CBCS SCHEME

USN | | | | | | | | | | |

18EC744

Seventh Semester B.E. Degree Examination, Feb./Mar. 2022
Cryptography

Time: 3 hrs. Max. Marks: 100

Note: ***Answer any FIVE full questions, choosing ONE full question from each module.***

Module-1

1 a. Draw the model of symmetric cryptosystem and explain in detail. **(08 Marks)**

b. Using Hill Cipher technique encrypt and decrypt the plain tent "Pay more money". Using the key. $\begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$ **(12 Marks)**

OR

2 a. Explain Euclidean algorithm for determining of GCD. If a = 24140, b = 16762 solve using. Euclidean algorithm to find GCD (a, b). **(08 Marks)**

b. Mention the modular arithmetic operation properties and prove the same. **(08 Marks)**

c. Find 11^7 mod 13 using modular Arithmetic. **(04 Marks)**

Module-2

3 a. With a neat diagram, explain fiestal encryption and decryption model. **(08 Marks)**

b. With a neat diagram, explain DES encryption algorithm. **(08 Marks)**

c. List the design features of fiestal network. **(04 Marks)**

OR

4 a. Explain with a neat diagram AES encryption and decryption process. **(08 Marks)**

b. Explain AES key expansion algorithm write the Pseudo code for the same. **(08 Marks)**

c. Describe the AES shift Rows Transformation. **(04 Marks)**

Module-3

5 a. What are Groups? Explain in detail with respect to its properties. **(06 Marks)**

b. Write a note on finite field of the form GF (P). **(06 Marks)**

c. Find the additive and multiplicative inverse of GF (8). **(08 Marks)**

OR

6 a. State and prove Fermat's Theorem. Also find 7^{18} mod 19 using it. **(08 Marks)**

b. With suitable explanation prove Euler's Theorem. **(07 Marks)**

c. Explain discrete logarithms for modular Arithmetic. **(05 Marks)**

Module-4

7 a. With a neat diagram, explain public-key cryptosystem secrecy and Authentication. **(10 Marks)**
b. Explain the steps involved for encryption and Decryption for RSA Algorithm. **(06 Marks)**
c. Perform encryption using RSA algorithm for $p = 5$, $q = 11$, $e = 3$, $m = 9$. **(04 Marks)**

OR

8 a. Explain Diffie-Hellman key exchange algorithm. **(07 Marks)**
b. Explain Elliptic curve over real numbers. **(07 Marks)**
c. Explain Elliptic curve cryptography. **(06 Marks)**

Module-5

9 a. Write an explanatory note on Liner Feedback shift registers. **(10 Marks)**
b. Explain the following with necessary diagrams :
i) Generalized Geffe Generator
ii) Threshold Generator
iii) Alternating stop and go generator. **(10 Marks)**

OR

10 a. Explain Additive Generators. Also explain fish and pike Additive Generator. **(10 Marks)**
b. With a neat diagram, explain the concept of Gifford. **(06 Marks)**
c. Write a short note on A5. **(04 Marks)**

* * * * *

Module 1

1 a **SYMMETRIC CIPHER MODEL**

A symmetric encryption scheme has **five** ingredients.

- **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.

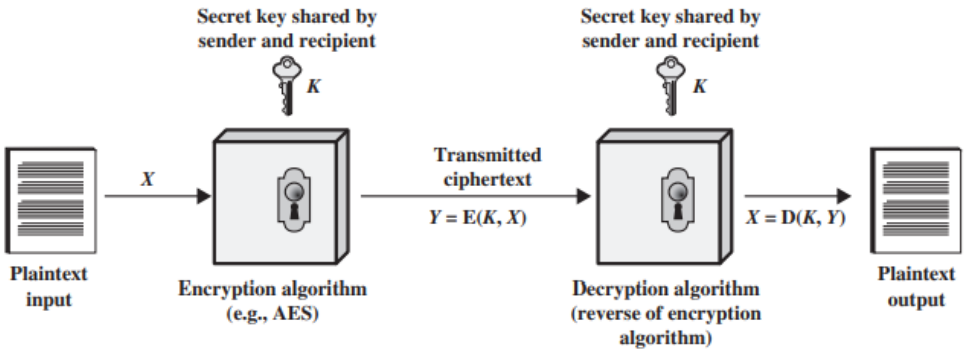


Figure Simplified Model of Symmetric Encryption

- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

There are **two** requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.
2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

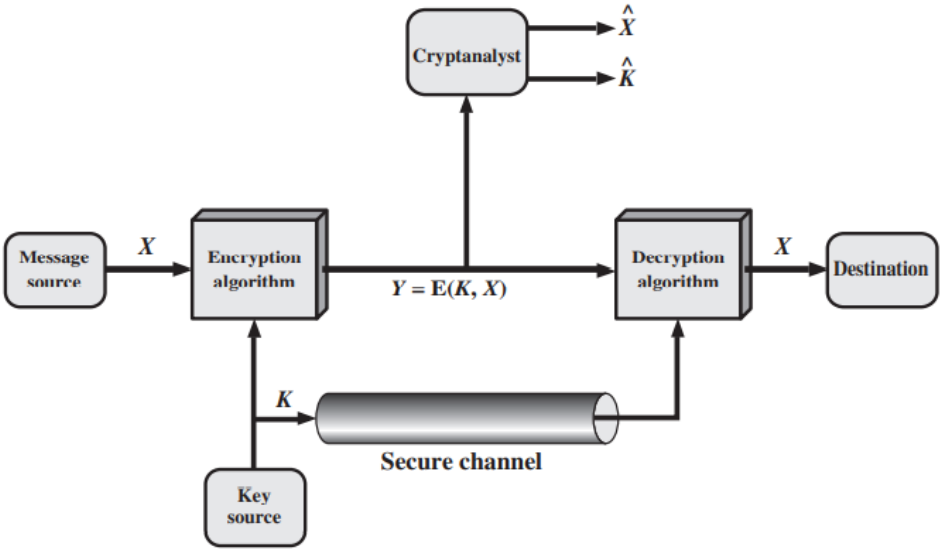


Figure Model of Symmetric Cryptosystem

We assume that it is impractical to decrypt a message on the basis of the ciphertext plus knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Let us take a closer look at the essential elements of a symmetric encryption scheme. A source produces a message in plaintext, $X = [X_1, X_2, \ldots . X_M]$. The M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet {0, 1} is typically used. For encryption, a key of the form $K = [K_1, K_2, \ldots . . K_J]$is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.

With the message X and the encryption key K as input, the encryption algorithm forms the ciphertext $Y = [Y_1, Y_2, \ldots . Y_M]$. We can write this as $Y = E(K, X)$

This notation indicates that Y is produced by using encryption algorithm E as a function of the plaintext X, with the specific function determined by the value of the key K. The intended receiver, in possession of the key, is able to invert the transformation: $X = D(K, Y)$

An opponent, observing Y but not having access to K or X, may attempt to recover X or K or both X and K. It is assumed that the opponent knows the encryption (E) and decryption (D) algorithms. If the opponent is interested in only this particular message, then the focus of the effort is to recover X by generating a plaintext estimate $\hat{X}$. Often, however, the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover K by generating an estimate $\hat{K}$.

1 b Plain text: PAY MORE MONEY

Key: $\begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$

Step1: Divide the plain text into block of 3(as here key is a 3X3 matrix)

PAY MOR EMO NEY

P	A	Y
15	0	24

M	O	R
12	14	17

E	M	O
4	12	14

N	E	Y
13	4	24

.

If it is not possible to make a group then add some filler letters 'X' to complete the group.

Step-2:

$C = KP \ mod \ 26$

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \times \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} mod \ 26$$

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix} \times \begin{bmatrix} 15 & 12 & 4 & 13 \\ 0 & 14 & 12 & 4 \\ 24 & 17 & 14 & 24 \end{bmatrix} mod 26$$

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} = \begin{bmatrix} 375 & 527 & 342 & 409 \\ 819 & 861 & 594 & 849 \\ 486 & 375 & 298 & 490 \end{bmatrix} mod \ 26 = \begin{bmatrix} 11 & 7 & 4 & 19 \\ 13 & 3 & 22 & 17 \\ 18 & 11 & 12 & 22 \end{bmatrix}$$

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} = C = \begin{bmatrix} L & H & E & T \\ N & D & W & R \\ S & L & M & W \end{bmatrix}$$

Plain Text: PAY MORE MONEY

Cipher Text: LNS HDLE WMTRW

2 a **Pseudo Code of the Euclidean Algorithm:**

Step 1: Let **a, b** be the two numbers

Step 2: a mod b = R

Step 3: Let **a = b** and **b = R**

Step 4: Repeat Steps 2 and 3 until **a mod b** is greater than 0

Step 5: GCD = b

Step 6: Finish

q	r_1	r_2	r
1	24140	16762	7378
2	16762	7378	2006
3	7378	2006	1360
1	2006	1360	646
2	1360	646	68
9	646	68	34
2	68	34	0
	34	0	

2 b

Property	Expression
Commutative Laws	$(a+b) mod\ n = (b+a)\ mod\ n$ $(a \times b) mod\ n = (b \times a)\ mod\ n$
Associative Laws	$[(a+b)+c] mod\ n = [a+(b+c)] mod\ n$ $[(a \times b) \times c] mod\ n = [a \times (b \times c)] mod\ n$
Distributive Law	$[a \times (b+c)] mod\ n = [(a \times b)+(a \times c)] mod\ n$ $[a+(b \times c)] mod\ n = [(a+b) \times (a+c)] mod\ n$
Identities	$(0+a) mod\ n = a\ mod\ n$ $(1 \times a) mod\ n = a\ mod\ n$
Inverse	$a + k = 0\ mod\ n \quad where\ k = (-a)$ $a \times k = 1\ mod\ n \quad where\ k = a^{-1}$

Let a = 1, b = 5, c = 3 and n = 8

Commutative Laws:

1. $(a+b) mod\ n = (1+5) mod\ 8 = 6 \quad (LHS)$
   $(b+a)\ mod\ n = (5+1) mod\ 8 = 6 \quad (RHS)$
   LHS = RHS (proved)
2. $(a \times b) mod\ n = (1 \times 5) mod\ 8 = 5 \quad (LHS)$
   $(b \times a)\ mod\ n = (5 \times 1) mod\ 8 = 5 \quad (RHS)$
   LHS = RHS (proved)

