➢ Conventional Encryption Algorithms

Conventional Encryption involves transforming plaintext messages into ciphertext messages that are to be decrypted only by the intended receiver. Both sender and receiver agree upon a secrete key to be used in encrypting and decrypting. Usually the secrete key is transmitted via public key encryption methods.

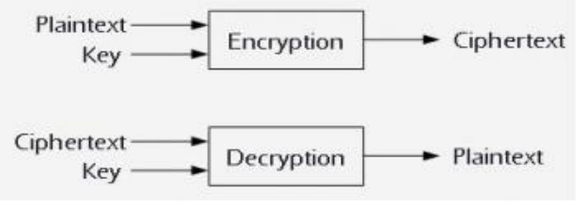


Figure 1: Flow Diagram

In conventional encryption, it is assumed that it is mathematically impossible to derive the plaintext from the ciphertext without the key.Therefore, it is essential that the key remains secret.

These encryption algorithms are used in practice due to their efficiency in encrypting/decrypting but these algorithms have vulnerabilities. One aspect of these vulnerabilities is the total number of keys available to choose from. Larger key domains reduce possibility of brute force attacks. The key length is another aspect of these vulnerabilities since they will produce periodic patterns in the ciphertext. Longer keys often reduce periodicity. The goal of conventional encryption algorithms is to produce truly randomized ciphertexts, such that the use of frequency analysis on individual ciphertext symbols or ciphertext blocks is useless.

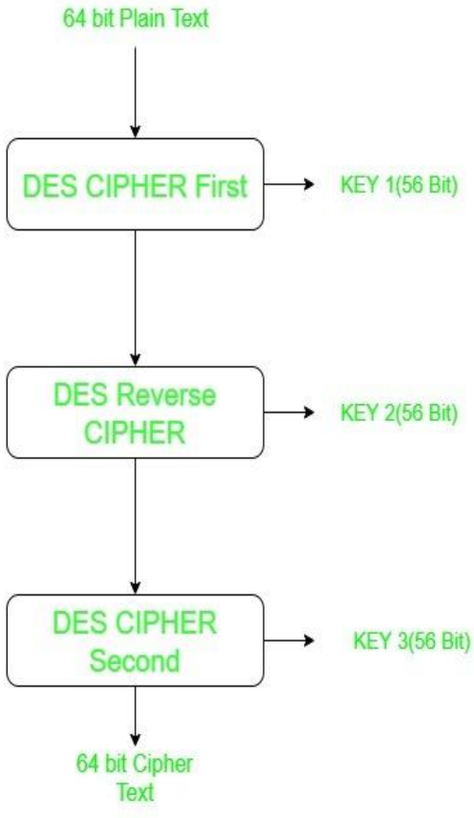
➢ Triples DES

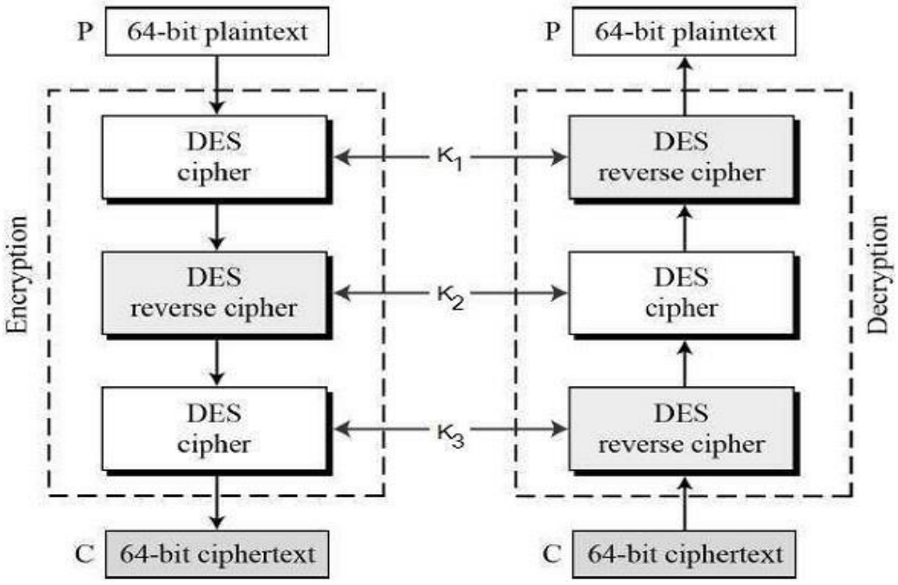
Triple DES − It is a variant scheme based on repeated DES applications. It is still a respected block ciphers but inefficient compared to the new faster block ciphers available.

Triple Data Encryption Standard (DES) is a type of computerized cryptography where block cipher algorithms are applied three times to each data block. The key size is increased in Triple DES to ensure additional security through encryption capabilities. Each block contains 64 bits of data. Three keys are referred to as bundle keys with 56 bits per key. There are three keying options in data encryption standards:

1. All keys being independent
2. Key 1 and key 2 being independent keys
3. All three keys being identical

Key option #3 is known as triple DES. The triple DES key length contains 168 bits but the key security falls to 112 bits.





The encryption scheme is illustrated as follows −

The encryption-decryption process is as follows −

- Encrypt the plaintext blocks using single DES with key K_1.
- Now decrypt the output of step 1 using single DES with key K_2.
- Finally, encrypt the output of step 2 using single DES with key K_3.
- The output of step 3 is the ciphertext.
- Decryption of a ciphertext is a reverse process. User first decrypt using K_3, then encrypt with K_2, and finally decrypt with K_1.

Due to this design of Triple DES as an encrypt–decrypt–encrypt process, it is possible to use a 3TDES (hardware) implementation for single DES by setting K_1, K_2, and K_3 to be the same value. This provides backwards compatibility with DES.

Second variant of Triple DES (2TDES) is identical to 3TDES except that $K_3$is replaced by K_1. In other words, user encrypt plaintext blocks with key K_1, then decrypt with key K_2, and finally encrypt with K_1 again. Therefore, 2TDES has a key length of 112 bits.

Triple DES systems are significantly more secure than single DES, but these are clearly a much slower process than encryption using single DES.

