G.S. Mandal's
Maharashtra Institute of Technology, Aurangabad
Department of Computer Science and Engineering

LAB MANUAL

CSE402: Cryptography and Network Security

(2019-20 Part-I)

Maharashtra Institute of Technology, Aurangabad
NH-211, MIT Campus, Satara Village Road, Aurangabad- 431 010 (M.S.); India.
Phone: (0240) 2375222; Fax: (0240) 2376618, E-mail: principalmitt@mit.asia
Website: www.btech.mit.asia

Department of Computer Science and Engineering

Vision

To develop the department as a center of excellence in the field of computer science and engineering by imparting knowledge & training to the students for meeting growing needs of the industry & society.

Mission

Providing quality education through a well-designed curriculum in tune with the challenging needs of software industry by providing state of the art facilities and to impart knowledge in the thrust areas of computer science and engineering.

Department of Computer Science and Engineering

Program Educational Objectives

PEO1: To prepare the students to achieve success in Computing Domain to create individual careers, innovations or to work as a key contributor to the private or Government sector and society.

PEO2: To develop the ability among the students to understand Computing and mathematical fundamentals and apply the principles of Computer Science for analyzing, designing and testing software for solving problems.

PEO3: To empower the students with ability to quickly reflect the changes in the new technologies in the area of computer software, hardware, networking and database management.

PEO4: To promote the students with awareness for lifelong learning, introduce them to professional practice, ethics and code of professionalism to remain continuous in their profession and leaders in technological society.

Program Specific Objectives

PSO1: Identify appropriate data structures and algorithms for a given contextual problem and develop programs to design and implement web applications.

PSO3: Design and manage the large databases and develop their own databases to solve real world problems and to design, build, manage networks and apply wireless techniques in mobile based applications.

PSO3: Design a variety of computer-based components and systems using computer hardware, system software, systems integration process and use standard testing tools for assuring the software quality.

Department of Computer Science and Engineering

Program Outcomes

PO1: Apply knowledge of mathematics, science, and engineering fundamentals to solve problems in Computer science and Engineering.

PO2: Identify, formulate and analyze complex problems.

PO3: Design system components or processes to meet the desired needs within realistic constraints for the public health and safety, cultural, societal and environmental considerations.

PO4: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data for valid conclusions.

PO5: Select and apply modern engineering tools to solve the complex engineering problem.

PO6: Apply knowledge to assess contemporary issues.

PO7: Understand the impact of engineering solutions in a global, economic, environmental, and societal context.

PO8: Apply ethical principles and commit to professional ethics and responsibilities.

PO9: Work effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10: Communicate effectively in both verbal and written form.

PO11: Demonstrate knowledge and apply engineering and management principles to manage projects and in multi-disciplinary environment.

PO12: To engage in life-long learning to adopt to the technological changes.

Department of Computer Science and Engineering

Course: CSE402 Cryptography and Network Security

Course Outcomes:

After Completing the course students will be able to

CO1 Understand security concepts and type of attacks and network security algorithms.

CO2 Apply symmetric and asymmetric key cryptography technique to encrypt and decrypt text.

CO3 Apply the knowledge of symmetric key algorithm

CO4 Apply the knowledge of public key algorithm

CO5 Apply Cryptography Hash Function for message authentication and to solve other applications.

CO6 Understand the concept of security with different key management things.

Mapping

Experiment No.	Blooms Level	Mapping To CO	Mapping To PO
1	3	CO2	2,5
2	3	CO2	2,5
3	3	CO2	2,5
4	3	CO2	2,5
5	3	CO3	5
6	3	CO3	5
7	3	CO3	5
8	3	CO4	5
9	3	CO5	6
10	2	CO6	5

Index

Contents	**Page No.**

Exp No.	**Title of Experiment**	**Page No**

Experiment#1

Pre-Assessment Questions: (optional)

Aim: Write a program to implement monoalphabatic cipher

Objective: Students should able to do program using monoalphabetic cipher method.

Outcomes: Students are able to apply there knowledge during the programming.

Theory:

A monoalphabetic substitution cipher, also known as a simple substitution cipher, relies on a fixed replacement structure. That is, the substitution is fixed for each letter of the alphabet. Thus, if "a" is encrypted to "R", then every time we see the letter "a" in the plaintext, we replace it with the letter "R" in the cipher text.

A simple example is where each letter is encrypted as the next letter in the alphabet: "a simple message" becomes "B TJNQMF NFTTBHF". In general, when performing a simple substitution manually, it is easiest to generate the cipher text alphabet first, and encrypt by comparing this to the plaintext alphabet.

The cipher text alphabet for the cipher where you replace each letter by the next letter in the alphabet

There are many different monoalphabetic substitution ciphers.

Assessment Questions:

1.what is the concept of monoalphabetic cipher?

2.is this method is easy to implement?

3.are you able to apply this method to encrypt your message?

Experiment#2

Pre-Assessment Questions: (optional)

Aim: Write a program to implement Ceaser cipher

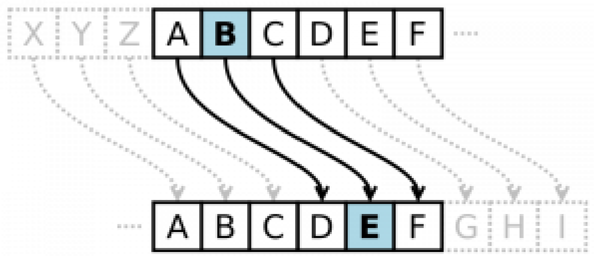
Objective: Student will able to use ceaser cipher technique.

Outcomes: Students are able to apply this technique to encrypt text

Theory: It is one of the simplest encryption technique in which each character in plain text is replaced by a character some fixed number of positions down to it.