Associative Laws:

1. $[(a+b)+c] mod\ n = [(1+5)+3] mod\ 8 = [6\ mod\ 8 + 3\ mod\ 8] mod\ 8 =$
   $[9] mod\ 8 = 1 \quad (LHS)$
   $[a+(b+c)] mod\ n = [1+(5+3)] mod\ 8 = [1\ mod\ 8\ + 0\ mod\ 8] mod\ 8 =$
   $[1] mod\ 8 = 1 \quad (RHS)$
   LHS = RHS (proved)
2. $[(a \times b) \times c] mod\ n = [(1 \times 5) \times 3] mod\ 8 = [5\ mod\ 8\ \times 3\ mod\ 8\ ] mod\ 8 =$
   $[15\ ] mod\ 8 = 7 \quad (LHS)$
   $[a \times (b \times c)] mod\ n = [1 \times (5 \times 3)] mod\ 8 = [1\ mod\ 8\ \times 15\ mod\ 8\ ] mod\ 8 =$
   $[7] mod\ 8 = 7 \quad (RHS)$
   LHS = RHS (proved)

Distributive Law:

1. $[a \times (b+c)] mod\ n = [1 \times (5+3)] mod\ 8 = [1\ mod\ 8 \times 8\ mod\ 8] = [0] mod\ 8 = 0 \quad (LHS)$
   $[(a \times b)+(a \times c)] mod\ n = [(1 \times 5)+(1 \times 3)] mod\ 8 = [5+3] mod\ 8 = 0 \quad (RHS)$
   LHS = RHS (proved)
2. $[a+(b \times c)] mod\ n = [1+(5 \times 3)] mod\ 8 = [1\ mod\ 8\ + 15\ mod\ 8] = [1+7] mod\ 8 = 0 \quad (LHS)$
   $[(a+b) \times (a+c)] mod\ n = [(1+5) \times (1+3)] mod\ 8 = [6\ mod\ 8 \times 4\ mod\ 8] =$
   $24\ mod\ 8 = 0 \quad (RHS)$

Identities:

1. $(0+a) mod\ n = (0+1) mod\ 8 = 1$
2. $(1 \times a) mod\ n = (1 \times 1) mod\ 8 = 1$

Inverse:

1. $b + k = 0\ mod\ n \quad where\ k = (-b) \qquad here\ k = -5$
   $[5+(-5)] mod\ 8 = 0$
2. $b \times k = 1\ mod\ n \quad where\ k = b^{-1} \qquad here\ k = 5$
   $[5 \times 5] mod\ 8 = [25] mod\ 8 = 1$

2 c $11^7\ mod\ 13 = \mathbf{1}1 \times 11 \times 11 \times 11 \times 11 \times 11 \times 11 = 19487171\ mod\ 13 = 2$

3 a **FEISTEL CIPHER STRUCTURE:**

1. The inputs to the encryption algorithm are a plaintext block of length $2w$ bits and a key K.
2. The plaintext block is divided into two halves, $L0$ and $R0$.
3. The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block.
4. Each round i has as inputs L_{i-1} *and* R_{i-1} derived from the previous round, as well as a subkey K_i derived from the overall K. The subkeys K_i are different from K and from each other.
5. 16 rounds are used, although any number of rounds could be implemented. All rounds have the same structure.
6. A **substitution** is performed on the left half of the data. This is done by applying a *round function* F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data.
7. The round function F has the same general structure for each round. The round function F is represented as $F(RE_i, K_{i+1})$
8. Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data.

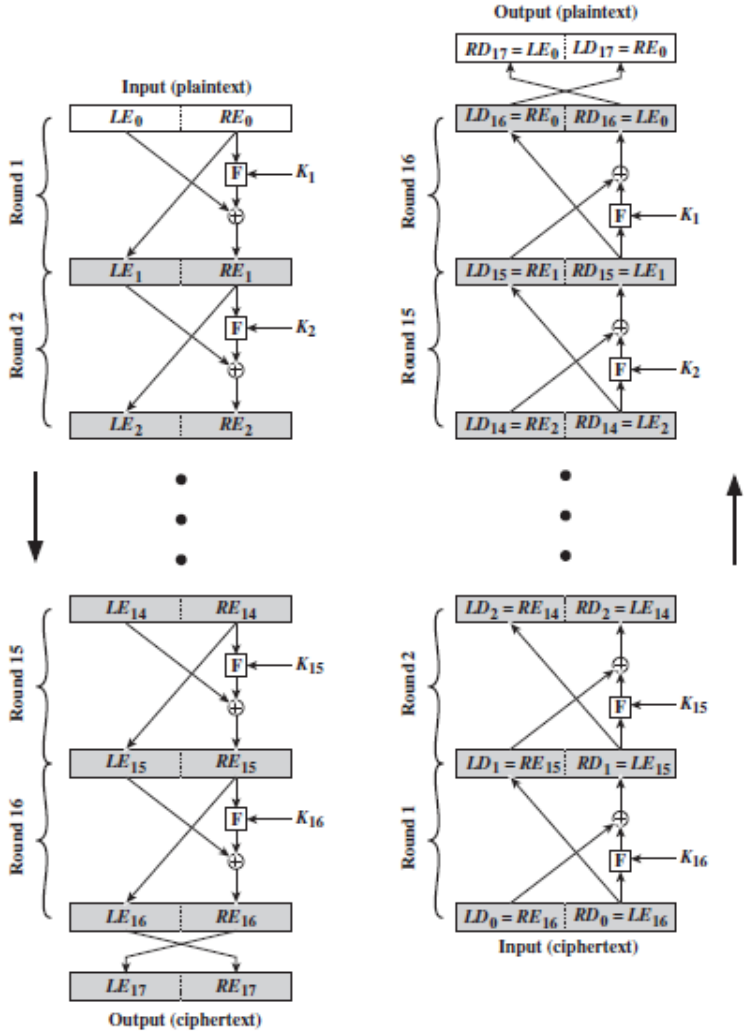


Figure: Feistel Encryption and Decryption (16 rounds)

9. Feistel network depends on the choice of the following parameters and design features:
   a) **Block size:** larger block sizes mean greater security, but it reduces encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.
   b) **Key size:** Larger key size means greater security but may decrease encryption decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered being inadequate and 128 bits has become a common size.
   c) **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.

d) **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
e) **Round function F:** Again, greater complexity generally means greater resistance to cryptanalysis.

10. There are two other considerations in the design of a Feistel cipher:
    a) **Fast software encryption/decryption:** Encryption is embedded in applications hence the speed of execution of the algorithm becomes a concern.
    b) **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.
11. **Feistel Decryption Algorithm:**
    a) Decryption with a Feistel cipher is same as the encryption process.
    b) In decryption the ciphertext is used as input to the algorithm, and the subkeys K_i are used in reverse order.
    c) That is, K_n is used in the first round, K_{n-1} in the second round, and so on, until K_1 is used in the last round. It is an advantage because no need to implement two different algorithms; one for encryption and one for decryption.
    d) For clarity, the notation LE_i *and* RE_i is used for data traveling through the encryption algorithm and LD_i *and* RD_i for data traveling through the decryption algorithm.
    e) The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. i.e. $RE_i||LE_i = LD_{16-i}||RD_{16-i}$
    f) Example: (for better clarity)

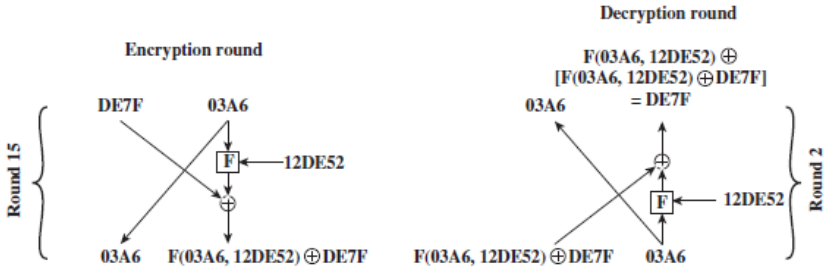


Figure: Feistel Example

3 b **DES Encryption :**

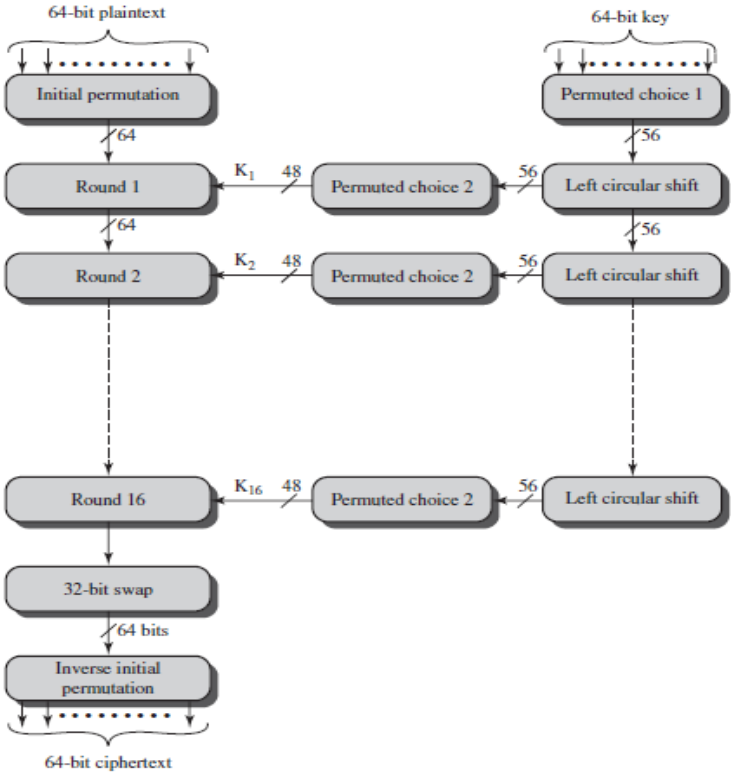


Figure: General Depiction of DES Encryption Algorithm

64 bit key is used but every 8th bit is the parity bit hence it is taken as 56 bit key. Initially the key is passed through the permutation function. For each 16 round, a sub key K_i is produced by the combination of left circular shift and permutation. The same permutation function is used in each round.
The plain text are processed through these phases

a) Initial Permutation
b) 16 rounds of same function
c) Swap
d) Final Permutation

