Name:Lisha Ingawale Roll No:2193140 Class: LY CSE IS 1

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 ≡ a1 mod m1

x ≡ a2 mod m2

x ≡ a3 mod m3

……

x ≡ ak mod mk

The Chinese Remainder Theorem states that the above equations have a 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 = m1 x m2 x m3…mk where M is common modulus Step II: Find M1 = M/m1, M2 = M/m2 and so on

Step III: Find multiplicative inverses for M1, M2 and so on

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

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

Example

X = 4 mod 5

X = 6 mod 8

X = 8 mod 9

Step I: M = 5 * 8* 9 = 360

Step II: M1 = M/m1 = 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)

q	r1	r2 r t1 t2	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. M1 inverse = -2 + 5 = 3

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

q	r1	r2 r t1 t2	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

M2 inverse = -3 + 8 = 5

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

q	r1	r2 r t1 t2	t
0	9	40 9 0 1	0
4	40	9 4 1 0	1
2	9	4 1 0 1	-2

M3 inverse = -2 + 9 = 7

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

X = (a1 x M1 x $M1^{-1}$ + a2 x M2 x $M2^{-1}$ + a3 x M3 x $M3^{-1}$) mod M X = (4*72*3 + 6*45*5 + 8*40*7) mod 360

X = (864 + 1350 + 2240) mod 360

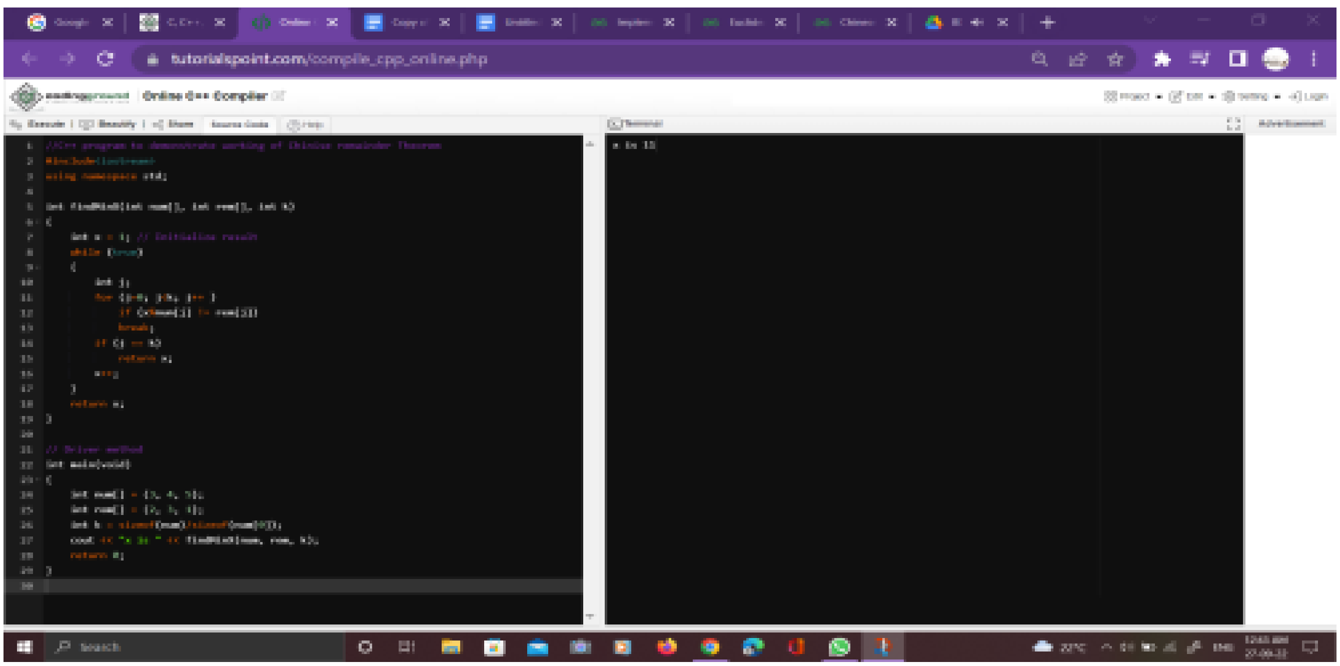
X = 4454 mod 360

X = 134

CODE:

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

// k is the 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)
int findMinX(int num[], int rem[], int k)
{
	int x = 1; // Initialize result

