Hope Foundation's

International Institute of Information Technology

Department of Computer Engineering

Laboratory Manual

Class	:	T.E. Computer Engineering
Course	:	2019 Pattern
Subject Code	:	310258
Subject	:	Laboratory Practice – II (Information Security)
Subject In charge	:	Dr. Uma Godase

Teaching Scheme

Practical : 2 Hrs/Week

Theory : 4 Hrs/Week

Examination Scheme

Term Work : 50 Marks

Practical : 25 Marks

Vision and Mission of the Institute

Vision

To be a premier academic institution that fosters diversity, value added education and research, leading to sustainable innovations and transforming learners into leaders.

Mission

To strive for academic excellence, knowledge enhancement, and critical thinking capabilities by adopting innovative and dynamic teaching learning pedagogies To enrich and leverage interactions and associations through industry academia partnerships To groom students so as to make them lifelong learners by helping them imbibe professional, entrepreneurial and leadership qualities To embrace an environment that allows all stakeholders to benefit from the technology enabled processes and systems

Vision and Mission of the Department

Vision

To be an eminent Department that provides exemplary education that would nurture innovative thinking, quality research and there by grooming students to become skilled professionals and researchers in various field of Computer Engineering

Mission

♦ To provide a conducive environment that offers an enriching learning experience and motivate students towards innovative thinking and research.

♦ To equip students with technical skills that would help them develop efficient software solutions.

♦ To enhance industry academia dialog enabling students to inculcate professional skills within them.

♦ To incorporate social and ethical awareness among our graduates to make them conscientious professionals.

Program Educational Objectives

PEO1	To develop technically competent Computer Engineers with proficient domain knowledge, research attitude, and innovative thinking.
PEO2	To foster committed graduates with a life-long learning attitude and entrepreneurship spirit for a successful career.
PEO3	To inculcate professional ethics, managerial, and communication skills to accomplish a worthy career in all spheres of Computer Engineering.
PEO4	To nurture socially aware engineers who can develop effective solutions for the betterment of the community and the nation.

Program Outcomes

PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
PO2	Problem analysis: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
PO4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	Communication: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	PO 12: Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO1	Professional Skills -The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexities.
PSO2	Problem Solving Skills - The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.
PSO3	Successful Career and Entrepreneurship - The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.

INDEX

Assignment 1

Implementation of Transposition Technique

AIM **:** Write a Java/C/C++/Python program to perform encryption and decryption using the method of transposition technique.

OBJECTIVE **:** To implement and understand the working of transposition technique.

THEORY **:**

Transposition Cipher is a cryptographic algorithm where the order of alphabets in the plaintext is rearranged to form a cipher text. A simple example for a transposition cipher is columnar transposition cipher where each character in the plain text is written horizontally with specified alphabet width. The cipher is written vertically, which creates an entirely different cipher text.

Encryption

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.
4. Any spare spaces are filled with some character (Example: _).
5. Finally, the message is read off in columns, in the order specified by the keyword.

Decryption

1. To decipher it, the recipient must work out the column lengths by dividing the message length by the key length.
2. Then, write the message out in columns again, then re-order the columns by reforming the key word.

INPUT **:** Plaintext

OUTPUT **:** Ciphertext

Assignment 2

Implementation of DES Algorithm

AIM **:** Write a Java/C/C++/Python program to implement DES (Data Encryption Standard) algorithm.

OBJECTIVE : To implement and understand the working of DES algorithm.

THEORY :

Data Encryption Standard (DES) is a block cipher algorithm that takes plain text in blocks of 64 bits and converts them to ciphertext using keys of 48 bits. It is a symmetric key algorithm, which means that the same key is used for encrypting and decrypting data.

The key has a "useful" length of 56 bits, which means that only 56 bits are used in the algorithm. The algorithm involves carrying out combinations, substitutions, and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions.