➢ International Data Encryption Algorithm

In cryptography, the **International Data Encryption Algorithm** (**IDEA**), originally called **Improved Proposed Encryption Standard** (**IPES**), is a symmetric-key block cipher In cryptography, block ciphers are very important in the designing of many cryptographic algorithms and are widely used to encrypt the bulk of data in chunks. By chunks, it means that the cipher takes a fixed size of the plaintext in the encryption process and generates a fixed size ciphertext using a fixed-length key. An algorithm's strength is determined by its key length. The Simplified **International Data Encryption Algorithm (IDEA)** is a **symmetric key block** cipher that

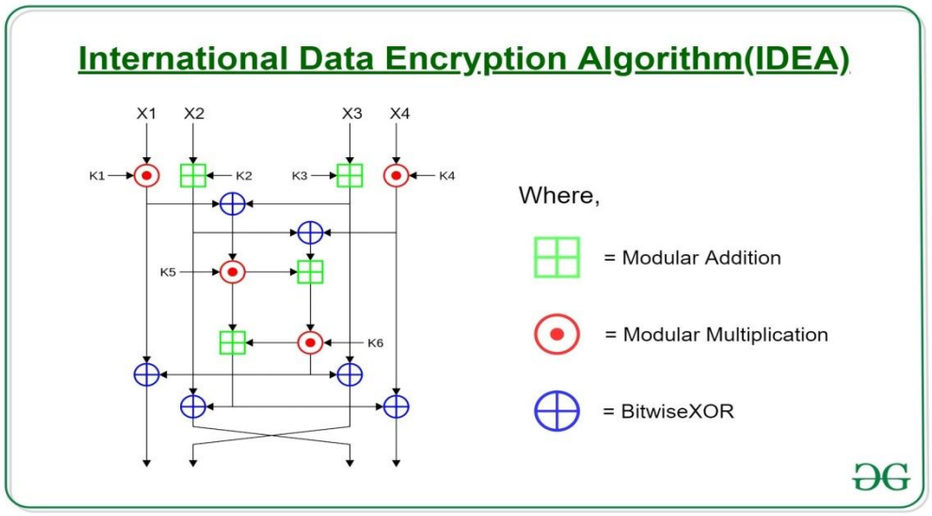
- uses a fixed-length plaintext of **16 bits** and
- encrypts them in **4 chunks of 4 bits** each
- to produce **16 bits ciphertext**.
- The length of the key used is **32 bits**.
- The key is also divided into 8 blocks of 4 bits each.

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

After 4 complete rounds, the final "half-round" consists of only the first 4 out of the 14 steps previously used in the full rounds. To perform these rounds, each binary notation must be converted to its equivalent decimal notation, perform the operation and the result obtained should be converted back to the binary representation for the final result of that particular step.

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. So, 4.5 rounds require 28 subkeys. The given key, 'K', directly gives the first 8 subkeys. By rotating the main key left by 6 bits between each group of 8, further groups of 8 subkeys are created, implying less than one rotation per round for the key (3 rotations).



	K1	K2	K3	K4	K5	K6
Round 1	1101	1100	0110	1111	0011	1111
Round 2	0101	1001*****	0001	1011	1100	1111
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. The steps involved are as follows:

1. X1 * K1
2. X2 + K2
3. X3 + K3

4. X4 * K4
5. Step 1 ^ Step 3
6. Step 2 ^ Step 4
7. Step 5 * K5
8. Step 6 + Step 7
9. Step 8 * K6
10. Step 7 + Step 9
11. Step 1 ^ Step 9
12. Step 3 ^ Step 9
13. Step 2 ^ Step 10
14. Step 4 ^ Step 10

The input to the next round is Step 11 || Step 13 || Step 12 || Step 14, which becomes X1 || X2 || X3 || X4. This swap between 12 and 13 takes place after each complete round, except the last complete round (4th round), where the input to the final half round is Step 11 || Step 12 || Step 13 || Step 14.

After last complete round, the half-round is as follows:

1. X1 * K1
2. X2 + K2
3. X3 + K3
4. X4 * K4

The final output is obtained by concatenating the blocks.

Example:

Key: 1101 1100 0110 1111 0011 1111 0101 1001
Plaintext: 1001 1100 1010 1100
Ciphertext: 1011 1011 0100 1011

Explanation:

The explanation is only for 1st complete round (the remaining can be implemented similarly) and the last half-round.

Round 1:

- From the plaintext: **X1 – 1001, X2 – 1100, X3 – 1010, X4 – 1100**

From the table above: **K1 – 1101, K2 – 1100, K3 – 0110, K4 – 1111, K5 – 0011, K6 – 1111**

```
(1001(9) * 1101(13))(mod 17) = 1111(15)
(1100(12) + 1100(12))(mod 16) = 1000(8)
(1010(10) + 0110(6))(mod 16) = 0000(0)

(1100(12) * 1111(15))(mod 17) = 1010(10)

(1111(15) ^ 0000(0)) = 1111(15)

(1000(8) ^ 1010(10)) = 0010(2)

(1111(15) * 0011(3))(mod 17) = 1011(11)

(0010(2) + 1011(11))(mod 16) = 1101(13)

(1101(13) * 1111(15))(mod 17) = 1000(8)

(1011(11) + 1000(8))(mod 16) = 0011(3)
```

(1000(8) ^ 1111(15)) = 0111(7)

(1000(8) ^ 0000(0)) = 1000(8)

(0011(3) ^ 1000(8)) = 1011(11)

(0011(3) ^ 1010(10)) = 1001(9)

- **Round 1 Output**: 0111 1011 1000 1001 (Step 12 and Step 13 results are interchanged*)*
- **Round 2:**
- From Round 1 output: **X1 – 0111, X2 – 1011, X3 – 1000, X4 – 1001**

- From the table above: **K1 – 0101, K2 – 1001, K3 – 0001, K4 – 1011, K5 – 1100, K6 – 1111**

- **Round 2 Output**: 0110 0110 1110 1100 *(Step 12 and Step 13 results are interchanged)*
- **Round 3:**
- From Round 2 Output: **X1 – 0110, X2 – 0110, X3 – 1110, X4 – 1100**

- From the table above: **K1 – 1101, K2 – 0110, K3 – 0111, K4 – 0111, K5 – 1111, K6 – 0011**

- **Round 3 Output**: 0100 1110 1011 0010 *(Step 12 and Step 13 results are interchanged)*
- **Round 4:**
- From Round 3 Output: **X1 – 0100, X2 – 1110, X3 – 1011, X4 – 0010**

- From the table above: **K1 – 1111, K2 – 0101, K3 – 1001, K4 – 1101, K5 – 1100, K6 – 0110**

- **Round 4 Output**: 0011 1110 1110 0100 *(Step 12 and Step 13 results are interchanged)*
- **Round 4.5:**
- From Round 4 Output: **X1 – 0011, X2 – 1110, X3 – 1110, X4 – 0100**

- From the table above: **K1 – 1111, K2 – 1101, K3 – 0110, K4 – 0111**

- **Round 4.5 Output**: 1011 1011 0100 1011 (Step 2 and Step 3 results are **not** interchanged)