	// As per the Chinese remainder theorem,
	// this loop will always break.
	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;
}
```



Applications

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

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) 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.

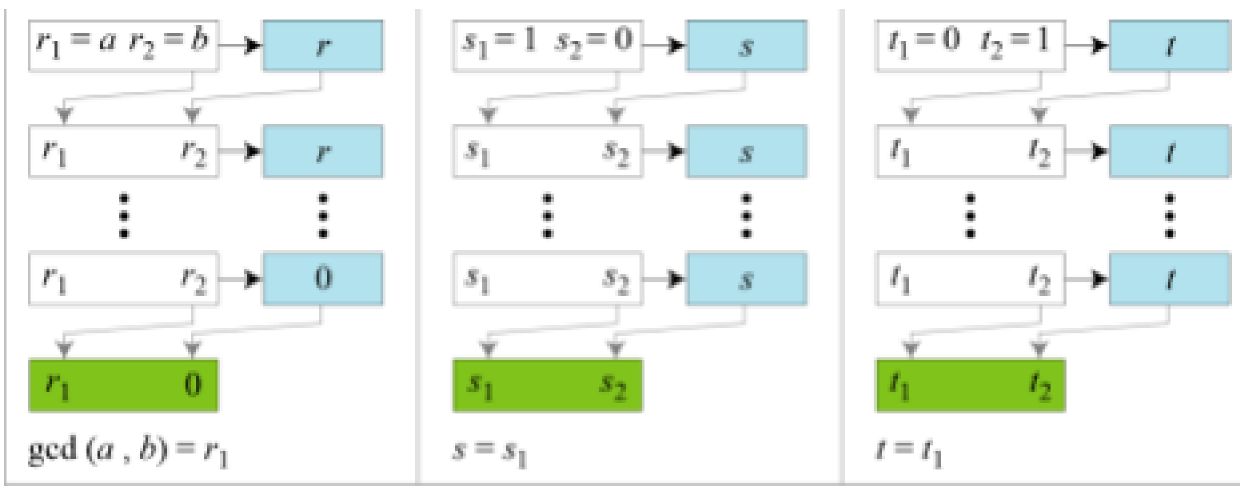


Figure: Extended Euclid's Algorithm Process

Algorithm:

Extend the algorithm to compute the integer coefficients x and y such that gcd (a, b) = ax + by

Extended-Euclid (a, b)

$$
\begin{aligned}
&r_1 \leftarrow a; \quad r_2 \leftarrow b; \\
&s_1 \leftarrow 1; \quad s_2 \leftarrow 0; \qquad \text{(Initialization)} \\
&t_1 \leftarrow 0; \quad t_2 \leftarrow 1;
\end{aligned}
$$

while $(r_2 > 0)$

{

$q \leftarrow r_1 / r_2;$