For example, if key is 3 then we have to replace character by another character that is 3 position down to it. Like A will be replaced by D, C will be replaced by F and so on.

For decryption just follow the reverse of encryption process.



Assessment Questions:

1.what is the concept of ceaser cipher?

2.are you able to implement it to encrypt your text.?

3.is it easy to use ?

Experiment#3

Pre-Assessment Questions: (optional)

Aim: Write a program to implement Affine cipher

Objective: student will able to do program using this technique

Outcomes: Student will able to apply this method for encryption of text

Theory:

The Affine cipher is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter. The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing which letter goes to which.

The whole process relies on working modulo m (the length of the alphabet used). In the affine cipher, the letters of an alphabet of size m are first mapped to the integers in the range 0 ⋯ m-1. The 'key' for the Affine cipher consists of 2 numbers, we'll call them a and b. The following discussion assumes the use of a 26 character alphabet (m = 26). a should be chosen to be relatively prime to m (i.e. a should have no factors in common with m).

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Assessment Questions:

1.what is the concept of Affine cipher

2.how to implement it.

Experiment#4

Pre-Assessment Questions: (optional)

Aim: Write a program to implement Rail fence Cipher technique

Objective: Student will able to a program to implement Rail fence Cipher encryption

Outcomes: Student will able to apply this technique for encryption.

Theory:

The railfence cipher is a very simple, easy to crack cipher. It is a transposition cipher that follows a simple rule for mixing up the characters in the plaintext to form the ciphertext. The railfence cipher offers essentially no communication security, and it will be shown that it can be easily broken even by hand.

Although weak on its own, it can be combined with other ciphers, such as a substitution cipher, the combination of which is more difficult to break than either cipher on it's own.

Many websites claim that the rail-fence cipher is a simpler "write down the columns, read along the rows" cipher. This is equivalent to using an un-keyed columnar transposition cipher.

Example

The key for the railfence cipher is just the number of rails. To encrypt a piece of text, e.g.

defend the east wall of the castle

We write it out in a special way on a number of rails (the key here is 3)

```
d . . . n . . . e . . . t . . . l . . . h . . . s . . .
. e . e . d . h . e . s . w . l . o . t . e . a . t . e
. . f . . . t . . . a . . . a . . . f . . . c . . . l .
```

The ciphertext is read off along the rows:

dnetlhseedheswloteateftaafcl

With a key of 4:

```
d . . . . . t . . . . . t . . . . . f . . . . . s . . .
. e . . . d . h . . . s . w . . . o . t . . . a . t . .
. . f . n . . . e . a . . . a . l . . . h . c . . . l .
. . . e . . . . . e . . . . . l . . . . . e . . . . . e
```

The ciphertext is again read off along the rows:

dttfsedhswotatfneaalhcleelee

Assessment Questions:

1.what is the concept of railfence technique

2.how to implement it.

Experiment#5

Pre-Assessment Questions: (optional)

Aim:User A want to send the message "Meet me very urgently" to user B by using DES algorithms encrypt it at sender end and decrypt it at receiver end.

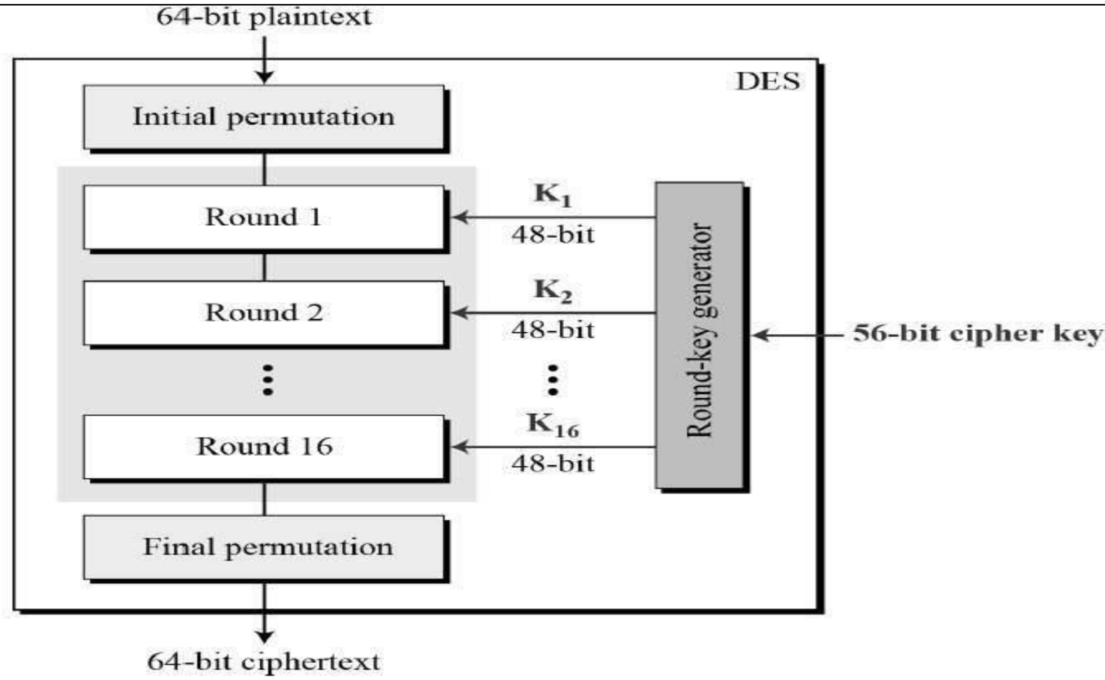
Objective: Student will able to implement DES algorithm

Outcomes: Student will able to apply this algo to encrypt the text

Theory:

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). General Structure of DES is depicted in the following illustration −