(0011(3) * 1111(15))(mod 17) = 1011(11)

(1110(14) + 1101(13))(mod 16) = 1011(11)

(1110(14) + 0110(6))(mod 16) = 0100(4)

(0100(4) * 0111(7))(mod 17) = 1011(11)

- **Final Ciphertext** is **1011 1011 0100 1011**

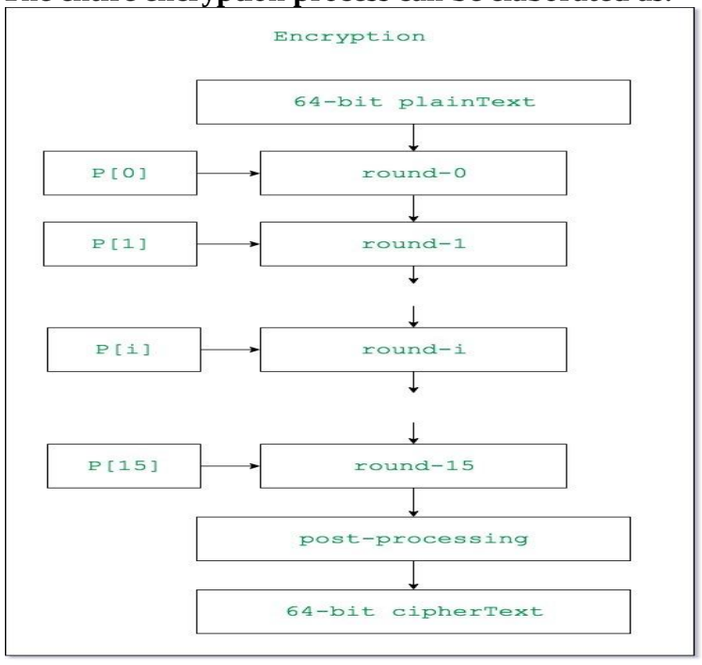
➢ Blowfish

Blowfish 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.

1. block Size: 64-bits
2. key Size: 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]

Blowfish Encryption Algorithm

The entire encryption process can be elaborated as:



Let's see each step one by one:

Step1: Generation of subkeys:

- 18 sub keys{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 hexadecimal representation of each of the subkeys is given by:

P[0] = "243f6a88"
P[1] = "85a308d3"
.
.
.
P[17] = "8979fb1b"

32-bit hexadecimal representation of initial values of sub-keys

P[0]	: 243f6a88	P[9]	: 38d01377
P[1]	: 85a308d3	P[10]	: be5466cf
P[2]	: 13198a2e	P[11]	: 34e90c6c
P[3]	: 03707344	P[12]	: c0ac29b7
P[4]	: a4093822	P[13]	: c97c50dd
P[5]	: 299f31d0	P[14]	: 3f84d5b5
P[6]	: 082efa98	P[15]	: b5470917
P[7]	: ec4e6c89	P[16]	: 9216d5d9
P[8]	: 452821e6	P[17]	: 8979fb1b

- Now each of the subkey is changed with respect to the input key as:

P[0] = P[0] xor 1st 32-bits of input key

P[1] = P[1] xor 2nd 32-bits of input key

.
.
.
P[i] = P[i] xor (i+1)th 32-bits of input key

(roll over to 1st 32-bits depending on the key length)

.
.
.

P[17] = P[17] xor 18th 32-bits of input key
(roll over to 1st 32-bits depending on key length)
The resultant P-array holds 18 subkeys that is used during the entire encryption process

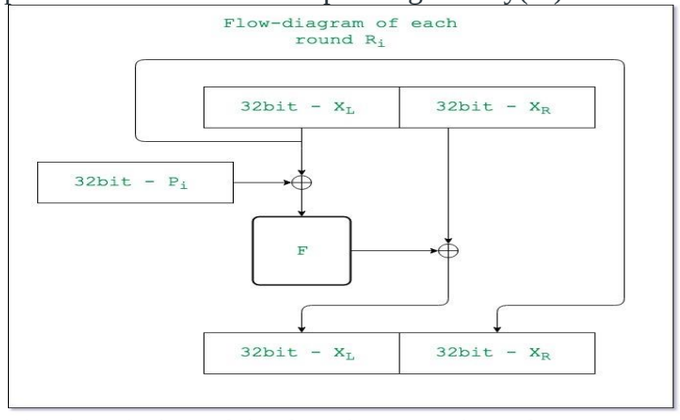
Step2: initialise Substitution Boxes:

- 4 Substitution boxes(S-boxes) are needed{S[0]…S[4]} in both encryption as well 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.

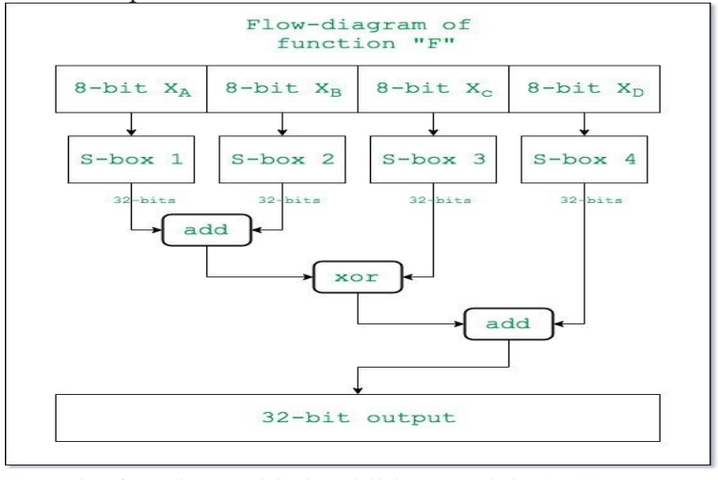
Step3: Encryption:

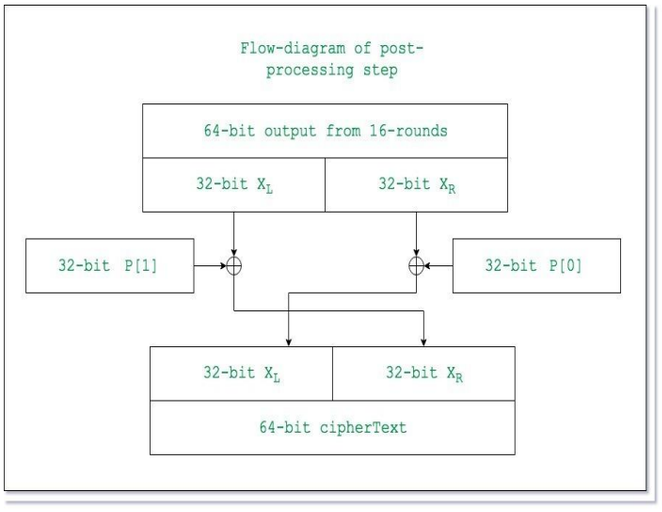
- The encryption function consists of two parts:

