

Name - Rugveda Mohan Bodke
Class - CSE IS 1
Roll no - 2193201
Enrollment no - MITU19BTCS0062

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: $M = 5 * 8 * 9 = 360$

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

$M_2 = M/m_2 = 360 / 8 = 45$

$M_3 = M/m_3 = 360 / 9 = 40$

Step III:

To find M_1 inverse, Solve for GCD (m_1, M_1) using Extended Euclidean Algorithm.
GCD (5, 72)

q	r_1	r_2	r	t_1	t_2	t
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. M_1 inverse = $-2 + 5 = 3$

To find M_2 inverse, Solve for GCD (m_2, M_2) using Extended Euclidean Algorithm. GCD (8, 45)

q	r_1	r_2	r	t_1	t_2	t
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

M_2 inverse = $-3 + 8 = 5$

To find M_3 inverse, Solve for GCD (m_3, M_3) using Extended Euclidean Algorithm. GCD (9, 40)

q	r_1	r_2	r	t_1	t_2	t
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

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

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

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

$$X = 4454 \bmod 360$$

$$X = 134$$

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 C++ program to demonstrate working of Chinese remainder
// Theorem
#include<iostream>
using namespace std;

int findMinX(int num[], int rem[], int k)
{
    int x = 1;
    while (true) {
        // Check if remainder of x % num[j] is
        // rem[j] or not (for all j from 0 to k-1)
        int j;
        for (j=0; j<k; j++)
            if (x%num[j] != rem[j])
                break;
        // If all remainders matched, we found x
        if (j == k)
            return x;
        // Else try next number
        x++;
    }
    return x;
}

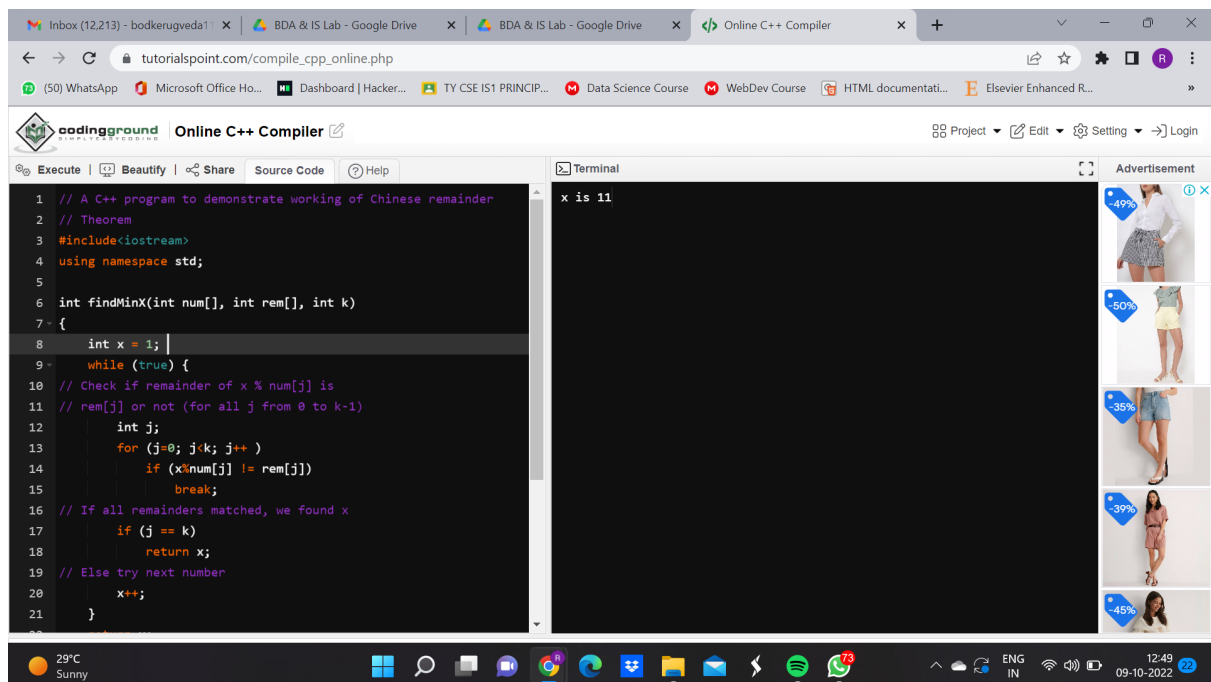
// Driver method
int main(void)
{
```

```

int num[] = {3, 4, 5};
int rem[] = {2, 3, 1};
int k = sizeof(num)/sizeof(num[0]);
cout << "x is " << findMinX(num, rem, k);
return 0;
}

```

OUTPUT (Screenshots):



The screenshot displays a web browser window with multiple tabs. The active tab is 'Online C++ Compiler' at the URL 'tutorialspoint.com/compile_cpp_online.php'. The browser's address bar and tabs are visible at the top. Below the browser, the 'codingground' logo and 'Online C++ Compiler' title are shown. The interface includes a 'Project' dropdown, 'Edit', 'Setting', and 'Login' links. The main area is divided into two panes: 'Source Code' on the left and 'Terminal' on the right. The 'Source Code' pane contains a C++ program that implements the Chinese Remainder Theorem. The 'Terminal' pane shows the output 'x is 11'. The program code is as follows:

