UNIT-II

SymmetrickeyCiphers: BlockCipherprinciples, DES, AES, Blowfish, RC5, IDEA, Block Cipher Operation, Streamciphers, RC4.

AsymmetrickeyCiphers: Principles of public key cryptosystems, RSA algorithm, ElgamalCryptography,Diffie-HellmanKey Exchange, Knapsack Algorithm.

SymmetrickeyCiphers:Block Cipher Design Principles

Block ciphers are built in the Feistel cipher structure. Block cipher has a specific number of rounds and keys for generating ciphertext.Block cipher is a type of encryption algorithm that processes fixed-size blocks of data, usually 64 or 128 bits, to produce ciphertext. The design of a block cipher involves several important principles to ensure the security and efficiency of the algorithm. Some of these principles are:

1. **Number of Rounds –** The number of Rounds is regularly considered in design criteria, it just reflects the number of rounds to be suitable for an algorithm to make it more complex, in DES we have 16 rounds ensuring it to be more secure while in AES we have 10 rounds which makes it more secure.
2. **Design of function F –** The complexity of cryptanalysis can be derived from the Round function i.e.. To increase the complexity of the round function, the avalanche effect is also included in the round function, as the change of a single bit in plain text would produce a mischievous output due to the presence of avalanche effect.
3. **Confusion and Diffusion:** The cipher should provide confusion and diffusion to make it difficult for an attacker to determine the relationship between the plaintext and ciphertext.
    - Confusion means that the ciphertext should be a complex function of the key and plaintext, making it difficult to guess the key.
    - Diffusion means that a small change in the plaintext should cause a significant change in the ciphertext, which makes it difficult to analyze the encryption pattern.
4. **Key Size:** The key size should be large enough to prevent brute-force attacks. A larger key size means that there are more possible keys, making it harder for an attacker to guess the correct one. A key size of 128 bits is considered to be secure for most applications.
5. **Key Schedule:** The key schedule should be designed carefully to ensure that the keys used for encryption are independent and unpredictable. The key schedule should also resist attacks that exploit weak keys or key-dependent properties of the cipher.
6. **Block Size:** The block size should be large enough to prevent attacks that exploit statistical patterns in the plaintext.
7. **Non-linearity:** The S-box used in the cipher should be non-linear to provide confusion. A linear S-box is vulnerable to attacks that exploit the linear properties of the cipher.
8. **Avalanche Effect:** The cipher should exhibit the avalanche effect, which means that a small change in the plaintext or key should cause a significant change in the ciphertext.
9. **Security Analysis:** The cipher should be analyzed for its security against various attacks such as differential cryptanalysis, linear cryptanalysis, and brute-force attacks.

Symmetric Encryption alogirthms:

In symmetric key cryptography,

- Both sender and receiver use a common secret key to encrypt and decrypt the message.
- The major issue is exchanging the secret key between the sender and the receiver.
- Attackers might intrude and know the secret key while exchanging it.

1)DES(Data Encryption Standard) algorithm:

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only).

DES Algorithm Steps

To put it in simple terms, DES takes 64-bit plain text and turns it into a 64-bit ciphertext. And since we're talking about asymmetric algorithms, the same key is used when it's time to decrypt the text.

The algorithm process breaks down into the following steps:

1. The process begins with the 64-bit plain text block getting handed over to an initial permutation (IP) function.
2. The initial permutation (IP) is then performed on the plain text.
3. Next, the initial permutation (IP) creates two halves of the permuted block, referred to as Left Plain Text (LPT) and Right Plain Text (RPT).
4. Each LPT and RPT goes through 16 rounds of the encryption process.
5. Finally, the LPT and RPT are rejoined, and a Final Permutation (FP) is performed on the newly combined block.
6. The result of this process produces the desired 64-bit ciphertext.

The encryption process step (step 4, above) is further broken down into five stages:

1. Key transformation
2. Expansion permutation
3. S-Box permutation
4. P-Box permutation
5. XOR and swap

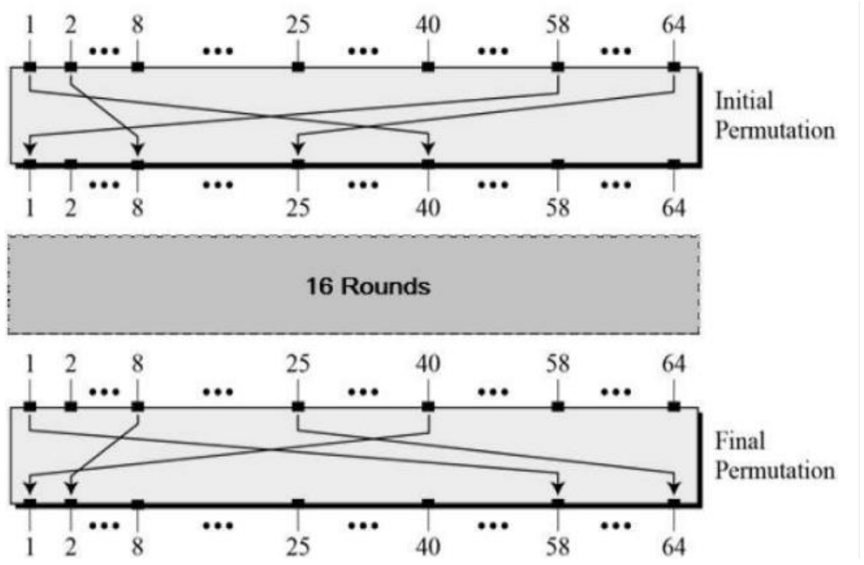
For decryption, we use the same algorithm, and we reverse the order of the 16 round keys.

Since DES is based on the Feistel Cipher, all that is required to specify DES is −

- Round function
- Key schedule
- Any additional processing − Initial and final permutation.

