

IS Lab File

Shweta Pardeshi

2183216

MITU18BTCS0178

Final Year B-Tech CSE IS-3

Batch A

ASSIGNMENT NO.1

TITLE: Chinese Remainder Theorem

PROBLEM STATEMENT: Implement a number theory such as Chinese remainder Theorem

OBJECTIVE:

To study & implement Chinese Remainder Theorem

THEORY:

Chinese Remainder Theorem is used to solve set of congruent equations with one variable but different modulus, which are relatively prime

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

$$x \equiv a_3 \pmod{m_3}$$

.....

$$x \equiv a_k \pmod{m_k}$$

The Chinese Remainder Theorem states that the above equations have unique solution if the moduli are relatively prime. Below are the steps needed to follow to solve set of congruent equations using Chinese Remainder Theorem

Step I: Find $M = m_1 \times m_2 \times m_3 \dots m_k$ where M is common modulus

Step II: Find $M_1 = M/m_1$, $M_2 = M/m_2$ and so on

Step III: Find multiplicative inverses for M_1 , M_2 and so on

Step IV: Put the values in the below equation to solve for X

$$X = (a_1 \times M_1 \times M_1^{-1} + a_2 \times M_2 \times M_2^{-1} + a_3 \times M_3 \times M_3^{-1}) \pmod{M}$$

Example

$$X = 4 \pmod{5}$$

$$X = 6 \pmod{8}$$

$$X = 8 \pmod{9}$$

Step I: $M = 5 \times 8 \times 9 = 360$

Step II: $M_1 = M/m_1 = 360 / 5 = 72$

$$M2 = M/m2 = 360 / 8 = 45$$

$$M3 = M/m3 = 360 / 9 = 40$$

Step III:

To find M1 inverse, Solve for GCD (m1, M1) using Extended Euclidean Algorithm.
GCD (5, 72)

<i>q</i>	<i>r1</i>	<i>r2</i>	<i>r</i>	<i>t1</i>	<i>t2</i>	<i>t</i>
0	5	72	5	0	1	0
14	72	5	2	1	0	1
2	5	2	1	0	1	-2

The inverse value cannot be negative, so add modulus into it to make it positive.

$$M1 \text{ inverse} = -2 + 5 = 3$$

To find M2 inverse, Solve for GCD (m2, M2) using Extended Euclidean Algorithm.
GCD (8, 45)

<i>q</i>	<i>r1</i>	<i>r2</i>	<i>r</i>	<i>t1</i>	<i>t2</i>	<i>t</i>
0	8	45	8	0	1	0
5	45	8	5	1	0	1
1	8	5	3	0	1	-1
1	5	3	2	1	-1	2
1	3	2	1	-1	2	-3

$$M2 \text{ inverse} = -3 + 8 = 5$$

To find M3 inverse, Solve for GCD (m3, M3) using Extended Euclidean Algorithm.
GCD (9, 40)

<i>q</i>	<i>r1</i>	<i>r2</i>	<i>r</i>	<i>t1</i>	<i>t2</i>	<i>t</i>
0	9	40	9	0	1	0
4	40	9	4	1	0	1
2	9	4	1	0	1	-2

$$M3 \text{ inverse} = -2 + 9 = 7$$

Step IV: Put the values in the below equation to solve for X

$$X = (a1 \times M1 \times M1^{-1} + a2 \times M2 \times M2^{-1} + a3 \times M3 \times M3^{-1}) \bmod M$$

$$X = (4 \times 72 \times 3 + 6 \times 45 \times 5 + 8 \times 40 \times 7) \bmod 360$$

$$X = (864 + 1350 + 2240) \bmod 360$$

$$X = 4454 \bmod 360$$

$$X = 134$$

Applications

The Chinese Remainder Theorem has several applications in cryptography. One is to solve the quadratic congruence and the other is to represent very large number in terms of list of small integers.

CODE:

```
# A Python3 program to demonstrate working of:
# Chinese remainder Theorem

# Returns modulo inverse of a with
# respect to m using extended Euclid Algorithm.
# multiplicative-inverse-under-modulo-m/
def inv(a, m):
    m0 = m
    x0 = 0
    x1 = 1

    if (m == 1):
        return 0

    # Apply extended Euclid Algorithm
    while (a > 1):
        # q is quotient
        q = a // m

        t = m

        # m is remainder now, process
        # same as Euclid's algo
        m = a % m
        a = t

        t = x0

        x0 = x1 - q * x0

        x1 = t

    # Make x1 positive
    if (x1 < 0):
        x1 = x1 + m0

    return x1

# k is size of num[] and rem[].
# Returns the smallest
# number x such that:
# x % num[0] = rem[0],
# x % num[1] = rem[1],
# .....
# x % num[k-2] = rem[k-1]
# Assumption: Numbers in num[]
# are pairwise coprime
# (gcd for every pair is 1)
```

```
def findMinX(num, rem, k):
    # Compute product of all numbers
    prod = 1
    for i in range(0, k):
        prod = prod * num[i]

    # Initialize result
    result = 0

    # Apply above formula
    for i in range(0, k):
        pp = prod // num[i]
        result = result + rem[i] * inv(pp, num[i]) * pp

    return result % prod

# Driver method
num = [3, 4, 5]
rem = [2, 3, 1]
k = len(num)
print("x is", findMinX(num, rem, k))
```

OUTPUT:

The screenshot shows a Python IDE window titled "Chinese remiander theorem". The command prompt shows the execution of the script: "D:\python docs\pythonIS_Labs\venv\Scripts\python.exe" "D:/python docs/pythonIS_Labs/Chinese remiander theorem.py". The output is "x is 11". Below the output, it says "Process finished with exit code 0".