$$
\begin{aligned}
&r \leftarrow r_1 - q \times r_2; \\
&r_1 \leftarrow r_2;\ r_2 \leftarrow r; \qquad \text{(Updating } r\text{'s)}
\end{aligned}
$$

$$
\begin{aligned}
&s \leftarrow s_1 - q \times s_2; \\
&s_1 \leftarrow s_2;\ s_2 \leftarrow s; \qquad \text{(Updating } s\text{'s)}
\end{aligned}
$$

$$
\begin{aligned}
&t \leftarrow t_1 - q \times t_2; \\
&t_1 \leftarrow t_2;\ t_2 \leftarrow t; \qquad \text{(Updating } t\text{'s)}
\end{aligned}
$$

}

$\gcd(a, b) \leftarrow r_1;\ s \leftarrow s_1;\ t \leftarrow t_1$

Example:
GCD (161, 28)

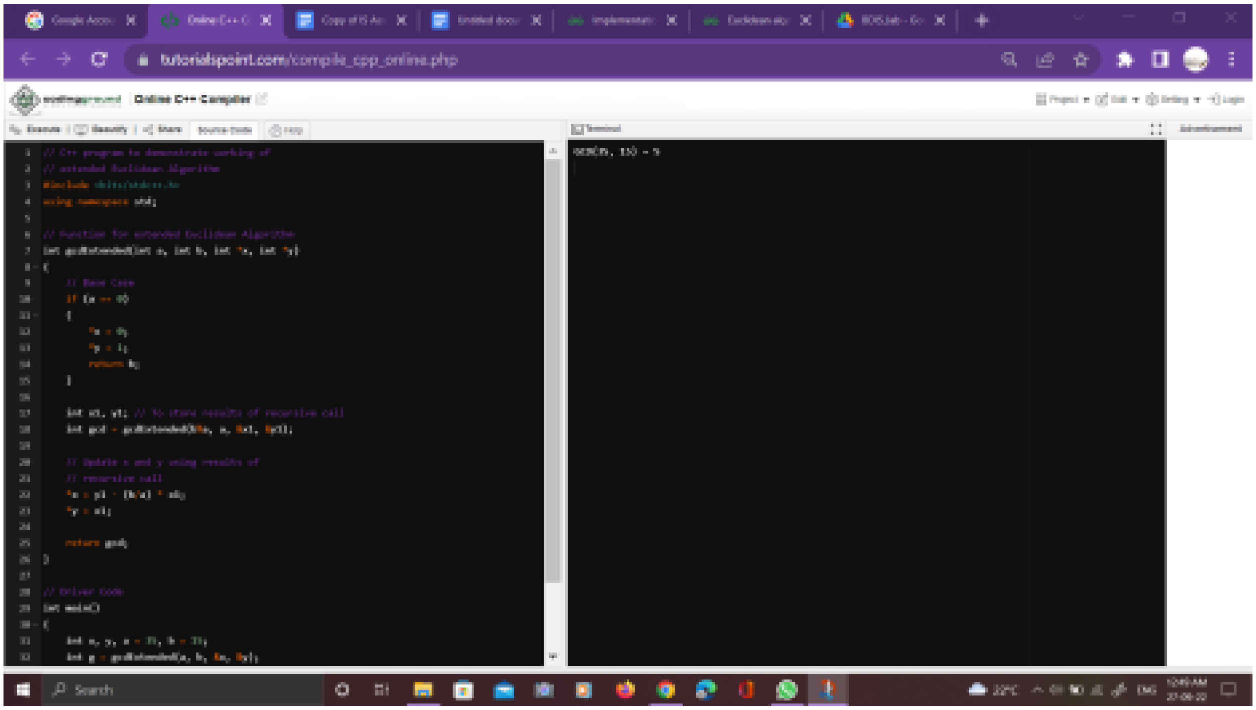
q	r1	r2	r s1 s2 s t1	t2	t
5	161	28	21 1 0 1 0	1	-5
1	28	21	7 0 1 -1 1	-5	6
3	21	7	0 1 -1 4 -5	6	-23
	7	0	-1 4 6	-23	

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

<u>CODE:</u>

