

**NETAJI SUBHAS UNIVERSITY**

**OF TECHNOLOGY**

**INFORMATION SECURITY**

ABHISHEK KUMAR SAH

2019UIT3001

**IT-1**

**INDEX**

|  |  |
| --- | --- |
| **S. NO** | **PRACTICAL** |
| **1.** | **Study of the features of firewall in providing network security and to set Firewall**  **security in windows. Students should know the following:**  **a) Know how to setup and configure a firewall on Operating System.**  **b) Know about the Windows Firewall with Advanced Security.**  **c) Know the Connection Security Rules**  **d) Know How to Start & Use the Windows Firewall with Advanced Security** |
| **2.** | **Demonstrate how to provide secure data storage, secure data transmission and for creating digital signatures Using tool GnuPG. (Download GPG4Win Tool). Create your public and private keys using Kloepatra Certificate management software. Check encryption –decryption of an email sent to you.** |
| **3.** | **Implement MD5 Algorithm. Take all the constants from the Figure attached implement in Python** |
| **4.** | **Implement SHA-256 Algorithm. Consider all the Constants and Tables as given in the text book.** |
| **5.** | **Plot an elliptic curve over finite field. Check whether the points lies on the elliptic curve or not.** |
| **6.** | **Perform the Elliptic Curve operations like Addition and Multiplication of two points and find the Inverse of a point also.** |
| **7.** | **Implement RSA Digital Signature Scheme.** |
| **8.** | **Implement Elgamal Digital Signature Scheme**. |
| **9.** | **Implement Schnorr Digital Signature Scheme** |
| **10.** | **Using Jcrypt tool (or any other equivalent) to demonstrate asymmetric, symmetric crypto algorithm** |
| **11.** | **Implement the Identity-based Encryption (IBE). Use the email address of the recipient to generate the key for a destination.** |
| **12.** | **To study and work with KF SENSOR Intrusion Detection Tool. Setup a honeypot and monitor the honeypot on the network.** |
| **13.** | **Configure Wireshark with a key to let you look inside encrypted SSL messages. You can read on the web how to do this. Once decrypted, you will be able to observe the HTTP protocol running on top of SSL, as well as the details of other SSL messages such as Alerts** |
| **14.** | **To build a Trojan and know the harmness of the trojan malwares in a computer system. When the trojan code executes, it will open MS-Paint, Notepad, Command Prompt, Explorer, calculator, infinitely. Note: Use Vmware to perform this experiment** |
| **15.** | **To work with Snort tool to demonstrate Intrusion Detection System. Download SNORT from snort.org.** |
| **16.** | **Implement a code to simulate buffer overflow attack.** |
| **17.** | **Use the Nessus tool to scan the network for vulnerabilities** |

**STUDY OF THE FEATURES OF FIREWALL IN PROVIDING NETWORK SECURITY AND TO SET FIREWALL SECURITY IN WINDOWS. STUDENTS SHOULD KNOW THE FOLLOWING:**

a) Know how to setup and configure a firewall on Operating System.

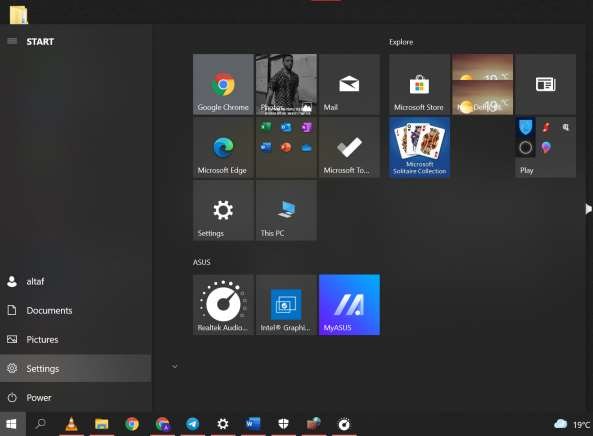
b) Know about the Windows Firewall with Advanced Security.

c) Know the Connection Security Rules.

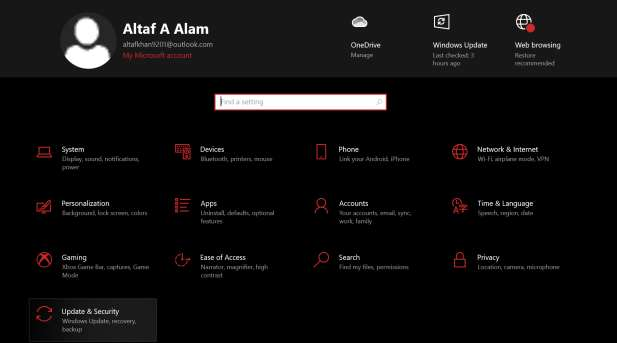
d) Know How to Start & Use the Windows Firewall with Advanced Security.

* **How to step and configure a firewall.**

1) Click on start button in windows 10 and then open up settings.



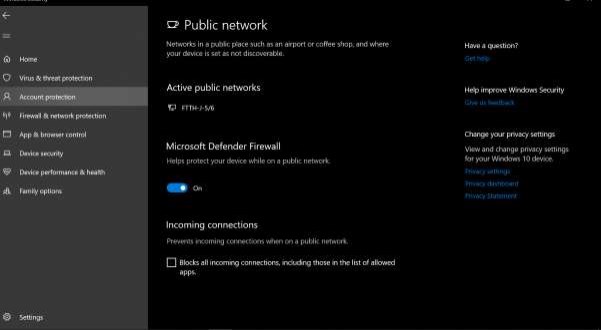
2) Click on windows update and security.



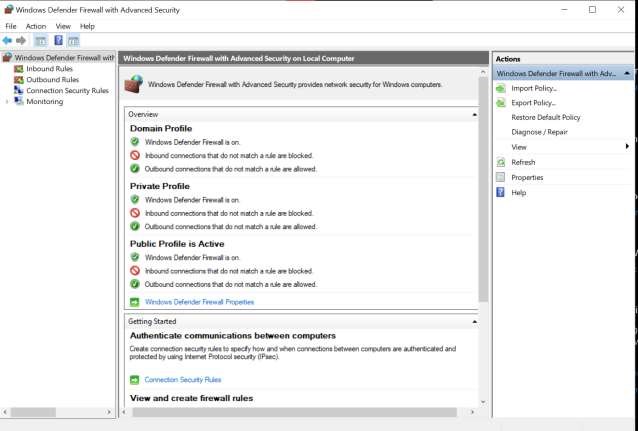
3) Click on windows security and then on Firewall and network protection.



4) Open up firewall and network security and click on public network

5) Now to configure firewall and network: slide on the slider of Microsoft Defender Firewall.

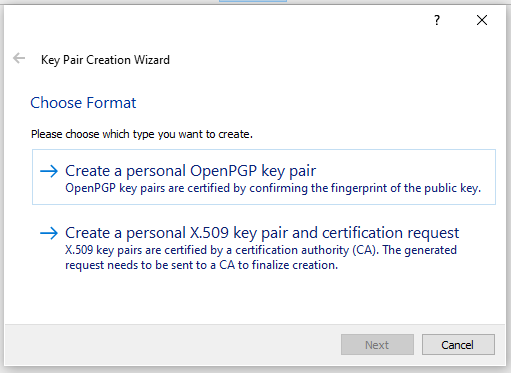
* **Window Defender Fireball with Advanced settings.**



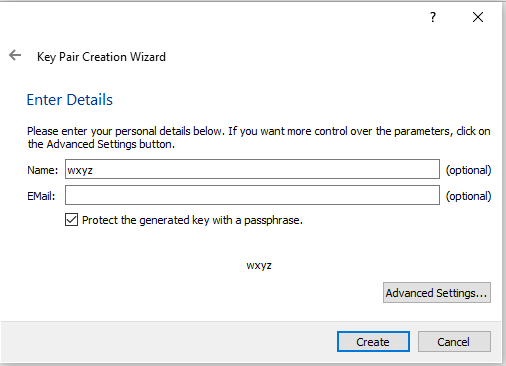
* **Connection Security Rules**
* Connection security involves the authentication of two computers before they begin communications and the securing of information sent between two computers. Windows Firewall with Advanced Security uses Internet Protocol security (IPsec) to achieve connection security by using key exchange, authentication, data integrity, and, optionally, data encryption.
* Connection security rules use IPsec to secure traffic while it crosses the network. We use connection security rules to specify that connections between two computers must be authenticated or encrypted. we have to create a firewall rule to allow network traffic protected by a connection security rule.

**DEMONSTRATE HOW TO PROVIDE SECURE DATA STORAGE, SECURE DATA TRANSMISSION AND FOR CREATING DIGITAL SIGNATURES USING TOOL GNUPG.**

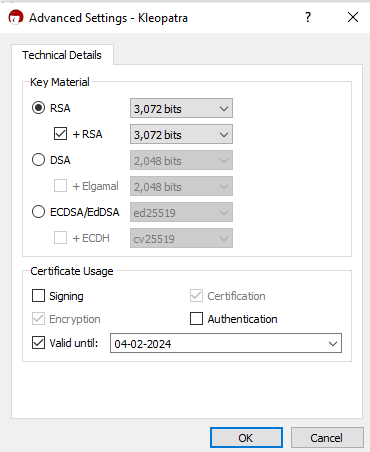
1. Click File->New Certificate:



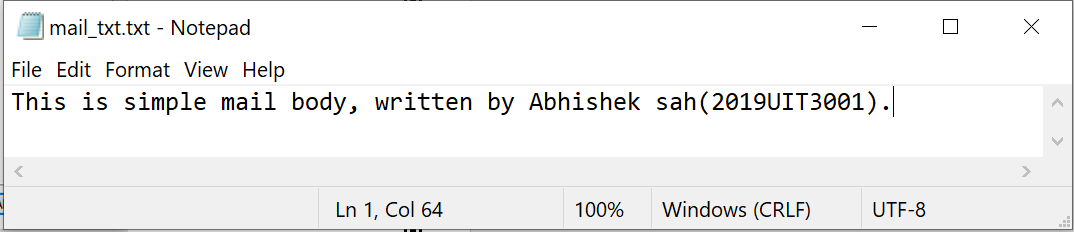
1. Choose and create a personal OpenPGP key pair. It will prompt to enter name and email.