CONCLUSION:

We have studied & implemented Chinese Reminder Theorem.

QUESTIONS:

Batch A:

1. Use the Chinese remainder theorem to find a solution to each of the following linear systems:

$$x \equiv 0 \pmod{2}$$

$$x \equiv 0 \pmod{3}$$

$$x \equiv 1 \pmod{5}$$

$$x \equiv 6 \pmod{7}$$
2. In a Chinese Reminder theorem, Let $N=210$ & let $n_1=5$, $n_2=6$, $n_3=7$. Compute $f_1(3,5,2)$. i.e $x_1=3$, $x_2=5$, $x_3=2$. Compute x .

Batch B:

1. Jessica breeds rabbits. She is not sure exactly how many she has today, but as she

was moving them about this morning she noticed some things. When she fed them in group of 5, she had 4 left over. When she bathed them in group of 8, she had group of 6 left over. She took them outside to wonder in groups of 9, but then last group consisted of only 8. Find out how many rabbits she has?

2. Find the integer that has a remainder of 3 when divided by 7 & 13, but is divisible by 12.

Batch C:

1. Determine the value of x using Chinese remainder theorem

$$X \equiv 1 \pmod{5}$$

$$X \equiv 6 \pmod{7}$$

$$X \equiv 8 \pmod{11}$$

2. Determine the value of x using Chinese remainder theorem

$$X \equiv 1 \pmod{13}$$

$$X \equiv 11 \pmod{23}$$

ASSIGNMENT NO.2

TITLE: Extended Euclidian Algorithm

PROBLEM STATEMENT: Implement Euclidean and Extended Euclidean algorithm to find out GCD and solve the inverse modproblem.

OBJECTIVES:

To study Euclidian & Extended Euclidian algorithm

THEORY:

The extended Euclidean algorithm is an extension to the Euclidean algorithm. Besides finding the greatest common divisor of integers a and b , as the Euclidean algorithm does, it also finds integers x and y (one of which is typically negative).

$$ax + by = \gcd(a, b) \text{ or } sa + tb = \gcd(a, b)$$

The extended Euclidean algorithm is particularly useful when a and b are coprime, since x is the multiplicative inverse of a modulo b , and y is the multiplicative inverse of b modulo a .

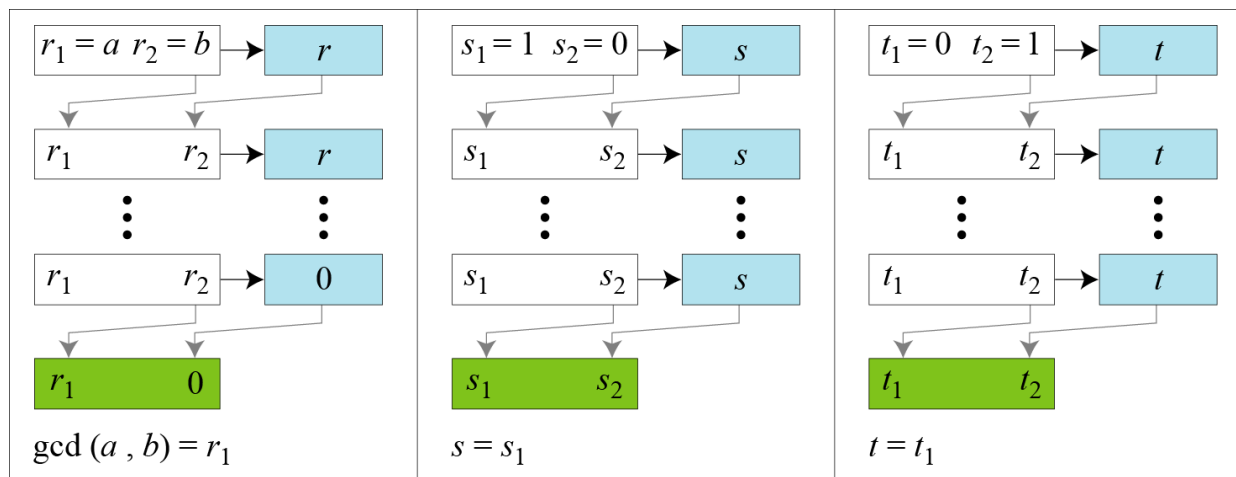


Figure: Extended Euclid's Algorithm Process

Algorithm:

Extend the algorithm to compute the integer coefficients x and y such that

$\gcd(a, b) = ax + by$

Extended-Euclid (a, b)

```

 $r_1 \leftarrow a; \quad r_2 \leftarrow b;$ 
 $s_1 \leftarrow 1; \quad s_2 \leftarrow 0;$ 
 $t_1 \leftarrow 0; \quad t_2 \leftarrow 1;$ 
(Initialization)
while ( $r_2 > 0$ )
{
   $q \leftarrow r_1 / r_2;$ 

   $r \leftarrow r_1 - q \times r_2;$ 
   $r_1 \leftarrow r_2; \quad r_2 \leftarrow r;$ 
  (Updating  $r$ 's)

   $s \leftarrow s_1 - q \times s_2;$ 
   $s_1 \leftarrow s_2; \quad s_2 \leftarrow s;$ 
  (Updating  $s$ 's)

   $t \leftarrow t_1 - q \times t_2;$ 
   $t_1 \leftarrow t_2; \quad t_2 \leftarrow t;$ 
  (Updating  $t$ 's)
}
 $\gcd(a, b) \leftarrow r_1; \quad s \leftarrow s_1; \quad t \leftarrow t_1$ 

