UNIVERSITY OF PETROLEUM & ENERGY STUDIES

College of Engineering Studies

Dehradun

Cryptography and Network Security

LAB FILE

Submitted By:

Name: RAHUL DHANOLA
Sap Id: 500075154
Roll no.: R177219139
Branch: B. Tech CSE AI&ML (2019-23)
Semester: VII
Batch: 4

EXPERIMENT NO – 1: Classical Encryption Techniques

Objective: - To understand the concept of passwords, Brute Force Techniques.

1. Develop a program to show the workings of substitution method.

```
// A C++ program to illustrate Substitution Method
#include <iostream>
using namespace std;

// This function receives text and shift and
// returns the encrypted text
string encrypt(string text, int s)
{
	string result = "";

	// traverse text
	for (int i = 0; i < text.length(); i++) {
		// apply transformation to each character
		// Encrypt Uppercase letters
		if (isupper(text[i]))
			result += char(int(text[i] + s - 65) % 26 + 65);

		// Encrypt Lowercase letters
		else
			result += char(int(text[i] + s - 97) % 26 + 97);
	}

	// Return the resulting string
	return result;
}

// Driver program to test the above function
int main()
{
	string text = "ATTACKATONCE";
	int s = 4;
	cout << "Text : " << text;
	cout << "\nShift: " << s;
	cout << "\nSubstitution: " << encrypt(text, s);
	return 0;
}
```

```
Text : ATTACKATONCE
Shift: 4
Substitution: EXXEGOEXSRGI
```

EXPERIMENT NO – 2: Shift Cipher Techniques

Objective: - To understand the concept of Shift Ciphers.

1. Implement a program to show the working of Caesar cipher.

```
// A C++ program to illustrate Caesar Cipher Technique
#include <iostream>
using namespace std;

// This function receives text and shift and
// returns the encrypted text
string encrypt(string text, int s)
{
	string result = "";

	// traverse text
	for (int i = 0; i < text.length(); i++) {
		// apply transformation to each character
		// Encrypt Uppercase letters
		if (isupper(text[i]))
			result += char(int(text[i] + s - 65) % 26 + 65);

		// Encrypt Lowercase letters
		else
			result += char(int(text[i] + s - 97) % 26 + 97);
	}

	// Return the resulting string
	return result;
}

// Driver program to test the above function
int main()
{
	string text = "ATTACKATONCE";
	int s = 4;
	cout << "Text : " << text;
	cout << "\nShift: " << s;
```

```
(base) PS C:\Users\HP\OneDrive\Desktop\LAB> cd "c:\Users\HP\OneDrive\Desktop\LAB\" ; if ($?) { g++ TEMP.cpp -o TEMP } ; if ($?) { .\TEMP }
Text : ATTACKATONCE
Shift: 4
Cipher: EXXEGOEXSRGI
```

```
	cout << "\nCipher: " << encrypt(text, s);
	return 0;
}
```

2. Implement a program to show the working of the Vigenère cipher.

```
// C++ code to implement Vigenere Cipher
#include<bits/stdc++.h>
using namespace std;
```

```
// This function generates the key in
// a cyclic manner until it's length isi'nt
// equal to the length of original text
string generateKey(string str, string key)
{
	int x = str.size();

	for (int i = 0; ; i++)
	{
		if (x == i)
			i = 0;
		if (key.size() == str.size())
			break;
		key.push_back(key[i]);
	}
	return key;
}

// This function returns the encrypted text
// generated with the help of the key
string cipherText(string str, string key)
{
	string cipher_text;

	for (int i = 0; i < str.size(); i++)
	{
		// converting in range 0-25
		char x = (str[i] + key[i]) %26;

		// convert into alphabets(ASCII)
		x += 'A';

		cipher_text.push_back(x);
	}
	return cipher_text;
}

// This function decrypts the encrypted text
// and returns the original text
string originalText(string cipher_text, string key)
{
	string orig_text;

	for (int i = 0 ; i < cipher_text.size(); i++)
	{
		// converting in range 0-25
		char x = (cipher_text[i] - key[i] + 26) %26;

		// convert into alphabets(ASCII)
		x += 'A';
		orig_text.push_back(x);
	}
```