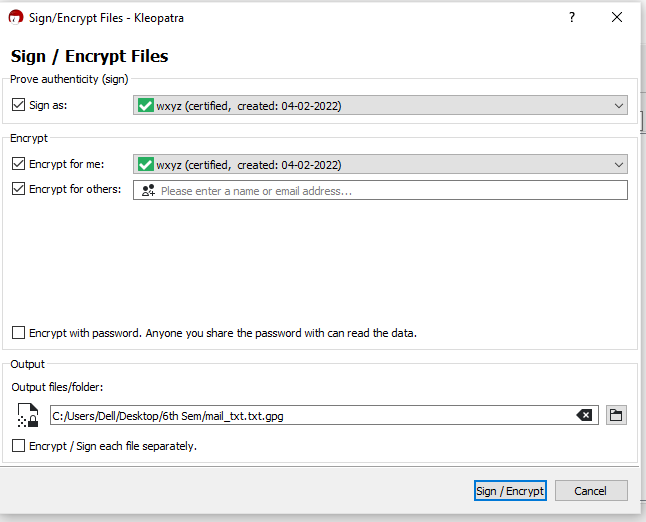
1. Click Advanced Settings and choose appropriate key and key size. After this step key will be created.



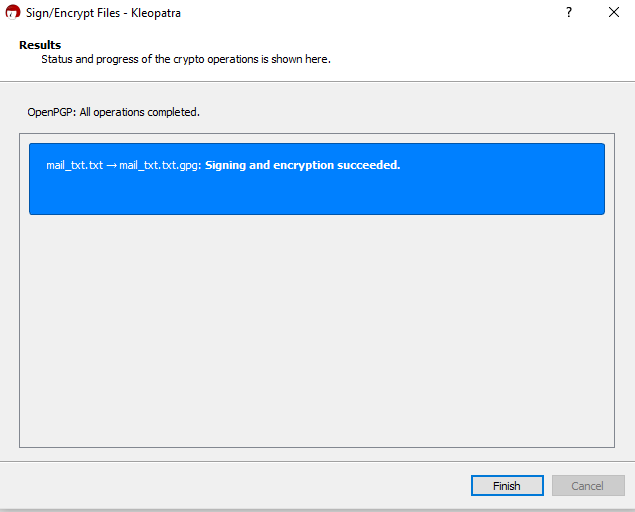
1. Create a text file mail\_txt and add some content to it.



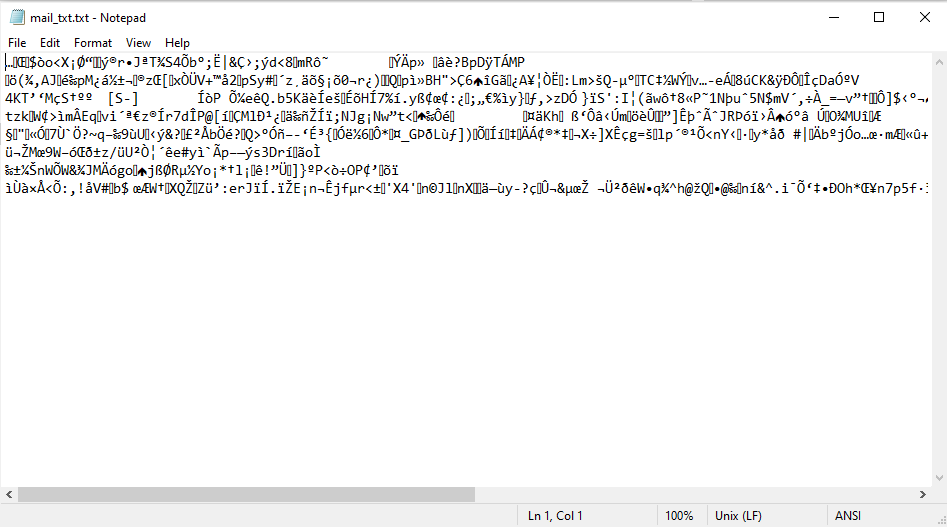
1. Click on Sign/Encrypt Files and select the text file created in the step above. Select the keys and the path to store the encrypted file.



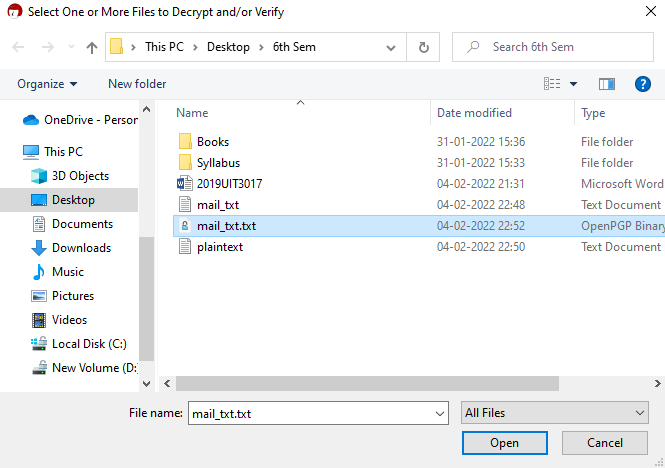
1. Click on Sign/Encrypt and click on finish.



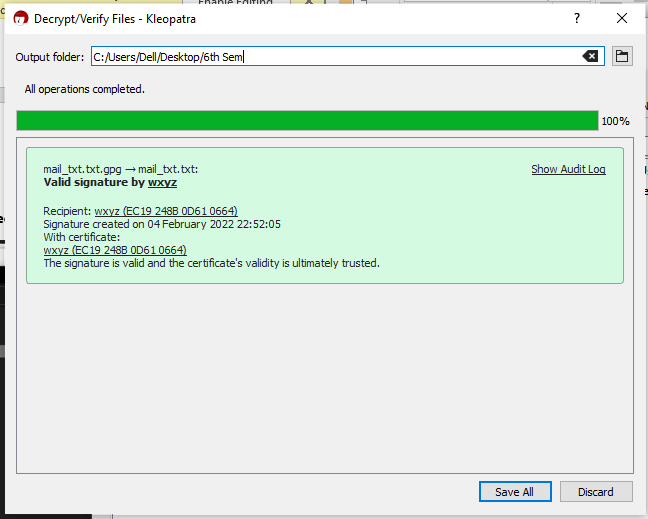
1. Now open the encrypted file to see the encrypted message.



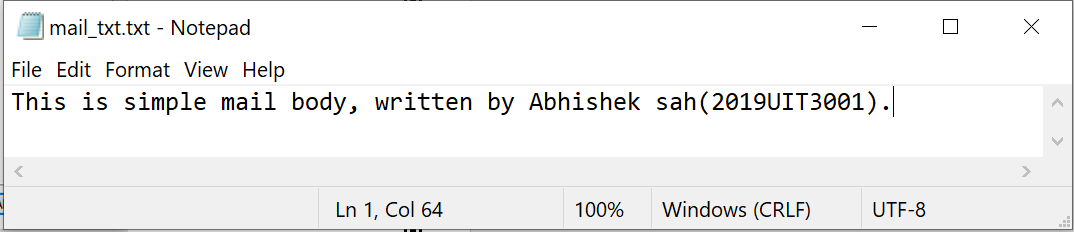
1. Now click on Decrypt/verify and select the encrypted file.



1. Select the path to store the decrypted file.



1. Open the decrypted will and it will have same content as the original file. So Encryption and Decryption is successful.



**IMPLEMENT MD5 ALGORITHM**

**MD5.h**

#ifndef md5\_h

#define md5\_h

#include <cstring>

#include <iostream>

**class** MD5

{

**public**:

**typedef** **unsigned** **int** size\_type; // must be 32bit

MD5();

MD5(**const** std::string& text);

**void** update(**const** **unsigned** **char** \*buf, size\_type length);

**void** update(**const** **char** \*buf, size\_type length);

MD5& finalize();

std::string hexdigest() **const**;

**friend** std::ostream& **operator**<<(std::ostream&, MD5 md5);

**private**:

**void** init();

**typedef** **unsigned** **char** uint1; // 8bit

**typedef** **unsigned** **int** uint4; // 32bit

**enum** {blocksize = 64}; // VC6 won't eat a const static int here

**void** transform(**const** uint1 block[blocksize]);

**static** **void** decode(uint4 output[], **const** uint1 input[], size\_type len);

**static** **void** encode(uint1 output[], **const** uint4 input[], size\_type len);

**bool** finalized;

uint1 buffer[blocksize];

uint4 count[2]; // 64bit counter for number of bits (lo, hi)

uint4 state[4]; // digest so far

uint1 digest[16]; // the result

// low level logic operations

**static** **inline** uint4 F(uint4 x, uint4 y, uint4 z);

**static** **inline** uint4 G(uint4 x, uint4 y, uint4 z);

**static** **inline** uint4 H(uint4 x, uint4 y, uint4 z);

**static** **inline** uint4 I(uint4 x, uint4 y, uint4 z);

**static** **inline** uint4 rotate\_left(uint4 x, **int** n);

**static** **inline** **void** FF(uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac);

**static** **inline** **void** GG(uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac);

**static** **inline** **void** HH(uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac);

**static** **inline** **void** II(uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac);

};

std::string md5(**const** std::string str);

#endif /\* md5\_h \*/

**Main.cpp**

/\* interface header \*/

#include "md5.h"

/\* system implementation headers \*/

#include <cstdio>

// Constants for MD5Transform routine.