```

Example:

GCD (161, 28)

q	r_1	r_2	r	s_1	s_2	s	t_1	t_2	t
5	161	28	21	1	0	1	0	1	-5
1	28	21	7	0	1	-1	1	-5	6
3	21	7	0	1	-1	4	-5	6	-23
	7	0		-1	4		6	-23	

Here GCD value we are getting as 7. Value of s is -1 and value of t is 6. If we put these values into equation,

$$ax + by = \gcd(a, b)$$

$$161 * (-1) + 28 * (6) = 7$$

This satisfies the equation for Extended Euclidean Algorithm

CODE:

```
# Python program to demonstrate working of extended
# Euclidean Algorithm

# function for extended Euclidean Algorithm
def gcdExtended(a, b):
    # Base Case
    if a == 0:
        return b, 0, 1


    gcd, x1, y1 = gcdExtended(b % a, a)

    # Update x and y using results of recursive
    # call
    x = y1 - (b // a) * x1
    y = x1

    return gcd, x, y

# Driver code
a, b = 35, 15
g, x, y = gcdExtended(a, b)
print("gcd(", a, ", ", b, ") = ", g)
print("x = ", x, "y = ", y)
```

OUTPUT:



```
Run: Extended euclidean theorem x
"D:\python docs\pythonIS_Labs\venv\Scripts\python.exe" "D:/python docs/pythonIS_Labs/Extended euclidean theorem.py"
gcd( 35 , 15 ) = 5
x = 1 y = -2
Process finished with exit code 0
```

CONCLUSION:

We have studied and implemented the Extended Euclidian algorithm.

QUESTIONS:

Batch A:

1. Determine integers x and y such that $\text{gcd}(421, 111) = 421x + 111y$.
2. Find x and y such that $51x + 100y = 1$.
3. How to compute multiplicative inverse using Extended Euclid's algorithm. Illustrate.

Batch B:

1. Using Euclidian algorithm calculate $\text{GCD}(48, 30)$ & $\text{GCD}(105, 80)$.
 2. What is the significance of Extended Euclidian algorithm with reference to RSA Algorithm?
-

3. Find the multiplicative inverse of 8 mod 11, using the Euclidean Algorithm.

Batch C:

1. Using Euclidean algorithm calculate $\gcd(16,20)$ and $\gcd(50,60)$.
 2. Find the multiplicative inverse of 43 mod 64, using the Euclidean Algorithm?
 3. Find x and y such that $97x + 20y = 1$.
-

ASSIGNMENT NO.3

TITLE: RSA Algorithm

PROBLEM STATEMENT: Implement RSA public key cryptosystem for key generation and cipher verification.

OBJECTIVES:

To understand,

1. Public key algorithm.
2. RSA algorithm
3. Concept of Public key and Private Key

THEORY:

Public Key Algorithm:

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristics:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristics:

- 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 encryption 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. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
 - **Cipher text:** 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 cipher texts.
 - **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.
-

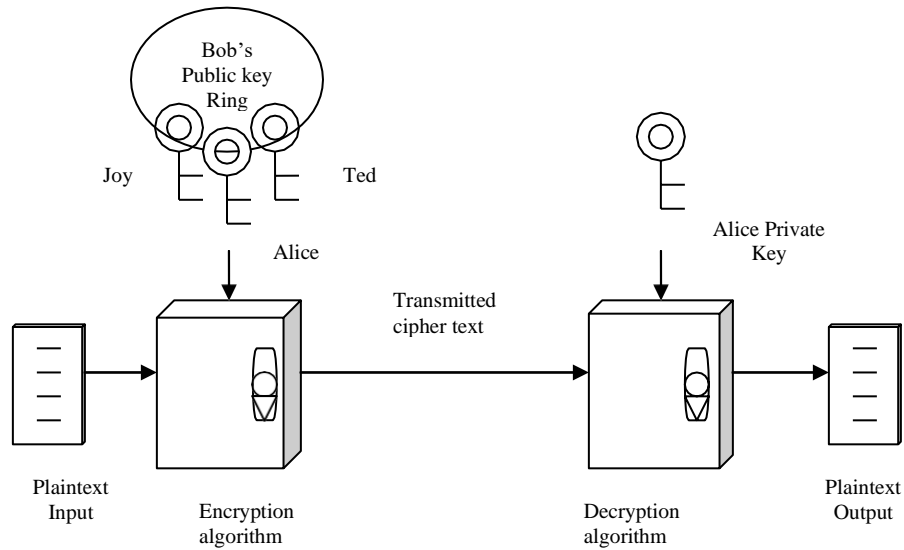


Figure: Public key cryptography

The essential steps are as the following:

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. As figure suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.

When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

The RSA Algorithm:

The scheme developed by Rivest, Shamir and Adleman makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n . That is the block size must be less than or equal to $\log_2(n)$; in practice the block size is I bits, where $2^i < n \leq 2^{i+1}$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C :

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

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

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 .
2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$.
3. It is feasible to determine d given e and n .