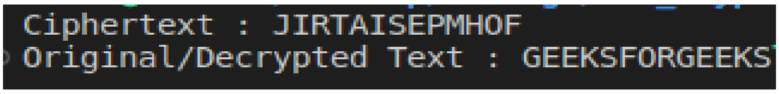
```
	return orig_text;
}

// Driver program to test the above function
int main()
{
	string str = "GEEKSFORGEEKS";
	string keyword = "DENJI";

	string key = generateKey(str, keyword);
	string cipher_text = cipherText(str, key);

	cout << "Ciphertext : "
		<< cipher_text << "\n";

	cout << "Original/Decrypted Text : "
		<< originalText(cipher_text, key);
	return 0;
}
```



EXPERIMENT NO – 3: Polyalphabetic Cipher Techniques

Objective: - To understand the concept of Polyalphabetic Ciphers.

1. Implement the polyalphabetic cipher techniques.

```
// C++ code to implement Vigenere Cipher
#include<bits/stdc++.h>
using namespace std;

// This function generates the key in
// a cyclic manner until it's length isi'nt
// equal to the length of original text
string generateKey(string str, string key)
{
	int x = str.size();

	for (int i = 0; ; i++)
	{
		if (x == i)
			i = 0;
		if (key.size() == str.size())
			break;
		key.push_back(key[i]);
	}
	return key;
}

// This function returns the encrypted text
// generated with the help of the key
```

```
string cipherText(string str, string key)
{
	string cipher_text;

	for (int i = 0; i < str.size(); i++)
	{
		// converting in range 0-25
		char x = (str[i] + key[i]) %26;

		// convert into alphabets(ASCII)
		x += 'A';

		cipher_text.push_back(x);
	}
	return cipher_text;
}

// This function decrypts the encrypted text
// and returns the original text
string originalText(string cipher_text, string key)
{
	string orig_text;

	for (int i = 0 ; i < cipher_text.size(); i++)
	{
		// converting in range 0-25
		char x = (cipher_text[i] - key[i] + 26) %26;

		// convert into alphabets(ASCII)
		x += 'A';
		orig_text.push_back(x);
	}
	return orig_text;
}

// Driver program to test the above function
int main()
{
	string str = "GEEKSFORGEEKS";
	string keyword = "DENJI";

	string key = generateKey(str, keyword);
	string cipher_text = cipherText(str, key);

	cout << "Ciphertext : "
		<< cipher_text << "\n";

	cout << "Original/Decrypted Text : "
		<< originalText(cipher_text, key);
	return 0;
}
```

```
Ciphertext : JIRTAISEPMHOF
Original/Decrypted Text : GEEKSFORGEEKS
```

EXPERIMENT NO – 4: Euclidean algorithms

Objective: - To understand the concept of Euclidean algorithms (Basic and Extended).

1. Implement the Euclidean algorithms.

```
// C++ program to demonstrate
// Basic Euclidean Algorithm

#include <bits/stdc++.h>
using namespace std;

// Function to return
// gcd of a and b
int gcd(int a, int b)
{
	if (a == 0)
		return b;
	return gcd(b % a, a);
}

// Driver Code
int main()
{
	int a = 10, b = 15;

	// Function call
	cout << "GCD(" << a << ", " << b << ") = " << gcd(a, b)
		<< endl;
	a = 35, b = 10;
	cout << "GCD(" << a << ", " << b << ") = " << gcd(a, b)
		<< endl;
	a = 31, b = 2;
	cout << "GCD(" << a << ", " << b << ") = " << gcd(a, b)
		<< endl;
	return 0;

}
```

```
GCD(10, 15) = 5
GCD(35, 10) = 5
GCD(31, 2) = 1
```