#define S11 7

#define S12 12

#define S13 17

#define S14 22

#define S21 5

#define S22 9

#define S23 14

#define S24 20

#define S31 4

#define S32 11

#define S33 16

#define S34 23

#define S41 6

#define S42 10

#define S43 15

#define S44 21

///////////////////////////////////////////////

// F, G, H and I are basic MD5 functions.

**inline** MD5::uint4 MD5::F(uint4 x, uint4 y, uint4 z) {

**return** x&y | ~x&z;

}

**inline** MD5::uint4 MD5::G(uint4 x, uint4 y, uint4 z) {

**return** x&z | y&~z;

}

**inline** MD5::uint4 MD5::H(uint4 x, uint4 y, uint4 z) {

**return** x^y^z;

}

**inline** MD5::uint4 MD5::I(uint4 x, uint4 y, uint4 z) {

**return** y ^ (x | ~z);

}

// rotate\_left rotates x left n bits.

**inline** MD5::uint4 MD5::rotate\_left(uint4 x, **int** n) {

**return** (x << n) | (x >> (32-n));

}

// FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.

// Rotation is separate from addition to prevent recomputation.

**inline** **void** MD5::FF(uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac) {

a = rotate\_left(a+ F(b,c,d) + x + ac, s) + b;

}

**inline** **void** MD5::GG(uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac) {

a = rotate\_left(a + G(b,c,d) + x + ac, s) + b;

}

**inline** **void** MD5::HH(uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac) {

a = rotate\_left(a + H(b,c,d) + x + ac, s) + b;

}

**inline** **void** MD5::II(uint4 &a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac) {

a = rotate\_left(a + I(b,c,d) + x + ac, s) + b;

}

//////////////////////////////////////////////

// default ctor, just initailize

MD5::MD5()

{

init();

}

// nifty shortcut ctor, compute MD5 for string and finalize it right away

MD5::MD5(**const** std::string &text)

{

init();

update(text.c\_str(), text.length());

finalize();

}

//////////////////////////////

**void** MD5::init()

{

finalized=**false**;

count[0] = 0;

count[1] = 0;

// load magic initialization constants.

state[0] = 0x67452301;

state[1] = 0xefcdab89;

state[2] = 0x98badcfe;

state[3] = 0x10325476;

}

//////////////////////////////

// decodes input (unsigned char) into output (uint4). Assumes len is a multiple of 4.

**void** MD5::decode(uint4 output[], **const** uint1 input[], size\_type len)

{

**for** (**unsigned** **int** i = 0, j = 0; j < len; i++, j += 4)

output[i] = ((uint4)input[j]) | (((uint4)input[j+1]) << 8) |

(((uint4)input[j+2]) << 16) | (((uint4)input[j+3]) << 24);

}

//////////////////////////////

// encodes input (uint4) into output (unsigned char). Assumes len is

// a multiple of 4.

**void** MD5::encode(uint1 output[], **const** uint4 input[], size\_type len)

{

**for** (size\_type i = 0, j = 0; j < len; i++, j += 4) {

output[j] = input[i] & 0xff;

output[j+1] = (input[i] >> 8) & 0xff;

output[j+2] = (input[i] >> 16) & 0xff;

output[j+3] = (input[i] >> 24) & 0xff;

}

}

//////////////////////////////

// apply MD5 algo on a block

**void** MD5::transform(**const** uint1 block[blocksize])

{

uint4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

decode (x, block, blocksize);

/\* Round 1 \*/

FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /\* 1 \*/

FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /\* 2 \*/

FF (c, d, a, b, x[ 2], S13, 0x242070db); /\* 3 \*/

FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /\* 4 \*/

FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /\* 5 \*/

FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /\* 6 \*/

FF (c, d, a, b, x[ 6], S13, 0xa8304613); /\* 7 \*/

FF (b, c, d, a, x[ 7], S14, 0xfd469501); /\* 8 \*/

FF (a, b, c, d, x[ 8], S11, 0x698098d8); /\* 9 \*/

FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /\* 10 \*/

FF (c, d, a, b, x[10], S13, 0xffff5bb1); /\* 11 \*/

FF (b, c, d, a, x[11], S14, 0x895cd7be); /\* 12 \*/

FF (a, b, c, d, x[12], S11, 0x6b901122); /\* 13 \*/

FF (d, a, b, c, x[13], S12, 0xfd987193); /\* 14 \*/

FF (c, d, a, b, x[14], S13, 0xa679438e); /\* 15 \*/

FF (b, c, d, a, x[15], S14, 0x49b40821); /\* 16 \*/

/\* Round 2 \*/

GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /\* 17 \*/

GG (d, a, b, c, x[ 6], S22, 0xc040b340); /\* 18 \*/

GG (c, d, a, b, x[11], S23, 0x265e5a51); /\* 19 \*/

GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /\* 20 \*/

GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /\* 21 \*/

GG (d, a, b, c, x[10], S22, 0x2441453); /\* 22 \*/

GG (c, d, a, b, x[15], S23, 0xd8a1e681); /\* 23 \*/

GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /\* 24 \*/

GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /\* 25 \*/

GG (d, a, b, c, x[14], S22, 0xc33707d6); /\* 26 \*/

GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /\* 27 \*/

GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /\* 28 \*/

GG (a, b, c, d, x[13], S21, 0xa9e3e905); /\* 29 \*/

GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /\* 30 \*/

GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /\* 31 \*/

GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /\* 32 \*/

/\* Round 3 \*/

HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /\* 33 \*/

HH (d, a, b, c, x[ 8], S32, 0x8771f681); /\* 34 \*/

HH (c, d, a, b, x[11], S33, 0x6d9d6122); /\* 35 \*/

HH (b, c, d, a, x[14], S34, 0xfde5380c); /\* 36 \*/

HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /\* 37 \*/

HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /\* 38 \*/

HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /\* 39 \*/

HH (b, c, d, a, x[10], S34, 0xbebfbc70); /\* 40 \*/

HH (a, b, c, d, x[13], S31, 0x289b7ec6); /\* 41 \*/

HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /\* 42 \*/

HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /\* 43 \*/

HH (b, c, d, a, x[ 6], S34, 0x4881d05); /\* 44 \*/

HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /\* 45 \*/

HH (d, a, b, c, x[12], S32, 0xe6db99e5); /\* 46 \*/

HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /\* 47 \*/

HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /\* 48 \*/

/\* Round 4 \*/

II (a, b, c, d, x[ 0], S41, 0xf4292244); /\* 49 \*/

II (d, a, b, c, x[ 7], S42, 0x432aff97); /\* 50 \*/

II (c, d, a, b, x[14], S43, 0xab9423a7); /\* 51 \*/

II (b, c, d, a, x[ 5], S44, 0xfc93a039); /\* 52 \*/

II (a, b, c, d, x[12], S41, 0x655b59c3); /\* 53 \*/

II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /\* 54 \*/

II (c, d, a, b, x[10], S43, 0xffeff47d); /\* 55 \*/

II (b, c, d, a, x[ 1], S44, 0x85845dd1); /\* 56 \*/

II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /\* 57 \*/

II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /\* 58 \*/

II (c, d, a, b, x[ 6], S43, 0xa3014314); /\* 59 \*/

II (b, c, d, a, x[13], S44, 0x4e0811a1); /\* 60 \*/

II (a, b, c, d, x[ 4], S41, 0xf7537e82); /\* 61 \*/

II (d, a, b, c, x[11], S42, 0xbd3af235); /\* 62 \*/

II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /\* 63 \*/

II (b, c, d, a, x[ 9], S44, 0xeb86d391); /\* 64 \*/

state[0] += a;

state[1] += b;

state[2] += c;

state[3] += d;

// Zeroize sensitive information.

memset(x, 0, **sizeof** x);

}

//////////////////////////////

// MD5 block update operation. Continues an MD5 message-digest

// operation, processing another message block

**void** MD5::update(**const** **unsigned** **char** input[], size\_type length)

{

// compute number of bytes mod 64

size\_type index = count[0] / 8 % blocksize;

// Update number of bits

**if** ((count[0] += (length << 3)) < (length << 3))

count[1]++;

count[1] += (length >> 29);

// number of bytes we need to fill in buffer

size\_type firstpart = 64 - index;

size\_type i;

// transform as many times as possible.

**if** (length >= firstpart)

{

// fill buffer first, transform

memcpy(&buffer[index], input, firstpart);

transform(buffer);

// transform chunks of blocksize (64 bytes)

**for** (i = firstpart; i + blocksize <= length; i += blocksize)

transform(&input[i]);

index = 0;

}

**else**

i = 0;

// buffer remaining input

memcpy(&buffer[index], &input[i], length-i);

}

//////////////////////////////

// for convenience provide a verson with signed char

**void** MD5::update(**const** **char** input[], size\_type length)

{

update((**const** **unsigned** **char**\*)input, length);

}

//////////////////////////////

// MD5 finalization. Ends an MD5 message-digest operation, writing the

// the message digest and zeroizing the context.