```
11 // rem[j] or not (for all j from 0 to k-1)
12 int j;
13 for (j=0; j<k; j++)
14     if (x%num[j] != rem[j])
15         break;
16 // If all remainders matched, we found x
17 if (j == k)
18     return x;
19 // Else try next number
20 x++;
21 }
22 return x;
23 }
24 // Driver method
25 int main(void)
26 {
27     int num[] = {3, 4, 5};
28     int rem[] = {2, 3, 1};
29     int k = sizeof(num)/sizeof(num[0]);
30     cout << "x is " << findMinX(num, rem, k);
31     return 0;
32 }
```

The 'Terminal' pane shows the output 'x is 11'. The bottom of the screen shows a Windows taskbar with various application icons, including the Start button, Search, File Explorer, and several communication apps. The system tray on the right shows the date and time as '12:50 09-10-2022'.

CONCLUSION:

We have studied & implemented the Chinese Remainder Theorem.

ASSIGNMENT NO.2

TITLE: Extended Euclidean Algorithm

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

OBJECTIVES:

To study Euclidean & Extended Euclidean 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 .

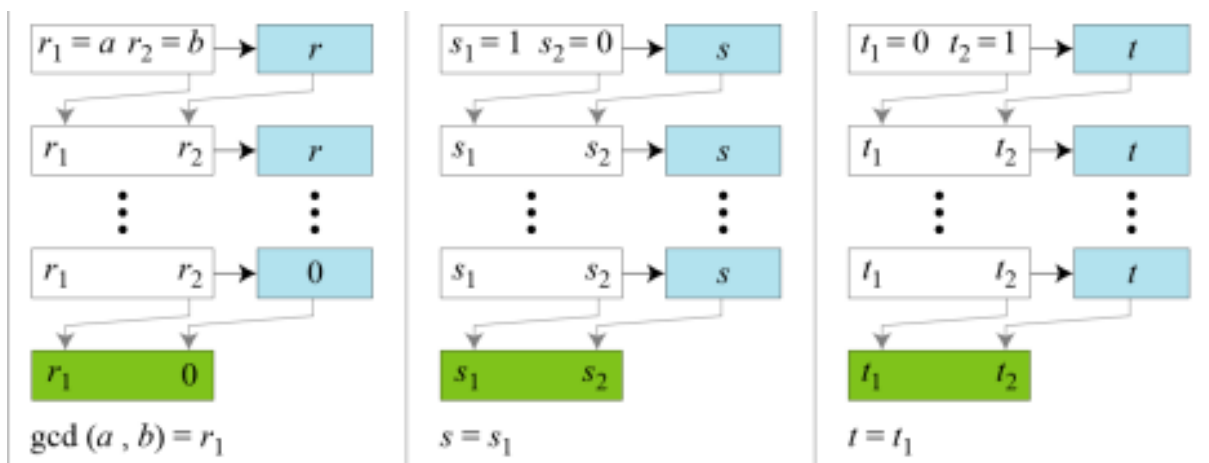


Figure: Extended Euclid's Algorithm Process

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,

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

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

This satisfies the equation for Extended Euclidean Algorithm

CODE:

```

// C program to demonstrate working of extended
// Euclidean Algorithm

```

```

#include <stdio.h>

// C function for extended Euclidean Algorithm
int gcdExtended(int a, int b, int *x, int *y)
{
    // Base Case
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExtended(b%a, a, &x1, &y1);

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

    return gcd;
}

// Driver Program
int main()
{
    int x, y;
    int a = 35, b = 15;
    int g = gcdExtended(a, b, &x, &y);
    printf("gcd(%d, %d) = %d", a, b, g);
    return 0;
}

```

OUTPUT (Screenshots) :