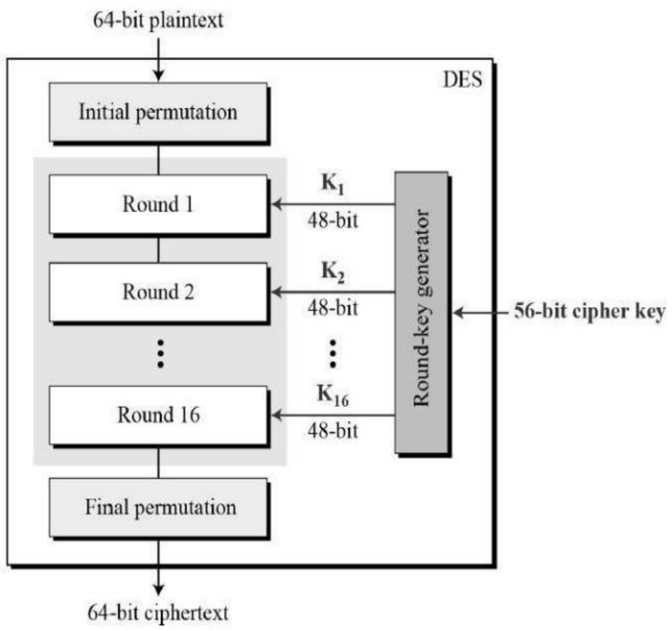
Initial and Final Permutation

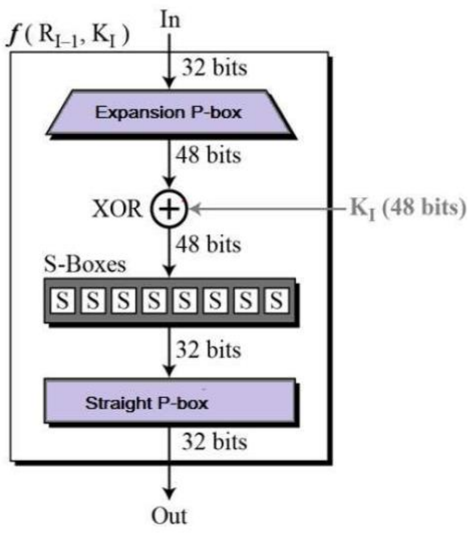
The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows −



<u>Round Function</u>

The heart of this cipher is the DES function, *f*. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.

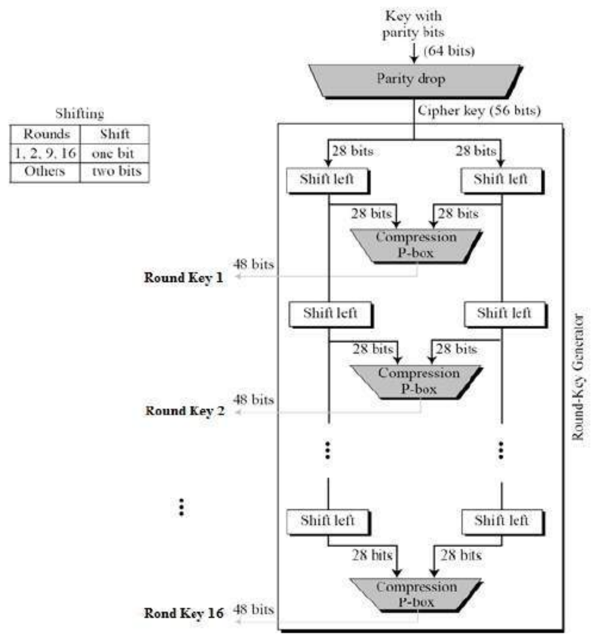




- **Expansion Permutation Box** − Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration −
  - **XOR (Whitener).** − After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.
  - **Substitution Boxes.** − The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output.
  - **Straight Permutation** − The 32 bit output of S-boxes is then subjected to the straight permutation.

<u>Key Generation</u>

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the following illustration −



-

The logic for Parity drop, shifting, and Compression P-box is given in the DES description.

DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- **Avalanche effect** − A small change in plaintext results in the very great change in the ciphertext.
- **Completeness** − Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

Applications of DES Algorithm

1. It is deployed when not-so-strong encryption is needed
2. It is used to develop a new form of DES, called Triple DES (using a 168-bit key formed using three keys)

2)Advanced Encryption Standard (AES) algorithm:

It is found at least six time faster than triple DES.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

The features of AES are as follows −

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C and Java

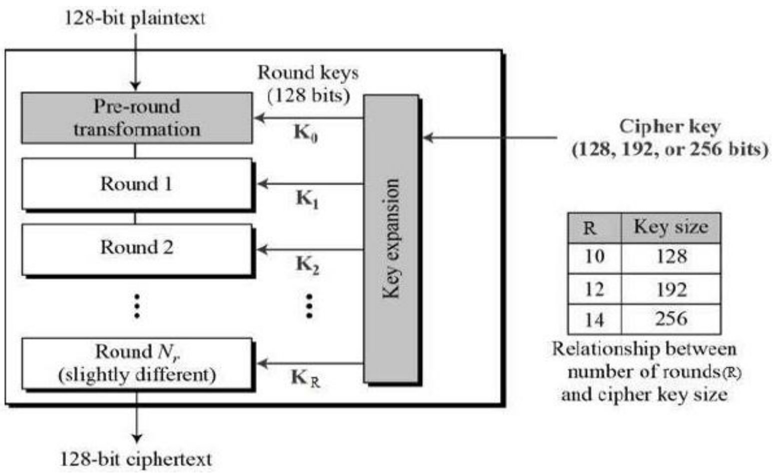
Operation of AES

AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

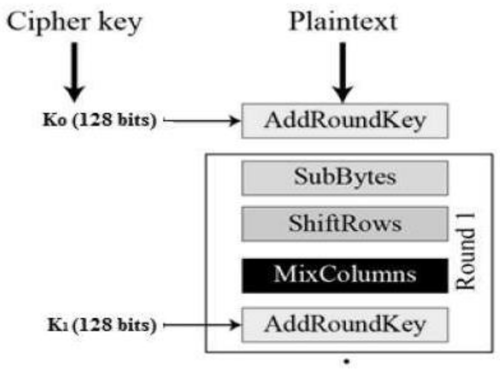
Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration −



Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

AES Analysis

In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of 'future-proofing' against progress in the ability to perform exhaustive key searches.

