to be compatible with error correcting codes



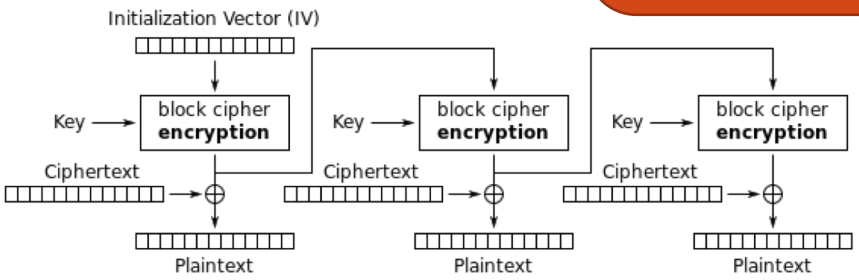
Output Feedback (OFB) mode encryption

5. OUTPUT FEEDBACK MODE (OFB)

- Very similar to the CFB mode
  - I.e. quite similar to a stream cipher
- Main difference: feed back of only the MSB from the output of the block cipher encryption algorithm, as opposed to feeding back the actual ciphertext byte

OFB MODE DECRYPTION

- As for CFB, only the encryption algorithm is used in both encryption and decryption
- This can be an important implementation-level detail for those block ciphers for which the encryption and the decryption algorithms are significantly different such as AES



Output Feedback (OFB) mode decryption

125

GALOIS/COUNTER (GCM) MODE

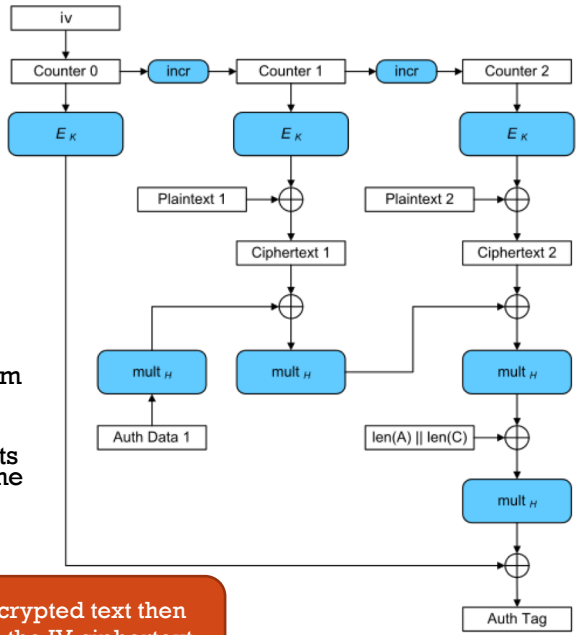
Widely used today in many protocols such as TLS

INTRODUCTION

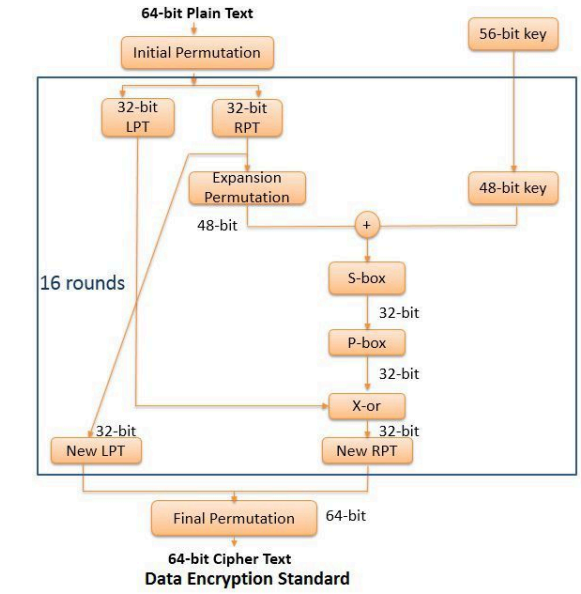
- Galois/Counter Mode (GCM) is a mode of operation for symmetric-key cryptographic block ciphers widely adopted for its **performance**
- GCM throughput rates for state-of-the-art, high-speed communication channels can be achieved with inexpensive hardware resources
- GCM is defined for block ciphers with a block size of 128 bits
- Galois Message Authentication Code (GMAC) is an authentication-only variant of the GCM which can form an incremental message authentication code
- Both GCM and GMAC can accept initialization vectors of arbitrary length

OVERVIEW

- Like in CTR, blocks are numbered sequentially
- The block number is combined with an IV and encrypted with a block cipher (usually AES)
- The result of this encryption is xor-ed with the plaintext to produce the ciphertext
- Like all counter modes, this is essentially a stream cipher, and so it is essential that a different IV is used for each stream that is encrypted
- The ciphertext blocks are considered coefficients of a polynomial which are then manipulated in the corresponding Galois field
- The result is then encrypted, producing an authentication tag that can be used to verify the integrity of the data



The encrypted text then contains the IV, ciphertext, and authentication tag



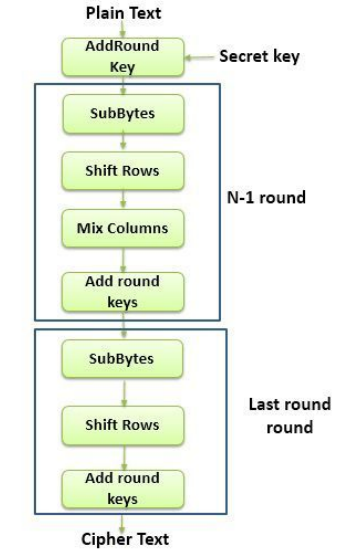
Data Encryption Standard

S. Ranise - Security & Trust (FBK)

DES: SUMMARY

- Initially, the 64-bit plaintext undergoes initial permutation which rearranges the bits to get 64-bit permuted input
- The 64 bit permuted input is divided into two halves i.e. 32-bit left portion and 32-bit right portion
- Each half undergoes sixteen rounds where each round applies the same Feistel transformation
- After completion of sixteen rounds, final permutation is applied and the 64-bit ciphertext is obtained

128



Advanced Encryption Standard

S. Ranise - Security & Trust (FBK)

AES: SUMMARY

- AES takes 128-bit plaintext and 128-bit secret key which together forms a 128-bit block which is depicted as 4 X 4 square matrix
- The 4 X 4 square matrix undergoes an initial transformation that is followed by 10 rounds
- 9 rounds are structured as follows:
  - **Subbytes** uses S-box to perform byte by byte substitution of the entire block (matrix)
  - **Shift Rows**: Rows of the matrix are shifted
  - **Mix Columns**: Columns of the matrix are shuffled from right to left
  - **Add round keys**: the Xor of the current block and the expanded key is performed
- The last (10^{th}) round involves Subbytes, Shift Rows, and Add round keys stages only and provides 16 bytes (128-bit) ciphertext

129

DES VS AES

- The basic difference between DES and AES is that the block in DES is divided into two halves before further processing whereas in AES entire block is processed to obtain ciphertext
- The DES algorithm works on the Feistel Cipher principle and the AES algorithm works on substitution and permutation principle
- The key size of DES is 56 bit which is comparatively smaller than AES which has 128,192, or 256-bit secret key
- The rounds in DES include Expansion Permutation, Xor, S-box, P-box, Xor and Swap whereas rounds in AES include Subbytes, Shiftrows, Mix columns, Addroundkeys
- DES is less secure than AES because of the small key size and the relatively small block size
- AES is comparatively faster than DES