```
1 // C program to demonstrate working of extended
2 // Euclidean Algorithm
3 #include <stdio.h>
4
5 // C function for extended Euclidean Algorithm
6 int gcdExtended(int a, int b, int *x, int *y)
7 {
8     // Base Case
9     if (a == 0)
10    {
11        *x = 0;
12        *y = 1;
13        return b;
14    }
15
16    int x1, y1; // To store results of recursive call
17    int gcd = gcdExtended(b%a, a, &x1, &y1);
18
19    // Update x and y using results of recursive
20    // call
21    *x = y1 - (b/a) * x1;
```

Terminal: gcd(35, 15) = 5

```
22    *y = x1;
23
24    return gcd;
25 }
26
27 // Driver Program
28 int main()
29 {
30     int x, y;
31     int a = 35, b = 15;
32     int g = gcdExtended(a, b, &x, &y);
33     printf("gcd(%d, %d) = %d", a, b, g);
34     return 0;
35 }
```

Terminal: gcd(35, 15) = 5

CONCLUSION:

We have studied and implemented the Extended Euclidean algorithm.

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.

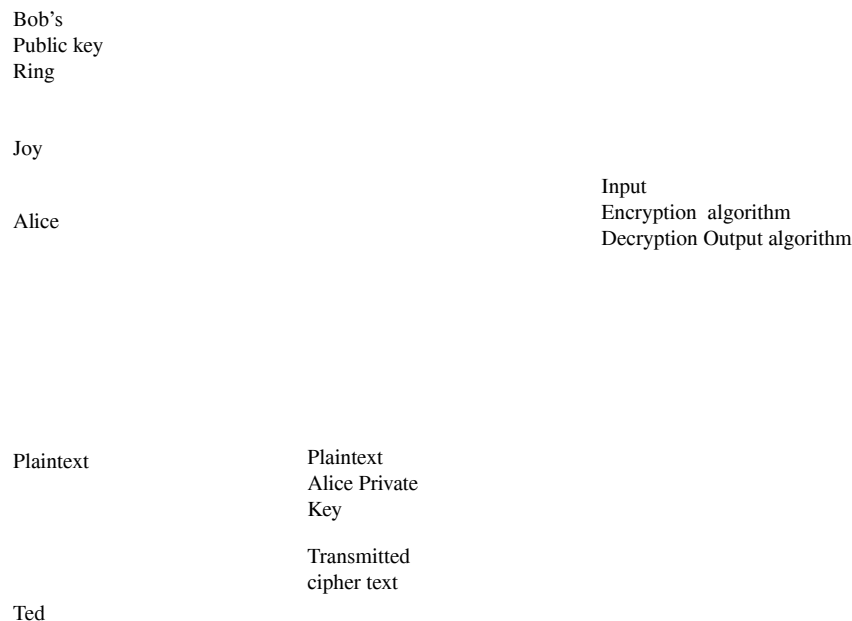


Figure: Public key cryptography

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 .

Key Generation

Select p, q p and q both prime, $p \neq q$

Calculate $n = p * q$

Calculate $\phi(n) = (p-1)(q-1)$

Select integer e $\gcd(\phi(n), e) = 1$; $1 < e < \phi(n)$

Calculate d $d = e^{(-1)} \bmod \phi(n)$

Public key $PU = \{e, n\}$

Private key $PR = \{d, n\}$

Encryption

Plaintext $M < n$

Ciphertext $C = M^e \bmod n$

Decryption

Ciphertext C

Plaintext $M = C^d \bmod n$

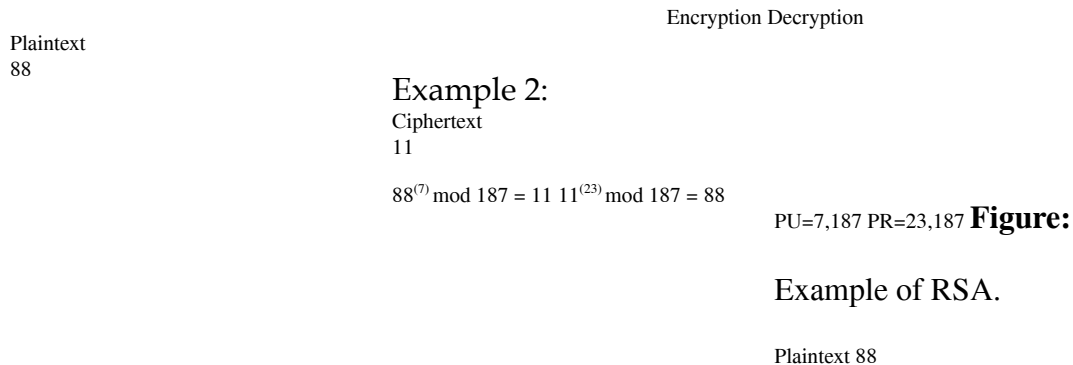
Figure: The RSA Algorithm

Example 1:

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 * 11 = 187$.

3. Calculate $\phi(n) = (p-1)(q-1) = 16 \cdot 10 = 160$.
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 $M=88$.



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

Step I: $n = 3 \cdot 11 = 33$

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

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,

$C = M^e \bmod n$

$= 2^3 \bmod 33$

$= 8 \bmod 33$

= 8

Step VI: For Decryption,

$M = C^d \bmod n$

$= 8^7 \bmod 33$

$= ((8^2 \bmod 33) (8^2 \bmod 33) (8^2 \bmod 33) (8^1 \bmod 33)) \bmod 33 = (31 * 31 * 31 * 8) \bmod 33$

$= (961 \bmod 33) (248 \bmod 33) \bmod 33$

$= 68 \bmod 33$

= 2

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:

```
#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 (Screenshots):

Online C Compiler

```
1 #include<stdio.h>
2 #include<math.h>
3 // Returns gcd of a and b
4 int gcd(int a, int h)
5 {
6     int temp;
7     while (1)
8     {
9         temp = a%h;
10        if (temp == 0)
11            return h;
12        a = h;
13        h = temp;
14    }
15 }
16 // Code to demonstrate RSA algorithm
17 int main()
18 {
19     // Two random prime numbers
20     double p = 3;
21     double q = 7;
```

Terminal

```
Message data = 20.000000
Encrypted data = 20.000000
Original Message Sent = 20.000000
```

Advertisements:

- Galaxy A53 5G
- PRICE DROP
- Galaxy S22
- PRICE DROP

29°C Mostly sunny 15:25 09-10-2022

Online C Compiler

```
21 double q = 7;
22 // First part of public key:
23 double n = p*q;
24 // Finding other part of public key.
25 // e stands for encrypt
26 double e = 2;
27 double phi = (p-1)*(q-1);
28 while (e < phi)
29 {
30     // e must be co-prime to phi and
31     // smaller than phi.
32     if (gcd(e, phi) == 1)
33         break;
34     else
35         e++;
36 }
37 // Private key (d stands for decrypt)
38 // choosing d such that it satisfies
39 // d*e = 1 + k * totient
40 int k = 2; // A constant value
41 double d = (1 + (k*phi))/e;
42 // Message to be encrypted
```

Terminal

```
Message data = 20.000000
Encrypted data = 20.000000
Original Message Sent = 20.000000
```

Advertisements:

- Galaxy A53 5G
- PRICE DROP
- Galaxy S22
- PRICE DROP

29°C Mostly sunny 15:25 09-10-2022

The screenshot displays the Online C Compiler interface. The left pane contains the source code for an RSA encryption and decryption program. The code includes comments for each step, such as calculating the totient, finding the private key 'd', and performing modular exponentiation. The right pane shows the terminal output, which confirms the encryption and decryption of the message '20.000000'.

```
33     break;
34     else
35         e++;
36 }
37 // Private key (d stands for decrypt)
38 // choosing d such that it satisfies
39 // d*e = 1 + k * totient
40 int k = 2; // A constant value
41 double d = (1 + (k*phi))/e;
42 // Message to be encrypted
43 double msg = 20;
44 printf("Message data = %lf", msg);
45 // Encryption c = (msg ^ e) % n
46 double c = pow(msg, e);
47 c = fmod(c, n);
48 printf("\nEncrypted data = %lf", c);
49 // Decryption m = (c ^ d) % n
50 double m = pow(c, d);
51 m = fmod(m, n);
52 printf("\nOriginal Message Sent = %lf", m);
53 return 0;
54 }
```

Terminal Output:

```
Message data = 20.000000
Encrypted data = 20.000000
Original Message Sent = 20.000000
```

CONCLUSION:

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

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:

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

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

Figure: Diffie Hellman Process

CODE:

```
from random import randint

if __name__ == '__main__':

    # Both the persons will be agreed upon the
    # public keys G and P
    # A prime number P is taken
    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-

```
The value of P : 23
The value of G : 9
The private key a for Lisha: 4
The private key b for Shewta: 3
Secret key for the Lisha is : 9
Secret key for the Shewta is : 9

...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION:

We have studied & implemented the Diffie Hellman key exchange algorithm.