EXPERIMENT NO-5: DES Encryption Techniques

Objective: - To understand the concept of Block Ciphers

1. Implement the Data Encryption Standards.

```
// C++ code for the above approach

#include <bits/stdc++.h>
using namespace std;
string hex2bin(string s)
{
	// hexadecimal to binary conversion
	unordered_map<char, string> mp;
	mp['0'] = "0000"; mp['1'] = "0001"; mp['2'] = "0010"; mp['3'] = "0011"; mp['4'] = "0100";
mp['5'] = "0101";
   mp['6'] = "0110"; mp['7'] = "0111"; mp['8'] = "1000"; mp['9'] = "1001"; mp['A'] = "1010"; mp['B']
= "1011";
   mp['C'] = "1100"; mp['D'] = "1101"; mp['E'] = "1110"; mp['F'] = "1111";
	string bin = "";
	for (int i = 0; i < s.size(); i++) {
		bin += mp[s[i]];
	}
	return bin;
}
string bin2hex(string s)
{
	// binary to hexadecimal conversion
	unordered_map<string, string> mp;
	mp["0000"] = "0"; mp["0001"] = "1"; mp["0010"] = "2"; mp["0011"] = "3"; mp["0100"] =
"4"; mp["0101"] = "5";
   mp["0110"] = "6"; mp["0111"] = "7"; mp["1000"] = "8"; mp["1001"] = "9"; mp["1010"] = "A";
mp["1011"] = "B";
   mp["1100"] = "C"; mp["1101"] = "D"; mp["1110"] = "E"; mp["1111"] = "F";
	string hex = "";
	for (int i = 0; i < s.length(); i += 4) {
		string ch = "";
		ch += s[i];
		ch += s[i + 1];
		ch += s[i + 2];
		ch += s[i + 3];
		hex += mp[ch];
	}
	return hex;
}

string permute(string k, int* arr, int n)
{
	string per = "";
	for (int i = 0; i < n; i++) {
		per += k[arr[i] - 1];
	}
```

```
	return per;
}

string shift_left(string k, int shifts)
{
	string s = "";
	for (int i = 0; i < shifts; i++) {
		for (int j = 1; j < 28; j++) {
			s += k[j];
		}
		s += k[0];
		k = s;
		s = "";
	}
	return k;
}

string xor_(string a, string b)
{
	string ans = "";
	for (int i = 0; i < a.size(); i++) {
		if (a[i] == b[i]) {
			ans += "0";
		}
		else {
			ans += "1";
		}
	}
	return ans;
}
string encrypt(string pt, vector<string> rkb,
			vector<string> rk)
{
	// Hexadecimal to binary
	pt = hex2bin(pt);

	// Initial Permutation Table
	int initial_perm[64]
		= { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44,
			36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22,
			14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57,
			49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35,
			27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13,
			5, 63, 55, 47, 39, 31, 23, 15, 7 };
	// Initial Permutation
	pt = permute(pt, initial_perm, 64);
	cout << "After initial permutation: " << bin2hex(pt)
		<< endl;

	// Splitting
	string left = pt.substr(0, 32);
	string right = pt.substr(32, 32);
```

```
    cout << "After splitting: L0=" << bin2hex(left)
        << " R0=" << bin2hex(right) << endl;

    // Expansion D-box Table
    int exp_d[48]
        = { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,
            8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17,
            16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
            24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1 };

    // S-box Table
    int s[8][4][16] = {
        { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5,9, 0, 7, 0, 15, 7, 4, 14, 2, 13, 1, 10, 6,12, 11,
9, 5, 3, 8, 4, 1, 14, 8, 13, 6,
  2,11, 15, 12, 9, 7, 3, 10, 5, 0, 15, 12, 8, 2,4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 },
  { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12,0, 5, 10, 3, 13, 4, 7, 15, 2, 8, 14, 12, 0,1, 10, 6, 9, 11, 5,
0, 14, 7, 11, 10, 4,
   13,1, 5, 8, 12, 6, 9, 3, 2, 15, 13, 8, 10, 1,3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 },

        { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12,7, 11, 4, 2, 8, 13, 7, 0, 9, 3, 4,6, 10, 2, 8, 5, 14, 12,