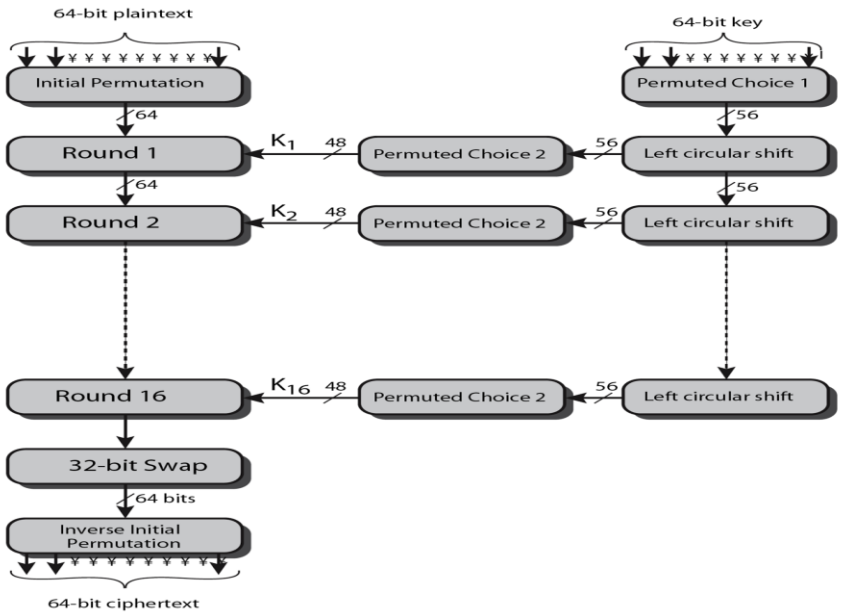

Figure 2.1 DES Algorithm

Generation of sub keys used in each round

- Permuted Choice 1(PC1): Convert 64 bit key to 56 bits by discarding every 8^{th} bit
- Divide 56-bits in two 28-bit halves.

- Apply left circular shift separately to each half either 1 or 2 places depending on the key rotation schedule

  round no. 1,2,9,16 – 1-bit shift

  remaining rounds – 2-bit shift
- These shifted 56 bits are input to the next round and Permuted Choice II is applied
- Permuted Choice 2 (PC2): Apply permutation to convert 56 bits into 48 bits which will be the sub key for that round.

DES Round Structure

- uses two 32-bit Left & Right halves
- as for any Feistel cipher we can describe as:

  Li = Ri–1

  Ri = Li–1 ⊕ F(Ri–1, Ki)
- F takes 32-bit R half and 48-bit sub key:
  1. Expansion: expands R to 48-bits using permutation E
  2. XOR: adds to sub key using XOR
  3. S- box: passes through 8 S-boxes to get 32-bit result
  4. P-box: finally permutes using 32-bit permutation P

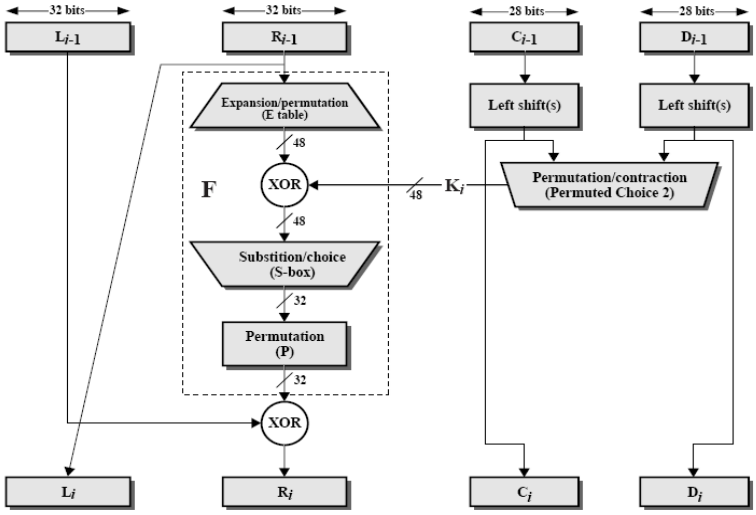

Figure 2.2 Details of Single round of DES

INPUT **:** Plain text

OUTPUT **:** Cipher text

Assignment 3

Implementation of AES Algorithm

AIM **:** Write a Java/C/C++/Python program to implement AES (Advanced Encryption Standard) algorithm.

OBJECTIVE : To implement and understand the working of AES.

THEORY :

The Advanced Encryption Standard (AES) was published by NIST (National Institute of Standards and Technology) in 2001.AES is symmetric block cipher. Its parameters are as below.