MD5& MD5::finalize()

{

**static** **unsigned** **char** padding[64] = {

0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

};

**if** (!finalized) {

// Save number of bits

**unsigned** **char** bits[8];

encode(bits, count, 8);

// pad out to 56 mod 64.

size\_type index = count[0] / 8 % 64;

size\_type padLen = (index < 56) ? (56 - index) : (120 - index);

update(padding, padLen);

// Append length (before padding)

update(bits, 8);

// Store state in digest

encode(digest, state, 16);

// Zeroize sensitive information.

memset(buffer, 0, **sizeof** buffer);

memset(count, 0, **sizeof** count);

finalized=**true**;

}

**return** \***this**;

}

//////////////////////////////

// return hex representation of digest as string

std::string MD5::hexdigest() **const**

{

**if** (!finalized)

**return** "";

**char** buf[33];

**for** (**int** i=0; i<16; i++)

sprintf(buf+i\*2, "%02x", digest[i]);

buf[32]=0;

**return** std::string(buf);

}

std::ostream& **operator**<<(std::ostream& out, MD5 md5)

{

**return** out << md5.hexdigest();

}

std::string md5(**const** std::string str)

{

MD5 md5 = MD5(str);

**return** md5.hexdigest();

}

**using** std::cout; **using** std::endl;

**int** main(**int** argc, **char** \*argv[])

{

cout << "MD5 of 'information': " << md5("information") << endl;

**return** 0;

}

**OUTPUT**

****

**IMPLEMENT SHA256 ALGORITHM**

**SHA256.h**

#ifndef SHA256\_H

#define SHA256\_H

#include <string>

using namespace std;

class SHA256

{

protected:

    typedef unsigned char uint8;

    typedef unsigned int uint32;

    typedef unsigned long long uint64;

    const static uint32 sha256\_k[];

    static const unsigned int SHA224\_256\_BLOCK\_SIZE = (512/8);

public:

    void init();

    void update(const unsigned char \*message, unsigned int len);

    void final(unsigned char \*digest);

    static const unsigned int DIGEST\_SIZE = ( 256 / 8);

protected:

    void transform(const unsigned char \*message, unsigned int block\_nb);

    unsigned int m\_tot\_len;

    unsigned int m\_len;

    unsigned char m\_block[2\*SHA224\_256\_BLOCK\_SIZE];

    uint32 m\_h[8];

};

string sha256(string input);

#define SHA2\_SHFR(x, n)    (x >> n)

#define SHA2\_ROTR(x, n)   ((x >> n) | (x << ((sizeof(x) << 3) - n)))

#define SHA2\_ROTL(x, n)   ((x << n) | (x >> ((sizeof(x) << 3) - n)))

#define SHA2\_CH(x, y, z)  ((x & y) ^ (~x & z))

#define SHA2\_MAJ(x, y, z) ((x & y) ^ (x & z) ^ (y & z))

#define SHA256\_F1(x) (SHA2\_ROTR(x,  2) ^ SHA2\_ROTR(x, 13) ^ SHA2\_ROTR(x, 22))

#define SHA256\_F2(x) (SHA2\_ROTR(x,  6) ^ SHA2\_ROTR(x, 11) ^ SHA2\_ROTR(x, 25))

#define SHA256\_F3(x) (SHA2\_ROTR(x,  7) ^ SHA2\_ROTR(x, 18) ^ SHA2\_SHFR(x,  3))

#define SHA256\_F4(x) (SHA2\_ROTR(x, 17) ^ SHA2\_ROTR(x, 19) ^ SHA2\_SHFR(x, 10))

#define SHA2\_UNPACK32(x, str)                 \

{                                             \

    \*((str) + 3) = (uint8) ((x)      );       \

    \*((str) + 2) = (uint8) ((x) >>  8);       \

    \*((str) + 1) = (uint8) ((x) >> 16);       \

    \*((str) + 0) = (uint8) ((x) >> 24);       \

}

#define SHA2\_PACK32(str, x)                   \

{                                             \

    \*(x) =   ((uint32) \*((str) + 3)      )    \

           | ((uint32) \*((str) + 2) <<  8)    \

           | ((uint32) \*((str) + 1) << 16)    \

           | ((uint32) \*((str) + 0) << 24);   \

}

#endif

SHA256.cpp

#include <bits/stdc++.h>

#include <fstream>

#include "sha256.h"

using namespace std;

const unsigned int SHA256::sha256\_k[64] = //UL = uint32

            {0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,

             0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,

             0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,

             0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,

             0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,

             0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,

             0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,

             0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,

             0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,

             0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,

             0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,

             0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,

             0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,

             0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,

             0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,

             0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2};

void SHA256::transform(const unsigned char \*message, unsigned int block\_nb)

{

    uint32 w[64];

    uint32 wv[8];

    uint32 t1, t2;

    const unsigned char \*sub\_block;

    int i;

    int j;

    for (i = 0; i < (int) block\_nb; i++) {

        sub\_block = message + (i << 6);

        for (j = 0; j < 16; j++) {

            SHA2\_PACK32(&sub\_block[j << 2], &w[j]);

        }

        for (j = 16; j < 64; j++) {

            w[j] =  SHA256\_F4(w[j -  2]) + w[j -  7] + SHA256\_F3(w[j - 15]) + w[j - 16];

        }

        for (j = 0; j < 8; j++) {

            wv[j] = m\_h[j];

        }

        for (j = 0; j < 64; j++) {

            t1 = wv[7] + SHA256\_F2(wv[4]) + SHA2\_CH(wv[4], wv[5], wv[6])

                + sha256\_k[j] + w[j];

            t2 = SHA256\_F1(wv[0]) + SHA2\_MAJ(wv[0], wv[1], wv[2]);

            wv[7] = wv[6];

            wv[6] = wv[5];

            wv[5] = wv[4];

            wv[4] = wv[3] + t1;

            wv[3] = wv[2];

            wv[2] = wv[1];

            wv[1] = wv[0];

            wv[0] = t1 + t2;

        }

        for (j = 0; j < 8; j++) {

            m\_h[j] += wv[j];

        }

    }

}

void SHA256::init()

{

    m\_h[0] = 0x6a09e667;

    m\_h[1] = 0xbb67ae85;

    m\_h[2] = 0x3c6ef372;

    m\_h[3] = 0xa54ff53a;

    m\_h[4] = 0x510e527f;

    m\_h[5] = 0x9b05688c;

    m\_h[6] = 0x1f83d9ab;

    m\_h[7] = 0x5be0cd19;

    m\_len = 0;

    m\_tot\_len = 0;

}

void SHA256::update(const unsigned char \*message, unsigned int len)

{

    unsigned int block\_nb;

    unsigned int new\_len, rem\_len, tmp\_len;

    const unsigned char \*shifted\_message;

    tmp\_len = SHA224\_256\_BLOCK\_SIZE - m\_len;

    rem\_len = len < tmp\_len ? len : tmp\_len;

    memcpy(&m\_block[m\_len], message, rem\_len);

    if (m\_len + len < SHA224\_256\_BLOCK\_SIZE) {

        m\_len += len;

        return;

    }

    new\_len = len - rem\_len;

    block\_nb = new\_len / SHA224\_256\_BLOCK\_SIZE;

    shifted\_message = message + rem\_len;

    transform(m\_block, 1);

    transform(shifted\_message, block\_nb);

    rem\_len = new\_len % SHA224\_256\_BLOCK\_SIZE;

    memcpy(m\_block, &shifted\_message[block\_nb << 6], rem\_len);

    m\_len = rem\_len;

    m\_tot\_len += (block\_nb + 1) << 6;

}

void SHA256::final(unsigned char \*digest)

{

    unsigned int block\_nb;

    unsigned int pm\_len;

    unsigned int len\_b;

    int i;

    block\_nb = (1 + ((SHA224\_256\_BLOCK\_SIZE - 9)

                     < (m\_len % SHA224\_256\_BLOCK\_SIZE)));

    len\_b = (m\_tot\_len + m\_len) << 3;

    pm\_len = block\_nb << 6;

    memset(m\_block + m\_len, 0, pm\_len - m\_len);

    m\_block[m\_len] = 0x80;

    SHA2\_UNPACK32(len\_b, m\_block + pm\_len - 4);

    transform(m\_block, block\_nb);

    for (i = 0 ; i < 8; i++) {

        SHA2\_UNPACK32(m\_h[i], &digest[i << 2]);

    }

}

string sha256(string input)

{

    unsigned char digest[SHA256::DIGEST\_SIZE];

    memset(digest,0,SHA256::DIGEST\_SIZE);

    SHA256 ctx = SHA256();

    ctx.init();

    ctx.update( (unsigned char\*)input.c\_str(), input.length());

    ctx.final(digest);

    char buf[2\*SHA256::DIGEST\_SIZE+1];

    buf[2\*SHA256::DIGEST\_SIZE] = 0;

    for (int i = 0; i < SHA256::DIGEST\_SIZE; i++)

        sprintf(buf+i\*2, "%02x", digest[i]);

    return string(buf);

}

int main(int argc, char \*argv[])

{

    string input = "secret";

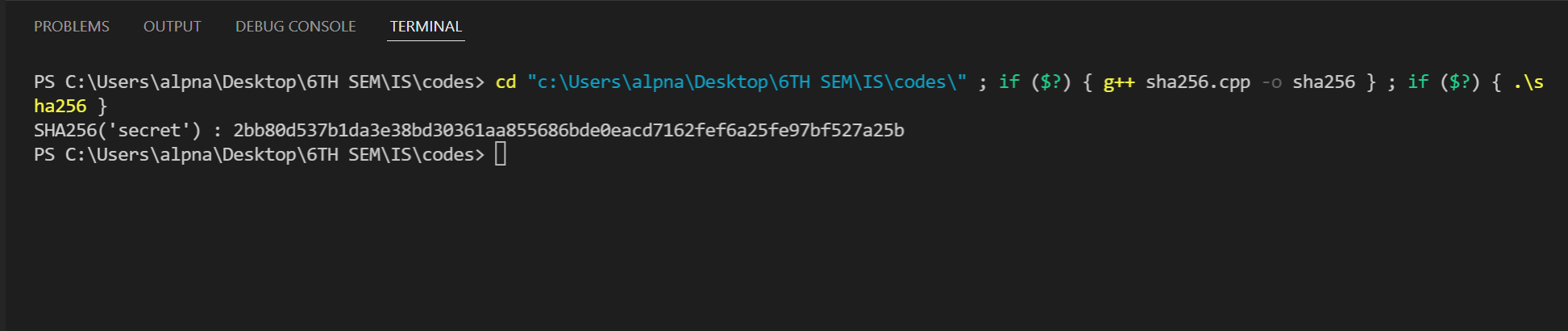
    string output1 = sha256(input);

    cout << "SHA256('"<< input << "') : " << output1 << endl;

    return 0;