11, 15, 1, 13,6, 4, 9, 8, 15, 3,
   0, 11, 1, 2, 12,5, 10, 14, 7, 1, 10, 13, 0, 6, 9, 8,7, 4, 15, 14, 3, 11, 5, 2, 12 },
   { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11,12, 4, 15, 13, 8, 11, 5, 6, 15, 0, 3, 4, 7,2, 12, 1, 10, 14, 9,
10, 6, 9, 0, 12, 11,
   7,13, 15, 1, 3, 14, 5, 2, 8, 4, 3, 15, 0, 6,10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 },
   { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13,0, 14, 9, 14, 11, 2, 12, 4, 7, 13, 1, 5, 0,15, 10, 3, 9, 8, 6,
4, 2, 1, 11, 10, 13,
   7,8, 15, 9, 12, 5, 6, 3, 0, 14, 11, 8, 12, 7,1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 },
   { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14,7, 5, 11, 10, 15, 4, 2, 7, 12, 9, 5, 6, 1,13, 14, 0, 11, 3, 8,
9, 14, 15, 5, 2, 8,
   12,3, 7, 0, 4, 10, 1, 13, 11, 6, 4, 3, 2, 12,9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 },
   { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5,10, 6, 1, 13, 0, 11, 7, 4, 9, 1, 10, 14, 3,5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3,
   7,14, 10, 15, 6, 8, 0, 5, 9, 2, 6, 11, 13, 8,1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 },
   { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5,0, 12, 7, 1, 15, 13, 8, 10, 3, 7, 4, 12, 5,6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12,
    14,2, 0, 6, 10, 13, 15, 3, 5, 8, 2, 1, 14, 7,4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 }
    };

    // Straight Permutation Table
    int per[32]
        = { 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23,
            26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27,
            3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

    cout << endl;
    for (int i = 0; i < 16; i++) {
        // Expansion D-box
        string right_expanded = permute(right, exp_d, 48);

        // XOR RoundKey[i] and right_expanded
        string x = xor_(rkb[i], right_expanded);
```

```
        // S-boxes
        string op = "";
        for (int i = 0; i < 8; i++) {
            int row = 2 * int(x[i * 6] - '0')
                      + int(x[i * 6 + 5] - '0');
            int col = 8 * int(x[i * 6 + 1] - '0')
                      + 4 * int(x[i * 6 + 2] - '0')
                      + 2 * int(x[i * 6 + 3] - '0')
                      + int(x[i * 6 + 4] - '0');
            int val = s[i][row][col];
            op += char(val / 8 + '0');
            val = val % 8;
            op += char(val / 4 + '0');
            val = val % 4;
            op += char(val / 2 + '0');
            val = val % 2;
            op += char(val + '0');
        }
        // Straight D-box
        op = permute(op, per, 32);

        // XOR left and op
        x = xor_(op, left);

        left = x;

        // Swapper
        if (i != 15) {
            swap(left, right);
        }
        cout << "Round " << i + 1 << " " << bin2hex(left)
            << " " << bin2hex(right) << " " << rk[i]
            << endl;
    }

    // Combination
    string combine = left + right;

    // Final Permutation Table
    int final_perm[64]
        = { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,
            15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22,
            62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36,
            4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11,
            51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58,
            26, 33, 1, 41, 9, 49, 17, 57, 25 };

    // Final Permutation
    string cipher
        = bin2hex(permute(combine, final_perm, 64));
    return cipher;
}
```

```
// Driver code
int main()
{