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Figure 3.1 shows the overall structure of AES. The input to the encryption and decryption algorithm is single 128-bit block. This block is depicted as a square matrix of bytes this block is copied into state array, which is modified at each stage of encryption or decryption. After the final stage, state is copied to an output matrix. These operations are given in figure.

Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words, each word is four bytes, and the total key schedule is 44 words for the 128-bit key.

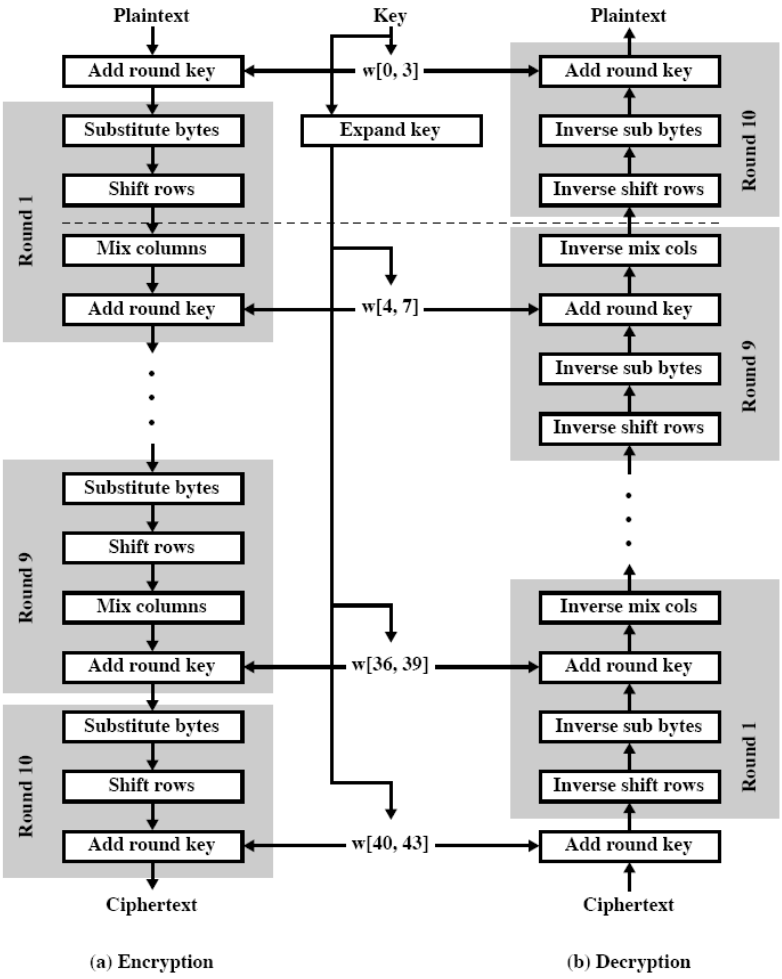

Fig 3.1 AES Encryption and Decryption

Four different stages are used, one permutation and three of substitution

- Substitute bytes: Uses S-box to perform a byte-by-byte substitution of the block.
- ShiftRows: A simple permutation
- MixColumns: A substitution that makes use of arithmetic.
- AddRoundKey: A simple bitwise XOR of the current block with the portion of expanded key.

1. **Substitute Bytes Transformation**

The forward substitute byte transformation, called SubBytes is a simple table lookup. AES defines 16*16 matrix of byte values, called S-box, that contains a permutation of all possible 256 8-bit values.

Each individual byte of State is mapped into a new byte in the following way: The leftmost 4 bits are used as arrow value and the rightmost 4 bits are used as column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value.

2. ShiftRows Transformation

Forward and Inverse Transformation

The forward shift row transformation called ShiftRows, is given in fig 6.5. The first row of state is not altered. For the second row, a 1-byte circular left shift is performed. For the third row a 2-byte circular left shift is performed. For the fourth row 3-byte circular left shift is performed.

Inverse shift row transformation, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a one-byte circular shift for the second row and so on.

3. MixColumns Transformation

Forward and Inverse Transformation

The forward mix column transformation called MixColumns, operates on each column individually. Each byte of column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by fig 6.6.

The Inverse mix column transformation, called InvMixColumn is defined by the multiplication.

4. AddRoundKey Transformation

Forward and Inverse transformation

In the forward add round key transformation called AddRoundKey, the 128 bits state are bitwise XORed with the 128 bits of round key.