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:



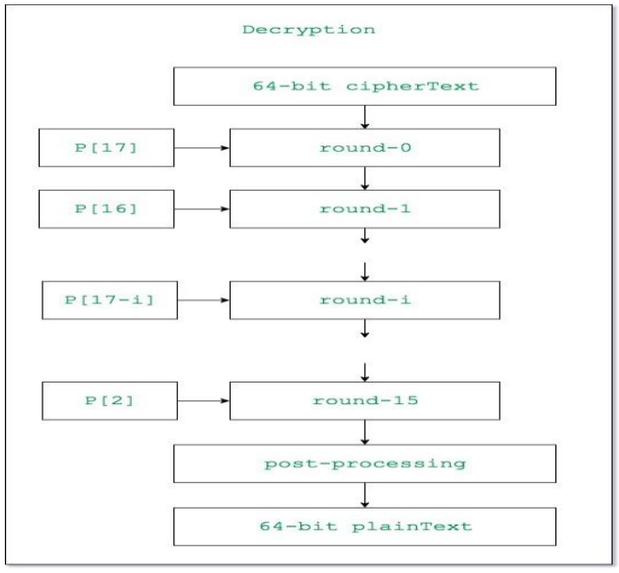


Here the function “add” is addition modulo 2^32.

b. Post-processing: The output after the 16 rounds is processed as follows:

Decryption

The decryption process is similar to that of encryption and the subkeys are used in reverse {P[17] – P[0]}. The entire decryption process can be elaborated as:



Let's see each step one by one:

Step1: Generation of subkeys:

- 18 subkeys {P[0]…P[17]} are needed in decryption process.
- 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 hexadecimal representation of each of the subkeys is given by:

P[0] = "243f6a88"
P[1] = "85a308d3"
.
.
.
P[17] = "8979fb1b"
P[0] = P[0] xor 1st 32-bits of input key

P[1] = P[1] xor 2nd 32-bits of input key
.
.
.
P[i] = P[i] xor (i+1) th 32-bits of input key
(roll over to 1st 32-bits depending on the key length)
.
.
.
P [17] = P[17] xor 18th 32-bits of input key
(roll over to 1st 32-bits depending on key length)

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

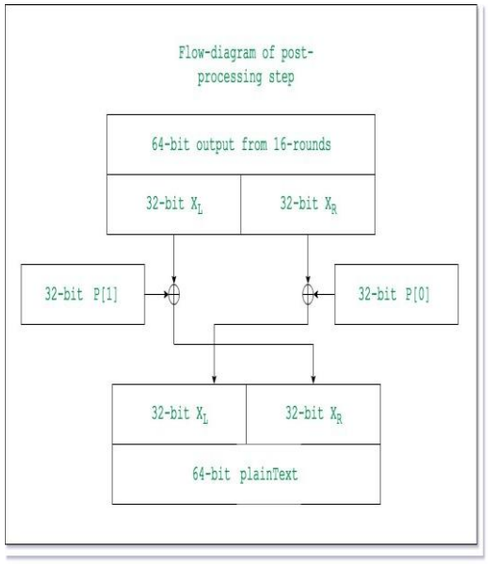
Step2: initialise Substitution Boxes:

- 4 Substitution boxes(S-boxes) are needed{S[0]…S[4]} in both encryption as well 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 initialised with the digits of pi(?) after initializing the P-array.

Step3: Decryption:

- The Decryption function also consists of two parts:

1. **Rounds:** The decryption also consists of 16 rounds with each round(Ri)(as explained above) taking inputs the cipherText(C.T.) from previous round and corresponding subkey(P[17-i])(i.e for decryption the subkeys are used in reverse).



2. **Post-processing:** The output after the 16 rounds is processed as follows:

Note: See encryption for the initial values of P-array.

- Now each of the subkeys is changed with respect to the input key as:

Advantages and Disadvantages of Blowfish Algorithm:

- Blowfish is a fast block cipher except when changing keys. Each new key requires a pre-processing equivalent to 4KB of text.
- It is faster and much better than DES Encryption.
- Blowfish uses a 64-bit block size which makes it vulnerable to birthday attacks.
- A reduced round variant of blowfish is known to be susceptible to known plain text attacks (2nd order differential attacks – 4 rounds).

Applications of Blowfish Algorithm:

- Bulk Encryption.
- Packet Encryption(ATM Packets)
- Password Hashing

➢ RC5

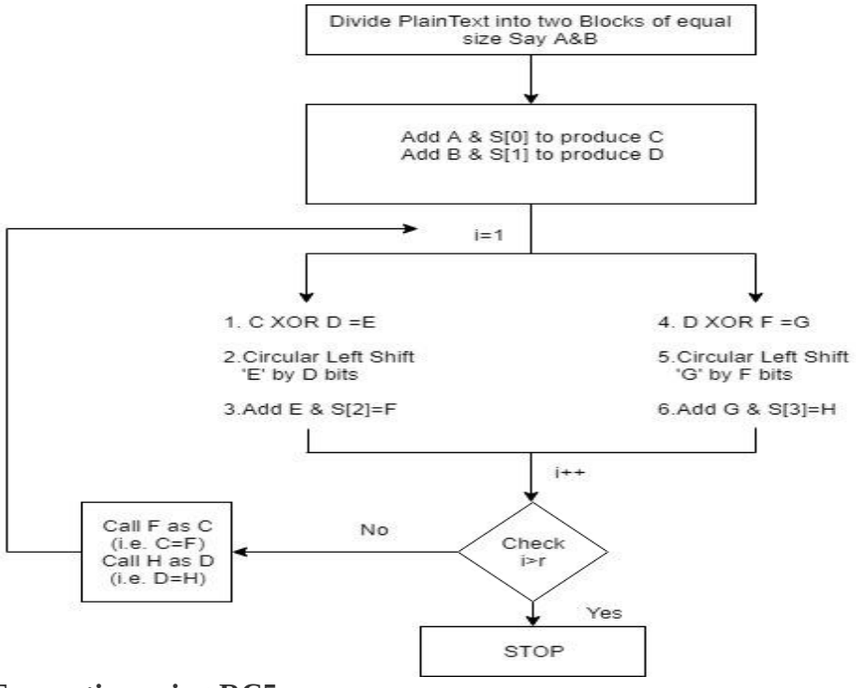
In cryptography, **RC5** is a symmetric-key block cipher notable for its simplicity. Designed by Ronald Rivest in 1994, *RC* stands for "Rivest Cipher", or alternatively, "Ron's Code" (compare RC2 and RC4).