However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

3)Blowfish algorithm:

It is an encryption technique designed by **Bruce Schneier** in 1993 as an alternative to DES Encryption Technique. It is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date. It is one of the first, secure block cyphers not subject to any patents and hence freely available for anyone to use. It is symmetric block cipher algorithm.

1. **blockSize**: 64-bits
2. **keySize**: 32-bits to 448-bits variable size
3. **number of subkeys**: 18 [P-array]
4. **number of rounds**: 16
5. **number of substitution boxes**: 4 [each having 512 entries of 32-bits each]

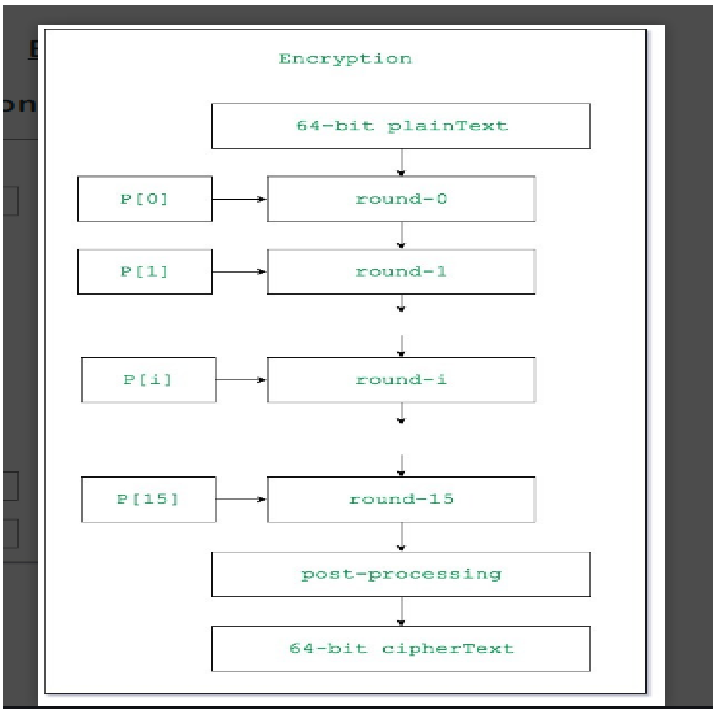
Step1: Generation of subkeys:

- 18 subkeys{P[0]…P[17]} are needed in both encryption as well as decryption process and the same subkeys are used for both the processes.
- These 18 subkeys are stored in a P-array with each array element being a 32-bit entry.
- It is initialized with the digits of pi(?).

The resultant P-array holds 18 subkeys that is used during the entire encryption process

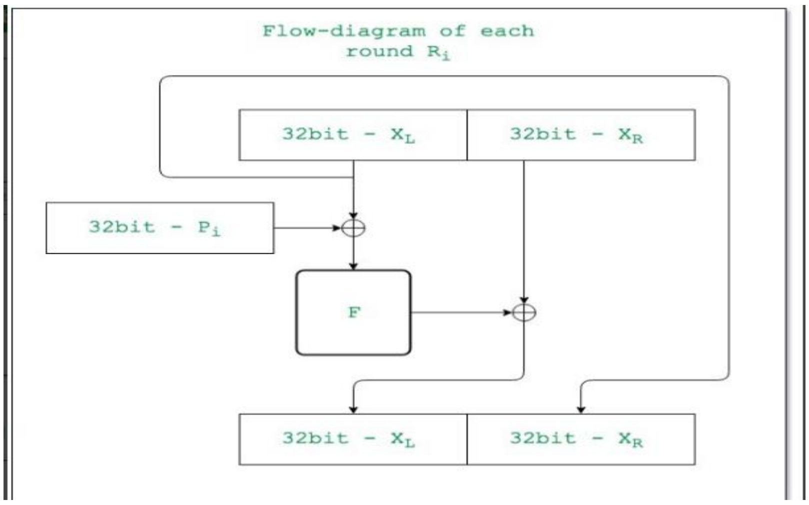
Step2: initialise Substitution Boxes:

- 4 Substitution boxes(S-boxes) are needed{S[0]…S[4]} in both encryption aswell as decryption process with each S-box having 256 entries{S[i][0]…S[i][255], 0&lei&le4} where each entry is 32-bit.
- It is initialized with the digits of pi(?) after initializing the P-array. You may find the **s-boxes** in here!

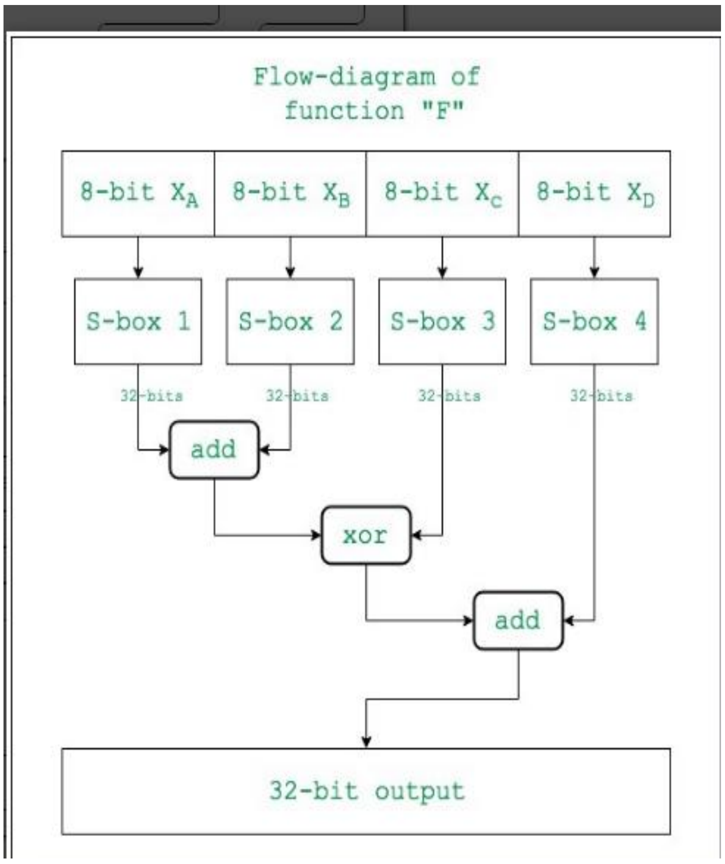


Step3: Encryption:

- The encryption function consists of two parts:
  a. Rounds: The encryption consists of 16 rounds with each round(Ri) taking inputs the plainText(P.T.) from previous round and corresponding subkey(Pi). The description of each round is as follows:



The description of the function ” F ” is as follows:



<u>4)RC5 Encryption Algorithm</u>

Word Size (bits)	**P (Hexadecimal)**	**Q (Hexadecimal)**
16	b7e1	9e37
32	b7e15163	9e3779b9
64	b7e151628aed2a6b	9e3779b97f4a7c15

Step-1: Here, Odd(x) is the odd integer nearest to x, e is the base of natural logarithms and is the golden ratio.

Step-2: Converting secret key K from bytes to words. Secret key K of size b bytes is used to initialize array L consisting of c words where c = b/u, u = w/8 and w = word size used for that particular instance of RC5. For example, if we choose w=32 bits and Key k is of size 96 bytes then, u=32/8=4, c=b/u=96/4=24. L is pre initialized to 0 value before adding secret key K to it.

Step-3: Initializing sub-key S. Sub-key S of size t=2(r+1) is initialized using magic constants P and Q.

```
S[0] = P
for i = 1 to 2(r+1)-1
    S[i] = S[i-1] + Q)
```

Step-4: Sub-key mixing. The RC5 encryption algorithm uses Sub key S. L is merely, a temporary array formed on the basis of user entered secret key. Mix in user's secret key with S and L.

Step-5: Encryption. We divide the input plain text block into two registers A and B each of size w bits. After undergoing the encryption process the result of A and B together forms the cipher text block. RC5 Encryption Algorithm:
Algorithm:

1. One time initialization of plain text blocks A and B by adding S[0] and S[1] to A and B respectively. These operations are mod .
2. XOR A and B. A=A^B
3. Cyclic left shift new value of A by B bits.
4. Add S[2*i] to the output of previous step. This is the new value of A.
5. XOR B with new value of A and store in B.
6. Cyclic left shift new value of B by A bits.
7. Add S[2*i+1] to the output of previous step. This is the new value of B.
8. Repeat entire procedure (except one time initialization) r times.

5)International Data Encryption Algorithm (IDEA)

The International Data Encryption Algorithm (IDEA) is a symmetric-key block cipher that was first introduced in 1991. It was designed to provide secure encryption for digital data and is used in a variety of applications, such as secure communications, financial transactions, and electronic voting systems.

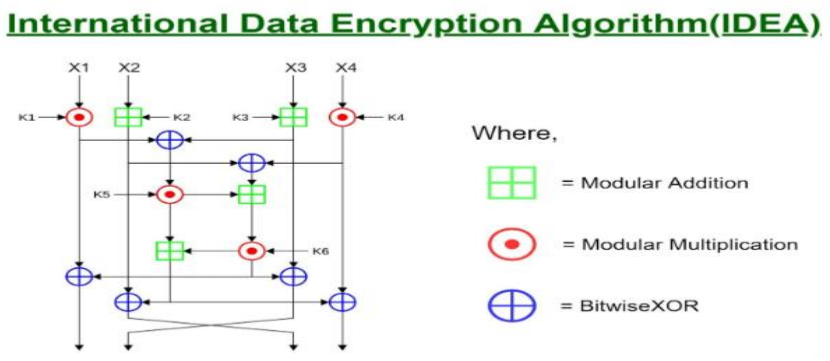
International Data Encryption Algorithm (IDEA) is a **symmetric key block** cipher that, it uses a fixed-length plaintext of **16 bits** and

- encrypts them in **4 chunks of 4 bits** each
- to produce **16 bits ciphertext**.
- The length of the key used is **32 bits**.
- The key is also divided into 8 blocks of 4 bits each.

This algorithm involves a series of 4 identical complete rounds and 1 half-round. Each complete round involves a series of 14 steps that includes operations like:

- Bitwise XOR
- Addition modulo
- Multiplication modulo +1

Key Schedule: 6 subkeys of 4 bits out of the 8 subkeys are used in each complete round, while 4 are used in the half-round.



	K1	K2	K3	K4	K5	K6
Round 1	1101	1100	0110	1111	0011	1111
Round 2	0101	1001*	0001	1011	1100	1111

	K1	K2	K3	K4	K5	K6
Round 3	1101	0110	0111	0111*	1111	0011
Round 4	1111	0101	1001	1101	1100	0110*
Round 4.5	1111	1101	0110	0111		

* denotes a shift of bits

Notations used in the 14 steps:

Symbol	Operation
*	Multiplication modulo +1
+	Addition modulo
^	Bitwise XOR

The 16-bit plaintext can be represented as **X1 || X2 || X3 || X4**, each of size 4 bits. The 32-bit key is broken into 8 subkeys denoted as K1 || K2 || K3 || K4 || K5 || K6 || K7 || K8, again of size 4 bits each. Each round of 14 steps uses the three algebraic operation-Addition modulo (2^4), Multiplication modulo (2^4)+1 and Bitwise XOR.

Block Cipher modes of Operation:

A block cipher processes the data blocks of fixed size. Usually, the size of a message is larger than the block size. Hence, the long message is divided into a series of sequential message blocks, and the cipher operates on these blocks one at a time.