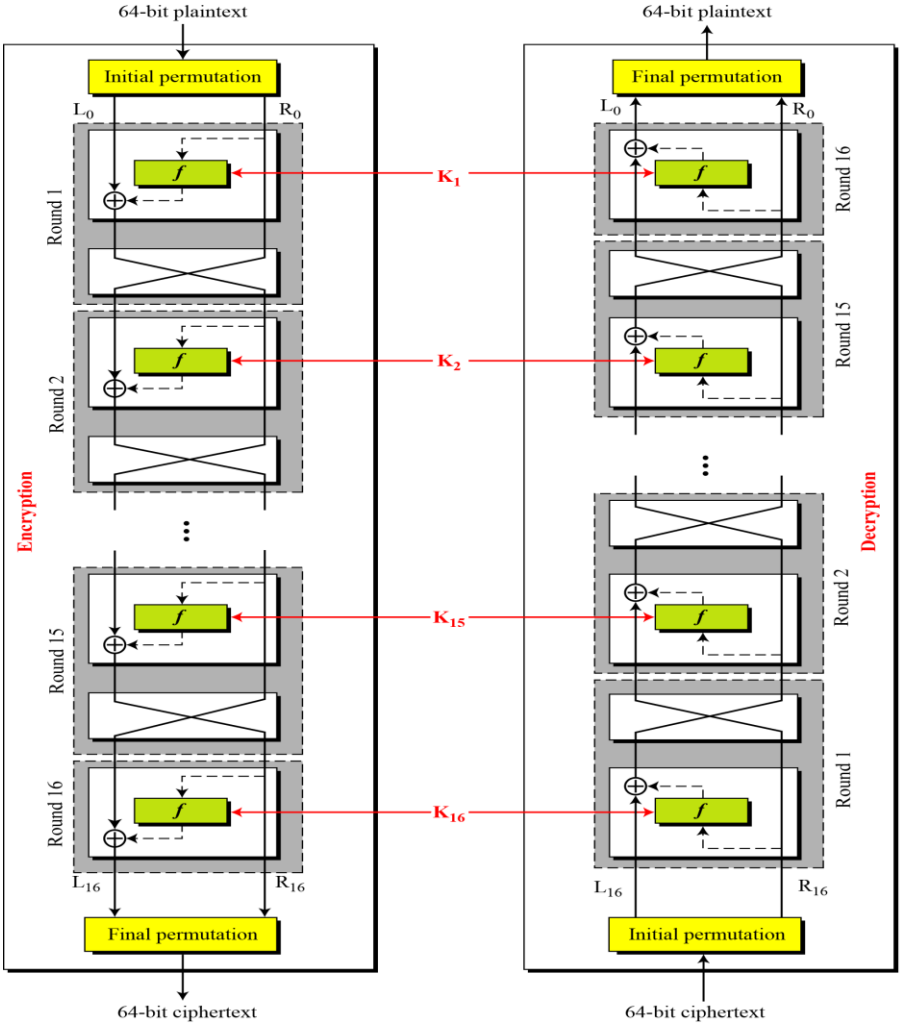


Figure: DES Encryption and Decryption

Initial Permutation and Final Permutation:

The input is 64 bit. These inputs are permuted according to a predefined rule. The permutation table contains a permutation of the number from 1 to 64. These permutation table and inverse permutation table can be designed such that the original bits can be restored.

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

DES Encryption:

a) In DES Encryption, there are two inputs to the encryption function:
   i. the plaintext to be encrypted
   ii. Key

b) In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.
c) The processing of the plaintext proceeds in three phases.
   i. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input.*
   ii. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions.
   iii. The left and right halves of the output are swapped to produce the **preoutput**.
   iv. Finally, the pre-output is passed through a permutation $[IP^{-1}]$ that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.
d) With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher.

Key Generation:

a) In DES, 56-bit key is used.
b) Initially, the key is passed through a permutation function.
a) Then, for each of the sixteen rounds, a *subkey* (K_i) is produced by the combination of a left circular shift and a permutation.
b) The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

DES Decryption:

a) As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.
b) Additionally, the initial and final permutations are reversed.

3 c Feistel network depends on the choice of the following parameters and design features:

a) **Block size:** larger block sizes mean greater security, but it reduces encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.
b) **Key size:** Larger key size means greater security but may decrease encryption decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered being inadequate and 128 bits has become a common size.
c) **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.
d) **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
e) **Round function F:** Again, greater complexity generally means greater resistance to cryptanalysis.

4 a AES doesn't use the Feistel structure. Feistel structure, half of the data block is used to modify the other half of the data block and then the halves are swapped. AES instead processes the entire data block as a single matrix during each round using substitutions and permutation. The key that is provided as input is expanded into an array of forty-four 32-bit words, **w**[*i*]. Four different stages are used, one of permutation and three of substitution:

a) **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block
b) **ShiftRows:** A simple permutation
c) **MixColumns:** A substitution that makes use of arithmetic over GF(28)
d) **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key

The cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. AddRoundKey stage makes use of the key.The cipher begins and ends with an AddRoundKey stage. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that A ⊕ B ⊕ B = A. In AES, the decryption algorithm is not identical to the encryption algorithm.

As all stages are reversible, it is easy to perform decryption to recover the plain text. Encryption and decryption going in opposite vertical directions. The first *N* - 1 rounds consist of four distinct transformation functions:

- SubBytes,
- ShiftRows,
- MixColumns,
- AddRoundKey

The final round contains only three transformations those are SubBytes, ShiftRows and AddRoundKey, and there is an initial single transformation (AddRoundKey) before the first round, which can be considered Round 0.

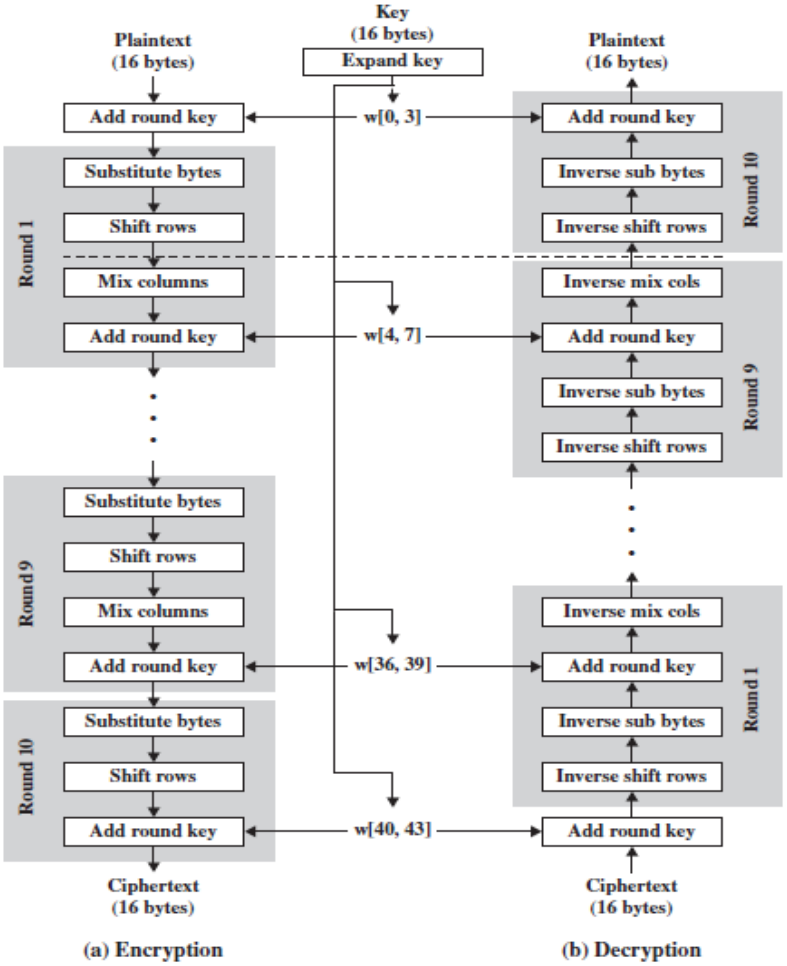
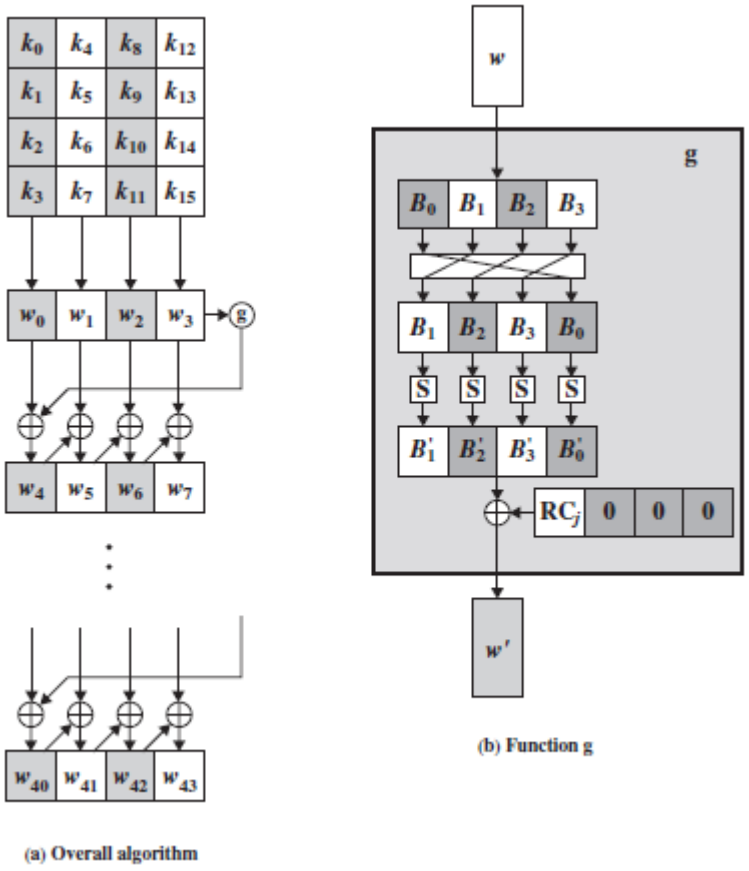


Figure: AES Encryption and Decryption

4 b **AES KEY EXPANSION:**

Key Expansion Algorithm:



1. The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes).
2. The key is copied into the first four words of the expanded key.
3. The remainder of the expanded key is filled in four words at a time.
4. Each added word **w**[i] depends on the immediately preceding word, **w**[i - 1], and the word four positions back, **w**[i - 4] and a simple XOR is used
5. For a word whose position in the **w** array is a multiple of 4, a more complex function 'g' is used.
6. The generation of the expanded key, using the symbol g to represent that complex function.
7. The function 'g' consists of the following sub-functions:
    a. Perform a one-byte left circular rotation. This means that an input word [B0, B1, B2, B3] is transformed into [B1, B2, B3, B0].
    b. Perform a byte substitution using the S-box table.
    c. The result of step 1 and step 2 is XORed with a Round Constant RC[j]

The round constant is a word in which the three rightmost bytes are always 0

Rcon Constants (Base 16)			
Round	Constant(Rcon)	Round	Constant(Rcon)
1	01 00 00 00	6	20 00 00 00
2	02 00 00 00	7	40 00 00 00
3	04 00 00 00	8	80 00 00 00
4	08 00 00 00	9	1B 00 00 00
5	10 00 00 00	10	36 00 00 00

$$\omega_{i+4} = \omega_i \oplus g(\omega_{i+3})$$
$$\omega_{i+5} = \omega_{i+4} \oplus \omega_{i+1}$$
$$\omega_{i+6} = \omega_{i+5} \oplus \omega_{i+2}$$
$$\omega_{i+7} = \omega_{i+6} \oplus \omega_{i+3}$$

4 c **<u>ShiftRows Transformation:</u>**

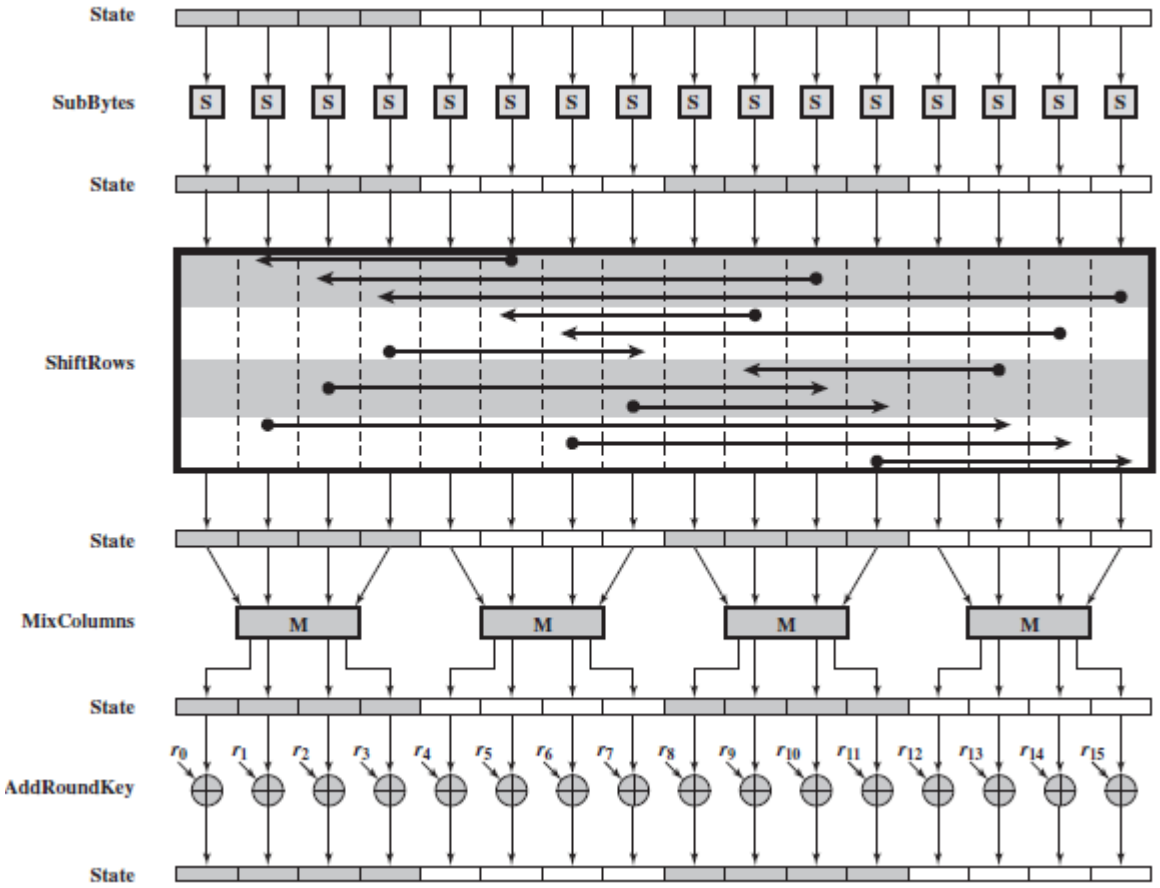


Figure: AES Encryption Round

1. The first row of **State** is not altered.
2. For the second row, a 1-byte circular left shift is performed.
3. For the third row, a 2-byte circular left shift is performed.
4. For the fourth row, a 3-byte circular left shift is performed.

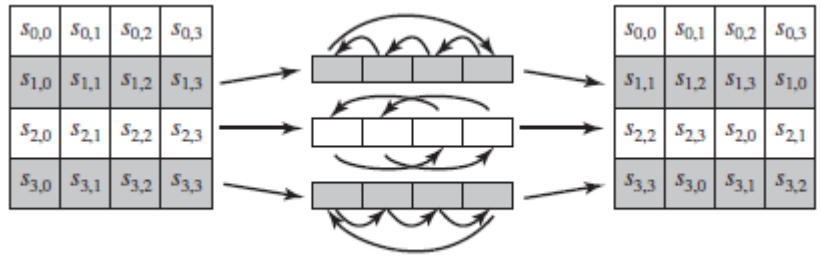
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

5. The **inverse shift row transformation** performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and so on.

Hint:



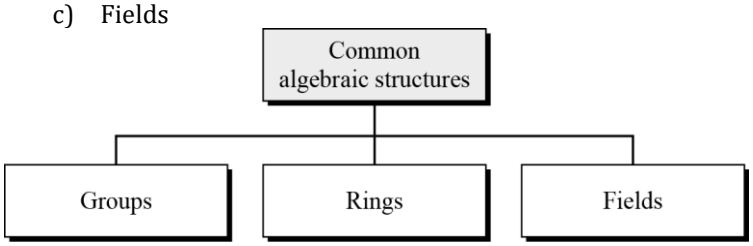
(a) Shift row transformation

Module 3

5 a **GROUP RING AND FIELDS:**

The combination of the set and the operations that are applied to the elements of the set is called an algebraic structure. There are 3 common algebraic structures

a) Group
b) Rings
c) Fields



GROUP:

1. A group (G) is a set of elements with a binary operation (•) that satisfies four properties (or axioms). It is denoted as **{G, • }**
2. A commutative group is also called abelian group. Abelian group is a group in which the operator satisfies the four properties for group plus an extra property i.e. commutativity property.
3. The 4 properties plus commutativity are defined as follows:

(A1) Closure:	If a and b belong to G, then $a \cdot b$ is also in G.
(A2) Associative:	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all a, b, c in G.
(A3) Identity element:	There is an element e in G such that $a \cdot e = e \cdot a = a$ for all a in G.
(A4) Inverse element:	For each a in G, there is an element a' in G such that $a \cdot a' = a' \cdot a = e$.

Abelian Group:

(A5) Commutative:	$a \cdot b = b \cdot a$ for all a, b in G.

Example: $\{Z_{10}, +\}$ is a commutative group/ abelian group.
The elements in Z_{10} *are* {0,1, 2,3,4,5,6,7,8,9}
Closure: It satisfies because any if any two elements are added, the resultant will be in this group.
Associativity: It doesn't matter in which order we apply the operation.
Indentity Element: '0' is the identity element.
Inverse: Additive inverse of each element exist.
Commutative: Commutative is satisfied because $a + b = b + a$
Hence $\{Z_{10}, +\}$ is an abelian group.

5 b **FINITE FIELD OF THE FORM GF(P):**

1. Although we have fields of infinite order, only finite fields is used in cryptography.

2. A finite field, with a finite number of elements are very important in cryptography.
3. For a field to be finite, the number of elements should be P^n where P is the prime and n is a positive integer.
4. The finite field of the order P^n is generally written as $GF(P^n)$, GF stands for Galois field, in honor of the mathematician who first studied finite fields.
5. There are 2 special cases:
   a) For $n = 1$, it is finite field $GF(P)$
   b) For $n > 1$, its finite field has different structure.
6. A Galois field, $GF(P^n)$ is a finite field with P^n element.

$GF(P)$Fields: When $n = 1$, we have $GF(P)$, it is same as Z_P
This field has the element i.e. $\{0,1,2,3, \ldots . (P-1)\}$ with two arithmetic operations (addition and multiplications).
$GF(2) = \{0,1\}$ *and operations are* $[+,\times]$

+	0	1
0	0	1
1	1	0

Addition

×	0	1
0	0	0
1	0	1

Multiplication

w	$-w$	w^{-1}
0	0	–
1	1	1

Inverses

In this case, the addition is equivalent to exclusive OR (XOR) operation and multiplication is equivalent to logical AND operation.
Example: Define $GF(5)$ with addition and multiplication operation.
$GF(5) = \{0,1,2,3,4\}$ *and operations are* $[+,\times]$

GF(5)
{0, 1, 2, 3, 4} [+ ×]

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Addition

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Multiplication

Additive inverse

a	0	1	2	3	4
−a	0	4	3	2	1

a	0	1	2	3	4
a^{-1}	—	1	3	2	4

Multiplicative inverse

5 c Arithmetic in $Z_8/GF(8)$:

Addition Table

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Multiplication Table

X	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

a	0	1	2	3	4	5	6	7
$-a$	0	7	6	5	4	3	2	1
a^{-1}	-	1	-	3	-	5	-	7

6 a **Fermat's Theorem:** Fermat's Theorem and Euler's Theorem are mostly used in public key cryptosystem. Fermat's Theorem and Euler's Theorem are helpful for quickly finding solution to exponentiations. Fermat's theorem has 2 versions of the theorem.
First Version: If 'P' is prime and 'a' is any integer but not divisible by 'P' then
$a^{P-1} mod\ P = 1$
Second Version: The second version removes the condition if 'P' is a prime and 'a' is any integer, then
$a^P mod\ P = a$
Example:
$7^{18} mod\ 19 = 1$
$3^5 mod\ 5 = 3$
$10^5 mod\ 5 = 0$
Proof:
$7^{18} mod\ 19 = 7^6\ mod\ 19 \times 7^6\ mod\ 19 \times 7^6\ mod\ 19$
$= 117649\ mod\ 19 \times 117649\ mod\ 19 \times 117649\ mod\ 19$
$= 1\ mod\ 19 \times 1\ mod\ 19 \times 1\ mod\ 19 = 1$