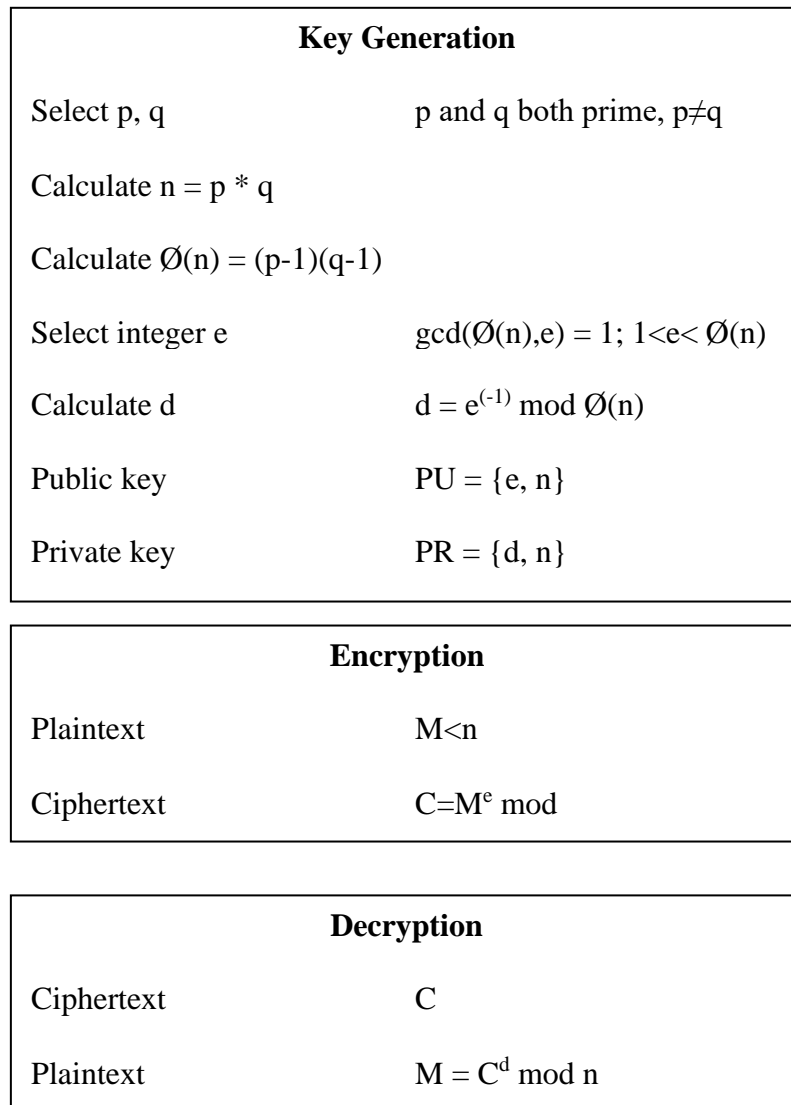


Figure: The RSA Algorithm

Example 1:

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 * 11 = 187$.
3. Calculate $\phi(n) = (p-1)(q-1) = 16 * 10 = 160$.
4. Select e such that relatively prime to $\phi(n)=160$ & less than $\phi(n)$; we choose $e = 7$.

5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$; 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 $M=88$.

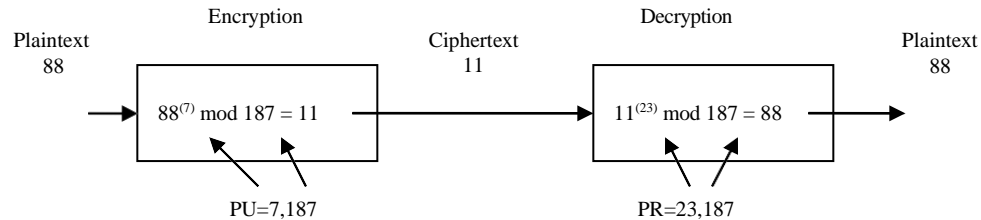


Figure: Example of RSA.

Example 2:

$$P = 7, Q = 13, M = 10.$$

Step I: $n = 3 * 11 = 33$

Step II: $\phi(n) = 2 * 10 = 20$

Step III: Select e , such that $\gcd(\phi(n), e) = 1$, $\gcd(20, 3) = 1$, So we can select $e = 3$

Step IV: To calculate d , solve for $\gcd(\phi(n), e)$ using extended Euclid's algorithm and pick up the value of t .

q	$r1$	$r2$	r	$t1$	$t2$	t
6	20	3	2	0	1	-6
1	3	2	1	1	-6	7

$$d = 7$$

Step V: For Encryption,

$$\begin{aligned} C &= M^e \bmod n \\ &= 2^3 \bmod 33 \\ &= 8 \bmod 33 \\ &= 8 \end{aligned}$$

Step VI: For Decryption,

$$\begin{aligned} M &= C^d \bmod n \\ &= 8^7 \bmod 33 \\ &= ((8^2 \bmod 33) (8^2 \bmod 33) (8^2 \bmod 33) (8^1 \bmod 33)) \bmod 33 \\ &= (31 * 31 * 31 * 8) \bmod 33 \\ &= (961 \bmod 33) (248 \bmod 33) \bmod 33 \\ &= 68 \bmod 33 \\ &= 2 \end{aligned}$$

Advantages:

1. Easy to implement.

Disadvantages:

1. Anyone can announce the public key.

Algorithm:

1. Start
2. Input two prime numbers p and q.
3. Calculate $n = pq$.
4. Calculate $\phi(n) = (p-1)(q-1)$.
5. Input value of e.
6. Determine d.
7. Determine PU and PR.
8. Take input plaintext.
9. Encrypt the plaintext and show the output.
10. Stop.