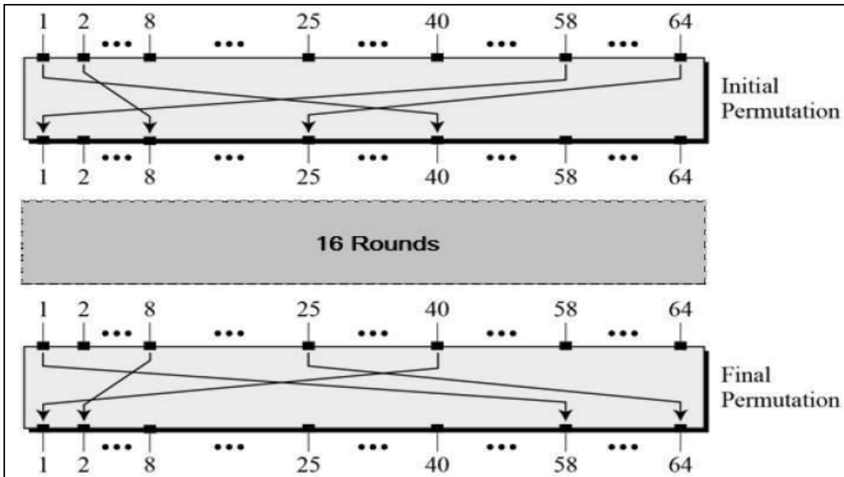


Since DES is based on the Feistel Cipher, all that is required to specify DES is −

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

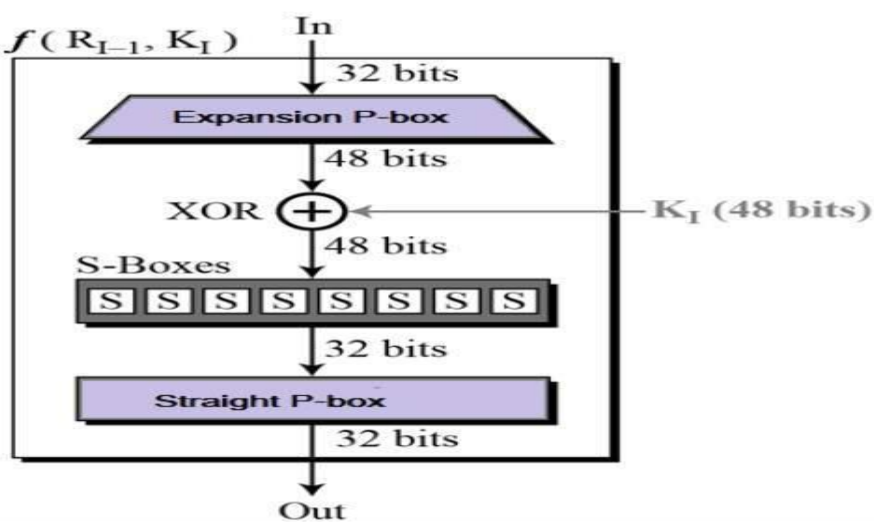
Initial and Final Permutation

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 −

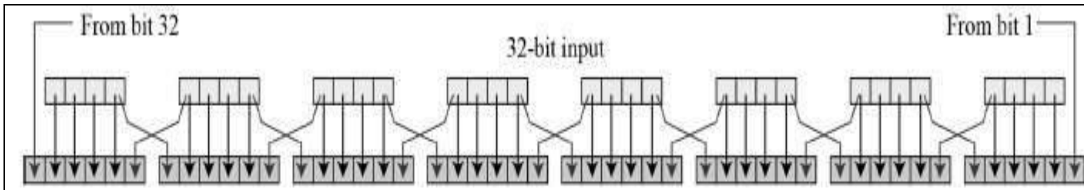


Round Function

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.



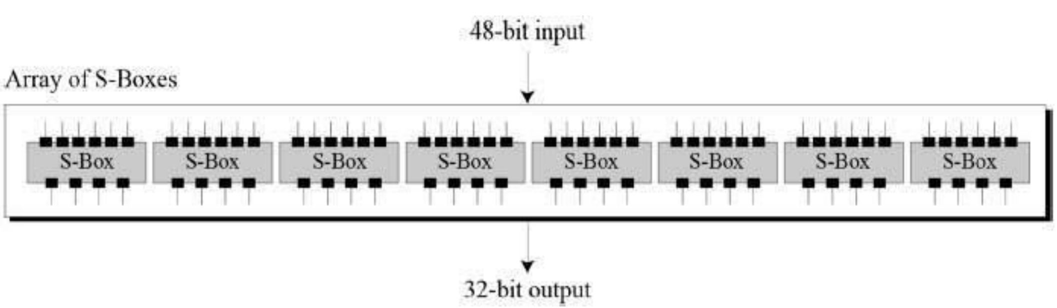
- Expansion Permutation Box – Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration –



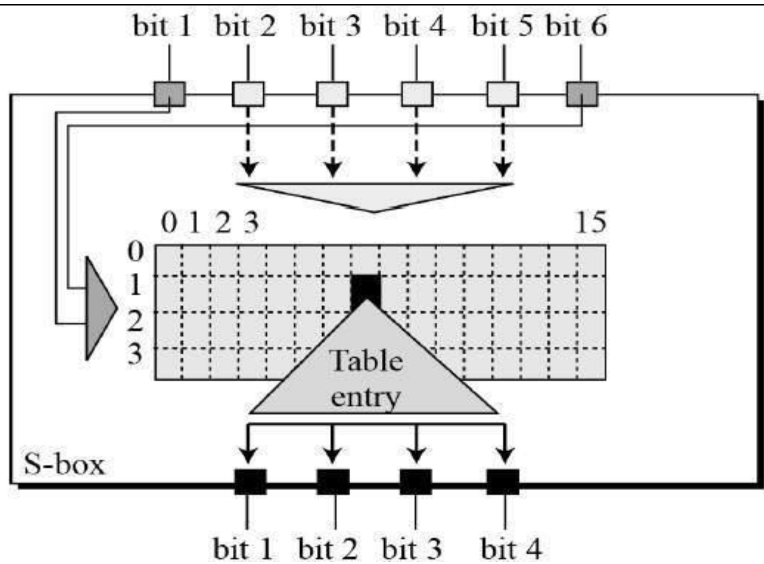
- The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown –

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

- 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. Refer the following illustration –



- The S-box rule is illustrated below –

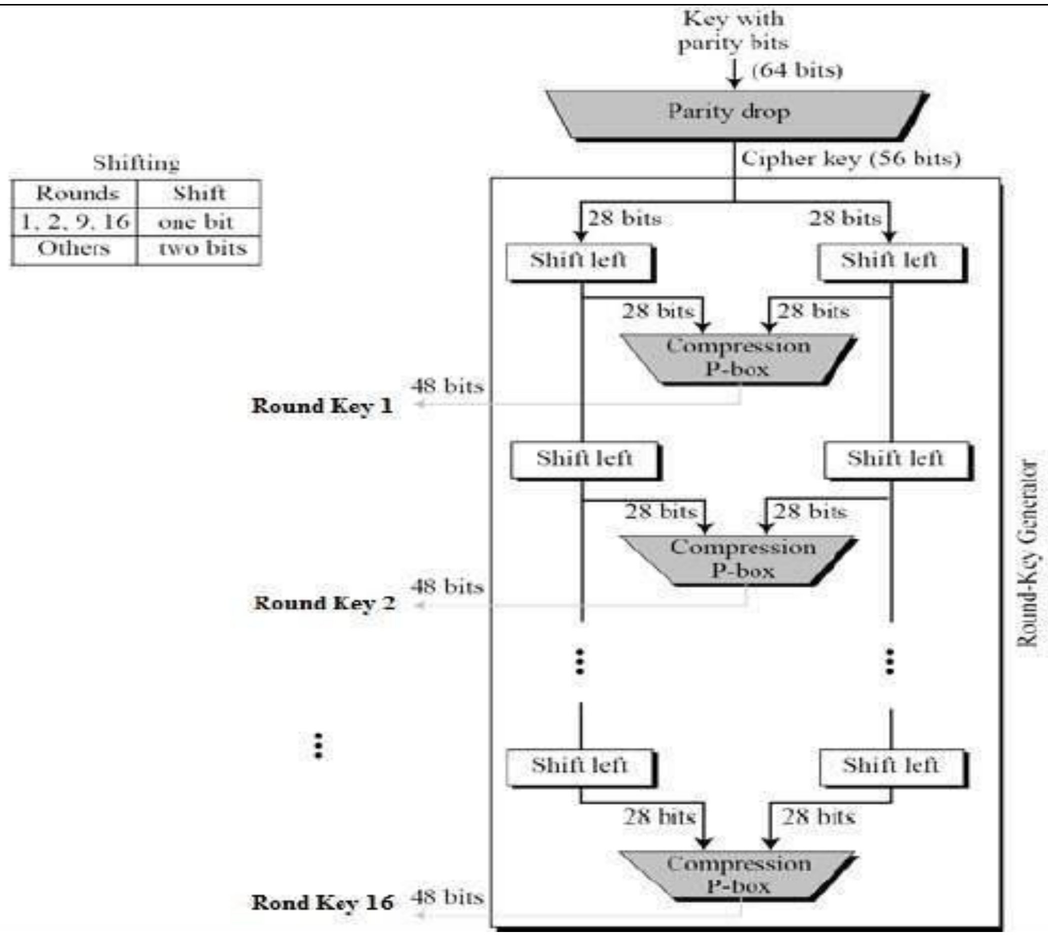


- There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.
- Straight Permutation – The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Key Generation

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 grate 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.

Assessment Questions:

1.what is the concept of DES Algorithm

2.how to implement it.

Experiment#6

Pre-Assessment Questions: (optional)

Aim: User C want to send message "welcome to cse" to user D by using AES algorithms encrypt it and decrypt it at receiver end.

Objective: Students will able to implement AES Algo

Outcomes: Student Will able to apply AES algo

Theory:

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). 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