1)Electronic Code Book (ECB) Mode

This mode is a most straightforward way of processing a series of sequentially listed message blocks.

Operation

- The user takes the first block of plaintext and encrypts it with the key to produce the first block of ciphertext.
- He then takes the second block of plaintext and follows the same process with same key and so on so forth.

The ECB mode is **deterministic**, that is, if plaintext block P1, P2,…, Pm are encrypted twice under the same key, the output ciphertext blocks will be the same.

In fact, for a given key technically we can create a codebook of ciphertexts for all possible plaintext blocks. Encryption would then entail only looking up for required plaintext and select the corresponding ciphertext. Thus, the operation is analogous to the assignment of code words in a codebook, and hence gets an official name − Electronic Codebook mode of operation (ECB). It is illustrated as follows −

Analysis of ECB Mode

In reality, any application data usually have partial information which can be guessed. For example, the range of salary can be guessed. A ciphertext from ECB can allow an attacker to guess the plaintext by trial-and-error if the plaintext message is within predictable.

For example, if a ciphertext from the ECB mode is known to encrypt a salary figure, then a small number of trials will allow an attacker to recover the figure. In general, we do not wish to use a deterministic cipher, and hence the ECB mode should not be used in most applications.

2)Cipher Block Chaining (CBC) Mode

CBC mode of operation provides message dependence for generating ciphertext and makes the system non-deterministic.

Operation

The operation of CBC mode is depicted in the following illustration. The steps are as follows

- Load the n-bit Initialization Vector (IV) in the top register.
- XOR the n-bit plaintext block with data value in top register.
- Encrypt the result of XOR operation with underlying block cipher with key K.
- Feed ciphertext block into top register and continue the operation till all plaintext blocks are processed.
- For decryption, IV data is XORed with first ciphertext block decrypted. The first ciphertext block is also fed into to register replacing IV for decrypting next ciphertext block.

Analysis of CBC Mode

In CBC mode, the current plaintext block is added to the previous ciphertext block, and then the result is encrypted with the key. Decryption is thus the reverse process, which involves decrypting the current ciphertext and then adding the previous ciphertext block to the result.

Advantage of CBC over ECB is that changing IV results in different ciphertext for identical message. On the drawback side, the error in transmission gets propagated to few further block during decryption due to chaining effect.

It is worth mentioning that CBC mode forms the basis for a well-known data origin authentication mechanism. Thus, it has an advantage for those applications that require both symmetric encryption and data origin authentication.

3)Cipher Feedback (CFB) Mode

In this mode, each ciphertext block gets ‘fed back’ into the encryption process in order to encrypt the next plaintext block.

Operation

The operation of CFB mode is depicted in the following illustration. For example, in the present system, a message block has a size ‘s’ bits where 1 < s < n. The CFB mode requires an initialization vector (IV) as the initial random n-bit input block. The IV need not be secret. Steps of operation are −

- Load the IV in the top register.
- Encrypt the data value in top register with underlying block cipher with key K.
- Take only ‘s’ number of most significant bits (left bits) of output of encryption process and XOR them with ‘s’ bit plaintext message block to generate ciphertext block.

- Feed ciphertext block into top register by shifting already present data to the left and continue the operation till all plaintext blocks are processed.
- Essentially, the previous ciphertext block is encrypted with the key, and then the result is XORed to the current plaintext block.
- Similar steps are followed for decryption. Pre-decided IV is initially loaded at the start of decryption.

Analysis of CFB Mode

CFB mode differs significantly from ECB mode, the ciphertext corresponding to a given plaintext block depends not just on that plaintext block and the key, but also on the previous ciphertext block. In other words, the ciphertext block is dependent of message.

CFB has a very strange feature. In this mode, user decrypts the ciphertext using only the encryption process of the block cipher. The decryption algorithm of the underlying block cipher is never used.

Apparently, CFB mode is converting a block cipher into a type of stream cipher. The encryption algorithm is used as a key-stream generator to produce key-stream that is placed in the bottom register. This key stream is then XORed with the plaintext as in case of stream cipher.

By converting a block cipher into a stream cipher, CFB mode provides some of the advantageous properties of a stream cipher while retaining the advantageous properties of a block cipher.

On the flip side, the error of transmission gets propagated due to changing of blocks.

4)Output Feedback (OFB) Mode

It involves feeding the successive output blocks from the underlying block cipher back to it. These feedback blocks provide string of bits to feed the encryption algorithm which act as the key-stream generator as in case of CFB mode.

The key stream generated is XOR-ed with the plaintext blocks. The OFB mode requires an IV as the initial random n-bit input block. The IV need not be secret.

The operation is depicted in the following illustration −

5)Counter (CTR) Mode

It can be considered as a counter-based version of CFB mode without the feedback. In this mode, both the sender and receiver need to access to a reliable counter, which computes a new shared value each time a ciphertext block is exchanged. This shared counter is not necessarily a secret value, but challenge is that both sides must keep the counter synchronized.

Operation

Both encryption and decryption in CTR mode are depicted in the following illustration. Steps in operation are −

- Load the initial counter value in the top register is the same for both the sender and the receiver. It plays the same role as the IV in CFB (and CBC) mode.
- Encrypt the contents of the counter with the key and place the result in the bottom register.
- Take the first plaintext block P1 and XOR this to the contents of the bottom register. The result of this is C1. Send C1 to the receiver and update the counter. The counter update replaces the ciphertext feedback in CFB mode.
- Continue in this manner until the last plaintext block has been encrypted.

- The decryption is the reverse process. The ciphertext block is XORed with the output of encrypted contents of counter value. After decryption of each ciphertext block counter is updated as in case of encryption.

Analysis of Counter Mode

It does not have message dependency and hence a ciphertext block does not depend on the previous plaintext blocks.