6 b **Euler's Theorem:** Euler's theorem can be thought as a generalization of a Fermat's theorem. The modulus in the Fermat's theorem is a prime, but the modulus in Euler's theorem is an integer.
Like Fermat's theorem there are 2 versions of Euler's theorem
First Version: If 'a' and 'n' are relatively prime then

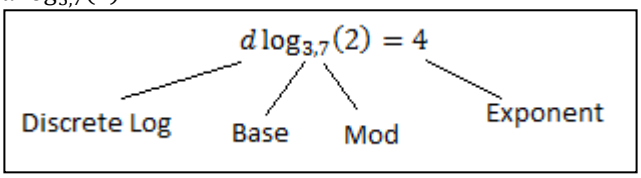
$$a^{\emptyset(n)} mod\ n = 1$$

Second Version: like Fermat's theorem, it removes the condition that 'a' and 'n' should be Co-prime.

$$a^{K.\emptyset(n)+1}\ mod\ n = a$$

Example:
$20^{62} mod\ 77 = 15$
$6^{24} mod\ 35 = 1$
$10^{5} mod\ 5 = 0$
Proof:
$6^{24} mod\ 35 = 1 = 6^{6}\ mod\ 35 \times 6^{6}\ mod\ 35 \times 6^{6}\ mod\ 35 \times 6^{6}\ mod\ 35$
$= 46656\ mod\ 35 \times 46656\ mod\ 35 \times 46656\ mod\ 35 \times 46656\ mod\ 35$
$= 1\ mod\ 35 \times 1\ mod\ 35 \times 1\ mod\ 35 \times 1\ mod\ 35 = 1$

6 c **Discrete Logarithm:** Discrete logarithm are the fundamental to a number of public key algorithm. We need a numerical procedure which is easy in one direction and hard in other way. This brings us to use modular arithmetic.
Example: $3^{5}\ mod\ 17 = 5$ $\quad but\ it\ is\ difficult\ to\ find\ \ 3^{x}\ mod\ 17 = 5$.
It is called discrete logarithm problem.
Modular Arithmetic
$5^{4}\ mod\ 7 = 2$
$d\log_{5,7}(2) = 4$



Example: $a^{i}\ mod\ 7 = x$ Write the possible exponent.
$d\log_{a,7}(x) = i \quad (here\ a\ has\ to\ be\ chosen\ wisely)$

a	a^1	a^2	a^3	a^4	a^5	a^6
1	1	1	1	1	1	1
2	2	4	1	2	4	1
3	3	2	6	4	5	1
4	4	2	1	4	2	1
5	5	4	6	2	3	1
6	6	1	6	1	6	1

As our modulus is 7 up to power 6 has to be written. Here 3 and 5 are the primitive root (generator), because it generated the entire set of integers of modulus.

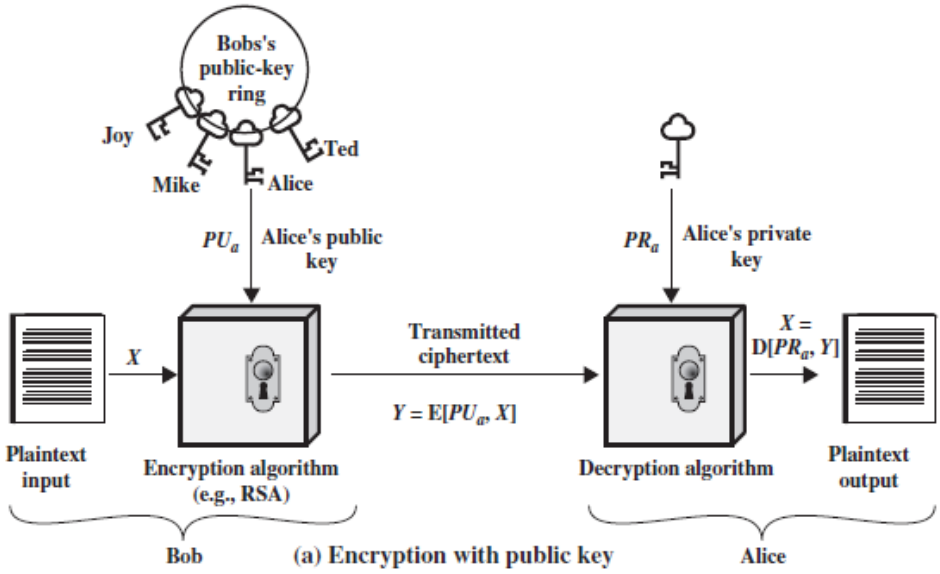
1. If we are using mod 7, then the base should be either 3 or 5, because they give unique answer, if we choose other than that, it will give multiple answers.
2. i.e. If we use exponentiation for encryption, then we have to use discrete logarithm for decryption. **If we want to find the discrete logarithm, then the base must be primitive root**.
3. Finding discrete logarithm is very hard, when we have large number, it takes a long time to find the answer **that's why it is used for security purpose.**
4. Not all the integers have primitive roots. The only integers with primitive roots are those of the form 2,4, P^{α} $and\ P^{2\alpha}$. Where
   $P\ is\ any\ odd\ Prime\ (i.e.\ all\ prime\ except\ 2)\ and\ \alpha\ is\ a\ positive\ integer$

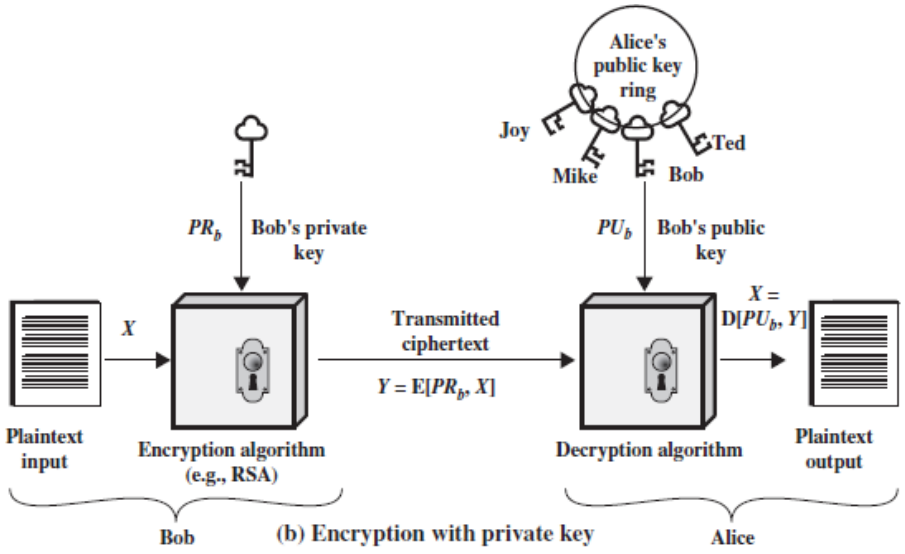
Module 4

7 a **Essential steps in public key cryptosystem:**

1. Each user generates a pair of key to be used for encryption and decryption.
2. Each user place one of the key in public register and other one is kept private. Each user maintains a collection of public keys obtained from others.
3. If Bob wants to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, it decrypts the message using its private key.
5. As long as the user's private key is protected the communication is secure. At any time a system can change its private key and publish the companion public key to replace its old public key.

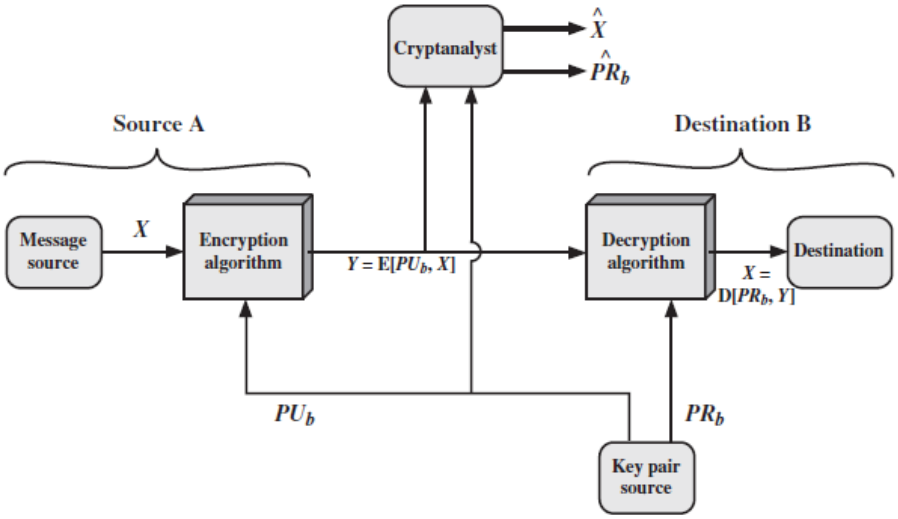
6. The key used in symmetric key is named as secret key and the 2 keys used in public key cryptography are named as public key and private key.
   $Notation\ Used:\ K_a = Secret\ key\ of\ sender\ 'A'$
   $PU_a = Public\ key\ of\ sender\ 'A'$ $PR_a = Private\ key\ of\ sender\ 'A'$