The operation is viewed as a column wise operation between the 4 bytes of a State column and one word of the round key. The inverse AddRoundKey transformation is identical to the forward AddRoundKey transformation, because the XOR operation is its own inverse.

Required Classes

Class KeyGenerator

Constructor Summary	
protected	**KeyGenerator**(KeyGeneratorSpi keyGenSpi, Provider provider, String algorithm) Creates a KeyGenerator object.igest with the specified algorithm name.

Required Methods	
static KeyGenerator	**getInstance**(String algorithm) Generates a KeyGenerator object for the specified algorithm
static KeyGenerator	**getInstance**(String algorithm, Provider provider) Generates a KeyGenerator object for the specified key algorithm from the specified provider.
static KeyGenerator	**getInstance**(String algorithm,String provider) Generates a KeyGenerator object for the specified key algorithm from the specified provider

Class SecretKeySpec

Constructor Summary	
protected	**SecretKeySpec**(byte[] key, int offset, int len, String algorithm) Constructs a secret key from the given byte array, using the first len bytes of key, starting at offset inclusive
protected	**SecretKeySpec**(byte[] key, String algorithm) Constructs a secret key from the given byte array.

Class Cipher

Constructor Summary	
Protected	**Cipher**(CipherSpi cipherSpi, Provider provider, String transformation)

	Creates a Cipher object.
Required Methods	
static Cipher	**getInstance**(String transformation) Generates a Cipher object that implements the specified transformation
void	**init**(int opmode,Certificate certificate) Initializes this cipher with the public key from the given certificate
byte[]	**doFinal**() Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized.

INPUT **:** Plain text

OUTPUT **:** Cipher text

Assignment 4

Implementation of RSA

AIM **:** Write a Java/C/C++/Python program to implement RSA algorithm.

OBJECTIVE :

To study

- Concept of Public key and Private Key.
- Public key algorithm
- Working of RSA algorithm

THEORY :

Asymmetric/Public Key Algorithm:

Public key algorithms were evolved to solve the problem of key distribution in symmetric algorithms. This is achieved by using one key for encryption and a different but related key for decryption. These algorithms are designed such that it is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key. Also in some algorithms, such as RSA, either of the two related keys can be used for encryption, with the other used for decryption.

A public key encryption scheme has six ingredients:

- **Plaintext:** This is readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The algorithm performs various transformations on the plaintext.
- **Public and private key:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

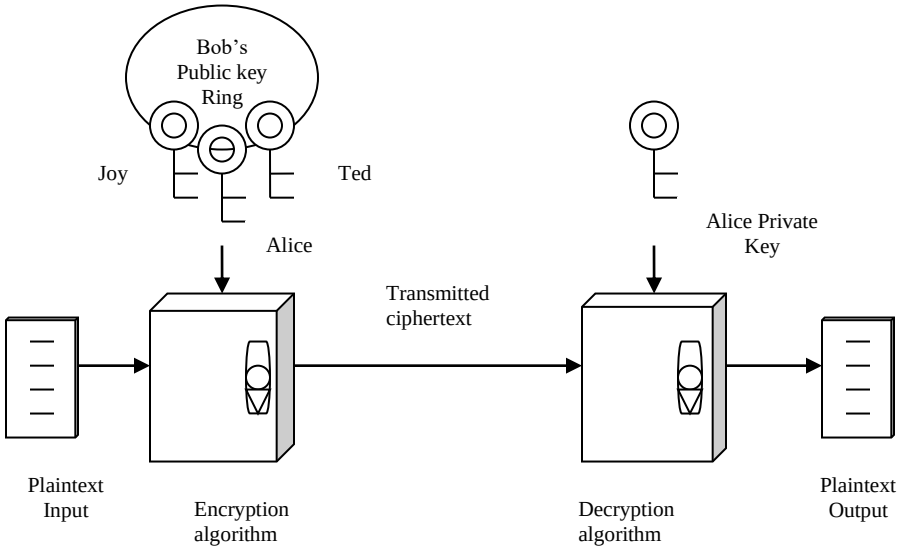

Figure 4.1: Public key cryptography

The essential steps in public key algorithm are as follows:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or the other accessible file. This is the public key. The companion key is kept private. Each user maintains a collection of public keys obtained from other parties participating in communication.
3. If A wishes to send a confidential message to B, A encrypts the message using B's public key.
4. When B receives the message, B decrypts it using B's private key. No other recipient can decrypt the message because only B knows B's private key.