}

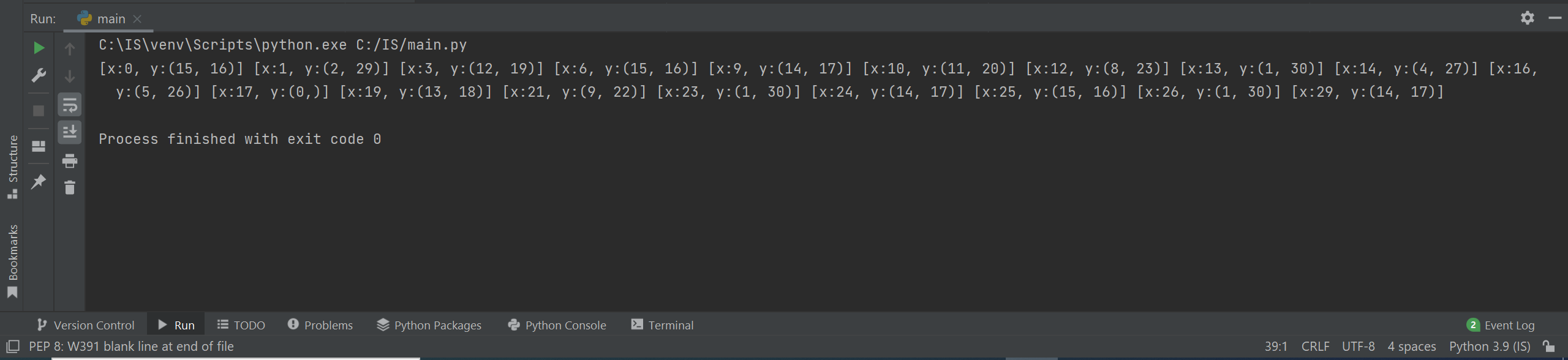
OUTPUT

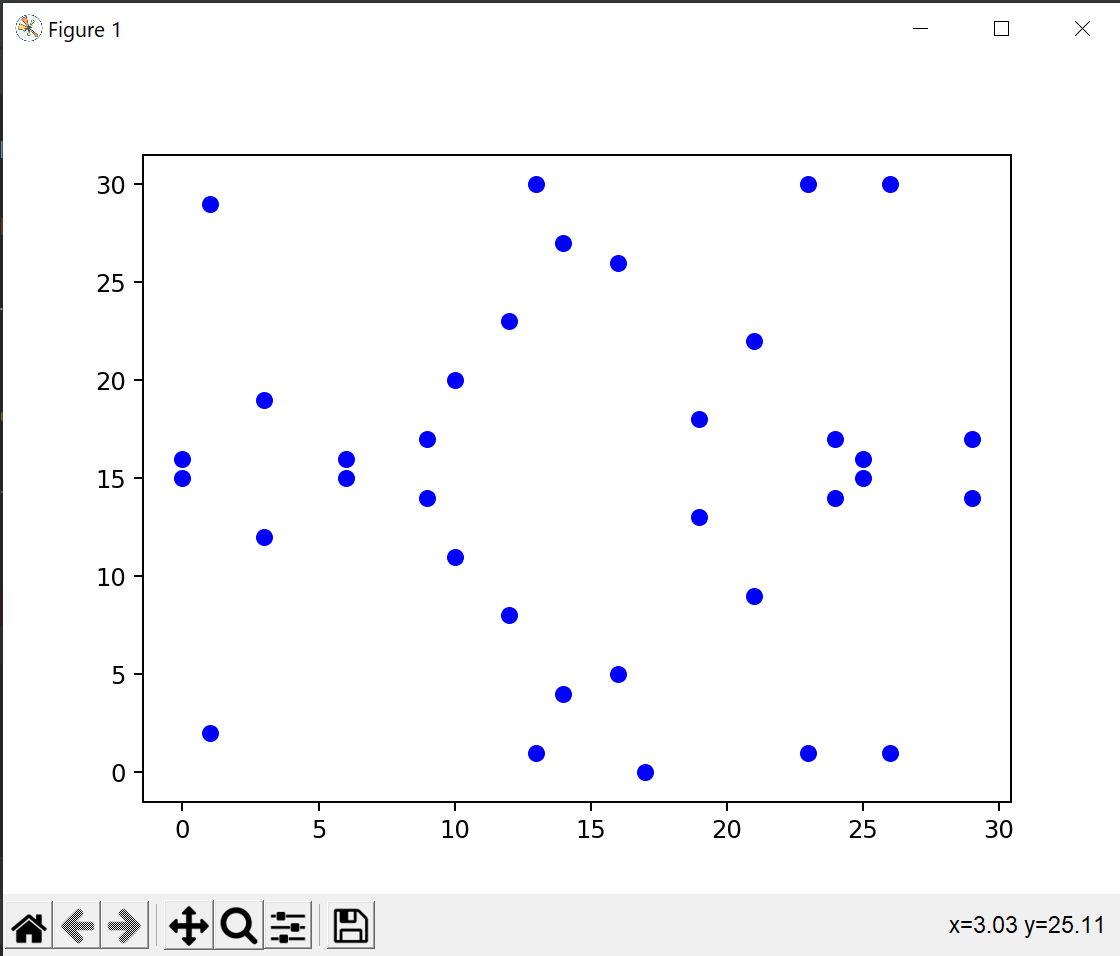


**PLOT AN ELLIPTIC CURVE OVER FINITE FIELD. CHECK WHETHER THE POINTS LIES ON THE ELLIPTIC CURVE OR NOT.**

import numpy as np  
import matplotlib.pyplot as plt  
  
def ecc(x, p, A, B):  
 assert (4\*A\*\*3 + 27\*B\*\*2) % p!= 0  
 return (x\*\*3 + A\*x + B) % p  
  
# Find the elements in x^2 that are equal to elements in y^2, which in finite field is to find sqrt  
def sqrt\_f(x, y2, p):  
 x2 = x\*\*2 % p  
 y = [(i, \*y\_i) for i, y\_i in enumerate([np.where(y2\_i == x2)[0] for y2\_i in y2]) if y\_i.size > 0]  
 return y  
  
  
# Order  
p = 31  
x = np.array(range(0, p))  
  
A = -5  
B = 8  
  
x2 = x\*\*2 % p  
y2 = ecc(x, p, A, B)  
  
# Compute the sqrt of the elliptic curve for plotting  
y = sqrt\_f(x, y2, p)  
  
fig = plt.figure(dpi=100)  
for y\_p in y:  
 [plt.scatter(y\_p[0], i, c='b') for i in y\_p[1:]]  
  
# Number of points and +1 for point at infinity  
len(sum([y\_p[1:] for y\_p in y], ())) + 1  
  
print(\*['[x:{}, y:{}]'.format(y\_p[0], y\_p[1:]) for y\_p in y])  
  
plt.show()