	// pt is plain text
	string pt, key;
	/*cout<<"Enter plain text(in hexadecimal): ";
	cin>>pt;
	cout<<"Enter key(in hexadecimal): ";
	cin>>key;*/

	pt = "123456ABCD132536";
	key = "AABB09182736CCDD";
	// Key Generation

	// Hex to binary
	key = hex2bin(key);

	// Parity bit drop table
	int keyp[56]
		= { 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34,
			26, 18, 10, 2, 59, 51, 43, 35, 27, 19, 11, 3,
			60, 52, 44, 36, 63, 55, 47, 39, 31, 23, 15, 7,
			62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37,
			29, 21, 13, 5, 28, 20, 12, 4 };

	// getting 56 bit key from 64 bit using the parity bits
	key = permute(key, keyp, 56); // key without parity

	// Number of bit shifts
	int shift_table[16] = { 1, 1, 2, 2, 2, 2, 2, 2,
							1, 2, 2, 2, 2, 2, 2, 1 };

	// Key- Compression Table
	int key_comp[48] = { 14, 17, 11, 24, 1, 5, 3, 28,
						15, 6, 21, 10, 23, 19, 12, 4,
						26, 8, 16, 7, 27, 20, 13, 2,
						41, 52, 31, 37, 47, 55, 30, 40,
						51, 45, 33, 48, 44, 49, 39, 56,
						34, 53, 46, 42, 50, 36, 29, 32 };

	// Splitting
	string left = key.substr(0, 28);
	string right = key.substr(28, 28);

	vector<string> rkb; // rkb for RoundKeys in binary
	vector<string> rk; // rk for RoundKeys in hexadecimal
	for (int i = 0; i < 16; i++) {
		// Shifting
		left = shift_left(left, shift_table[i]);
		right = shift_left(right, shift_table[i]);
```

```
            // Combining
            string combine = left + right;

            // Key Compression
            string RoundKey = permute(combine, key_comp, 48);

            rkb.push_back(RoundKey);
            rk.push_back(bin2hex(RoundKey));
        }

        cout << "\nEncryption:\n\n";
        string cipher = encrypt(pt, rkb, rk);
        cout << "\nCipher Text: " << cipher << endl;

        cout << "\nDecryption\n\n";
        reverse(rkb.begin(), rkb.end());
        reverse(rk.begin(), rk.end());
        string text = encrypt(cipher, rkb, rk);
        cout << "\nPlain Text: " << text << endl;
}
```

```
Encryption:

After initial permutation: 14A7D67818CA18AD
After splitting: L0=14A7D678 R0=18CA18AD

Round 1 18CA18AD 5A78E394 194CD072DE8C
Round 2 5A78E394 4A1210F6 4568581ABCCE
Round 3 4A1210F6 B8089591 06EDA4ACF5B5
Round 4 B8089591 236779C2 DA2D032B6EE3
Round 5 236779C2 A15A4B87 69A629FEC913
Round 6 A15A4B87 2E8F9C65 C1948E87475E
Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0
Round 8 A9FC20A3 308BEE97 34F822F0C66D
Round 9 308BEE97 10AF9D37 84BB4473DCCC
Round 10 10AF9D37 6CA6CB20 02765708B5BF
Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5
Round 12 FF3C485F 22A5963B C2C1E96A4BF3
Round 13 22A5963B 387CCDAA 99C31397C91F
Round 14 387CCDAA BD2DD2AB 251B8BC717D0
Round 15 BD2DD2AB CF26B472 3330C5D9A36D
Round 16 19BA9212 CF26B472 181C5D75C66D

Cipher Text: C0B7A8D05F3A829C

Decryption

After initial permutation: 19BA9212CF26B472
After splitting: L0=19BA9212 R0=CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D
Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D
Round 3 387CCDAA 22A5963B 251B8BC717D0
Round 4 22A5963B FF3C485F 99C31397C91F
Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3
Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5
Round 7 10AF9D37 308BEE97 02765708B5BF
Round 8 308BEE97 A9FC20A3 84BB4473DCCC
Round 9 A9FC20A3 2E8F9C65 34F822F0C66D
Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0
Round 11 A15A4B87 236779C2 C1948E87475E
Round 12 236779C2 B8089591 69A629FEC913
Round 13 B8089591 4A1210F6 DA2D032B6EE3
Round 14 4A1210F6 5A78E394 06EDA4ACF5B5
Round 15 5A78E394 18CA18AD 4568581ABCCE
Round 16 14A7D678 18CA18AD 194CD072DE8C

Plain Text: 123456ABCD132536
```