Unlike many schemes, RC5 has a variable block size (32, 64 or 128 bits), key size (0 to 2040 bits) and number of rounds (0 to 255). The original suggested choice of parameters was a block size of 64 bits, a 128-bit key and 12 rounds.

A key feature of RC5 is the use of data-dependent rotations; one of the goals of RC5 was to prompt the study and evaluation of such operations as a cryptographic primitive. RC5 also consists of a number of modular additions and eXclusive OR (XOR)s. The general structure of the algorithm is a Feistel-like network. The encryption and decryption routines can be specified in a few lines of code. The key schedule, however, is more complex, expanding the key using an essentially one-way function with the binary expansions of both e and the golden ratio as sources of "nothing up my sleeve numbers". The tantalising simplicity of the algorithm together with the novelty of the data-dependent rotations has made RC5 an attractive object of study for cryptanalysts. The RC5 is basically denoted as RC5-w/r/b where w=word size in bits, r=number of rounds, b=number of 8-bit bytes in the key.

RC-5 Algorithm

- In RC-5, the word size (i.e. input plaintext block size), number of rounds and number of keys are not fixed i.e. all can be of variable length.
- Once w, r, k (word size, number of rounds, number of keys) are finalized then they remain same for all the rounds.
- Plain text can be 32 bits, 64 bits or 128 bits
- Number of rounds can be between 0-255
- Key size can be between 0 to 255 bytes



Encryption using RC5

Encryption involved several rounds of a simple function. 12 or 20 rounds seem to be recommended, depending on security needs and time considerations.

We initialize the counter to 1 and perform some permutation and combination using addition and XOR

The algorithm works into two phases:

a. First it starts with phase one

b. Output of phase one become input of phase two

We divide the plaintext block into two equal parts A and B

Then they are XOR with two subkeys S{0} and S{1}

C=A+S[0]C=A+S[0]

D=B+S[1]D=B+S[1]

for i = 1 to r do:

1. C ⊕ D = E
2. perform circular left shift on E by D bits
3. add E and S[2 * i] and store the result in F which is input for step 4
4. D ⊕ F = G
5. perform circular left shift on G by F bits
6. add G and S[2 * i + 1] and store the result in H
7. If i< r

Call F as C and H as D and repeat the steps from 1 to 7

else stop

- Once both the phases are completed the counter is incriminated and we check if it is greater than the number of rounds, if yes then the algorithm terminals and if no then the algorithm iterates.

Decryption:

Decryption is a straightforward reversal of the encryption process

Example:

```
Key : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Plain Text : 00000000 00000000
Cipher Text : EEDBA521 6D8F4B15
```

RC5 is a block cipher and addresses two word blocks at a time. Depending on input plain text block size, number of rounds and key size, various instances of RC5 can be defined and each instance is denoted as RC5-w/r/b where w=word size in bits, r=number of rounds and b=key size in bytes.
Allowed values are:

Parameter	Possible Value
block/word size (bits)	16, 32, 64
Number of Rounds	0 – 255
Key Size (bytes)	0 – 255

Note – Since at a time, RC5 uses 2 word blocks, the plain text block size can be 32, 64 or 128 bits.
Notation used in the algorithm:

Symbol	Operation
x <<< y	Cyclic left shift of x by y bits
+	Two's complement addition of words where addition is modulo 2^w
^	Bit wise Exclusive-OR

Step-1: Initialization of constants P and Q
RC5 makes use of 2 magic constants P and Q whose value is defined by the word size w.

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

For any other word size, P and Q can be determined as:

```
P = Odd((e-2)2^w)
Q = Odd((ϕ-2)2^w)
```

Here, Odd(x) is the odd integer nearest to x, e is the base of natural logarithms and ϕ 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.

```
for i=b-1 to 0
    L[i/u] = (L[u/i] <<< 8) + K[i]
```

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.

```
i = j = 0
A = B = 0
do 3 * max(t, c) times:
    A = S[i] = (S[i] + A + B) <<< 3
    B = L[j] = (L[j] + A + B) <<< (A + B)
    i = (i + 1) % t
    j = (j + 1) % c
```

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:

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^w.
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.

```
A = A + S[0]
B = B + S[1]
for i = 1 to r do:
   A = ((A ^ B) <<< B) + S[2 * i]
   B = ((B ^ A) <<< A) + S[2 * i + 1]
return A, B
```

Alternatively, RC5 Decryption can be defined as:

```
for i = r down to 1 do:
   B = ((B - S[2 * i + 1]) >>> A) ^ A
   A = ((A - S[2 * i]) >>> B) ^ B
B = B - S[1]
A = A - S[0]
return A, B .
```

➢ Characteristics of advanced symmetric Block Ciphers:

Symmetric Block Cipher FactsA block cipher takes a fixed-length number of bits, referred to as a block, and encrypts them all at once. Be aware of the following facts regarding symmetric key block ciphers:

- Block ciphers are fast.
- Block ciphers can process large amounts of data. They do not process small amounts of data well.
- Block ciphers use a substitution and transposition function.
- A *round* refers to data going through one substitution and transposition process.
- Large amounts of data processed through a block cipher may begin to show patterns in the cipher text.

Advanced Encryption Standard (AES) is an iterative symmetric key block cipher that was developed as a replacement to DES in 2001. AES:

- Uses the Rijndael Block Cipher which is resistant to all known attacks.
- Uses a variable-length block and key length (128-, 192-, or 256-bit keys).

Block cipher is an algorithm that operates on bits called block.

The characteristics of Advanced symmetric block ciphers are -

1) **Key dependent S-boxes** - S-box performs substitution and depicts the relationship between key and cipher text.
2) **Data dependent rotation** - Data dependent rotation results in differences in the amount of rotation.
3) **Variable plain text** or cipher text block length
4) **Operations on both plain and ciphered data**

➢ Confidentiality Using Conventional Encryption

➢ Confidentiality

- Providing confidentiality through the use of secret-key encryption has historically been the focus of cryptology.
- This topic remains important in itself, though other considerations have emerged in the last few decades.
- An understanding of the issues involved here clarifies those in other applications of encryption andhelps to motivate the development of public-key encryption.

Placement of Encryption Function

Issues involved:

What should be encrypted?

Where should encryption be done?