```
// C++ program to demonstrate working of
```

```
// extended Euclidean Algorithm
#include <bits/stdc++.h>
using namespace std;

// 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 Code
int main()
{
	int x, y, a = 35, b = 15;
	int g = gcdExtended(a, b, &x, &y);
	cout << "GCD(" << a << ", " << b
		<< ") = " << g << endl;
	return 0;
}
```



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 a 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.

Bob's	Ted	Input
Public key Ring	Plaintext	Encryption
Joy	Alice Private Key	algorithm
Alice	Transmitted cipher	Decryption Output
Plaintext	text	algorithm

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 log2 (n); in practice the block size is I bits, where 2i<n<=2i+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 mod n and C^d mod 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≠q

Calculate n = p * q

Calculate Ø(n) = (p-1)(q-1)

Select integer e gcd(Ø(n),e) = 1; 1<e< Ø(n)

Calculate d d = $e^{(-1)}$ mod Ø(n)

Public key PU = {e, n}

Private key PR = {d, n}

Encryption

Plaintext M<n

Ciphertext C=M^e mod n

Decryption

Ciphertext C

Plaintext M = C^d mod 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 Ø(n) = (p-1)(q-1) = 16*10 = 160.
4. Select e such that it is relatively prime to Ø(n)=160 & less than Ø(n); we choose e = 7.
5. Determine d such that de ≡ 1 (mod 160) and d < 160. The correct value is d = 23; d can be calculated using Euclid's extended 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.

Encryption Decryption

Plaintext Step I: n = 3 * 11 = 33 mod 187 = 88

88 Step II: Ø(n) = 2 * 10 = 20 PU=7,187 PR=23,187

Ciphertext Figure: Example of

Example 2: 11 RSA. Plaintext 88

P = 7, Q = 13, M = 10. $88^{(7)}$ mod 187 = 11 $11^{(23)}$

Step III: Select e, such that gcd (Ø(n),e) = 1, gcd(20,3) = 1, So we can select e =3 Step IV: To calculate d, solve for gcd (Ø(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 mod n
= 2^3 mod 33
= 8 mod 33
= 8
Step VI: For Decryption,
M = C^d mod n
= 8^7 mod 33
= ((8^2 mod 33) (8^2 mod 33) (8^2 mod 33) (8^1 mod 33)) mod 33 = (31* 31 *31 *8) mod 33
= (961 mod 33) (248 mod 33) mod 33
= 68 mod 33
= 2

Advantages:

1. Easy to implement.

Disadvantages:

1. Anyone can announce the public key.

Algorithm:

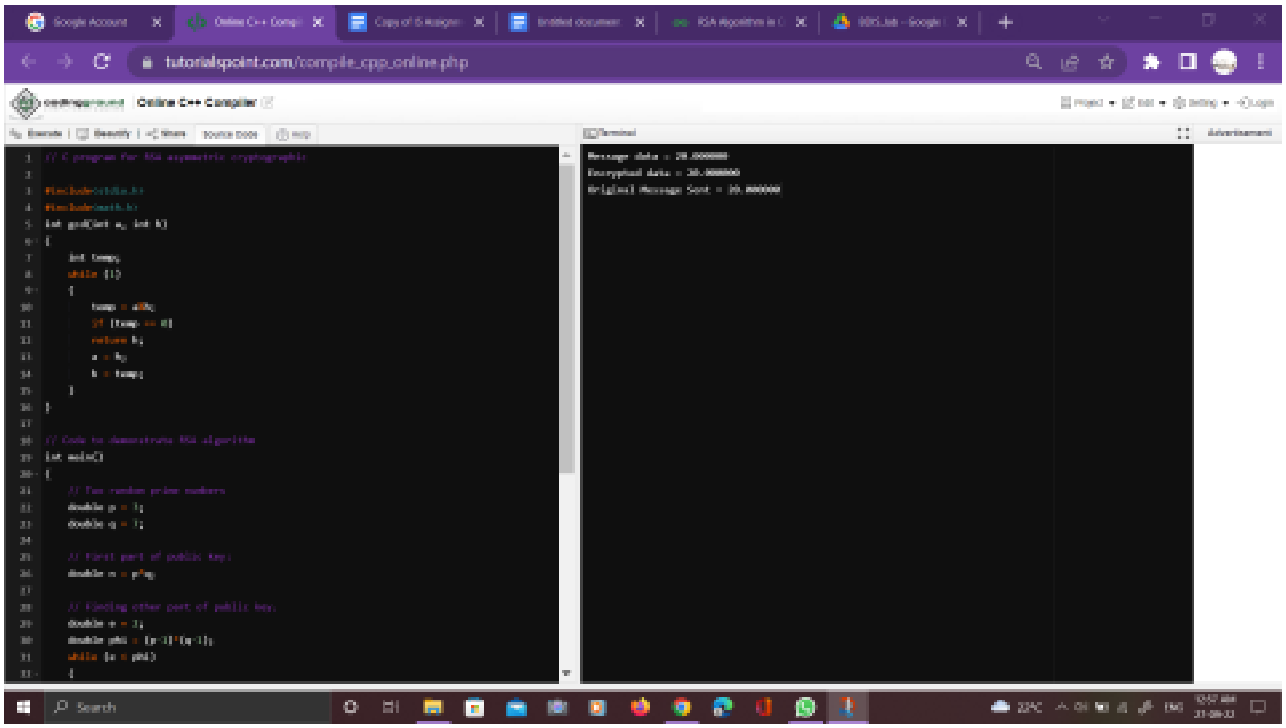
1. Start
2. Input two prime numbers p and q.
3. Calculate n = pq.
4. Calculate Ø(n) = (p-1)(q-1).
5. Input value of e.
6. Determine d.
7. Determine PU and PR.
8. Take input plaintext.
9. Encrypt the plaintext and show the output.
10. Stop.

CODE:

```
// C program for RSA asymmetric cryptographic
```

```c
#include<stdio.h>
#include<math.h>
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.
	double e = 2;
	double phi = (p-1)*(q-1);
	while (e < phi)
	{
		if (gcd(e, phi)==1)
			break;
		else
			e++;
	}
	int k = 2;
	double d = (1 + (k*phi))/e;
	double msg = 20;
	printf("Message data = %lf", msg);
```