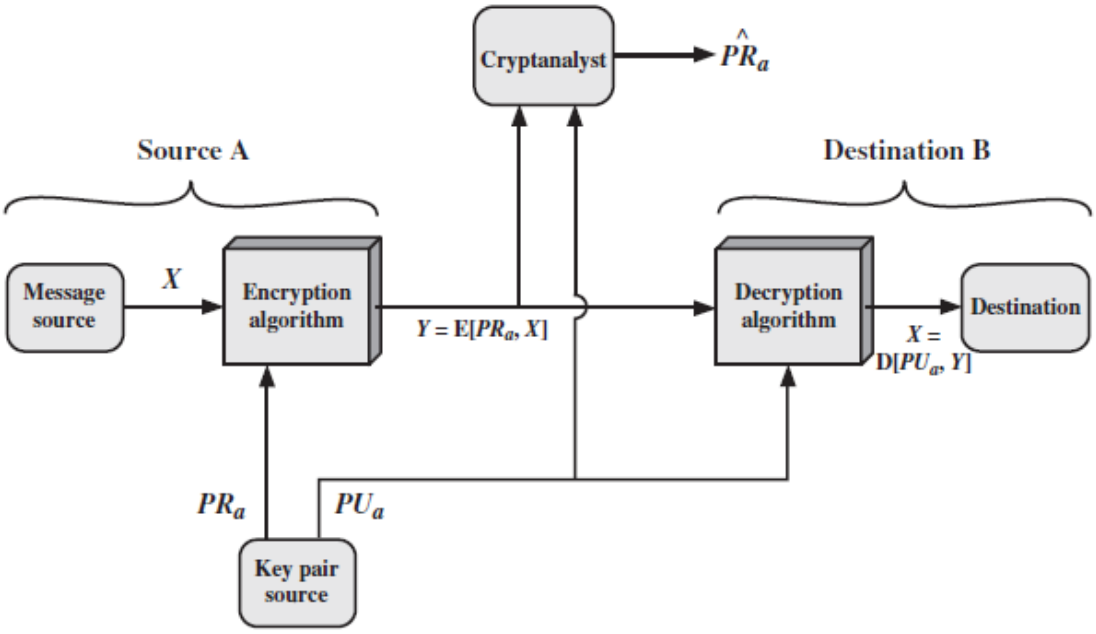
Public Key Cryptosystem-Secrecy:

1. Source 'A' sends the plaintext $X = [X_1, X_2, \dots \dots X_m]$. The m element of X is some alphabet in the message.
2. As the message is intended for user 'B' , 'B' generates 2 keys
   a) Private Key (PR_b)
   b) Public Key (PU_b) and PU_b is publicly available so that it is accessible by A.
3. With the message X and encryption key PU_b, sender forms the cipher text
   $Y = [Y_1, Y_2, \dots \dots Y_N]$ $\quad where\ \ Y = E(PU_b, X)$
4. At the receiver, the intended receiver matches the key and find the message
   $X = D(PR_b, Y)$
5. It is assumed that the cryptanalysts have the knowledge of encryption $(E)$and decryption (D)algorithms. If the cryptanalyst is interested only in this particular message, then its focus is to recover X, by generating a plaintext estimate $\hat{X}$. But if Cryptanalyst is interested in being able to read future message as well, it will try to recover PR_b, by generating an estimate $\widehat{PR_b}$.
6. Either of the 2 keys can be used for encryption, with other being used for decryption. This above scheme provides confidentiality.
7. As anybody can encrypt the message using $B's$ public key and claim to be came from $'A'$.



Public-Key Cryptosystem: Secrecy

Public Key Cryptosystem-Authentication:



Public-Key Cryptosystem: Authentication

1. If $'A'$ wants to communicate to $'B'$, then $'A'$ encrypt the message using $A's$ private key.
2. $'B'$ can decrypt the message using $A's$ public key.
3. As message was encrypted using $A's$ private key, only $'A'$ could prepare the message. This entire message serves as a digital signature.
4. It is important to alter the message without access to $A's$ private key. So this message is authenticated both in terms of source and data integrity.
5. The encryption and decryption can be represented as :
   $Y = E(PR_a, X)$
   $X = D(PU_a, Y)$
6. This public key encryption doesn't provide confidentiality because all will have $A's$ public key hence can decrypt the message easily.
7. It is safe from alteration but not from eavesdropping.

7 b Description of the Algorithm:

1. RSA uses the expression with exponentials.
2. Plain text is encrypted in block.
3. The plain text M and cipher text C can be represented as
   $C = M^e \ mod \ n$
   $M = C^d \ mod \ n = (M^e)^d \ mod \ n = M^{ed} \ mod \ n$

4. Both the sender and receiver know the value of n.
5. Sender knows the value of e and only receiver know the value of d.
6. This is a public key encryption algorithm with a public key of $PU = \{e, n\}$ and a private key of $PR = \{d, n\}$
7. For the encryption algorithm to satisfy for public key encryption, the following requirements must be met.
    a) It is possible to find Message (M) if e, d and n are known i.e. $M^{ed}\ mod\ n = M$ $for\ all\ M < n$
    b) It is relatively easy to calculate $M^e\ mod\ n$ and $C^d\ mod\ n\ \ for\ all\ value\ of\ M < n$
    c) It is infeasible to determine d given $e\ \ and\ n$.
8. The equation $M^{ed}\ mod\ n = M$ is only valid if $e\ \ and\ d$ are multiplicative inverse $modulo\ \emptyset(n)$. Where $\emptyset(n)$ is the Euler totient function. i.e. $ed\ mod\ \emptyset(n) = 1$ means $e\ and\ d$ should be relatively prime.

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

The RSA Algorithm

7 **c** $n = pq = 5 \times 11 = 55$
$\emptyset(n) = (p-1) \times (q-1) = 4 \times 10 = 40$
$e = 3\ \ and\ \ m = 9$
$ed\ mod\ \emptyset(n) \equiv 1 => d = e^{-1}\ mod\ \emptyset(n) => d = 3^{-1}\ mod\ 40 => d = -13\ mod\ 40 =$ **27**

q	r_1	r_2	r	t_1	t_2	$t = t_1 - qt_2$
13	40	3	1	0	1	-13
3	3	1	0	1	-13	40
	1	0		**-13**	40	

$PU = \{3,55\}\ \ and\ \ PR = \{27,55\}$
$C = M^e\ mod\ n => C = 9^3 mod\ 55 = 14$
$M = C^d\ mod\ n = 14^{27}\ mod\ 55 = 9$

$14^{27}\ mod\ 55$

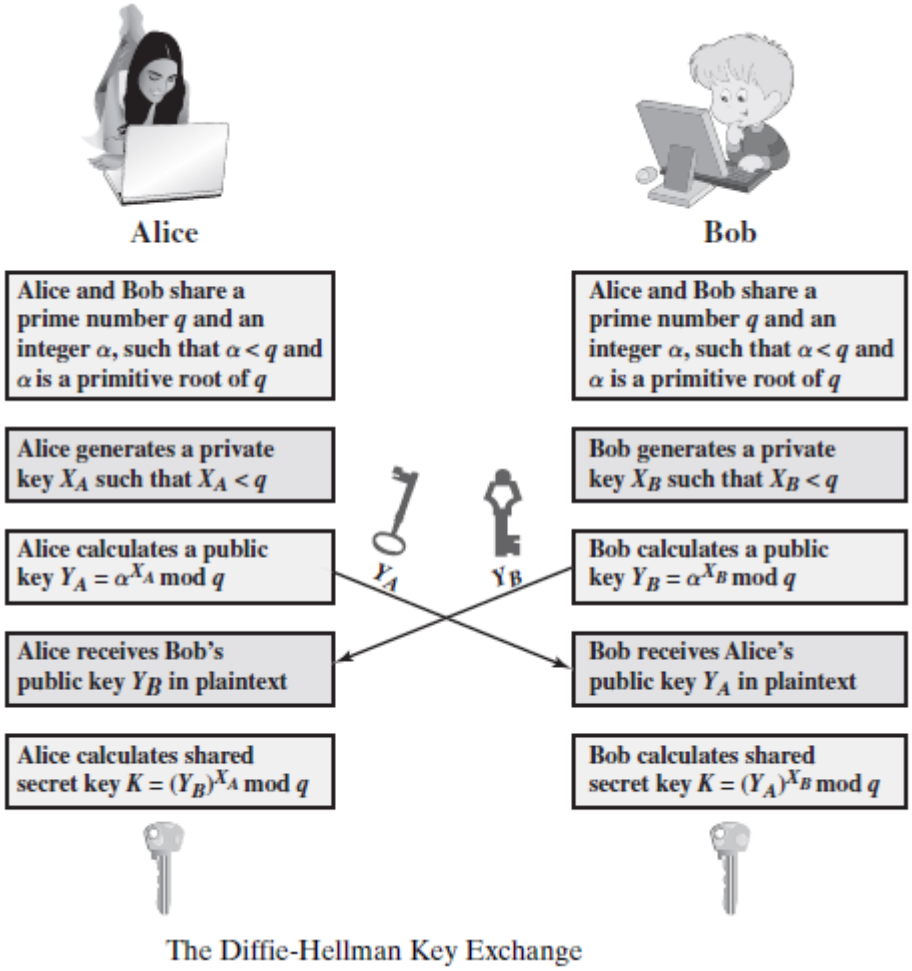
$(27)_{10} = (11011)_2$
$1: 14\ mod\ 55 = 14$
$1: (14)^2 \times 14\ mod\ 55 = 49$
$0: (49)^2 mod\ 55 = 36$
$1: (36)^2 \times 14\ mod\ 55 = 49$
$1: (49)^2 \times 14\ mod\ 55 = 9$

8 **a** **<u>Diffie Hellman Key Exchange Algorithm:</u>**

1. In this scheme, there are two publicly known numbers those are: a prime number q and an integer α that is a primitive root of q.
2. User A selects a random integer $X_A < q$ and compute $Y_A = \alpha^{X_A}\ mod\ q$.

3. User B selects a random integer $X_B < q$ and compute $Y_B = \alpha^{X_B} \bmod q$.
4. User A computes the key as $K_A = {Y_B}^{X_A} \bmod q$
5. User B computes the key as $K_B = {Y_A}^{X_B} \bmod q$

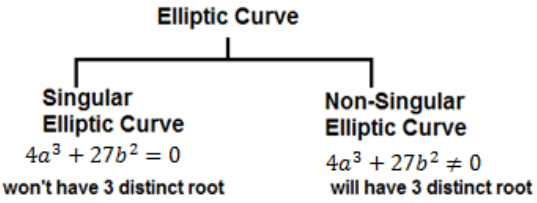
$K_A = {Y_B}^{X_A} \bmod q$
$K_A = (\alpha^{X_B} \bmod q)^{X_A} \bmod q$
$K_A = (\alpha^{X_B})^{X_A} \bmod q$
$K_A = \alpha^{X_B X_A} \bmod q$
$K_A = (\alpha^{X_A})^{X_B} \bmod q$
$K_A = (\alpha^{X_A} \bmod q)^{X_B} \bmod q$
$K_A = {Y_A}^{X_B} \bmod q$
$K_A = K_B$



The Diffie-Hellman Key Exchange

8 b **<u>ELLIPTIC CURVE OVER REAL NUMBER:</u>**

1. Elliptic curves are not ellipses.
2. It is named because; they are described by cubic equation.
3. The elliptic curves is represented as $y^2 = x^3 + ax + b$
4. To plot such a curve, we need to compute $y = \sqrt{x^3 + ax + b}$
5. For given value of a *and* b, the plot consists of +ve and –ve values of y for each values of x. Hence the curve is symmetrical about $y = 0$.
6. An elliptic curve has a single element denoted as 0 and called the point at infinity.
7. Elliptic curve is represented as $E(a, b)$
8. The points on the curves are used for cryptography.