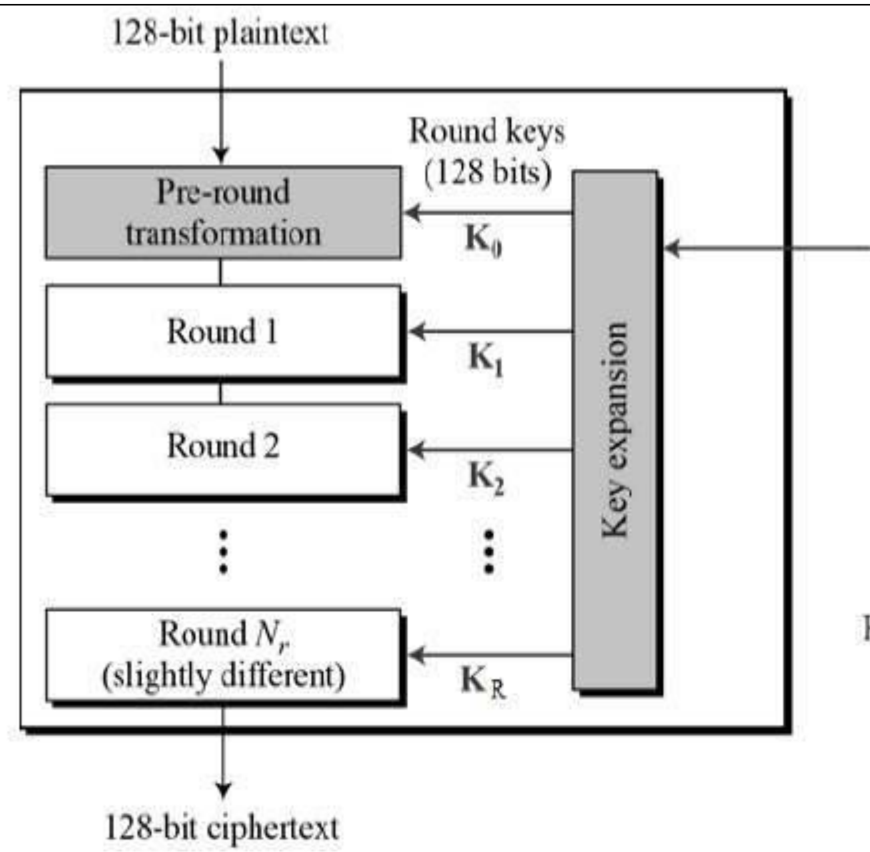
Operation of AES

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 −



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Assessment Questions:

1.what is the concept of AES Algorithm

2.how to implement it.

Experiment#7

Pre-Assessment Questions: (optional)

Aim: user A want to communicate with user B but it should be confidential by using Blowfish Algorithms send encrypt message

Objective: Student will able to implement blowfish algo

Outcomes: Student will able to understand the concept of blowfish

Theory:

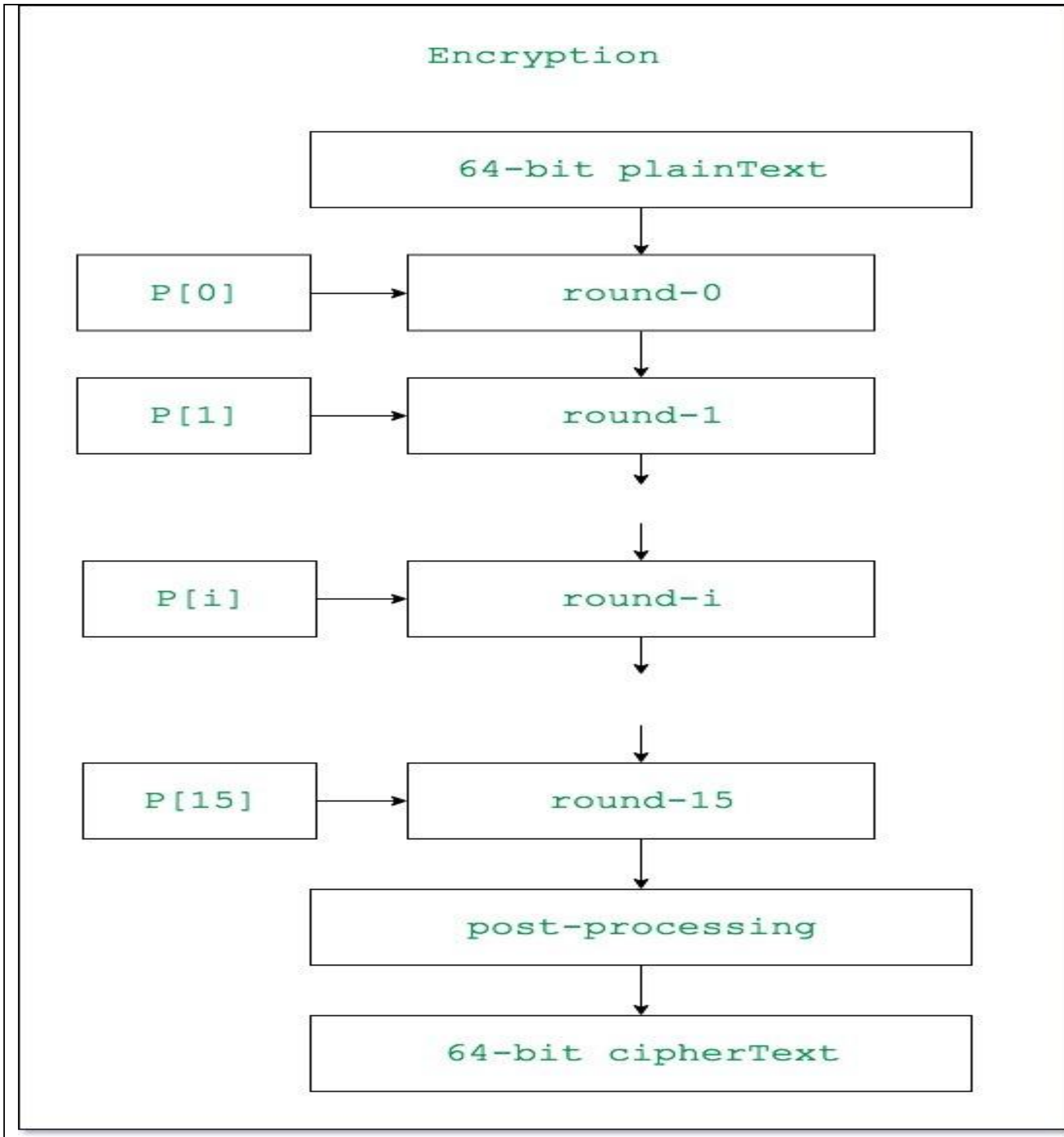
Blowfish is an encryption technique designed by **Bruce Schneier** in 1993 as an alternative to DES Encryption Technique. It is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date. It is one of the first, secure block cyphers not subject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

1. **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 subsitution boxes**: 4 [each having 512 entries of 32-bits each]

Blowfish Encryption Algorithm

The entire encryption process can be elaborated as:



Blowfish symmetric block cipher algorithm encrypts block data of 64-bits at a time.it will follows the feistel network and this algorithm is divided into two parts.