Like CFB mode, CTR mode does not involve the decryption process of the block cipher. This is because the CTR mode is really using the block cipher to generate a key-stream, which is encrypted using the XOR function. In other words, CTR mode also converts a block cipher to a stream cipher.

The serious disadvantage of CTR mode is that it requires a synchronous counter at sender and receiver. Loss of synchronization leads to incorrect recovery of plaintext.

However, CTR mode has almost all advantages of CFB mode. In addition, it does not propagate error of transmission at all.

Stream Ciphers:

1)RC4 is a stream cipher and variable-length key algorithm. This algorithm encrypts one byte at a time (or larger units at a time). A key input is a pseudorandom bit generator that produces a stream 8-bit number that is unpredictable without knowledge of input key, The output of the generator is called key-stream, is combined one byte at a time with the plaintext stream cipher using X-OR operation.

Example:

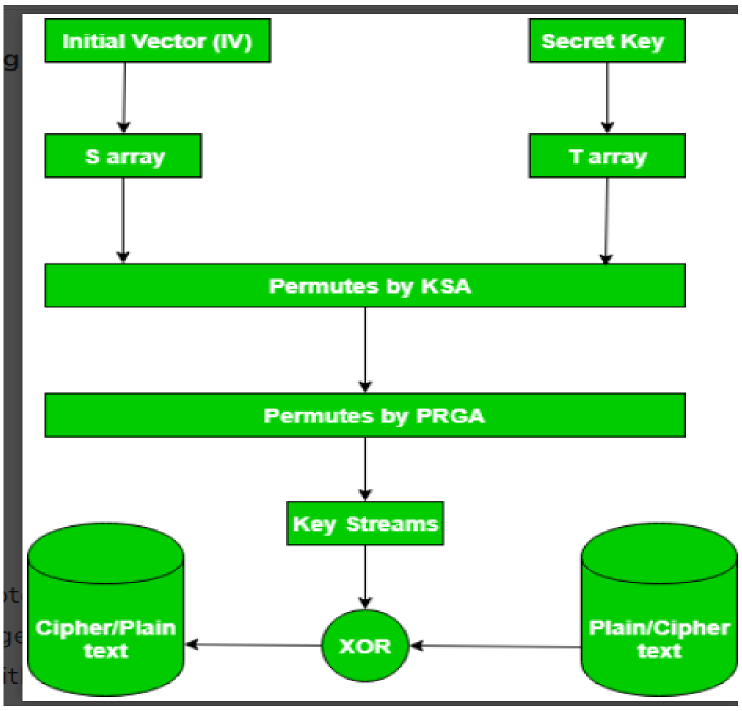
RC4 Encryption

10011000 ? 01010000 = 11001000

Key-Generation Algorithm – A variable-length key from 1 to 256 bytes is used to initialize a 256-byte state vector S, with elements S[0] to S[255]. For encryption and decryption, a byte k is generated from S by selecting one of the 255 entries in a systematic fashion, then the entries in S are permuted again.

Key-Scheduling Algorithm: Initialization: The entries of S are set equal to the values from 0 to 255 in ascending order, a temporary vector T, is created. If the length of the key k is 256 bytes, then k is assigned to T. Otherwise, for a key with length(k-len) bytes, the first k-len elements of T as copied from K, and then K is repeated as many times as necessary to fill T. use T to produce the initial permutation of S. Starting with S[0] to S[255], and for each S[i] algorithm swap it with another byte in S according to a scheme dictated by T[i], but S will still contain values from 0 to 255 :

Pseudo random generation algorithm (Stream Generation): Once the vector S is initialized, the input key will not be used. In this step, for each S[i] algorithm swap it with another byte in S according to a scheme dictated by the current configuration of S. After reaching S[255] the process continues, starting from S[0] again



Advantages:

1. **Fast and efficient:** RC4 is a very fast and efficient encryption algorithm, which makes it suitable for use in applications where speed and efficiency are critical.
2. **Simple to implement:** RC4 is a relatively simple algorithm to implement, which means that it can be easily implemented in software or hardware.
3. **Variable key size:** RC4 supports variable key sizes, which makes it flexible and adaptable for different security requirements.
4. **Widely used:** RC4 has been widely used in various applications, including wireless networks, secure sockets layer (SSL), virtual private networks (VPN), and file encryption.

Disadvantages:

1. **Vulnerabilities:** RC4 has several known vulnerabilities that make it unsuitable for new applications. For example, there is a bias in the first few bytes of the keystream, which can be exploited to recover the key.
2. **Security weaknesses:** RC4 has some inherent weaknesses in its design, which make it less secure than other encryption algorithms, such as AES or ChaCha20.
3. **Limited key length:** The maximum key length for RC4 is 2048 bits, which may not be sufficient for some applications that require stronger encryption.
4. **Not recommended for new applications:** Due to its vulnerabilities and weaknesses, RC4 is no longer recommended for use in new applications. Other more secure stream cipher algorithms, such as AES-CTR or ChaCha20, should be used instead.

Asymmetric keyCiphers:

In asymmetric encryption,

- Sender and receiver use different keys to encrypt and decrypt the message.
- The famous asymmetric encryption algorithms are-

Principles of public key cryptosystems

Public key Cryptosystem − Asymmetric algorithms depends on one key for encryption and a distinct but related key for decryption. These algorithms have the following characteristics which are as follows −

- It is computationally infeasible to decide the decryption key given only information of the cryptographic algorithm and the encryption key.
- There are two related keys such as one can be used for encryption, with the other used for decryption.

A public key encryption scheme has the following ingredients which are as follows −

- **Plaintext** − This is the readable message or information that is informer into the algorithm as input.
- **Encryption algorithm** − The encryption algorithm performs several conversion on the plaintext.
- **Public and Private keys** − This is a set of keys that have been selected so that if one can be used for encryption, and the other can be used for decryption.
- **Ciphertext** − This is scrambled message generated as output. It based on the plaintext and the key. For a given message, there are two specific keys will create two different ciphertexts.
- **Decryption Algorithm** − This algorithm get the ciphertext and the matching key and create the original plaintext.