EXPERIMENT-7: Public key Cryptography:

Objective: - To understand the concept of secret key, cipher and plain text.

1. Design a system, which will demonstrate the working of RSA public key cryptography.

```
#include <stdio.h>
#include <math.h>
using namespace std;

// 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*e = 1 + k * totient
	int k = 2; // A constant value
	double d = (1 + (k * phi)) / e;

	// Message to be encrypted
```

```
	double msg = 12;

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

	// Encryption c = (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;
}
```

```
Message data = 12.000000
Encrypted data = 3.000000
```

EXPERIMENT-8: Diffie Hellman Key Exchange Algorithm:

Objective: - To understand the concept of exchanging keys through Diffie Hellman.

1. Design a system, which will demonstrate the working of Diffie Hellman.

```
/* This program calculates the Key for two persons
using the Diffie-Hellman Key exchange algorithm using C++ */
#include <cmath>
#include <iostream>
using namespace std;

// Power function to return value of a ^ b mod P
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;

	// Both the persons will be agreed upon the
	// public keys G and P
	P = 23; // A prime number P is taken
	cout << "The value of P : " << P << endl;
```

```
        G = 9; // A primitive root for P, G is taken
        cout << "The value of G : " << G << endl;

        // Alice will choose the private key a
        a = 4; // a is the chosen private key
        cout << "The private key a for Alice : " << a << endl;

        x = power(G, a, P); // gets the generated key

        // Bob will choose the private key b
        b = 3; // b is the chosen private key
        cout << "The private key b for Bob : " << b << endl;

        y = power(G, b, P); // gets the generated key

        // Generating the secret key after the exchange
        // of keys
        ka = power(y, a, P); // Secret key for Alice
        kb = power(x, b, P); // Secret key for Bob
        cout << "Secret key for the Alice is : " << ka << endl;

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

        return 0;
}
```

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

Experiment No 9: Hash Function

Objective: - To understand the concept of Integrity, Non-repudiation and message digest.