1. Key-expansion
2. Data Encryption

Key-expansion:

It will converts a key of at most 448 bits into several subkey arrays totaling 4168 bytes. Blowfish uses large number of subkeys.

These keys are generate earlier to any data encryption or decryption.

The p-array consists of 18, 32-bit subkeys:

P1,P2,…………,P18

Four 32-bit S-Boxes consists of 256 entries each

S1,0, S1,1,………. S1,255

S2,0, S2,1,……….. S2,255

S3,0, S3,1,……….. S3,255

S4,0, S4,1,..............S4,255

Generating the Subkeys: The subkeys are calculated using the Blowfish algorithm:

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of pi (less the initial 3): P1 = 0x243f6a88, P2 = 0x85a308d3, P3 = 0x13198a2e, P4 = 0x03707344, etc.
2. XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P14). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits. (For every short key, there is at least one

equivalent longer key; for example, if A is a 64-bit key, then AA, AAA, etc., are equivalent keys.)

3. Encrypt the all-zero string with the Blowfish algorithm, using the subkeys described in steps (1) and (2).
4. Replace P1 and P2 with the output of step (3).
5. Encrypt the output of step (3) using the Blowfish algorithm with the modified subkeys.
6. Replace P3 and P4 with the output of step (5).
7. Continue the process, replacing all entries of the P array, and then all four S-boxes in order, with the output of the continuously changing Blowfish algorithm.

In total, 521 iterations are required to generate all required subkeys. Applications can store the subkeys rather than execute this derivation process multiple times.

Data Encryption:

It is having a function to iterate 16 times of network. Each round consists of key-dependent permutation and a key and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookup tables for each round

Algorithm: Blowfish Encryption

Divide x into two 32-bit halves: xL, xR

For i = 1to 16:

xL = XL XOR Pi

xR = F(XL) XOR Xr

Swap XL and xR

Swap XL and xR (Undo the last swap.)

xR = xR XOR P17

xL = xL XOR P18

Recombine xL and xR

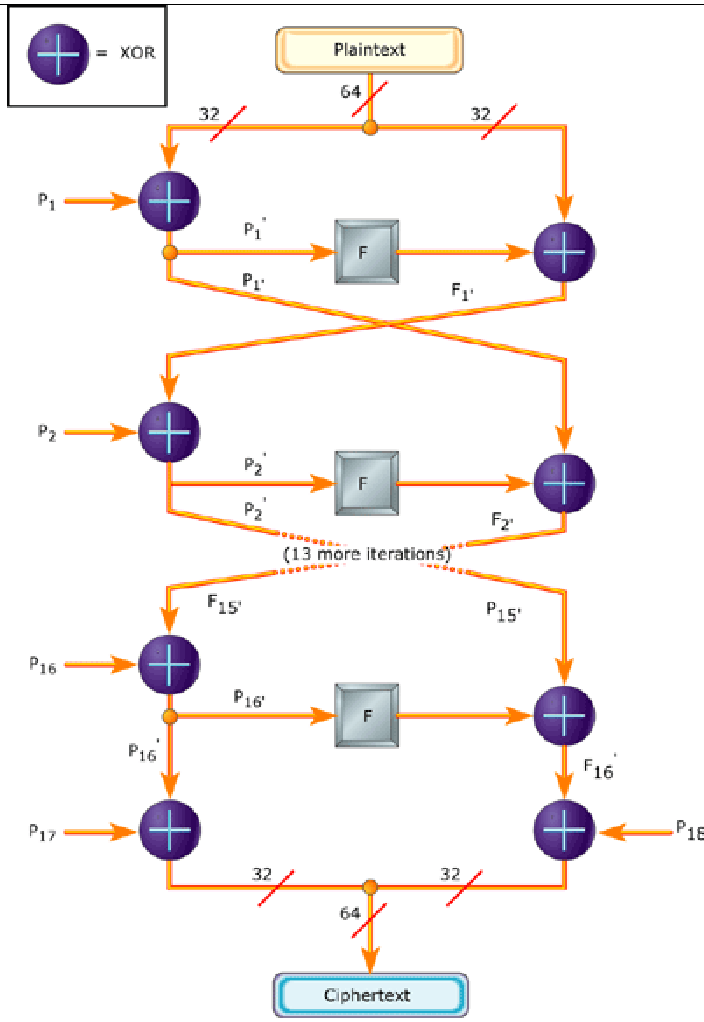


Fig 2: Blowfish Encryption

Assessment Questions:

1.what is the concept of blowfish Algorithm

2.how to implement it.

Experiment#8

Pre-Assessment Questions: (optional)

Aim: user A want to communicate to user B but they want to user Asymmetric Key Cryptography by using RSA algorithms send message to each other.encrypt message at sender side and decrypt it at receiver side.

Objective: Student will able to implement RSA algo

Outcomes: Student will able to apply RSA algo

Theory:

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

RSA is one of the first practical public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and different from the decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1978. Clifford Cocks, an English mathematician working for the UK intelligence agency GCHQ, had developed an equivalent system in 1973, but it was not declassified until 1997.[1] A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime numbers can feasibly decode the message.[2] Breaking RSA encryption is known as the RSA problem; whether it is as hard as the factoring problem remains an open question. RSA is a relatively slow algorithm, and because of this it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed.