9. There are 2 types of elliptic curves.

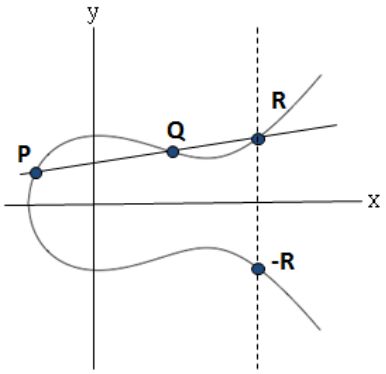
a) Singular Elliptic curve
b) Non-Singular Elliptic curve

10. In cryptography we are interested in non-singular elliptic curve, because it has 3 distinct roots.

Geometric Description of Addition:

If three points on an elliptic curve lie on a straight line, their sum is *O.* From this definition, we can define the rules of addition over an elliptic curve.

1. *O serves as the additive identity such that P + O = P.*
2. if *P = (x, y), then -P = (x, -y) such that P + (-P) = P - P = O.*
3. To add two points *P and Q* , draw a straight line between them and find the third point of intersection *R. such that P + Q = -R.*
4. If two points, *P and –P* are joined by a vertical line, will intersect the curve at the infinity point. i.e. *P + (-P) = O*
5. To find 2Q, draw the tangent line and find the other point of intersection *S.* Such that *Q + Q = 2Q = -S.*



Algebraic Description of Addition:

If the coordinates of P and Q are $P = (x_P, y_P) and\ Q = (x_Q, y_Q)$ then the coordinates of R will be

$If \quad P \neq Q \ \ (i.e. P + Q = R)$	$If \quad P = Q \ \ (i.e. P + P = 2P)$
$x_R = \Delta^2 - x_P - x_Q$ $y_R = \Delta(x_P - x_R) - y_P$ $Where\ \Delta = \left(\frac{y_Q - y_p}{x_Q - x_P}\right)$	$x_R = \Delta^2 - 2x_P$ $y_R = \Delta(x_P - x_R) - y_P$ $Where\ \Delta = \left(\frac{3{x_P}^2 + a}{2y_P}\right)$

8 c ELLIPTIC CURVE CRYPTOGRAPHY:

1. Principal attraction of ECC compared to RSA is; it provides equal security for a smaller key size.
2. It reduces the processing overhead.
3. Though ECC has been explained before some years, confidence level in ECC is not yet as high as that in RSA.
4. ECC is more difficult to explain than either RSA or Diffie-Hellman.
5. Before understanding the ECC, the concept of abelian group should be known.

Abelian Group: An abelian group denoted as **{*G*, • },** is a set of elements with a binary operations denoted by **•.** If these following axioms are obeyed.

(A1) Closure:	If a and b belong to G, then $a \cdot b$ is also in G.
(A2) Associative:	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all a, b, c in G.
(A3) Identity element:	There is an element e in G such that $a \cdot e = e \cdot a = a$ for all a in G.
(A4) Inverse element:	For each a in G, there is an element a' in G such that $a \cdot a' = a' \cdot a = e$.
(A5) Commutative:	$a \cdot b = b \cdot a$ for all a, b in G.

6. Public key cryptosystem uses the abelian group.
7. E.g. Diffie Hellman Key exchange uses multiplication operation i.e.
   a) In Diffie Hellman algorithm encryption is done as $a^K\ mod\ q$

$$a^K\ mod\ q = \underbrace{(a \times a \times a \ldots \times a)}_{K\ times}\ mod\ q$$

To attack Diffie-Hellman, the attacker must determine K given $a\ and\ a^K$ this is the discrete logarithm problem.

b) Whereas, for elliptic curve cryptography, the operation used is addition i.e.

$a \times K = \underbrace{(a + a + \cdots + a)}_{K\ times}\ mod\ q$

To Attack ECC, the attacker must determine K given $a\ and\ (a \times K)$

Elliptic curves can be defined as

(i) Elliptic curve over real Number

(ii) Elliptic curve over Z_p

(iii) Elliptic curves over $GF(2^m)$

Module 5

9 a LINEAR FEEDBACK SHIFT REGISTERS (LFSR):

Shift register sequences are used in both cryptography and coding theory. E.g., stream ciphers based on shift registers have been used in military cryptography. A **feedback shift register** is made up of **two** parts:

- A shift registers
- Feedback function.

The shift register is a sequence of bits. (If it is n-bits long, it is called an n-bit shift register.). Each time a bit is needed, all the bits in the registers are shifted 1 bit to the right. The new left-most bit is computed as a function of the other bits. The output of the shift register is the 1 bit, often the least significant bit. The period of a shift register is the length of the output sequence before it starts repeating. Cryptographers have liked stream ciphers made up of shift registers.

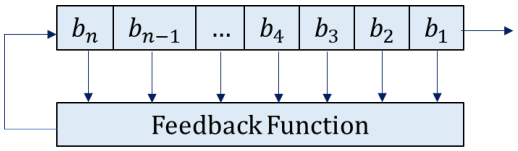


Figure: feedback shift register

- The simplest kind of feedback shift register is a **linear feedback shift register (LFSR)**. The feedback function is simply the XOR of certain bits in the register, the list of these bits is called **tap sequence** also called as **Fibonacci configuration**.

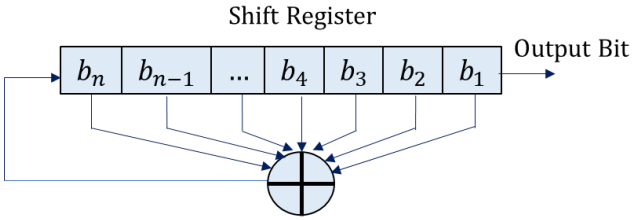


Figure: Linear feedback shift register

- The simple 4-bit LFSR can be shown in figure below. Here the first and the fourth bits are tapped. If it is initialized with the value 1111, it produces the following sequence of internal states before repeating.

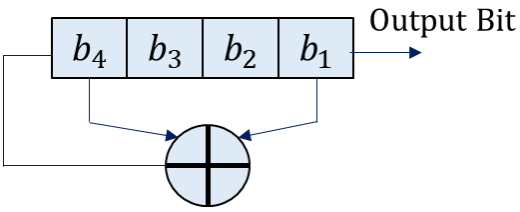


Figure: 4-bit LFSR

S.N.	Register state	Output
1	1111	1
2	0111	1
3	1011	1
4	0101	1
5	1010	0
6	1101	1
7	0110	0
8	0011	1
9	1001	1
10	0100	0
11	0010	0
12	0001	1
13	1000	0
14	1100	0
15	1110	0
16	1111	0

The output sequence is the string of least significant bits.: 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 .....

- The shift registers filled with zeros will cause the LFSR to output a never-ending stream of zeros – this is particularly not useful.
- Only LFSR with certain tap sequences will cycle through all $2^n - 1$ internal states; these are maximal period LFSRs. The resulting output is called an **m-sequence**.
- For maximal period LFSR, the polynomial formed from a tap sequence plus the constant 1 must be primitive polynomial mod 2.
- A primitive polynomial of degree n is an irreducible polynomial if it divides $x^{2^{n-1}} + 1$ but not $x^d + 1$ for any d that divides $2^n - 1$.
- There is no easy way to generate the primitive polynomial mod 2 for a given degree. The easiest way is to choose a random polynomial and test whether it is primitive. But it is complicated as- something like testing random numbers for primality.
- E.g. the polynomial (32, 7, 5, 3, 2, 1, 0) means that the following polynomial is primitive modulo 2. (Polynomial is: $x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$).
- It's easy to turn this into maximal-period LFSR. The first number is the length of the LFSR. The last number is always 0 and can be ignored. All the number except 0, specify the tap sequence. It means, if you take a 32-bit shift register and generate the new bit by XORing the thirty-second, seventh, fifth, third, second and first bits together, the resulting LFSR will be maximal length, it will cycle through $2^{32} - 1$ values before repeating. The C code for the LFSR is:

```
Int LFSR(){
     Static unsigned long ShiftRegister = 1;
     /* Anything but 0. */
     ShiftRegister = ((((ShiftRegister >> 31)
                      ^ (ShiftRegister >> 6)
                      ^ (ShiftRegister >> 4)
                      ^ (ShiftRegister >> 2)
                      ^ (ShiftRegister >> 1)
                      ^  ShiftRegister))
                      &   0×00000001)
                      <<  31)
                      | (ShiftRegister >> 1);
     Return ShiftRegister & 0×00000001;
}
```

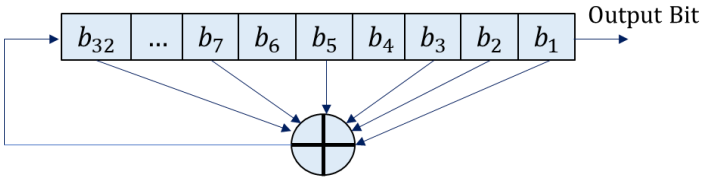


Figure: 32-bit long maximal-length LFSR

- This code is more complicated when the shift register is longer than the computer's word size.
- E.g. if (a, b, 0) is primitive, then (a, a-b, 0) is also primitive. If (a, b, c, d, 0) is primitive, then (a, a-d, a-c, a-b, 0) is also primitive. Mathematically
  If $x^a + x^b + 1$ is primitive, so is $x^a + x^{a-b} + 1$
  If $x^a + x^b + x^c + x^d + 1$ is primitive, so is $x^a + x^{a-d} + x^{a-c} + x^{a-b} + 1$
- Primitive polynomials are faster in software, because only two bits of the shift register have to be XORed to generate each new bit.