CODE :

```
// C program for RSA asymmetric cryptographic
// algorithm. For demonstration values are
// relatively small compared to practical
// application
#include<stdio.h>
#include<math.h>

// Returns gcd of a and b
int gcd(int a, int h)
{
    int temp;
    while (1)
    {
        temp = a%h;
        if (temp == 0)
            return h;
        a = h;
        h = temp;
    }
}

// Code to demonstrate RSA algorithm
int main()
```

```

{
    // Two random prime numbers
    double p = 3;
    double q = 7;

    // First part of public key:
    double n = p*q;

    // Finding other part of public key.
    // e stands for encrypt
    double e = 2;
    double phi = (p-1)*(q-1);
    while (e < phi)
    {
        // e must be co-prime to phi and
        // smaller than phi.
        if (gcd(e, phi)==1)
            break;
        else
            e++;
    }

    // Private key (d stands for decrypt)
    // choosing d such that it satisfies
    //  $d \cdot e = 1 + k \cdot \text{totient}$ 
    int k = 2; // A constant value
    double d = (1 + (k*phi))/e;

    // Message to be encrypted
    double msg = 20;

    printf("Message data = %lf", msg);

    // Encryption  $c = (\text{msg}^e) \% n$ 
    double c = pow(msg, e);
    c = fmod(c, n);
    printf("\nEncrypted data = %lf", c);

    // Decryption  $m = (c^d) \% n$ 
    double m = pow(c, d);
    m = fmod(m, n);
    printf("\nOriginal Message Sent = %lf", m);

    return 0;
}

```


OUTPUT:

Message data = 12.000000

Encrypted data = 3.000000

Original Message Sent = 12.000000

CONCLUSION:

We have studied and implemented the public key algorithm that is RSA algorithm.

QUESTIONS:**Batch A:**

1. What are the principle elements of a public key cryptosystem?
2. Differentiate public key and conventional encryption?
3. Perform encryption and decryption using RSA Alg. for the following. $P=7$; $q=11$; $e=17$; $M=8$.

Batch B:

1. Specify the applications of the public key cryptosystem?
2. Define Euler's totient function or phi function and their applications?
3. Perform encryption and decryption using RSA Alg. for the following. $P=3$; $q=11$; $e=17$; $M=12$.

Batch C:

1. Give the significance of Extended Euclid's Algorithm with respect to RSA.
 2. How to calculate private key d in RSA algorithm?
 3. Calculate cipher text using RSA algorithm. Given data is as follows: Prime numbers P , Q as 13, 17 & the plain text to be sent is 12. Assume public key as 19.
-

ASSIGNMENT NO.4

TITLE: Diffie Hellman Key Exchange

PROBLEM STATEMENT: Implement Diffie Hellman key exchange algorithm for secret key generation and distribution of public key

OBJECTIVE:

1. To learn the basics of key management.
2. To study & implement Diffie Hellman key exchange algorithm.

THEORY:

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

Algorithm:

There are two publicly known numbers: a prime number q and an integer α that is a primitive root of q .

Suppose the users A and B wish to exchange a key.

User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A} \bmod q$. Similarly User B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha^{X_B} \bmod q$.

Each side keeps the value X private and makes the value Y available publicly to the other side.

User A computes the key K as $K = Y_B^{X_A} \bmod q$

User B computes the key K as $K = Y_A^{X_B} \bmod q$. These two calculations produce identical results.

Eg.

1. Users Alice & Bob who wish to swap keys:
2. Agree on prime $q=353$ and $\alpha=3$
3. Select random secret keys:
 - A chooses $X_A=97$, B chooses $X_B=233$
4. Compute public keys:
 - $Y_A=3^{97} \bmod 353 = 40$ (Alice)
 - $Y_B=3^{233} \bmod 353 = 248$ (Bob)