Assignment No.5

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

THEORY:

SHA-1 Hash

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. To calculate cryptographic hashing value in Java, MessageDigest Class is used, under the package java.security. MessageDigest Class provides following cryptographic hash function to find hash value of a text as follows:

- MD2
- MD5
- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

These algorithms are initialized in a static method called getInstance(). After selecting the algorithm the message digest value is calculated and the results are returned as a byte array. BigInteger class is used to convert the resultant byte

array into its signum representation. This representation is then converted into a hexadecimal format to get the expected MessageDigest. Examples:

*Input : hello world Output : 2aae6c35c94fcfb415dbe95f408b9ce91ee846ed Input :
GeeksForGeeks Output : addf120b430021c36c232c99ef8d926aea2acd6b Below*

program shows the implementation of SHA-1 hash in Java.

CODE:

```
# Python 3 code to demonstrate  
# SHA hash algorithms.
```

```
import hashlib
```

```
# initializing string  
str = "GeeksforGeeks"
```

```
# encoding GeeksforGeeks using encode()  
# then sending to SHA256()  
result = hashlib.sha256(str.encode())
```

```
# printing the equivalent hexadecimal value.  
print("The hexadecimal equivalent of SHA256 is : ")  
print(result.hexdigest())
```

```
print ("\r")
```

```
# initializing string  
str = "GeeksforGeeks"
```

```
# encoding GeeksforGeeks using encode()  
# then sending to SHA384()  
result = hashlib.sha384(str.encode())
```

```
# printing the equivalent hexadecimal value.
```

```
print("The hexadecimal equivalent of SHA384 is : ")
print(result.hexdigest())
```

```
print ("\r")
```

```
# initializing string
str = "GeeksforGeeks"
```

```
# encoding GeeksforGeeks using encode()
# then sending to SHA224()
result = hashlib.sha224(str.encode())
```

```
# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA224 is : ")
print(result.hexdigest())
```

```
print ("\r")
```

```
# initializing string
str = "GeeksforGeeks"
```

```
# encoding GeeksforGeeks using encode()
# then sending to SHA512()
result = hashlib.sha512(str.encode())
```

```
# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA512 is : ")
print(result.hexdigest())
```

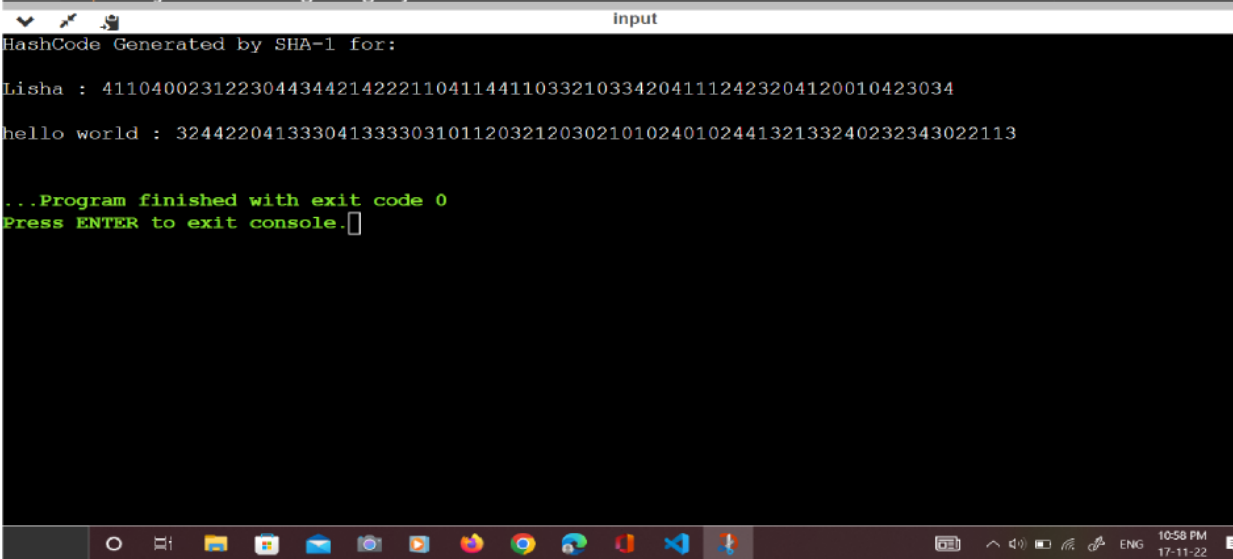
```
print ("\r")
```

```
# initializing string
str = "GeeksforGeeks"
```

```
# encoding GeeksforGeeks using encode()
# then sending to SHA1()
result = hashlib.sha1(str.encode())
```

```
# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA1 is : ")
print(result.hexdigest())
```

OUTPUT:



```
Input
HashCode Generated by SHA-1 for:
Lisha : 411040023122304434421422211041144110332103342041112423204120010423034
hello world : 32442204133304133330310112032120302101024010244132133240232343022113

...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION: Hence, we have successfully implemented the SHA-1 algorithm.