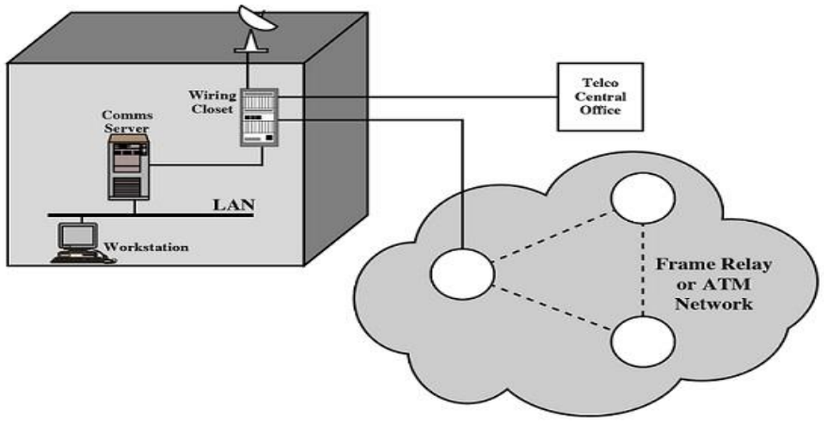
Two approaches :

Encryption

End-to-end encryption

To make the decisions, one should first examine thepotential locations of security attacks.

Points of Vulnerability

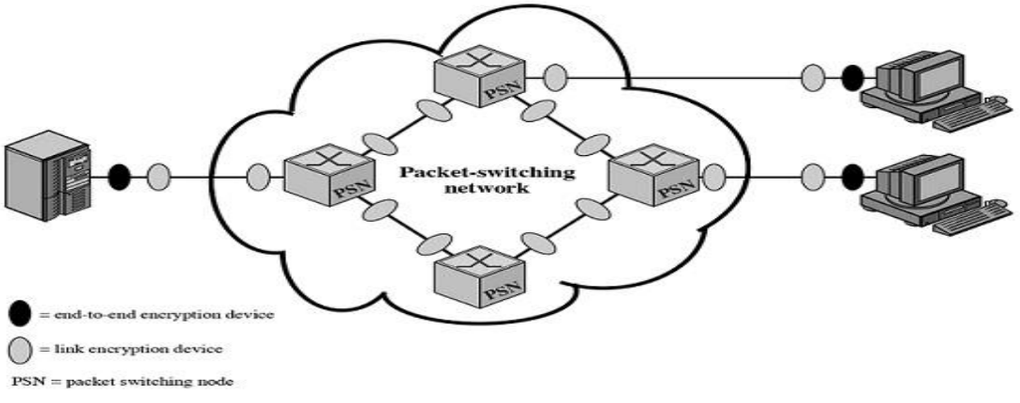


Locations for Confidentiality Attacks

Consider a user workstation in a typical business organization. The points of vulnerability include:

- The LAN that the workstation is attached to: *eavesdropping* on the LAN, which is typically a *broadcast* network.
- The Wiring closet: tapping the wires.
- Communications links out of the Wiring closet: invasive or inductive tapping.
- Processors along the path to the outside: modifying the hardware or software, etc.

Encryption in Packet-Switching Networks



Link Encryption

- Each vulnerable communications link is equipped onboth ends with an encryption device. Thus, all traffic over all communications links is secured.
- The message must be decrypted each time it enters a packet switch. Thus, the message is vulnerable at each switch.

- Many keys must be provided. However, each key needsbe distributed to only two nodes.

End-to-End Encryption

- The encryption process is carried out at the two end systems. The source and the destination share a key.

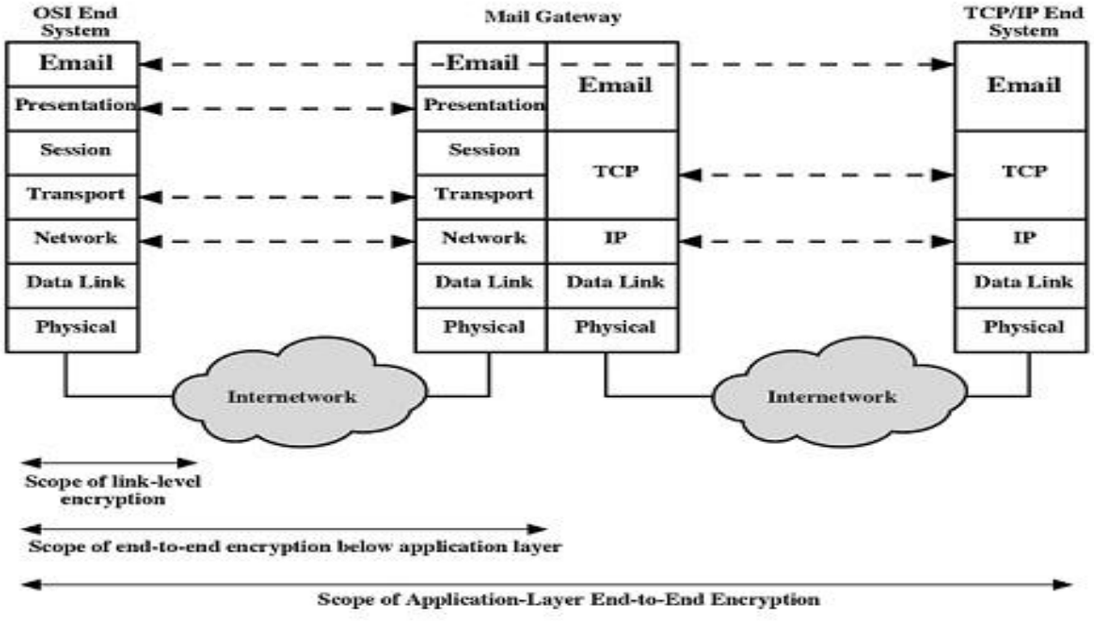
- This plan seems to secure the transmission againstattacks on the network links or switches. There is, however, still a weak spot.
- The source may encrypt only the user data portion, butmust leave the header in the clear.
- With end-to-end encryption, the user data are secure,but the traffic pattern is not. A certain degree of authentication is also provided.