```
double c = pow(msg, e);
c = fmod(c, n);
printf("\nEncrypted data = %lf", c);
double m = pow(c, d);
m = fmod(m, n);
printf("\nOriginal Message Sent = %lf", m);

return 0;
```




```
}
```

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 XA < q and computes YA = α^{X}A mod q. Similarly User B independently selects a random integer XB < q and computes YB = α^{X}B mod 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^{xA} mod q

User B computes the key K as K = Y_A^{xB} mod q

These two calculations produce identical results.

Eg.

1. Users Alice & Bob who wish to swap keys:
2. Agree on prime q=353 and α=3
3. Select random secret keys:

– A chooses XA=97, B chooses XB=233

4. Compute public keys:

– YA=3^{97} mod 353 = 40 (Alice)

– YB=3^{233} mod 353 = 248 (Bob)

5. Compute shared session key as:

KAB= Y_B^{xA} mod 353 = 248^{97} = 160 (Alice)

KAB= Y_A^{xB} mod 353 = 40^{233} = 160 (Bob)

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{XA} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{XB} \bmod q$

Calculation of Secret Key by User A

$K = (Y_B)^{XA} \bmod q$

Calculation of Secret Key by User B

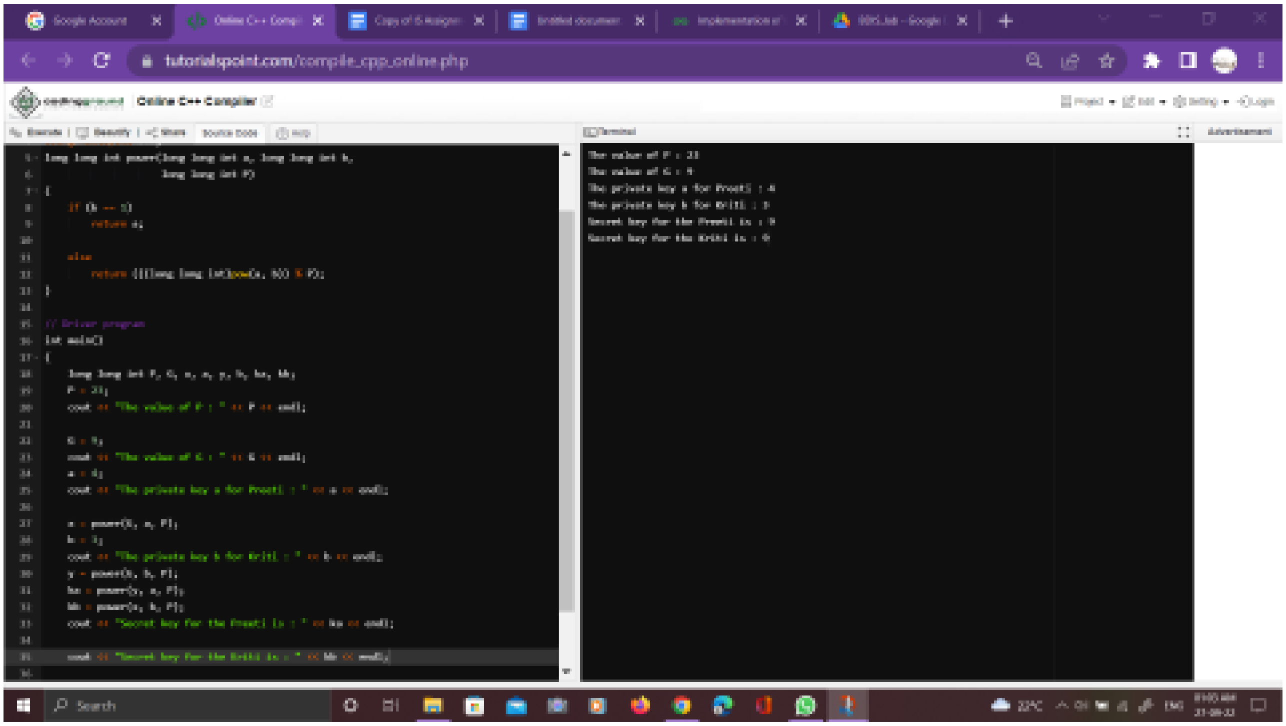
$K = (Y_A)^{XB} \bmod q$

Figure: Diffie Hellman Process

CODE:

```
/* the Diffie-Hellman Key exchange algorithm using C++ */
#include <cmath>
#include <iostream>
using namespace std;
long long int power(long long int a, long long int b,
                                        long long int P)
{
        if (b == 1)
                return a;
        else
```

```
        return (((long long int)pow(a, b)) % P);
}

// Driver program
int main()
{
    long long int P, G, x, a, y, b, ka, kb;
    P = 23;
    cout << "The value of P : " << P << endl;

    G = 9;
    cout << "The value of G : " << G << endl;
    a = 4;
    cout << "The private key a for Preeti : " << a << endl;

    x = power(G, a, P);
    b = 3;
    cout << "The private key b for Kriti : " << b << endl;
    y = power(G, b, P);
    ka = power(y, a, P);
    kb = power(x, b, P);
    cout << "Secret key for the Preeti is : " << ka <<

    endl; cout << "Secret key for the Kriti is : " << kb <<

    endl;

    return 0;
}
```



CONCLUSION:

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

Assignment No.5

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

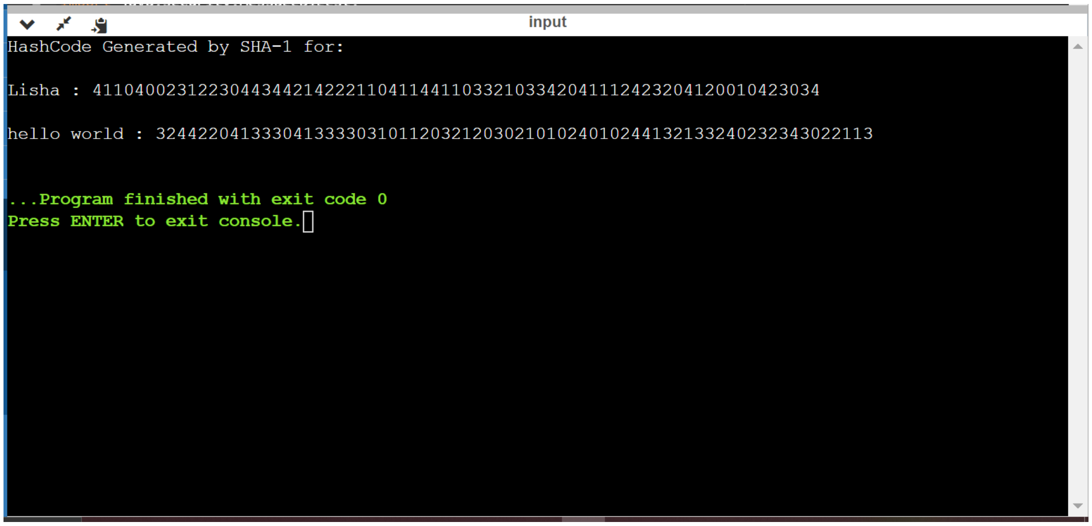
Below program shows the implementation of SHA-1 hash in Java.

Code:

```
// Java program to calculate SHA-1 hash value
```

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Main {
	public static String encryptThisString(String input)
	{
		try {
			// getInstance() method is called with algorithm SHA-1
			MessageDigest md = MessageDigest.getInstance("SHA-1");

			// digest() method is called
			// to calculate message digest of the input string
			// returned as array of byte
			byte[] messageDigest = md.digest(input.getBytes());

			// Convert byte array into signum representation
			BigInteger no = new BigInteger(1, messageDigest);

			// Convert message digest into hex value
			String hashtext = no.toString(5);

			// Add preceding 0s to make it 32 bit
			while (hashtext.length() < 32) {
				hashtext = "0" + hashtext;
			}

			// return the HashText
			return hashtext;
		}

		// For specifying wrong message digest algorithms
		catch (NoSuchAlgorithmException e) {
			throw new RuntimeException(e);
		}
	}

	// Driver code
	public static void main(String args[]) throws
											NoSuchAlgorithmException
	{
```

```
                System.out.println("HashCode Generated by SHA-1 for: ");

                String s1 = "Lisha";
                System.out.println("\n" + s1 + " : " + encryptThisString(s1));

                String s2 = "hello world";
                System.out.println("\n" + s2 + " : " + encryptThisString(s2));
        }
}
```



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