5. Compute shared session key as:

$$K_{AB} = Y_B^{x_A} \bmod 353 = 248^{97} = 160 \text{ (Alice)}$$

$$K_{AB} = Y_A^{x_B} \bmod 353 = 40^{233} = 160 \text{ (Bob)}$$

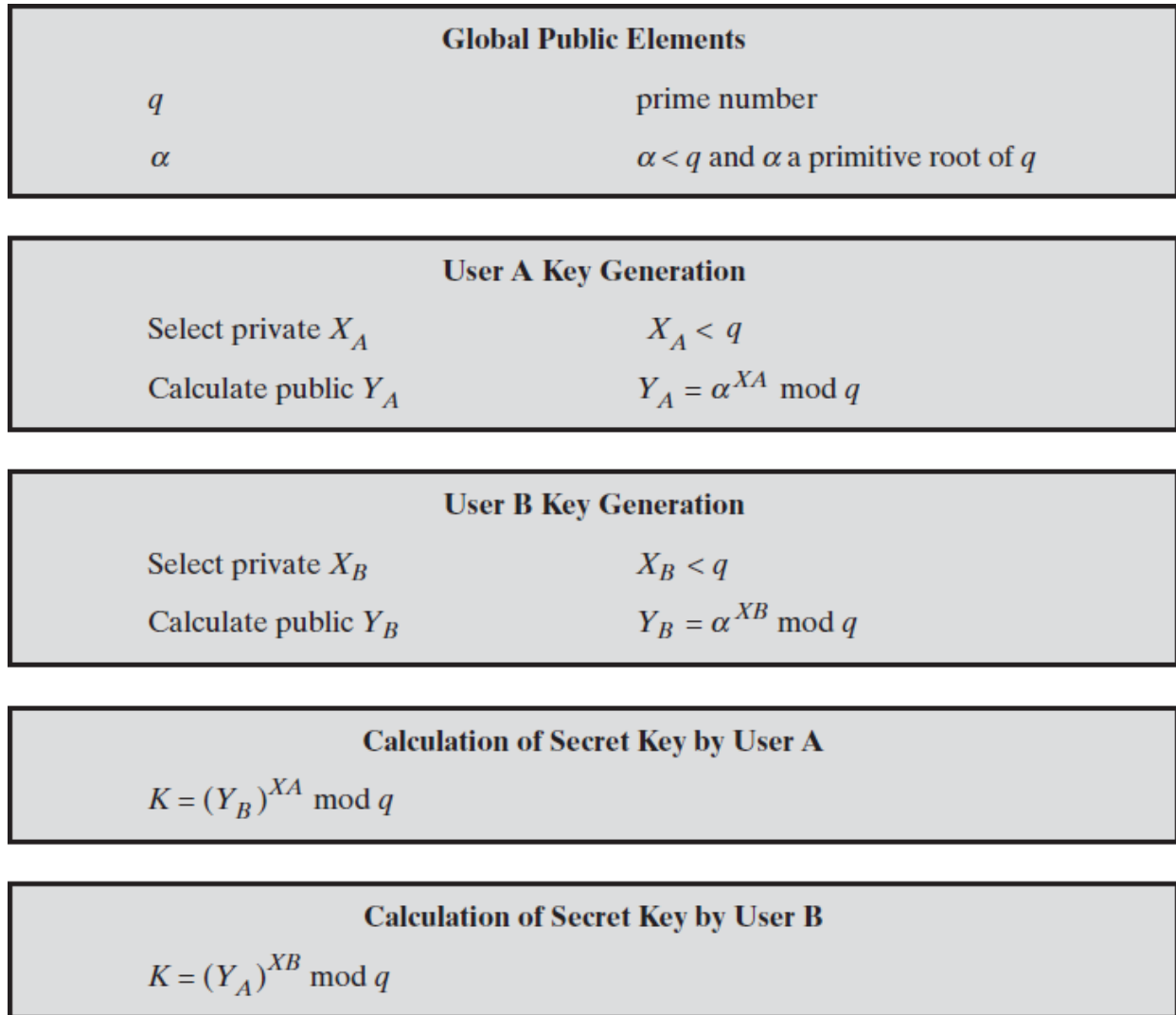


Figure: Diffie Hellman Process

CODE:

```
from random import randint
P = 23
# A primitive root for P, G is taken
G = 9

print('The Value of P is :%d' % (P))
print('The Value of G is :%d' % (G))

# Alice will choose the private key a
```

```

a = 4
print('The Private Key a for Alice is :%d' % (a))

# gets the generated key
x = int(pow(G, a, P))

# Bob will choose the private key b
b = 3
print('The Private Key b for Bob is :%d' % (b))

# gets the generated key
y = int(pow(G, b, P))

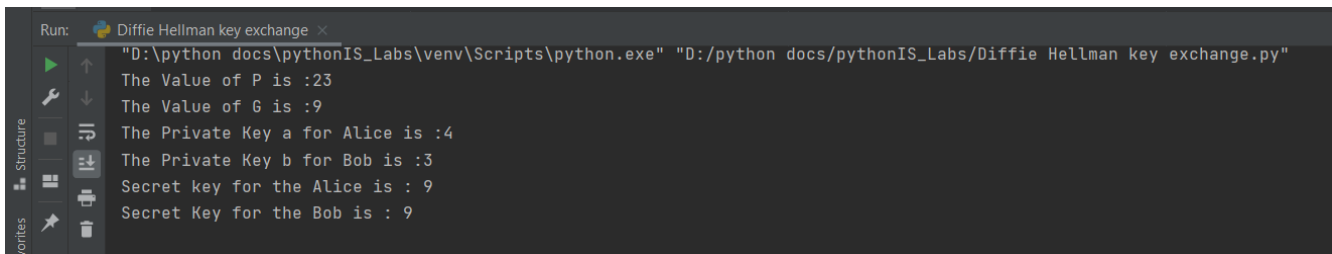
# Secret key for Alice
ka = int(pow(y, a, P))

# Secret key for Bob
kb = int(pow(x, b, P))

print('Secret key for the Alice is : %d' % (ka))
print('Secret Key for the Bob is : %d' % (kb))

```

OUTPUT:



```

Run: Diffie Hellman key exchange x
"D:\python docs\pythonIS_Labs\venv\Scripts\python.exe" "D:/python docs/pythonIS_Labs/Diffie Hellman key exchange.py"
The Value of P is :23
The Value of G is :9
The Private Key a for Alice is :4
The Private Key b for Bob is :3
Secret key for the Alice is : 9
Secret Key for the Bob is : 9

```

CONCLUSION:

We have studied & implemented Diffie Hellman key exchange algorithm.