Asymmetric Encryption Algorithms:

1)RSA Algorithm:

The RSA algorithm is a public-key signature algorithm founded by Ron Rivest, Adi Shamir, and Leonard Adleman. RSA can also encrypt and decrypt general data to securely exchange information along with managing digital signature verification.

The RSA algorithm is based on the complexity contained in the factorization of large numbers. The RSA algorithm depends on the fact that there is no effective method to factor very large numbers. Therefore, deducing an RSA key would take a large amount of time and processing power.

RSA algorithm is asymmetric cryptography algorithm as it operate on two different keys such as public key and private key. The public key is likely to everyone, and private key remains private. The public key includes two numbers, one of which is a multiplication of two large prime numbers.

There are the following steps in RSA Algorithm which are as follows −

- **Generating the keys**
  - Choose two large prime numbers, such as P and Q. The prime numbers required to be large so that they will be complex for someone to figure out.
  - Compute N = P x Q
  - Choose the public key (i.e., the Encryption key) E such that it is not a factor of (P-1) and (Q-1).
  - Choose the private key (i.e., the decryption key) D such that the following equation is true −

    $(D \times E) \bmod (P - 1) \times (Q - 1) = 1$
  - For encryption, compute the cipher text (CT) from the plain text (PT) as follows −

    $CT = PT^E \bmod N$
  - Send CT as the cipher text to the receiver.
  - For encryption, compute the plain text (PT) from the cipher text (CT) as follows −

$$PT = CT^D \bmod N$$

- **Encryption/Decryption Function** − Once it can generate the keys, and it can pass the parameters to the functions that calculate the ciphertext and plaintext using the respective key.
  - If the plaintext is m, ciphertext = me mod n.
  - If the ciphertext is c, plaintext = cd mod n
- For example, where p = 17 and q=13. Value of e can be 5 as it satisfies the condition 1 < e < (p-1) (q-1).
  N = p * q = 91
  D = e-1 mod (p-1) (q-1) = 29
  Public Key pair = (91, 5)
  Private Key pair = (91, 29)
  If the plaintext (m) value is 10, it can encode it utilizing the formula me mod n = 82.
  To decrypt this ciphertext(c) back to original data, it should use the formula cd mod n = 29.

2)ElGamal Cryptosystem

Along with RSA, there are other public-key cryptosystems proposed. Many of them are based on different versions of the Discrete Logarithm Problem.

ElGamal cryptosystem, called Elliptic Curve Variant, is based on the Discrete Logarithm Problem. It derives the strength from the assumption that the discrete logarithms cannot be found in practical time frame for a given number, while the inverse operation of the power can be computed efficiently.

Generation of ElGamal Key Pair
Each user of ElGamal cryptosystem generates the key pair through as follows −

- **Choosing a large prime p.** Generally a prime number of 1024 to 2048 bits length is chosen.
- **Choosing a generator element g.**
  - This number must be between 1 and p âˆ' 1, but cannot be any number.
  - It is a generator of the multiplicative group of integers modulo p. This means for every integer m co-prime to p, there is an integer k such that g^k=a mod n. For example, 3 is generator of group 5 (Z_5 = {1, 2, 3, 4}).

N	**3^n**	**3^n mod 5**
1	3	3
2	9	4
3	27	2
4	81	1

- **Choosing the private key.** The private key x is any number bigger than 1 and smaller than pâˆ'1.
- **Computing part of the public key.** The value y is computed from the parameters p, g and the private key x as follows −

$y = g^x$ mod p

- **Obtaining Public key.** The ElGamal public key consists of the three parameters (p, g, y).

For example, suppose that p = 17 and that g = 6 (It can be confirmed that 6 is a generator of group Z_{17}). The private key x can be any number bigger than 1 and smaller than 71, so we choose x = 5. The value y is then computed as follows −

$y = 6^5 \bmod 17 = 7$

- Thus the private key is 62 and the public key is (17, 6, 7).

Encryption and Decryption

The generation of an ElGamal key pair is comparatively simpler than the equivalent process for RSA. But the encryption and decryption are slightly more complex than RSA.

ElGamal Encryption

Suppose sender wishes to send a plaintext to someone whose ElGamal public key is (p, g, y), then −

- Sender represents the plaintext as a series of numbers modulo p.
- To encrypt the first plaintext P, which is represented as a number modulo p. The encryption process to obtain the ciphertext C is as follows −
  - Randomly generate a number k;
  - Compute two values C1 and C2, where −

$C1 = g^k \bmod p$

$C2 = (P^*y^k) \bmod p$

- Send the ciphertext C, consisting of the two separate values (C1, C2), sent together.
- Referring to our ElGamal key generation example given above, the plaintext P = 13 is encrypted as follows −
  - Randomly generate a number, say k = 10
  - Compute the two values C1 and C2, where −

$C1 = 6^{10} \bmod 17$

$C2 = (13^* 7^{10}) \bmod 17 = 9$

- Send the ciphertext C = (C1, C2) = (15, 9).

ElGamal Decryption

- To decrypt the ciphertext (C1, C2) using private key x, the following two steps are taken −
  - Compute the modular inverse of $(C1)^x$ modulo p, which is $(C1)^{-x}$, generally referred to as decryption factor.
  - Obtain the plaintext by using the following formula −

C2 Ã— $(C1)^{-x}$ mod p = Plaintext

- In our example, to decrypt the ciphertext C = (C1, C2) = (15, 9) using private key x = 5, the decryption factor is

$15^{-5} \bmod 17 = 9$

- Extract plaintext P = (9 Ã— 9) mod 17 = 13.

ElGamal Analysis

In ElGamal system, each user has a private key x. and has **three components** of public key − **prime modulus p, generator g, and public $Y = g^x$ mod p**. The strength of the ElGamal is based on the difficulty of discrete logarithm problem.

The secure key size is generally > 1024 bits. Today even 2048 bits long key are used. On the processing speed front, Elgamal is quite slow, it is used mainly for key authentication protocols.