**OUTPUT**

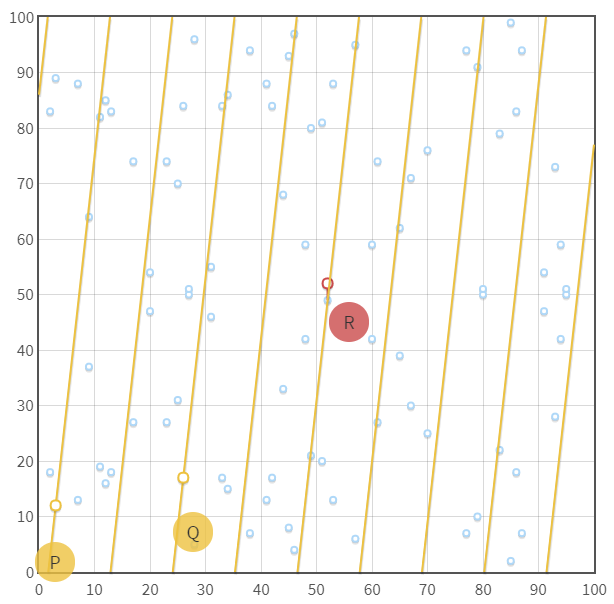
****

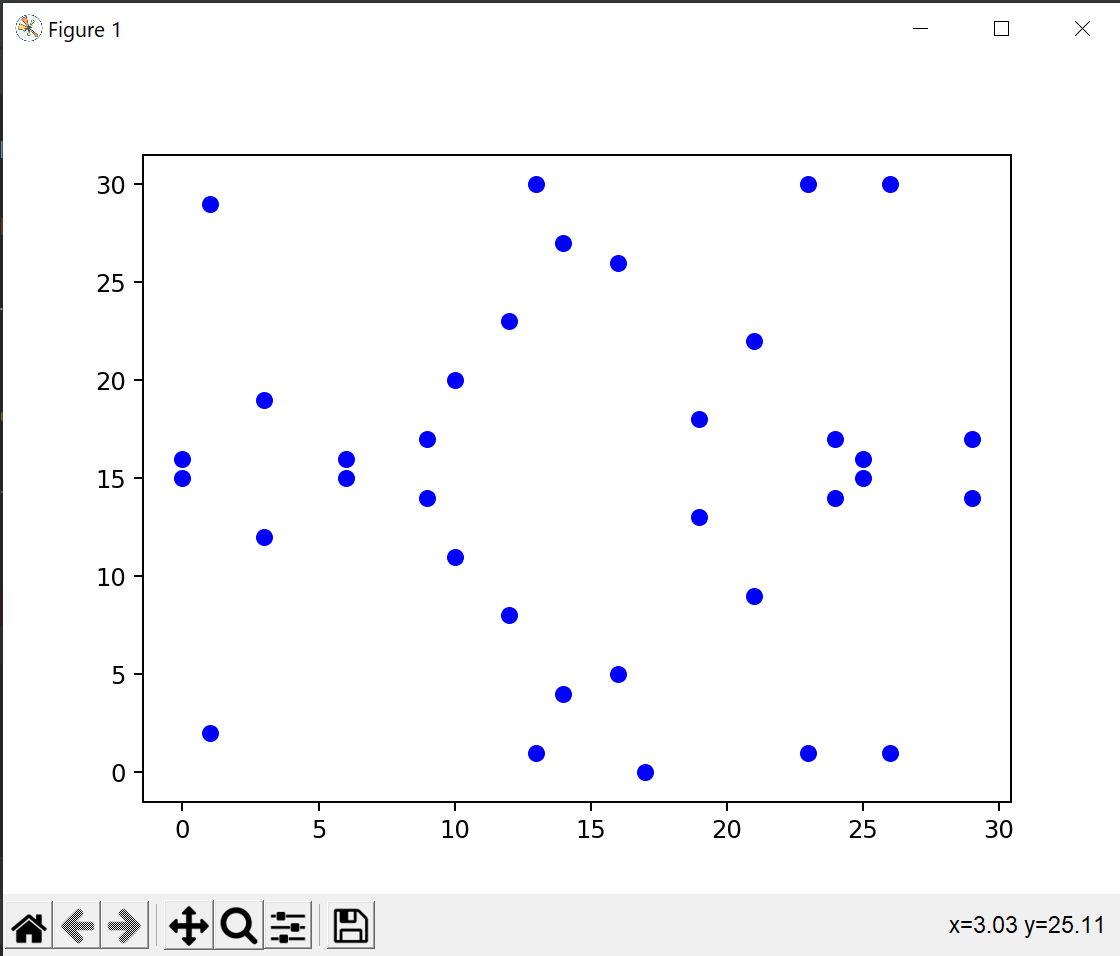
****

PERFORM THE ELLIPTIC CURVE OPERATIONS LIKE ADDITION AND MULTIPLICATION OF TWO POINTS AND FIND THE INVERSE OF A POINT ALSO.

import numpy as np  
import matplotlib.pyplot as plt  
  
  
def ecc(x, p, A, B):  
 assert (4 \* A \*\* 3 + 27 \* B \*\* 2) % p != 0  
 return (x \*\* 3 + A \* x + B) % p  
  
  
# Find the elements in x^2 that are equal to elements in y^2, which in finite field is to find sqrt  
def sqrt\_f(x, y2, p):  
 x2 = x \*\* 2 % p  
 y = [(i, \*y\_i) for i, y\_i in enumerate([np.where(y2\_i == x2)[0] for y2\_i in y2]) if y\_i.size > 0]  
 return y  
  
  
# Order  
p = 31  
x = np.array(range(0, p))  
  
A = -5  
B = 8  
  
x2 = x \*\* 2 % p  
y2 = ecc(x, p, A, B)  
  
# Compute the sqrt of the elliptic curve for plotting  
y = sqrt\_f(x, y2, p)  
  
fig = plt.figure(dpi=100)  
for y\_p in y:  
 [plt.scatter(y\_p[0], i, c='b') for i in y\_p[1:]]  
  
  
def get\_2P(P, p, A, B):  
 x, y = P  
  
 lam = (3 \* x \*\* 2 + A) \* eea(2 \* y, p) % p  
 nu = (-x \*\* 3 + A \* x + 2 \* B) \* eea(2 \* y, p) % p  
 # n = (x\*\*4 - 2\*A\*x\*\*2 - 8\*B\*x + A\*\*2) % p  
 # d = (4 \* (x \*\* 3 + A \* x + B)) % p  
 # d = eea(d, p)  
 return (lam \*\* 2 - 2 \* x) % p, (-lam \*\* 3 + 2 \* lam \* x - nu) % p  
 # return n \* d % p, ecc(n \* d % p, p, A, B)  
  
  
def get\_PQ(P, Q, p, A, B):  
 x1, y1 = P  
 x2, y2 = Q  
  
 lam = (y2 - y1) \* eea((x2 - x1) % p, p) % p  
 nu = (y1 \* x2 - y2 \* x1) \* eea((x2 - x1) % p, p) % p  
  
 return (lam \*\* 2 - x1 - x2) % p, (-lam \*\* 3 + lam \* (x1 + x2) - nu) % p  
  
  
def sum\_on\_E(P, Q, p, A, B):  
 # We have a point at infinity  
 if (P == 0):  
 return Q  
 if (Q == 0):  
 return P  
 if (P == Q):  
 if P[0] == 0:  
 # Point at infinity  
 return 0  
 val = get\_2P(P, p, A, B)  
 return val  
 else:  
 if P[0] == Q[0]:  
 return 0  
  
 val = get\_PQ(P, Q, p, A, B)  
 return val  
  
  
# Extended Euclidean Algorithm for finding inverse of x mod p  
def eea(x, p, debug=False):  
 # Quotient  
 q = int(p / x)  
 # Remainder  
 r = r\_old = p % x  
  
 q\_s = [q]  
 a\_s = [0, 1]  
 i = 0  
 while True:  
 if debug:  
 print(f'{q \* x + r} = {q} \* {x} + {r}')  
 if i > 1:  
 a\_i = (a\_s[i - 2] - a\_s[i - 1] \* q\_s[i - 2]) % p  
 a\_s.append(a\_i)  
 if r == 0:  
 if r\_old != 1:  
 assert "No inverse"  
 i = i + 1  
 break  
 r\_old = r  
 q = int(x / r)  
 q\_s.append(q)  
 r = x % r  
 x = r\_old  
 i = i + 1  
  
 a\_i = (a\_s[i - 2] - a\_s[i - 1] \* q\_s[i - 2]) % p  
 return a\_i  
  
  
plt.show()

**OUTPUT**



****

**IMPLEMENT THE RSA DIGITAL SIGNATURE SCHEME**

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

int modu(int b, unsigned int exp, unsigned int m){

int x = 1;

int i;

int power = b % m;

for (i = 0; i < sizeof(int) \* 8; i++) {

 int least\_bit = 0x00000001 & (exp >> i);

 if (least\_bit)

 x = (x \* power) % m;

 power = (power \* power) % m;

 } return x;

}

int modI(int a, int m)

{

int temp = m;

int y = 0, x = 1;

if (m == 1)

 return 0;

 while (a > 1)

 { int q = a / m;

 int t = m;

 m = a % m, a = t;

 t = y;

 y = x - q \* y;

 x = t;

 } if (x < 0)

 x += temp;

 return x;

}

int gcd(int a, int b)

{

if (a == 0 || b == 0)

 return 0;

if (a == b)

 return a;

if (a > b)

 return gcd(a-b, b);

 return gcd(a, b-a);

}

int Prime(int num){

 int flag = 1;

 for(int i=2;i<=sqrt(num);i++)

 {

 if(num%i==0)

 {

 flag = 0;

 return flag;

 }

 }

 return flag;

}

int lcm(int a, int b)

{

 return (a\*b)/gcd(a, b);

}

int main(){

int msg; char m;

 cout<<"\n Enter the character to be encrypted: ";

 cin>>m;

msg = (int)m;

 cout<<"\n The corresponding ASCII value of the character is"<<msg;

int p,q, random; int i=0; int a[2];

 srand (time(NULL));

 generate:

 while(i<2){

 random = rand() % 40 + 3;

 if(Prime(random)){

 a[i]=random;

 i++;

 } }

 i=0;p=a[0];q=a[1];

 if(p==q){

 goto generate;

}

 cout<<"\n The Random Prime Numbers are: "<<p<<" and "<<q;

int n; n = p\*q;

int phi = (p-1)\*(q-1);

int lambda = lcm(p-1,q-1);

int e;

 vector<int> tot;

 for(int i=3;i<lambda;i++)

 { if(gcd(i,lambda) == 1){

 tot.push\_back(i);

 }

}

int size = tot.size();

int ran = rand() % size;

e = tot[ran];

 cout<<"\n The modulus is: "<<n;

 cout<<"\n The phi(n) is: "<<phi;

 cout<<"\n The lambda(n) is:"<<lambda;

 cout<<"\n The toitient is: "<<e;

 cout<<"\n The public key is: ("<<n<<","<<e<<")";

long long int encrypt;

 encrypt = modu(msg,e,n);

 cout<<"\n The Cipher text is: "<<(char)encrypt;

 cout<<"\n The ASCII value of Cipher Text is: "<<encrypt;

long long int d = modI(e,lambda);

 if(d==e){

 cout<<"\n";

 goto generate;

}

 cout<<"\n The private key is: "<<d;

long long int decrypted;

 decrypted = modu(encrypt,d,n);

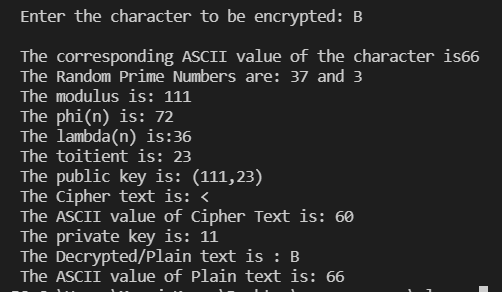
 cout<<"\n The Decrypted/Plain text is : "<<(char)decrypted;

 cout<<"\n The ASCII value of Plain text is: "<<decrypted;

return 0;

}

**OUTPUT**

****

**IMPLEMENT ELGAMAL DIGITAL SIGNATURE SCHEME**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int checkIfPrime (int newPrime);

int checkRoot (int newPrime, int newRoot);

int computeYa (int newRoot, int newXa, int newPrime);

int computeInverseK (int userK, int newPrime);

int computeS2 (int inverseK, int userHash, int userXa, int userS1, int newPrime);

int computeV2 (int Ya, int S1, int S2, int newPrime);

int main ()