QUESTIONS:

Batch A:

1. What is primitive root of a number? Explain with suitable example.
2. Explain Key management with respect to key generation.

3. Let $p = 37$ and $g = 13$. Let Alice pick $a = 10$. Let Bob pick $b = 7$. Find of the secret key K .

Batch B:

1. What is Man in the middle attack? Explain with respect to Diffie-Hellman Algorithm.
2. Given the values of $p=11$ and $g=2$. Alice chooses a secret integer whose value is 9 and Bob chooses a secret integer whose value is 4. Find of the secret key K
3. Explain Key management with respect to key distribution.

Batch C:

1. Do we share a key in Diffie-Hellman algorithm or do we create it?
 2. Show that 2 is primitive root of 11.
 3. Explain Key management with respect to key storage.
-

ASSIGNMENT NO.5

TITLE: MD5/SHA1 Algorithm

PROBLEM STATEMENT: Implement MD5/SHA-1 algorithms for verifying and maintaining the integrity of information.

OBJECTIVE:

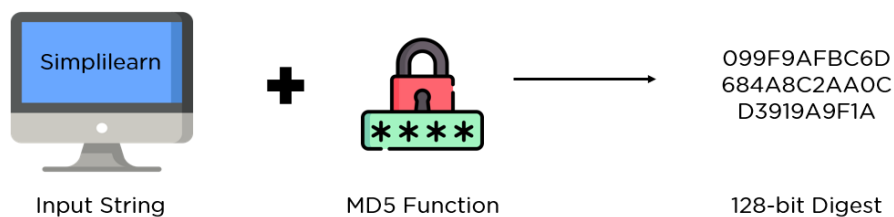
To study & implement MD5/ SHA1 algorithm.

THEORY:

Cryptographic hashes are used in day-day life like in digital signatures, message authentication codes, manipulation detection, fingerprints, checksums (message integrity check), hash tables, password storage and much more. They are also used in sending messages over network for security or storing messages in databases.

What is the MD5 Algorithm?

MD5 (Message Digest Method 5) is a cryptographic hash algorithm used to generate a 128-bit digest from a string of any length. It represents the digests as 32-digit hexadecimal numbers.



Ronald Rivest designed this algorithm in 1991 to provide the means for digital signature verification. Eventually, it was integrated into multiple other frameworks to bolster security indexes.

Input into Hash Function	MD5 Hash Value/Digest
Cryptography	d2fc0657a64a3291826136c7712abbe7
Cryptographyabc123	c56db83ab5482b4e94536f4a29b21de0
Cryptographyxyz456	783b10b483435e05f3f2705bdd5a825c

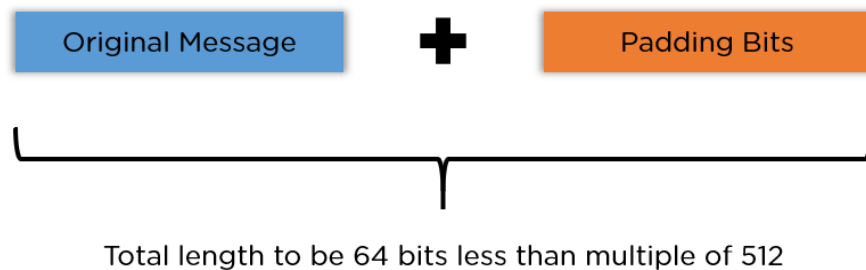
The digest size is always 128 bits, and thanks to hashing function guidelines, a minor change in the input string generate a drastically different digest. This is essential to prevent similar hash generation as much as possible, also known as a hash collision.

Steps in MD5 Algorithm

There are four major sections of the algorithm:

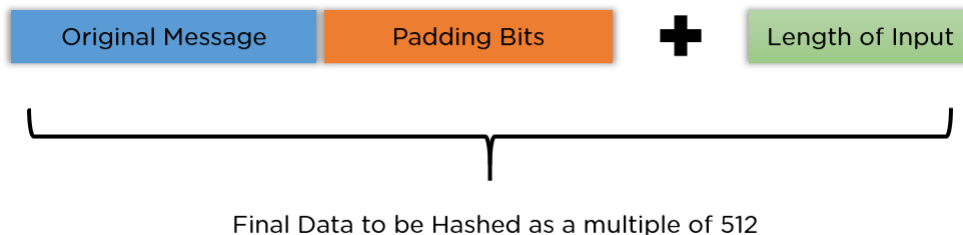
Padding Bits

When you receive the input string, you have to make sure the size is 64 bits short of a multiple of 512. When it comes to padding the bits, you must add one (1) first, followed by zeroes to round out the extra characters.



Padding Length

You need to add a few more characters to make your final string a multiple of 512. To do so, take the length of the initial input and express it in the form of 64 bits. On combining the two, the final string is ready to be hashed.