Link vs. End-to-End Encryptions

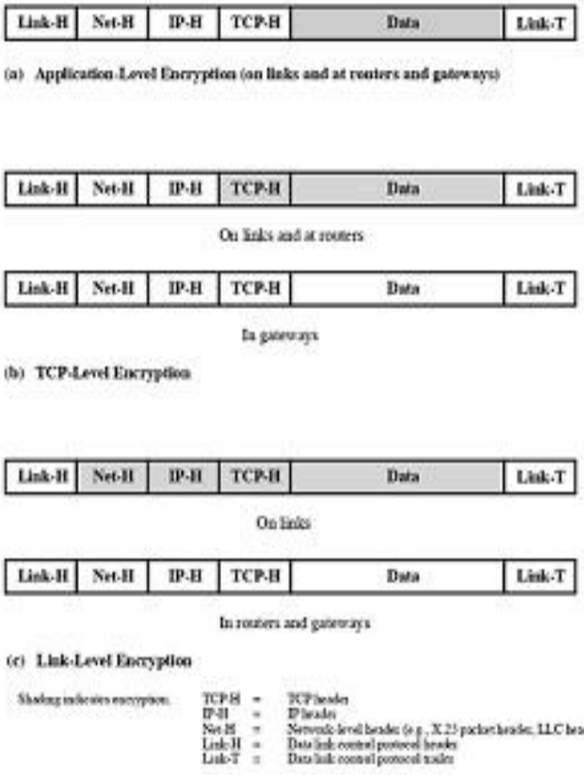
Link Encryption	End-to-End Encryption
Security within End Systems and Intermediate Systems	
Message exposed in sending host Message exposed in intermediate nodes	Message encrypted in sending host Message encrypted in intermediate nodes
Role of User	
Applied by sending host Transparent to user Host maintains encryption facility One facility for all users Can be done in hardware All or no messages encrypted	Applied by sending process User applies encryption User must determine algorithm Users selects encryption scheme Software implementation User chooses to encrypt, or not, for each message
Implementation Concerns	
Requires one key per (host-intermediate node) pair and (intermediate node-intermediate node) pair Provides host authentication	Requires one key per user pair Provides user authentication

Deploying End-to-End Encryption

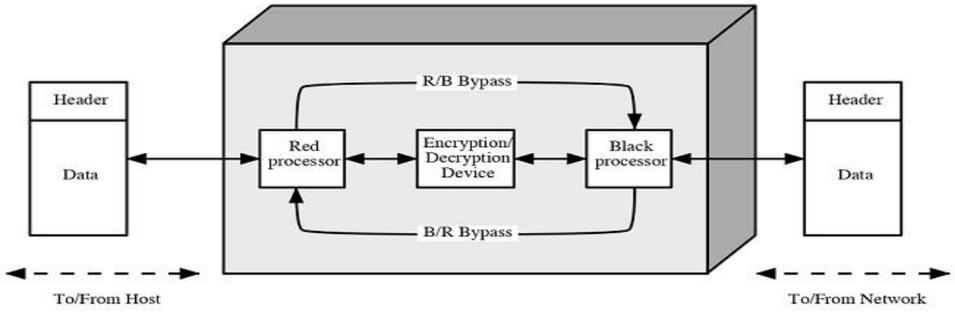
Store-and-Forward Communications



Encryption and Protocol Layers



Front-End Processor Function

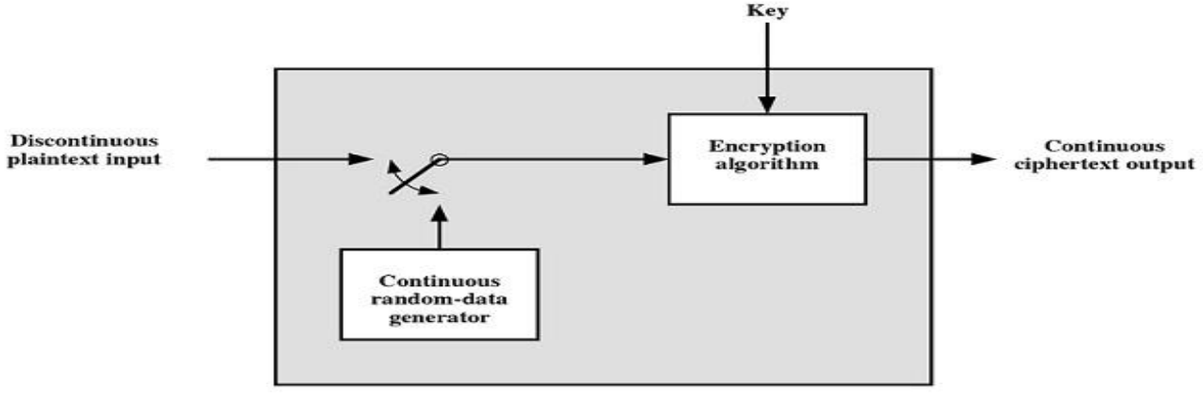


Traffic Confidentiality

Types of information that can be derived from a traffic analysis attack:

- Identities of partners
- How frequently the partners are communicating
- Message pattern, message length, or quantity of messages
- Events correlated with conversations between particular partners
- Messages of a *covert channel*

Traffic Padding



Countering Traffic Analysis

- Link encryption approach
  - packet headers already encrypted
  - further strength via traffic padding
- End-to-end encryption approach: available measures more limited
  - padding out data units to a uniform length
  - inserting null messages randomly

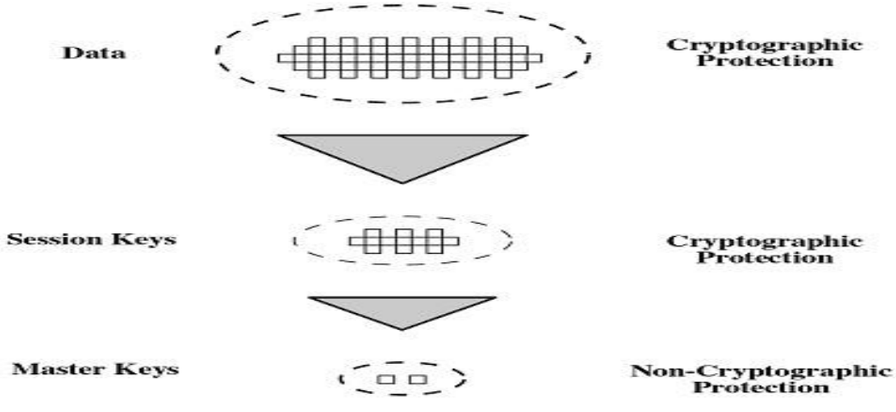
The Key Distribution Problem

- For symmetric encryption to work, the two parties of an exchange must share the same key and that key must be protected.
- Frequent key changes may be desirable to limit the amount of data compromised.
- The strength of a cryptographic system rests with the technique for solving the key distribution problem—delivering a key to the two parties of an exchange.
- The scale of the problem depends on the number of communication pairs.