Operation

The RSA algorithm involves four steps: key generation, key distribution, encryption and decryption. A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integer m: and that even knowing e and n or even m it can be extremely difficult to find d. Additionally, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies: RSA involves a *public key* and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The public key is represented by the integers n and e; and, the private key, by the integer d (although n is also used during the decryption process; so, it might be considered a part of the private key, too). m represents the message (previously prepared with a certain technique explained below).

Key generation

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random, and should be similar in magnitude but 'differ in length by a few digits'[2] to make factoring harder. Prime integers can be efficiently found using a primality test.
2. Compute $n = pq$.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute $\lambda(n) = \text{lcm}(\lambda(p), \lambda(q)) = \text{lcm}(p - 1, q - 1)$, where λ is Carmichael's totient function. This value is kept private.
4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; i.e., e and $\lambda(n)$ are coprime.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; i.e., d is the modular multiplicative inverse of e (modulo $\lambda(n)$).

   - This is more clearly stated as: solve for d given $d \cdot e \equiv 1 \pmod{\lambda(n)}$.
   - e having a short bit-length and small Hamming weight results in more efficient

encryption – most commonly $e = 2^{16} + 1 = 65{,}537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.[14]

- e is released as the public key exponent.
- d is kept as the private key exponent.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the modulus n and the private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d.

Alternatively, as in the original RSA paper,[2] the Euler totient function $\phi(n) = (p - 1)(q - 1)$ can be used instead of $\lambda(n)$ for calculating the private exponent d. This works because $\phi(n)$ is always divisible by $\lambda(n)$ (a consequence of applying Lagrange's theorem to the multiplicative group of integers modulo pq), and thus any d satisfying $d \cdot e \equiv 1 \pmod{\phi(n)}$ also satisfies $d \cdot e \equiv 1 \pmod{\lambda(n)}$. However, computing d modulo $\phi(n)$ will sometimes yield a result that is larger than necessary (i.e. $d > \lambda(n)$). Most RSA implementations will accept exponents generated using either method (if they use the private exponent d at all, rather than using the optimized decryption method based on the Chinese remainder theorem described below), but some standards like FIPS 186-4 may require that $d < \lambda(n)$. Any "oversized" private exponents not meeting that criterion may always be reduced modulo $\lambda(n)$ to obtain a smaller equivalent exponent.

Since any common factors of $(p - 1)$ and $(q - 1)$ are present in the factorisation of $n - 1 = pq -$

$1 = (p-1)(q-1) + (p-1) + (q-1)$,[15] it is recommended that $(p-1)$ and $(q-1)$ have only very small common factors, if any besides the necessary 2.[2][16][17]

Note: The authors of the original RSA paper carry out the key generation by choosing d and then computing e as the modular multiplicative inverse of d (modulo $\phi(n)$). Since it is beneficial to use a small value for e (i.e. 65,537) in order to speed up the encryption function, current implementations of RSA, such as PKCS#1 choose e and compute d instead.[2][18]

Key distribution

Suppose that Bob wants to send a secret message to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message and, Alice must use her private key to decrypt the message. To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret route. Alice's private key (d), is never distributed.

Key Genration :

- Select p,q…….. p and q both are the prime numbers, p ≠ q.
- Calculate n=p × q
- Calculate q(n) = (p-1) (q-1)
- Select integer….g(d ((n), e)) =1 & 1< e < (n)
- Calculate d; d= e-1 mod (n)

- Public Key, PU= {e, n}
- Private Key, PR ={d,n}

b) Encryption :

- Plaintext : m<n< p="">
- Ciphertext: C

c) Decryption:

- Ciphertext: C
- Plaintext : M= Cd mod n
- Note 1 : (n) -> Euler's totient function
- Note 2: Relationship between C and d is expressed as:

ed (mod (n))=1

ed = 1 mod (n)

d = $e-1$

- mod (n)

6.**Example:**

- **Key Generation :**
  1. Select 2 prime numbers -> p=17 and q=11
  2. Calculate n = p × q =17 × 11=187
  3. Calculate = 16 × 10= 160 Select 'e' such that e is relatively prime to (n)=160 and e <

4. Determine d such that :

   de =1 mod (n)

   d × 7 = 1 mod 160

   ↓

161

$d = e - 1 \ \ mod \ \ (n)$[161/7= $div.(d)$23 and remainder (mod) =1 d=23

1. Then the resulting keys are public key :

   PU = {7, 187 }

   PR = {23, 187 }

   Let M=88 for encryption

   $C=887mod(187)88mod187=88882mod187=7744mod187=77884mod187=59969536mod187=$ 132

$887mod187$

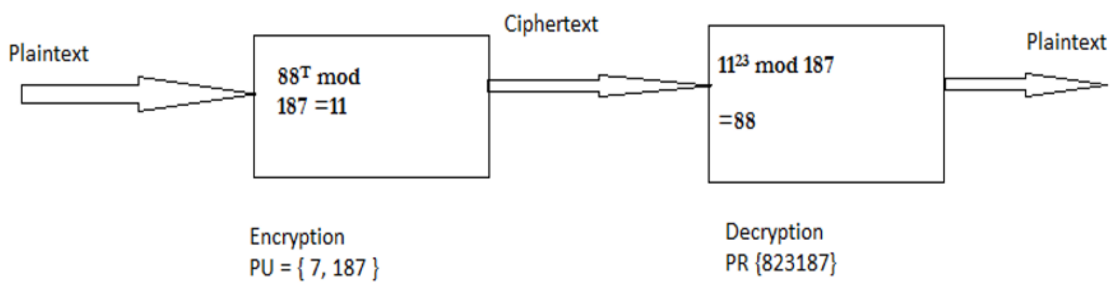
$=(884mod187)\times(882mod187)\times(88mod187)mod187=(132\times77\times88)mod187=894432mod187=11$

1.