Initialize MD Buffer

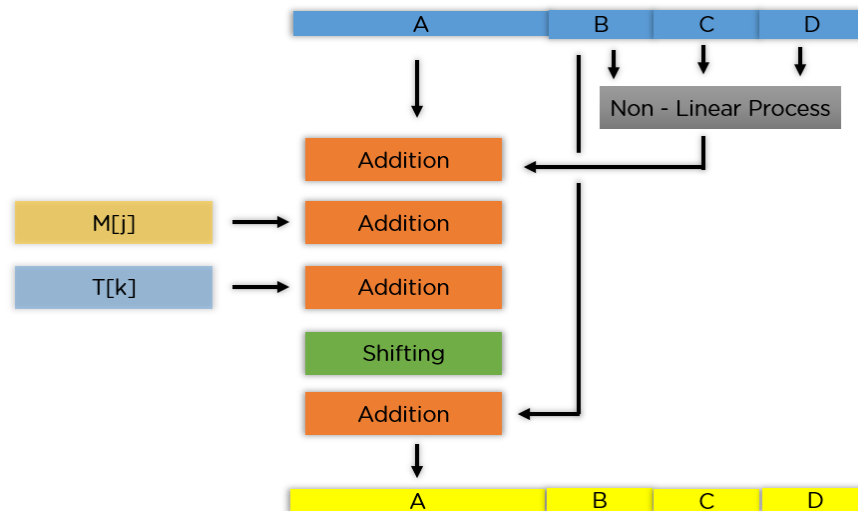
The entire string is converted into multiple blocks of 512 bits each. You also need to initialize four different buffers, namely A, B, C, and D. These buffers are 32 bits each and are initialized as follows:

A = 01 23 45 67; B = 89 ab cd ef ; C = fe dc ba 98 ; D = 76 54 32 10

Process Each Block

Each 512-bit block gets broken down further into 16 sub-blocks of 32 bits each. There are four rounds of operations, with each round utilizing all the sub-blocks, the buffers, and a constant array value.

This constant array can be denoted as T[1] -> T[64]. Each of the sub-blocks are denoted as M[0] -> M[15].



According to the image above, you see the values being run for a single buffer A. The correct order is as follows:

- It passes B, C, and D onto a non-linear process.
 - The result is added with the value present at A.
 - It adds the sub-block value to the result above.
 - Then, it adds the constant value for that particular iteration.
-

- There is a circular shift applied to the string.
- As a final step, it adds the value of B to the string and is stored in buffer A.

The steps mentioned above are run for every buffer and every sub-block. When the last block's final buffer is complete, you will receive the MD5 digest.

The non-linear process above is different for each round of the sub-block.

Round 1: (b AND c) OR ((NOT b) AND (d))

Round 2: (b AND d) OR (c AND (NOT d))

Round 3: b XOR c XOR d

Round 4: c XOR (b OR (NOT d))

MD5 in Python

- There are many hash functions defined in the “hashlib” library in python.
- Functions associated:
 - encode (): Converts the string into bytes to be acceptable by hash function.
 - digest (): Returns the encoded data in byte format.
 - hexdigest (): Returns the encoded data in hexadecimal format.
 -

Difference Between MD5 and SHA-1

S.NO	MD5	SHA1
1.	MD5 stands for Message Digest.	While SHA1 stands for Secure Hash Algorithm.
2.	MD5 can have 128 bits length of message digest.	Whereas SHA1 can have 160 bits length of message digest.
3.	The speed of MD5 is fast in comparison of SHA1's speed.	While the speed of SHA1 is slow in comparison of MD5's speed.
4.	To make out the initial message the	On the opposite hand, in SHA1 it'll be

S.NO	MD5	SHA1
	aggressor would want 2^{128} operations whereas exploitation the MD5 algorithmic program.	2^{160} that makes it quite troublesome to seek out.
5.	MD5 is simple than SHA1.	While SHA1 is more complex than MD5.
6.	MD5 provides indigent or poor security.	While it provides balanced or tolerable security.
7.	In MD5, if the assailant needs to seek out the 2 messages having identical message digest then assailant would need to perform 2^{64} operations.	Whereas in SHA1, assailant would need to perform 2^{80} operations which is greater than MD5.
8.	MD5 was presented in the year 1992.	While SHA1 was presented in the year 1995.

CODE:

```
# working of MD5 (string - hexadecimal) Shweta Pardeshi

import hashlib

# initializing string
str2hash = "MITADTUniversity "

# encoding MITADTUniversity using encode()
# then sending to md5()
result = hashlib.md5(str2hash.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of hash is : ", end="")
print(result.hexdigest())
```

The above code takes string and converts it into the byte equivalent using `encode()` so that it can be accepted by the hash function. The md5 hash function encodes it and then using `hexdigest()`, hexadecimal equivalent encoded string is printed.

OUTPUT:



The screenshot shows a terminal window titled "Run: MD5" with a dark background. The terminal displays the command to run a Python script and its output. The command is: `"D:\python docs\pythonIS_Labs\venv\Scripts\python.exe" "D:/python docs/pythonIS_Labs/MD5.py"`. The output is: `The hexadecimal equivalent of hash is : bf3e164722250e0769db585fe2b1dd28`. Below the output, it says `Process finished with exit code 0`. The IDE interface includes a sidebar with "Structure" and "Favorites" views, and a bottom status bar with tabs for "Run", "TODO", "Problems", "Terminal", and "Python Console".

```
Run: MD5 x
"D:\python docs\pythonIS_Labs\venv\Scripts\python.exe" "D:/python docs/pythonIS_Labs/MD5.py"
The hexadecimal equivalent of hash is : bf3e164722250e0769db585fe2b1dd28

Process finished with exit code 0

Run  TODO  Problems  Terminal  Python Console
```

CONCLUSION:

We have studied & implemented MD5 algorithm.