RSA Algorithm

RSA (which stands for Rivest, Shamir and Adleman who first publicly described it), an algorithm for public-key cryptography involves three steps key generation, encryption and decryption.

RSA is a block cipher with each block having a binary value less than some number n. That is the block size must be less than or equal to log2 (n). Encryption & decryption are of the following form, for some plaintext block M and ciphertext block C:

$C = M^e \bmod n$

$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$

Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus, this is a public-key encryption algorithm with a public key of PU = {e, n} and a private key of PR = {d, n}. For this algorithm to be satisfactory for public key encryption, the following requirements must meet:

1. It is possible to find values of e, d, n such that $M^{ed} = M \bmod n$ for all $M<n$.
2. It is relatively easy to calculate M^e and C^d for all values of $M<n$.
3. It is infeasible to determine d given e and n.

Algorithm

1. Key generation

The keys (public key and private key) for the RSA algorithm are generated as:

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 of similar bit-length. 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
3. Compute $\varphi(n) = (p – 1)(q – 1)$, where φ is Euler's totient function
4. Choose an integer e such that $1 < e < \varphi(n)$ and gcd $(e, \varphi(n)) = 1$, i.e. e and $\varphi(n)$ are co-prime.
   - e is released as the public key exponent.
   - e having a short bit-length and small Hamming weight results in more efficient encryption - most commonly 0x10001 = 65537. However, small values of e (such as 3) have been shown to be less secure in some settings.
5. Determine $d = e–1 \bmod \varphi(n)$; i.e. d is the multiplicative inverse of e mod $\varphi(n)$.
   - This is more clearly stated as solve for the value of d given $(d*e) \bmod \varphi(n) = 1$
   - This is often computed using the extended Euclidean algorithm.

i. d is kept as the private key exponent.

Public key : PU = {e, n}

Private Key : PR = {d, n}

2. Encryption

Alice transmits her public key (e, n) to Bob and keeps the private key secret. Bob then wishes to send message M to Alice. He computes the ciphertext c corresponding to

$$C = M^e \bmod n$$

This can be done quickly using the method of exponentiation by squaring. Bob then transmits C to Alice.

3. Decryption

Alice can recover M from C by using her private key exponent d via computing

$$M = C^d \bmod n$$

Example

1. Select two prime numbers, p = 17 and q = 11.
2. Calculate n = pq = 17*11 = 187.
3. Calculate Ø(n) = (p-1)(q-1) = 16*10 = 160.
4. Select e such that relatively prime to Ø(n) = 160 & less than Ø(n); choose e = 7.
5. Determine d such that de ≡ 1 (mod 160) and d < 160. The correct value is d = 23, because 23*7 = 161 = 10*160+1; d can be calculated using the extended Euclid's algorithm.

The resulting keys are public key PU = {7, 187} and private key PR = {23, 187}. The example shows the use of these keys for plaintext input of M=88.

INPUT:

- Two prime numbers p = 17 and q = 11.
- Select e = 7.
- Plaintext = 88.

OUTPUT:

- PU = 7, 187
- PR = 23, 187
- Ciphertext = 11

Assignment 5

Implementation of MD – 5 Algorithm

AIM **:** Calculate the message digest of a text using MD-5 algorithm in Java.

OBJECTIVE : To implement and understand the working of MD - 5

THEORY :

MD5 processes a variable length message into a fixed-length output of 128 bits. MD - 5 is designed to be computationally infeasible to:

1. Obtain the original message, given its message digest and
2. Find two messages producing the same message digest

Following are the important steps in execution of MD - 5

Step 1. Padding

The first step in MD - 5 is to add padding to the end of the original message in such a way that the length of the message is 64 bits short of multiple of 512.Padding is always added even if message is already 64 bit short of multiple of 512.

Step 2. Append length

The length of the message excluding the length of the padding is now calculated and appended to the end of the padding as a 64-bit block.

Step 3. Divide the input into 512-bit block

The input message is now divided into blocks, each of length 512 bits. These blocks become the input to the message digest processing logic.

Step 4. Initialize chaining variables

Five chaining variables A through D are initialized. Each chaining variable is 32 bits in length.