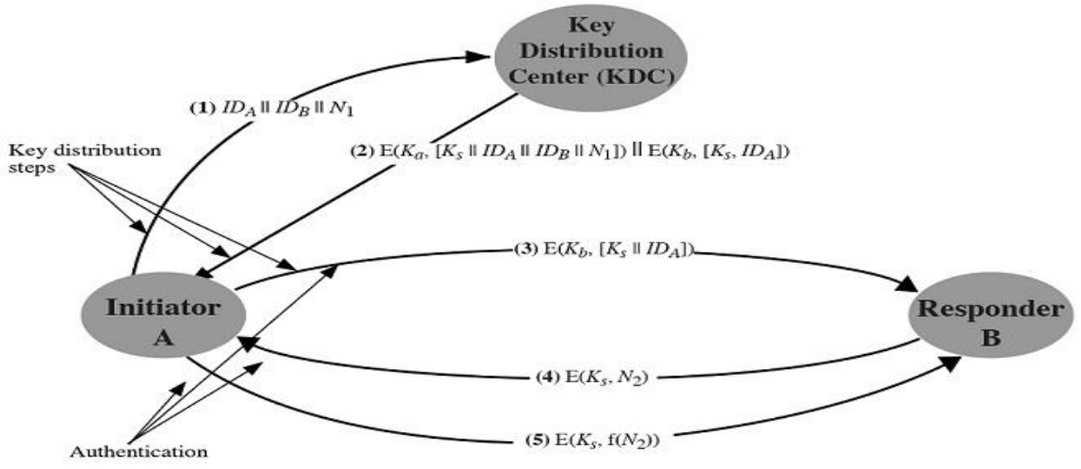
Using a Key Distribution Center

- A key distribution center is responsible for distributing keys to pairs of users as needed.
- Each user must share a unique key with the key distribution center for purposes of key distribution.
- At least two levels of keys must be used: session keys and master keys.
- If there are N end users, $N(N-1)/2$ session keys are needed at any one time, but only N master keys are required.

Key Hierarchy



Key Distribution Scenario



Hierarchical Key Control

- For large networks, a single KDC is inadequate.
- In a hierarchy of KDCs, each local KDC is responsible for a small domain.
- If the two parties are within the same local domain, their KDC is responsible for key distribution.
- Otherwise, the two corresponding local KDCs can communicate through a global KDC. Any of the three KDCs involved can select the key.
- Advantages: distributing the effort of master key distribution and isolating the damage of a fault.

Session Key Lifetime

- Two competing considerations in determining the lifetime of a session key:
  - The more frequently session keys are changed, the more secure they are.
  - The distribution of session keys delays the start of an exchange and places a burden on network capacity.
- The decision can be based on whether the communication protocol is connection-oriented or connectionless.

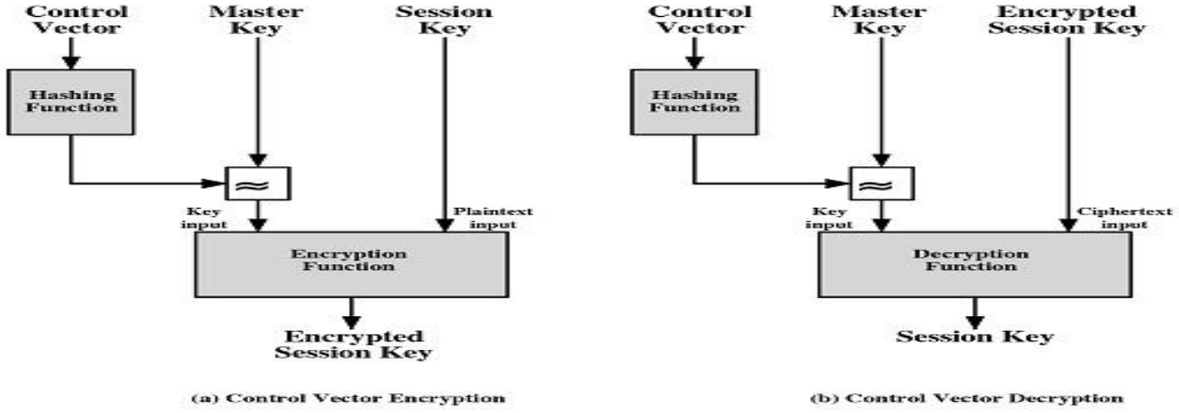
Decentralized Key Control

- The KDC must be trusted and be protected from subversion.
- This requirement can be avoided if the key distribution is fully decentralized.
- A fully decentralized key control, though not feasible for large networks, may be useful within a local context.
- A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution.

Controlling Key Usage

- It may be desirable to impose some control on the way in which automatically distributed keys are used.
- Possible types of session keys include: data-encrypting key, PIN-encrypting key, file-encrypting key, etc.
- Key use controlling schemes:
  - Tags
  - Control vectors

Control Vector



Pseudorandom Numbers

- True random numbers are hard to come by.
- Cryptographic applications typically use algorithmic techniques for random number generation.
- These algorithms are deterministic and therefore produce sequence of numbers that are not statistically random.
- If the algorithm is good, the resulting sequences will pass reasonable tests for randomness.
- Such numbers are often referred to as pseudorandom numbers.

The Use of Random Numbers

- Random numbers are used by a number of security algorithms for:
  - Nonces (used in authentication protocols)
  - Session key generation (by the KDC or an end system)
  - Key generation for the RSA algorithm
- Two requirements: randomness and unpredictability.

Cryptographical Generation

- **Cyclic encryption**: use an arbitrary block cipher. Full-period generating functions are easily obtained.
- **DES Output Feedback Mode**: the successive 64-bit outputs constitute a sequence of pseudorandom numbers.
- **ANSI X9.17 Pseudorandom number generator (PRNG)**: make use of triple DES. Employed in financial security applications and PGP.

The Linear Congruential Method

m	the modulus	$m > 0$
a	the multiplier	$0 \leq a < m$
c	the increment	$0 \leq c < m$
X_0	the starting value (seed)	$0 \leq X_0 < m$

- Iterative equation: $X_{n+1} = (aX_n + c) \bmod m$
- Larger values of m imply higher potential for a long period.
- For example, $X_{n+1} = (7^5 X_n) \bmod (2^{31} - 1)$ has a period of $2^{31} - 2$.
- What are the weakness and the remedy?

The Blum Blum Shub (BBS) Generator

- Choose two large prime numbers p and q such that $p \equiv q \equiv 3 \pmod 4$. Let $n = p \times q$.
- Choose a random number s relatively prime to n.
- Bit sequence generating algorithm:

$$X_0 = s^2 \bmod n$$
for $i = 1$ **to** ∞
$$X_i = (X_{i-1})^2 \bmod n$$
$$B_i = X_i \bmod 2$$

- The BBS generator passes the next-bit test.

Example Operation of BBS Generator

i	X_i	B_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0

i	X_i	B_i
11	137922	0
12	123175	1
13	8630	0
14	114386	0
15	14863	1
16	133015	1
17	106065	1
18	45870	0
19	137171	1
20	48060	0