Due to higher processing efficiency, Elliptic Curve variants of ElGamal are becoming increasingly popular.

3)Diffie Hellman Key Exchange-

- This algorithm is used to exchange the secret key between the sender and the receiver.
- This algorithm facilitates the exchange of secret key without actually transmitting it.

Let-

- Private key of the sender = X_s
- Public key of the sender = Y_s
- Private key of the receiver = X_r
- Public key of the receiver = Y_r

Using Diffie Hellman Algorithm, the key is exchanged in the following steps-

Step-01:

One of the parties choose two numbers ‘a’ and ‘n’ and exchange with the other party.

- ‘a’ is the primitive root of prime number ‘n’.
- After this exchange, both the parties know the value of ‘a’ and ‘n’.

Step-02:

Both the parties already know their own private key.

- Both the parties calculate the value of their public key and exchange with each other.

Sender calculate its public key as-

$\mathbf{Y_s = a^{X_s} \bmod n}$

Receiver calculate its public key as-

$\mathbf{Y_r = a^{X_r} \bmod n}$

Step-03:

Both the parties receive public key of each other.

- Now, both the parties calculate the value of secret key.

Sender calculates secret key as-

Secret key = $\mathbf{(Y_r)^{X_s} \bmod n}$

Receiver calculates secret key as-

Secret key = $\mathbf{(Y_s)^{X_r} \bmod n}$

Finally, both the parties obtain the same value of secret key.

Example:

Suppose that two parties A and B wish to set up a common secret key (D-H key) between themselves using the Diffie Hellman key exchange technique. They agree on 7 as the modulus and 3 as the primitive root. Party A chooses 2 and party B chooses 5 as their respective secrets. Their D-H key is-

Solution-

Given-

- n = 7
- a = 3
- Private key of A = 2
- Private key of B = 5

Step-01:

Both the parties calculate the value of their public key and exchange with each other.

Public key of A

= $3^{\text{private key of A}}$ mod 7

= 3^2 mod 7

= 2

Public key of B

= $3^{\text{private key of B}}$ mod 7

= 3^5 mod 7

= 5

Step-02:

Both the parties calculate the value of secret key at their respective side.

Secret key obtained by A

= $5^{\text{private key of A}}$ mod 7

= 5^2 mod 7

= 4

Secret key obtained by B

= $2^{\text{private key of B}}$ mod 7

= 2^5 mod 7

= 4

Finally, both the parties obtain the same value of secret key.

The value of common secret key = 4.

4)Knapsack Encryption Algorithm is the first general public key cryptography algorithm. It is developed by **Ralph Merkle** and **Mertin Hellman** in 1978. As it is a Public key cryptography, it needs two different keys. One is Public key which is used for Encryption process and the other one is Private key which is used for Decryption process. In this algorithm we will use two different knapsack problems in which one is easy and other one is hard. The easy knapsack is used as the private key and the hard knapsack is used as the public key. The easy knapsack is used to derived the hard knapsack.

For the easy knapsack, we will choose a **Super Increasing knapsack problem**. Super increasing knapsack is a sequence in which every next term is greater than the sum of all preceding terms.

Example –

{1, 2, 4, 10, 20, 40} is a super increasing as
1<2, 1+2<4, 1+2+4<10, 1+2+4+10<20 and 1+2+4+10+20<40.

Derive the Public key

- **Step-1:**
  Choose a super increasing knapsack {1, 2, 4, 10, 20, 40} as the private key.
- **Step-2:**
  Choose two numbers n and m. Multiply all the values of private key by the number n and then find modulo m. The value of m must be greater than the sum of all values in private key.
- **Step-3:**
  Calculate the values of Public key using m and n.

1x31 mod(110) = 31
2x31 mod(110) = 62
4x31 mod(110) = 14
10x31 mod(110) = 90
20x31 mod(110) = 70
40x31 mod(110) = 30

- Thus, our public key is {31, 62, 14, 90, 70, 30}
  And Private key is {1, 2, 4, 10, 20, 40}.
  - Now take an example for understanding the process of encryption and decryption.
  - **Example –**
    Lets our plain text is 100100111100101110.
  - **1. Encryption :**
    As our knapsacks contain six values, so we will split our plain text in a groups of six:
  - 100100 111100 101110

Multiply each values of public key with the corresponding values of each group and take their sum.

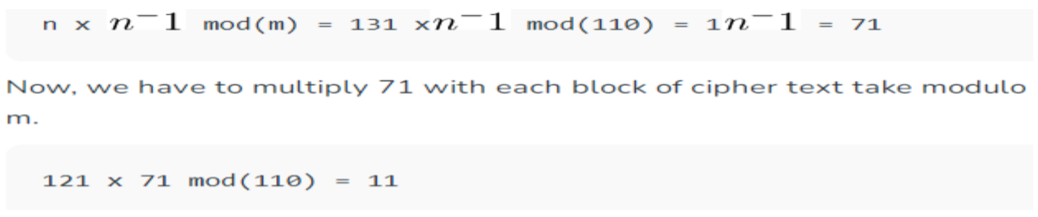
100100 {31, 62, 14, 90, 70, 30}
1x31+0x62+0x14+1x90+0x70+0x30 = 121

111100 {31, 62, 14, 90, 70, 30}
1x31+1x62+1x14+1x90+0x70+0x30 = 197

101110 {31, 62, 14, 90, 70, 30}
1x31+0x62+1x14+1x90+1x70+0x30 = 205
So, our cipher text is 121 197 205.

2. Decryption :
The receiver receive the cipher text which has to be decrypt. The receiver also knows the values of m and n.

First we need to find n-1which is multiplicative inverse of n mod m.



```
n x n⁻1 mod(m) = 131 xn⁻1 mod(110) = 1n⁻1 = 71
```

Now, we have to multiply 71 with each block of cipher text take modulo m.

```
121 x 71 mod(110) = 11
```