Step 5. Process Block

This process is divided into following sub steps

Step 5.1

Copy the chaining variables A–D into variables a-d. The combination of a-d called abcd will be considered as a single register for storing the temporary intermediate as well as the final results.

Step 5.2

Now divide the current 512-bit block into 16 sub blocks, each consisting of 32 bits.

Step 5.3

MD-5 has 4 rounds each consisting of 16 steps or iteration. Each round takes the current 512-bit block, the register abcd and a constant t as the three inputs. It then updates the contents of the register abcd using Md-5 algorithm steps.

Step 5.4

MD-5 consist of four rounds, each round containing 16 iterations. This makes a total of 64 iterations. The logical operation of MD-5 is shown in following Figure

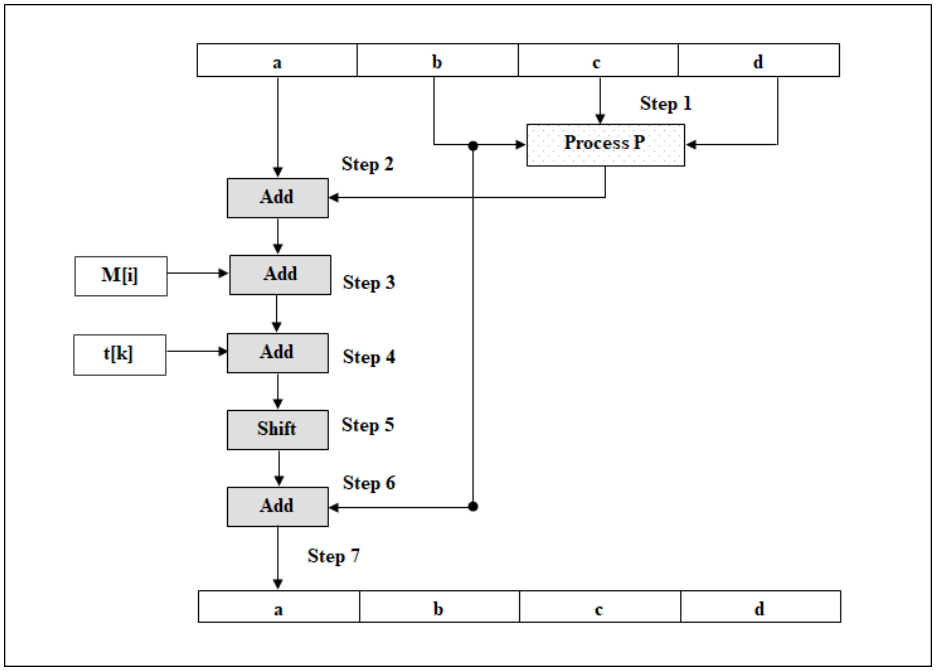

Figure 5.1 MD-5 Steps

Required Classes

1. **Class Message Digest (java.security.MessageDigest)**

This MessageDigest class provides applications the functionality of a message digest algorithm, such as MD5 or SHA. Message digests are secure one-way hash functions that take arbitrary-sized data and output a fixed-length hash value.

Constructor Summary	
protected	**MessageDigest(**String algorithm**)** Creates a message digest with the specified algorithm name.
Required Methods	

static MessageDigest	**getInstance**(String algorithm) Generates a MessageDigest object that implements the specified digest algorithm.
static MessageDigest	**getInstance**(String algorithm,Provider provider) Generates a MessageDigest object implementing the specified algorithm, as supplied from the specified provider, if such an algorithm is available from the provider.
static MessageDigest	**getInstance**(String algorithm, String provider) Generates a MessageDigest object implementing the specified algorithm, as supplied from the specified provider, if such an algorithm is available from the provider.
void	**update**(byte[] input) Updates the digest using the specified byte.
void	**update**(byte[] input) Updates the digest using the specified array of bytes.
void	**update**(byte[] input, int offset, int len) Updates the digest using the specified array of bytes, starting at the specified offset.

Steps:

Generating Digest

1. Get Instance of MessageDigest class for MD - 5 algorithm using getInstance method of MessageDigest class.
2. Convert the plaintext into bytes and pass it to update () method of MessageDigest class.
3. Use digest () method of MessageDiegest class to generate digest which returns the result in bytes format.

INPUT **:** Plaintext

OUTPUT **:** Message Digest