The polynomials are generally **sparse**, means they have very few co-efficient. Sparseness is always a source of weakness, sometimes enough to break the algorithm. It is better to use **dense** primitive polynomial, those with a lot of coefficients for cryptographic applications. Generating dense primitive polynomials modulo 2 is not easy.

9 b i) **<u>Generalized Geffe Generator:</u>**

Instead of choosing between two LFSRs, this scheme chooses between k LFSRs, where k is power of 2. There are $k + 1$ LFSRs total. LFSR-1 must be clocked $\log_2 k$ times faster than the other k LFSRs. Though this scheme is complex than Gaffe generator, same kind of correlation attack is possible.

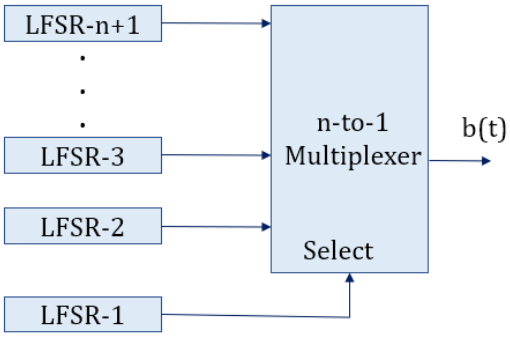


Figure: Generalized Geffe generator

ii) Threshold Generator:

- This generator tries to get around the security problems of the previous generators by using a variable number of LFSRs. The theory is that if we use a lot of LFSRs, it's hard to break the cipher.
- It is implemented by taking a large number of LFSRs and the length of all the LFSRs are relatively prime and all the feedback polynomials are primitive.
- If more than half of the output bits are 1, then the output of the generator is 1. If more than half of the output bits are 0, then the output of the generator is 0.
- The output of the generator is represented as:

$$b = (a_1 \hat{} a_2) \oplus (a_1 \hat{} a_3) \oplus (a_2 \hat{} a_3)$$

- It is very similar to Geffe generator, except that it has a large linear complexity of $n_1n_2 + n_1n_3 + n_2n_3$ *where* n_1, n_2 *and* n_3 are the length of the first, second and third LFSRs.
- This falls to correlation attack.

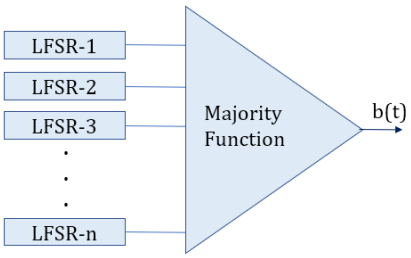


Figure: Threshold generator

iii) Alternating Stop-and-Go Generator:

- This generator, uses three LFSRs of different length. LFSR-2 is clocked when the output of LFSR-1 is 1; LFSR-3 is clocked when the output of LFSR-1 is 0.
- The output of the generator is the XOR of LFSR-2 and LFSR-3.
- This generator has a long period and large linear complexity. The correlation attack is possible against LFSR-1.

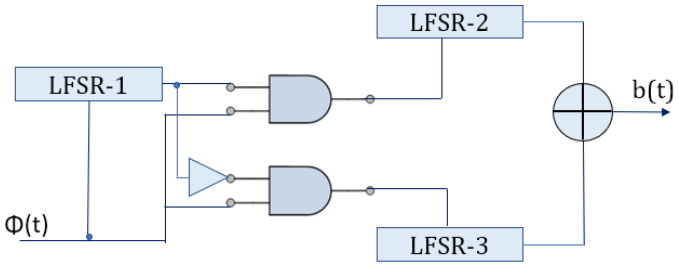


Figure: Alternating stop-and-go generator

10 a ADDITIVE GENERATORS:

- **Additive generators** are extremely efficient because they produce random words instead of random bits. They are not secure on their own, but can be used as building blocks or secure generators.
- The initial state of the generator is an array of n-bit words: 8-bit words, 16-bit words, 32-bit words. The initial state is the key. The *i*th word of the generator is

$$X_i = (X_{i-a} + X_{i-b} + X_{i-c} + \cdots + X_{i-m}) mod\ 2^n$$

- If the coefficients a, b, c, … m are chosen right, the period of this generator is at least $2^n - 1$.

- **Example:** (55,24,0) is a primitive polynomial mod 2. This means that the following additive generator is maximal length.

$$X_i = (X_{i-55} + X_{i-24}) mod\ 2^n$$

This works because, primitive polynomial has three coefficients. If it has more coefficient, then we need some additional requirements to make it maximal length.

- **Fish:**
  - Fish is an additive generator based on techniques used in the shrinking generator. It produces a stream of 32-bit words which can be XORed with the plaintext stream to produce ciphertext, or XORed with ciphertext stream to produce plaintext.
  - The algorithm is named as it is Fibonacci Shrinking generator.
  - First, it uses two additive generators. The key is the initial values of these generators.

$$A_i = (A_{i-55} + A_{i-24}) mod\ 2^{32}$$
$$B_i = (B_{i-52} + A_{i-19}) mod\ 2^{32}$$

  - These sequences are shrunk, as a pair, depending on the least significant bit of B_i: if it is 1, use the pair; if it is 0, ignore the pair.
  - C_j is the sequence of used words from A_i and D_j is the sequence of used words from B_i. These words are used in pairs- $C_{2j}, C_{2j+1}, D_{2j}\ and\ D_{2j+1}$- to generate two 32-bit output words: $K_{2j}\ and\ K_{2j+1}$.

$$E_{2j} = C_{2j} \oplus (D_{2j} \hat{} D_{2j+1})$$
$$F_{2j} = D_{2j+1} \hat{} (E_{2j} \hat{} C_{2j+1})$$
$$K_{2j} = E_{2j} \oplus F_{2j}$$
$$K_{2i} = C_{2i+1} \oplus F_{2j}$$

  - This algorithm is fast, Unfortunately, it is also insecure; an attack has a work factor of about 2^{40}.

- **Pike:**
  - Pike is the leaner, meaner version of Fish, developed by Ross Anderson, the man who broke Fish.
  - It uses three additive generators. For example:

$$A_i = (A_{i-55} + A_{i-24}) mod\ 2^{32}$$
$$B_i = (B_{i-57} + A_{i-7}) mod\ 2^{32}$$
$$C_i = (C_{i-58} + C_{i-19}) mod\ 2^{32}$$

  - To generate the keystream word, look at the additional carry bits.
  - If all the three agree, then clock all three generators. If they don't, then just clock the two generators that agree. Save the carry bit for the next time. The final output is the XOR of the three generators.
  - Pike is faster than Fish, as on average it requires 2.75 steps per output rather than 3 steps.

10 b GIFFORD:

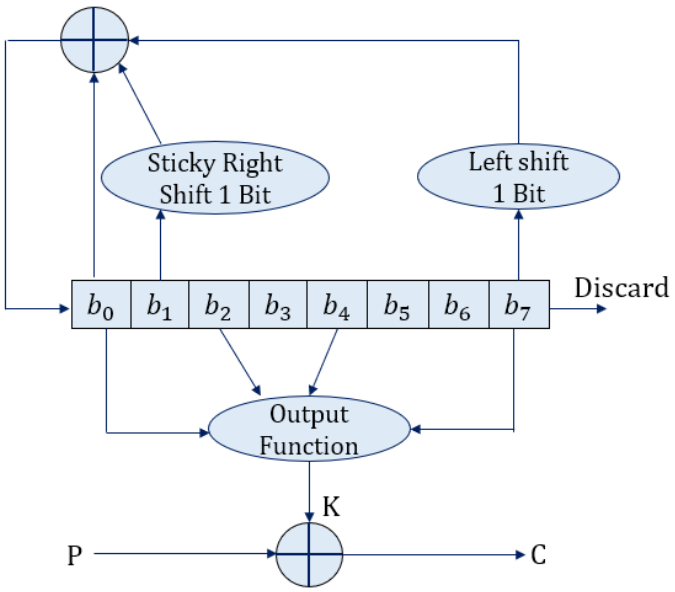


Figure: Gifford

- It was developed by David Gifford. It was used to encrypt news wire reports in Boston area from 1984 until 1988.
- The algorithm has a single 8-byte register: $b_0, b_1, \ldots \ldots b_7$.
- The key is the initial stage of the register.
- The algorithm works in OFB (output feedback); the plaintext doesn't affect the algorithm at all.
- To generate the key byte k_i, concatenate b_0 *and* b_2 and concatenate b_4 *and* b_7. Multiply the two together to get a 32-bit number. The third byte from the left is k_i.
- To update the register, take b_1 and sticky right shift it 1 bit. (Sticky right shift: the left most bit is both shifted and also remains in place.). Take b_7 and shift it 1 bit to the left; there should be a 0 in the right-most bit position. Take the XOR of the modified b_1, the modified b_7 and b_0. Shift the original register 1byte to the right and put this byte in the left most position.
- This algorithm was broken in 1994. It concludes that, the feedback polynomial isn't primitive and can be attacked.

10 c A5:

- A5 is a stream cipher used to encrypt GSM (Group Special Mobile). That is the non-American standard for digital cellular mobile telephones. It is used to encrypt the link from the telephone to the base station. The rest of the link is unencrypted, the telephone company can easily eavesdrop on our conversation.
- Originally, it was designed to prohibit export of phones to some country. It is being discussed now, whether A5 might harm export sales. There is also a rumor as, various NATO intelligence agencies had a catfight in the mid-1980s over whether GSM encryption should be strong or weak. Germans wanted strong cryptography, as they were sitting near the Soviet Union. But other counties overruled them. A5 is a French design.
- A British telephone company gave all the documentation to Bradford University without a nondisclosure agreement. It leaked and was eventually posted to the internet.
- A5 consists of three LFSRs; the register lengths are 19,22 and 23, all the feedback polynomials are sparse. The output is the XOR of the three LFSRs. A5 uses variable clock control. Each register is clocked based on its own middle bit, XORed with the inverse threshold function of the middle bits of all three registers. Generally, two LFSRs clock in each round.
- There is a trivial attack requiring 2^{40} encryptions: Guess the contents of the first two LFSRs, then try to determine the third LFSR from the key stream.
- A5 is very efficient. It passes all known statistical tests, its only weakness is, its registers are very short enough to make exhaustive search feasible.
- Variant of A5 with longer shift registers and denser feedback polynomial should be secure.