{

    int p = 0, q = 0, randXa = 0, UserAYa = 0, userS1 = 0, userS2 = 0;

    int primeCheck = 0, primitiveCheck = 0, userInt = 0;

    int hashValue = -1, randK = 0, inverseK = 0;

    char userChoice;

    char userMessage[256];

    int gcd;

    int V1 = 0, V2 = 0;

    printf ("Welcome to the ElGamal digital signature this works with primes up to 17\n\nThe global elements of this signature are a prime q\nand p, which is the primitive root of q\n");

    while (primeCheck == 0)

    {

        printf ("\nEnter a prime number q: ");

        scanf ("%d", &q);

        primeCheck = checkIfPrime (q);

        if (primeCheck == 1)

            printf (" This is in fact prime\n");

        else

        {

            printf (" Whoops, not prime\n");

            primeCheck = 0;

        }

    }

    while (primitiveCheck == 0)

    {

        printf ("\nEnter a primitive root of p: ");

        scanf ("%d", &p);

        primitiveCheck = checkRoot (q, p);

        if (primitiveCheck == 1) printf ("\nThis is in fact a primitive root\n");

        else

        {

            printf ("\nWhoops, not a primitive root\n");

            primeCheck = 0;

        }

    }

    printf ("\n\nNext User A generates a private/public key pair. First your random integer is generated");

    printf ("\nWould you like to enter a number greater than 1 but less than %d, y for yes or n for no: ", q - 1);

    scanf (" %c", &userChoice);

    if (userChoice == 'n')

    {

        while ((randXa <= 1) || (randXa >= q - 1))

        {

        randXa = rand ();

        }

    }

    else

    {

        while (randXa == 0)

        {

        printf ("Enter your random integer: ");

        scanf ("%d", &randXa);

        if ((randXa <= 1) || (randXa >= q - 1))

        randXa = 0;

        }

    }

    printf ("\n\nNext we computer Ya = p^Xa mod q");

    UserAYa = computeYa (p, randXa, q);

    printf ("\nUser A's Ya is %d", UserAYa);

    printf ("\nA's private key Xa is %d, A's public key is {q, p, Ya} {%d,%d,%d}\n",

    randXa, q, p, UserAYa);

    while (getchar () != '\n'); //Clears the buffer

    printf ("\n\nPlease enter your message:");

    scanf ("%[^\n]", userMessage);

    printf ("\nNext we are gonna give the user Message M a hash value m.");

    printf ("\nWould you like to enter one? y/n: ");

    scanf (" %c", &userChoice);

    if (userChoice == 'n')

    {

        while ((hashValue < 0) || (hashValue > q - 1))

        {

        hashValue = rand ();

        }

    }

    else

    {

        while (hashValue == -1)

        {

        printf ("Enter your hash value: ");

        scanf ("%d", &hashValue);

        if ((hashValue < 0) || (hashValue > q - 1)) hashValue = -1;

        }

    }

    printf ("\nThe hash value is %d", hashValue);

    printf ("\n\nA then forms a digital signature as follows");

    while (randK < 1)

    {

        while ((randK < 1) || (randK > q - 1))

        {

        randK = rand ();

        }

        for (int i = 1; i <= randK && i <= (q - 1); i++)

        {

            if ((randK % i == 0) && ((q - 1) % i == 0))

            gcd = i;

        }

        if (gcd != 1)

        randK = 0;

    }

    printf ("\n The a random integer K is generated %d\n", randK);

    printf ("\n Next we compute S1 = p^K mod q\n");

    userS1 = computeYa (p, randK, q);

    printf (" S1 is %d\n", userS1);

    printf ("\n Next we compute K^-1 mod(q-1)");

    inverseK = computeInverseK (randK, q);

    printf ("\n K^-1 is %d because %d\*%d mod %d = 1", inverseK, inverseK,

    randK, q - 1);

    printf ("\n\n Next we compute S2 = K^-1(m-XaS1)mod(q-1)");

    userS2 = computeS2 (inverseK, hashValue, randXa, userS1, q);

    printf ("\n S2 is %d because it is %d\*(%d - %d\*%d)mod(%d) ", userS2,

    inverseK, hashValue, randXa, userS1, q - 1);

    printf ("\n\nThe signature consists of (S1, S2) which are %d , %d", userS1,

    userS2);

    //Verification

    printf ("\nWould you like to verify? y/n: ");

    scanf (" %c", &userChoice);

    if (userChoice == 'y')

    {

        printf ("\n\nTo check first we calculate V1 = p^m mod q");

        V1 = computeYa (p, hashValue, q);

        printf ("\n\nThen we calculate V2 = (Ya^S1 \* S1^S2)mod q");

        V2 = computeV2 (UserAYa, userS1, userS2, q);

        printf ("\n\nFinally, we check whether they equal each other");

        if (V1 == V2) printf ("\nThe signature is valid V1 %d == V2 %d\n\n", V1, V2);

        else printf ("\nSignature is invalid V1 %d != V2 %d\n\n", V1, V2);

    }

    return 0;

}

int computeV2 (int Ya, int S1, int S2, int newPrime)

{

    int newV2 = 0;

    long long int exYa = 1, exS1 = 1;

    while (S2 != 0)

    {

        exS1 \*= S1;

        --S2;

    }

    while (S1 != 0)

    {

        exYa \*= Ya;

        --S1;

    }

    exYa = exYa % newPrime;

    exS1 = exS1 % newPrime;

    newV2 = exYa \* exS1;

    newV2 = newV2 % newPrime;

    return newV2;

}

int computeS2 (int inverseK, int userHash, int userXa, int userS1, int newPrime)

{

 int newS2 = 0;

 newS2 = inverseK \* (userHash - (userXa \* userS1));

 newS2 = newS2 % (newPrime - 1);

 if (newS2 < 0)

    newS2 = newS2 + (newPrime - 1);

 return newS2;

}

int computeInverseK (int userK, int newPrime)

{

 int inverseK = 0;

 int testValue, posValue = 1;

 while (inverseK == 0)

 {

    testValue = (posValue \* userK) % (newPrime - 1);

    if (testValue == 1)

    {

        inverseK = posValue;

    }

    else

    {

        posValue++;

    }

 }

 return inverseK;

}

int computeYa (int newRoot, int newXa, int newPrime)

{

 long long int newYa = 1;

 while (newXa != 0)

 {

    newYa \*= newRoot;

    --newXa;

 }

 newYa = newYa % newPrime;

 return (int) newYa;

}

// Checks if number is prime

int checkIfPrime (int newPrime)

{

 if (newPrime <= 1)

    return 0;

 if (newPrime > 2)

    for (int factor = 2; factor < newPrime; factor++)

    {

    if (newPrime % factor == 0)

    return 0;

    }

 return 1;

}

int checkRoot (int newPrime, int newRoot)

{

 long long int primeCombo = 0, modCheck = 1;

 int power;

 char result;

 if (newRoot >= newPrime) return 0;

 for (int factor = 1; factor < newPrime; factor++) primeCombo = primeCombo + factor;

 printf ("\n Starting PrimeCombo %d", primeCombo);

 for (int exponent = 1; exponent < newPrime; exponent++)

 {

    power = exponent;

    while (power != 0){modCheck \*= (long long int) newRoot; --power;}

    modCheck = modCheck % (long long int) newPrime;

    printf ("\n%d^%d mod %d is %lli", newRoot, exponent, newPrime, modCheck);

    modCheck = 1;

 }

 printf ("\nDid any value repeat?(y for yes or n for no) ");

 scanf (" %c", &result);

 if (result == 'y')

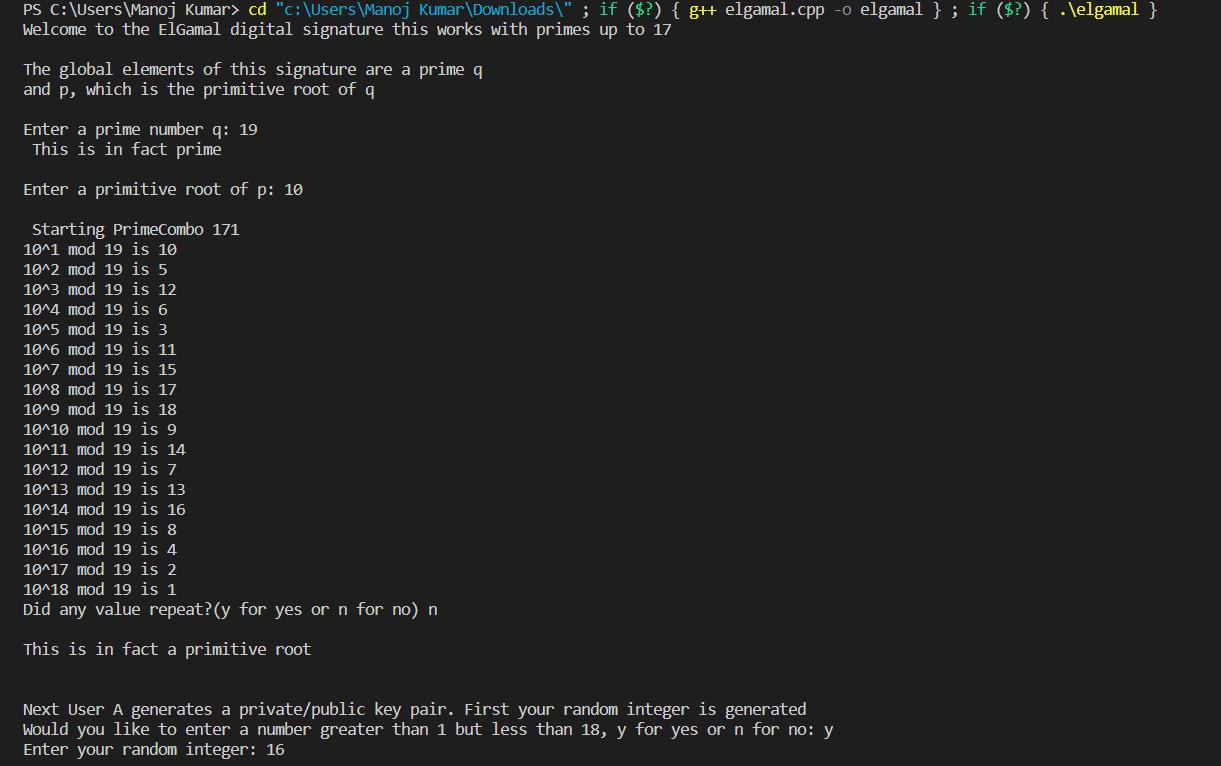
    return 0;