1. Write a program to demonstrate the working of SHA-512.

```
#include "SHA512.h"
#include <stdio.h>
#include <string>
#include <string.h>
#include <iostream>
#include <iomanip>
#include <cstdint>
#include <sstream>

typedef unsigned long long uint64;

/**
```

```
 * SHA512 class constructor
 */
SHA512::SHA512(){
}

/**
 * SHA512 class destructor
 */
SHA512::~SHA512(){
}

/**
 * Returns a message digest using the SHA512 algorithm
 * @param input message string used as an input to the SHA512 algorithm, must be < size_t bits
 */
std::string SHA512::hash(const std::string input){
	size_t nBuffer; // amount of message blocks
	uint64** buffer; // message block buffers (each 1024-bit = 16 64-bit words)
	uint64* h = new uint64[HASH_LEN]; // buffer holding the message digest (512-bit = 8 64-
bit words)

	buffer = preprocess((unsigned char*) input.c_str(), nBuffer);
	process(buffer, nBuffer, h);

	freeBuffer(buffer, nBuffer);
	return digest(h);
}

/**
 * Preprocessing of the SHA512 algorithm
 * @param input message in byte representation
 * @param nBuffer amount of message blocks
 */
uint64** SHA512::preprocess(const unsigned char* input, size_t &nBuffer){
	// Padding: input || 1 || 0*k || l (in 128-bit representation)
	size_t mLen = strlen((const char*) input);
	size_t l = mLen * CHAR_LEN_BITS; // length of input in bits
	size_t k = (896-1-l) % MESSAGE_BLOCK_SIZE; // length of zero bit padding (l + 1 + k =
896 mod 1024)
	nBuffer = (l+1+k+128) / MESSAGE_BLOCK_SIZE;

	uint64** buffer = new uint64*[nBuffer];

	for(size_t i=0; i<nBuffer; i++){
		buffer[i] = new uint64[SEQUENCE_LEN];
	}

	uint64 in;
	size_t index;

	// Either copy existing message, add 1 bit or add 0 bit
	for(size_t i=0; i<nBuffer; i++){
```

```
		for(size_t j=0; j<SEQUENCE_LEN; j++){
			in = 0x0ULL;
			for(size_t k=0; k<WORD_LEN; k++){
				index = i*128+j*8+k;
				if(index < mLen){
					in = in<<8 | (uint64)input[index];
				}else if(index == mLen){
					in = in<<8 | 0x80ULL;
				}else{
					in = in<<8 | 0x0ULL;
				}
			}
			buffer[i][j] = in;
		}
	}

	// Append the length to the last two 64-bit blocks
	appendLen(l, buffer[nBuffer-1][SEQUENCE_LEN-1], buffer[nBuffer-
1][SEQUENCE_LEN-2]);
	return buffer;
}

/**
 * Processing of the SHA512 algorithm
 * @param buffer array holding the preprocessed
 * @param nBuffer amount of message blocks
 * @param h array of output message digest
 */
void SHA512::process(uint64** buffer, size_t nBuffer, uint64* h){
	uint64 s[WORKING_VAR_LEN];
	uint64 w[MESSAGE_SCHEDULE_LEN];

	memcpy(h, hPrime, WORKING_VAR_LEN*sizeof(uint64));

	for(size_t i=0; i<nBuffer; i++){
		// copy over to message schedule
		memcpy(w, buffer[i], SEQUENCE_LEN*sizeof(uint64));

		// Prepare the message schedule
		for(size_t j=16; j<MESSAGE_SCHEDULE_LEN; j++){
			w[j] = w[j-16] + sig0(w[j-15]) + w[j-7] + sig1(w[j-2]);
		}
		// Initialize the working variables
		memcpy(s, h, WORKING_VAR_LEN*sizeof(uint64));

		// Compression
		for(size_t j=0; j<MESSAGE_SCHEDULE_LEN; j++){
			uint64 temp1 = s[7] + Sig1(s[4]) + Ch(s[4], s[5], s[6]) + k[j] + w[j];
			uint64 temp2 = Sig0(s[0]) + Maj(s[0], s[1], s[2]);

			s[7] = s[6];
			s[6] = s[5];
```

```
                s[5] = s[4];
                s[4] = s[3] + temp1;
                s[3] = s[2];
                s[2] = s[1];
                s[1] = s[0];
                s[0] = temp1 + temp2;
            }

            // Compute the intermediate hash values
            for(size_t j=0; j<WORKING_VAR_LEN; j++){
                h[j] += s[j];
            }
        }

}

/**
 * Appends the length of the message in the last two message blocks
 * @param l message size in bits
 * @param lo pointer to second last message block
 * @param hi pointer to last message block
 */
void SHA512::appendLen(size_t l, uint64& lo, uint64& hi){
        lo = l;
        hi = 0x00ULL;
}

/**
 * Outputs the final message digest in hex representation
 * @param h array of output message digest
 */
std::string SHA512::digest(uint64* h){
        std::stringstream ss;
        for(size_t i=0; i<OUTPUT_LEN; i++){
            ss << std::hex << std::setw(16) << std::setfill('0') << h[i];
        }
        delete[] h;
        return ss.str();
}

/**
 * Free the buffer correctly
 * @param buffer array holding the preprocessed
 * @param nBuffer amount of message blocks
 */
void SHA512::freeBuffer(uint64** buffer, size_t nBuffer){
        for(size_t i=0; i<nBuffer; i++){
            delete[] buffer[i];
        }

        delete[] buffer;
}
```

```
int main(int argc, char *argv[]){

	SHA512 sha512;

	std::stringstream ss;
	ss << argv[1];

	std::cout << "SHA512:" << sha512.hash(ss.str()) << std::endl;

	return 0;
}
```

```
SHA512:cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327af927da3e
```

E.O.F