- **For Decryption :**

  $M=Cdmod187=1123mod187111mod187=11112mod187=121114mod187=14641/187=55118$ $mod187=214358881mod187=331123mod187$

$=(118 \, mod 187 \times 118 \, mod 187 \times 114 \, mod 187 \times 112 \, mod 187 \times 111 \, mod 187) \, mod 187 = (33 \times 33 \times 55 \times 81 \times 11) \, mod 187 = 79720245 \, mod 187 = 88$



•

Figure Solution of Above example

Assessment Questions:

1.what is the concept of RSA Algorithm

2.how to implement it.

Experiment#9

Pre-Assessment Questions: (optional)

Aim: Write a program to implement Secure Hash Algorithm

Objective: student will able to implement Secure Hash Algorithm

Outcomes: student will able to apply Secure Hash Algorithm

Theory:

SHA-1 or Secure Hash Algorithm 1 is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value. This hash value is known as a message digest. This message digest is usually then rendered as a hexadecimal number which is 40 digits long. It is a U.S. Federal Information Processing Standard and was designed by the United States National Security Agency.

SHA-1 is now considered insecure since 2005. Major tech giants browsers like Microsoft, Google, Apple and Mozilla have stopped accepting SHA-1 SSL certificates by 2017.

Experiment#10

Pre-Assessment Questions: (optional)

Aim: Write a program to implement digital Signature

Objective: Student will able to implement digital signature

Outcomes: Student will able to apply digital signature with example

Theory:

Java – Digital Signatures example

1. Generate a Public-Private Key Pair

The code to generate Public-Private Key Pair is identical to the one used in Asymmetric Cryptography example, please refer to Step 1 or download the source code at the end of the article that includes all sources

2. Sign the message

Next we have to write our message and then sign it. The message and the signature can be separate files but in our example we add them to a List of byte[] and write them as Object to the file.

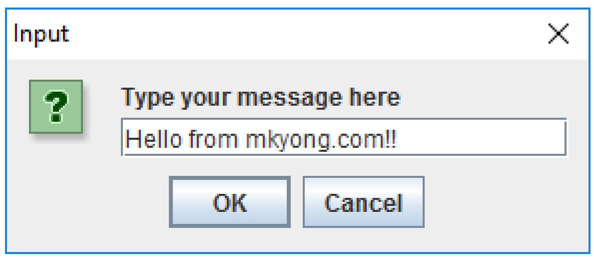
Message.java

```
package com.mkyong.sender;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.nio.file.Files;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.Signature;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JOptionPane;

public class Message {
	private List<byte[]> list;

	//The constructor of Message class builds the list that will be written to the file.
	//The list consists of the message and the signature.
	public Message(String data, String keyFile) throws InvalidKeyException, Exception {
		list = new ArrayList<byte[]>();
```

```
        list.add(data.getBytes());
        list.add(sign(data, keyFile));
    }

    //The method that signs the data using the private key that is stored in keyFile path
    public byte[] sign(String data, String keyFile) throws InvalidKeyException, Exception{
        Signature rsa = Signature.getInstance("SHA1withRSA");
        rsa.initSign(getPrivate(keyFile));
        rsa.update(data.getBytes());
        return rsa.sign();
    }

    //Method to retrieve the Private Key from a file
    public PrivateKey getPrivate(String filename) throws Exception {
        byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePrivate(spec);
    }

    //Method to write the List of byte[] to a file
    private void writeToFile(String filename) throws FileNotFoundException, IOException {
        File f = new File(filename);
        f.getParentFile().mkdirs();
        ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename));
      out.writeObject(list);
        out.close();
        System.out.println("Your file is ready.");
    }

    public static void main(String[] args) throws InvalidKeyException, IOException, Exception{
        String data = JOptionPane.showInputDialog("Type your message here");
```

```
        new Message(data, "MyKeys/privateKey").writeToFile("MyData/SignedData.txt");
    }
}
```

Output:



Your file is ready.

3. Verify the Signature

The receiver has the file *(he knows it is a List of 2 byte arrays; the message and the signature)* and wants to verify that the message comes from the expected source with a pre-shared Public Key. VerifyMessage.java

```
package com.mkyong.receiver;

import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
```

```
import java.nio.file.Files;
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.Signature;
import java.security.spec.X509EncodedKeySpec;
import java.util.List;

public class VerifyMessage {
	private List<byte[]> list;

	@SuppressWarnings("unchecked")
	//The constructor of VerifyMessage class retrieves the byte arrays from the File
	//and prints the message only if the signature is verified.
	public VerifyMessage(String filename, String keyFile) throws Exception {
		ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename));
	    this.list = (List<byte[]>) in.readObject();
	    in.close();

	    System.out.println(verifySignature(list.get(0), list.get(1), keyFile) ? "VERIFIED MESSAGE" +
	      "¥n----------------¥n" + new String(list.get(0)) : "Could not verify the signature.");
	}

	//Method for signature verification that initializes with the Public Key,
	//updates the data to be verified and then verifies them using the signature
	private boolean verifySignature(byte[] data, byte[] signature, String keyFile) throws Exception {
		Signature sig = Signature.getInstance("SHA1withRSA");
		sig.initVerify(getPublic(keyFile));
		sig.update(data);

		return sig.verify(signature);
	}
```

```
    //Method to retrieve the Public Key from a file
    public PublicKey getPublic(String filename) throws Exception {
        byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());
        X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePublic(spec);
    }

    public static void main(String[] args) throws Exception{
        new VerifyMessage("MyData/SignedData.txt", "MyKeys/publicKey");
    }
}
```

Output:

```
VERIFIED MESSAGE
----------------
Hello from mkyong.com!!
```

Assessment Questions:

1. what is the use of digital signature
2. how to implement it.