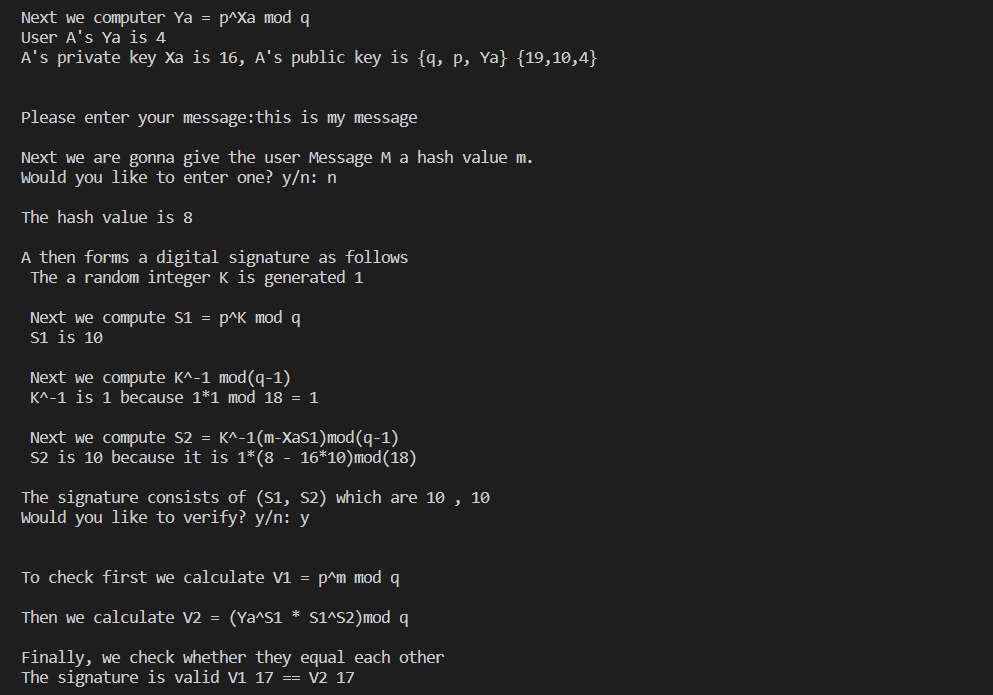
 else

    return 1;

}

**OUTPUT**

****

****

**IMPLEMENT SCHNORR DIGITAL SIGNATURE SCHEME**

#include <bits/stdc++.h>

using namespace std;

#define ld long long

ld ModInv(ld b, ld n)

{

  ld r1 = n, r2 = b, t1 = 0, t2 = 1;

  while (r2 > 0)

  {

    ld q = r1 / r2;

    ld r = r1 - q \* r2;

    r1 = r2;

    r2 = r;

    ld t = t1 - q \* t2;

    t1 = t2;

    t2 = t;

  }

  if (t1 < 0)

    t1 += n;

  return t1;

}

ld powerLL(ld x, ld n, ld MOD)

{

  ld result = 1;

  while (n)

  {

    if (n & 1)

      result = result \* x % MOD;

    n = n / 2;

    x = x \* x % MOD;

  }

  return result;

}

ld powerStrings(string sa, string sb, ld MOD)

{

  ld a = 0, b = 0;

  for (int i = 0; i < sa.length(); i++)

    a = (a \* 10 + (sa[i] - '0')) % MOD;

  for (int i = 0; i < sb.length(); i++)

    b = (b \* 10 + (sb[i] - '0')) % (MOD - 1);

  return powerLL(a, b, MOD);

}

ld mod(string num, ld a)

{

  ld res = 0;

  // One by one process all digits of 'num'

  for (int i = 0; i < num.length(); i++)

    res = (res \* 10 + (int)num[i] - '0') % a;

  return res;

}

int main()

{

  ld p, q, e0, d, m, r, MOD;

  cout << "Enter the value of p, q, e0, d, m and r respecitively: ";

  cin >> p >> q >> e0 >> d >> m >> r;

  // 2267 103 2 30 1000 11

  ld a, b, t, bef\_v, bef\_s1, e1, e2, s1, s2, v;

  e1 = powerStrings(to\_string(e0), to\_string((p - 1) / q), p);

  if (e1 < 0)

    e1 += p;

  e2 = powerStrings(to\_string(e1), to\_string(d), p);

  if (e2 < 0)

    e2 += p;

  ////////// For concate  S1 we are finding both components

  a = powerStrings(to\_string(e1), to\_string(r), p);

  if (a < 0)

    a += p;

  bef\_s1 = stoll(to\_string(m) + to\_string(a), NULL, 10);

  s1 = 200;

  s2 = (r + ((d \* s1) % q)) % q;

  //////////////// fro concate V we are finding both components

  a = powerStrings(to\_string(e1), to\_string(s1), p);

  if (a < 0)

    a += p;

  b = powerStrings(to\_string(e2), to\_string(s2), p);

  if (b < 0)

    b += p;

  t = (a \* b) % p;

  bef\_v = stoll(to\_string(m) + to\_string(t), NULL, 10);

  cout << "e1=" << e1 << "\n";

  cout << "e2=" << e2 << "\n";

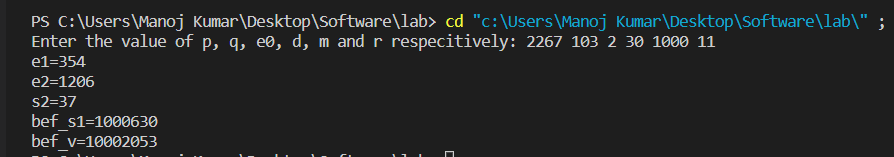
  cout << "s2=" << s2 << "\n";

  cout << "bef\_s1=" << bef\_s1 << "\n";

  cout << "bef\_v=" << bef\_v << "\n";

}

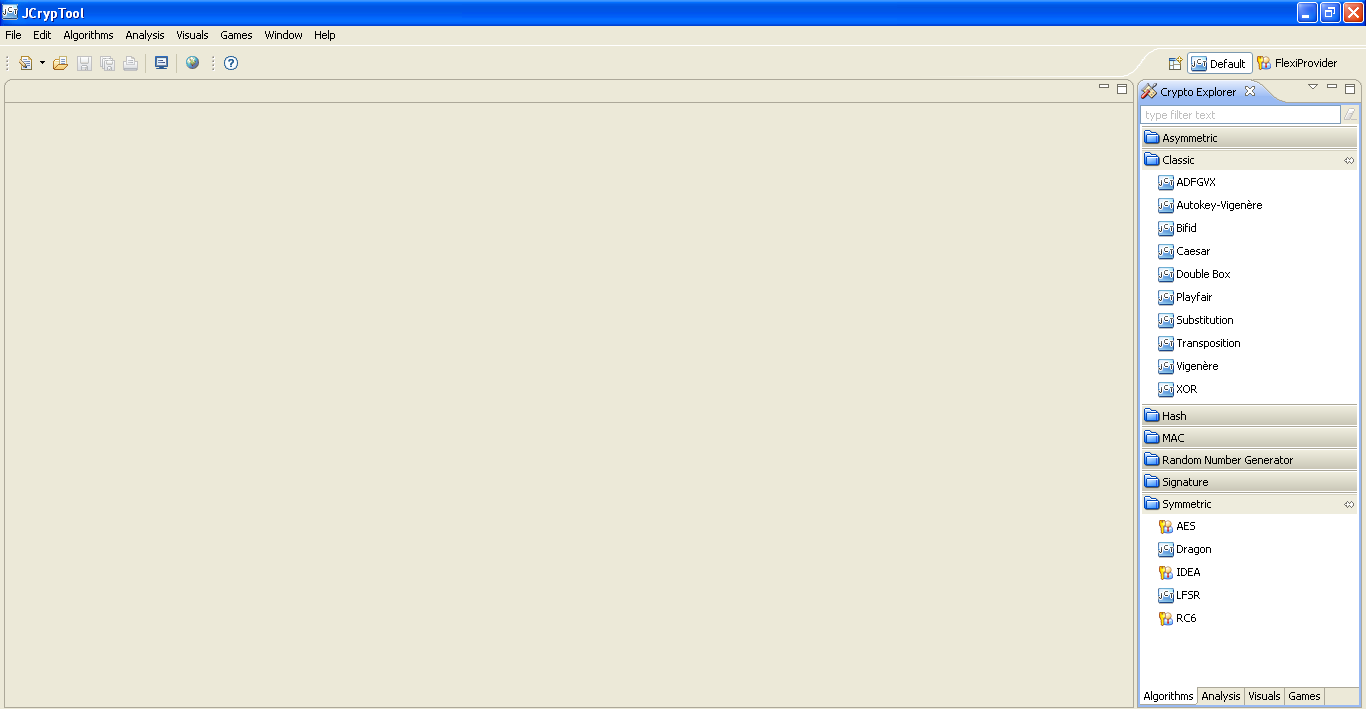
**OUTPUT**

****

**USING JCRYPT TOOL (OR ANY OTHER EQUIVALENT) TO DEMONSTRATE ASYMMETRIC, SYMMETRIC CRYPTO ALGORITHM**

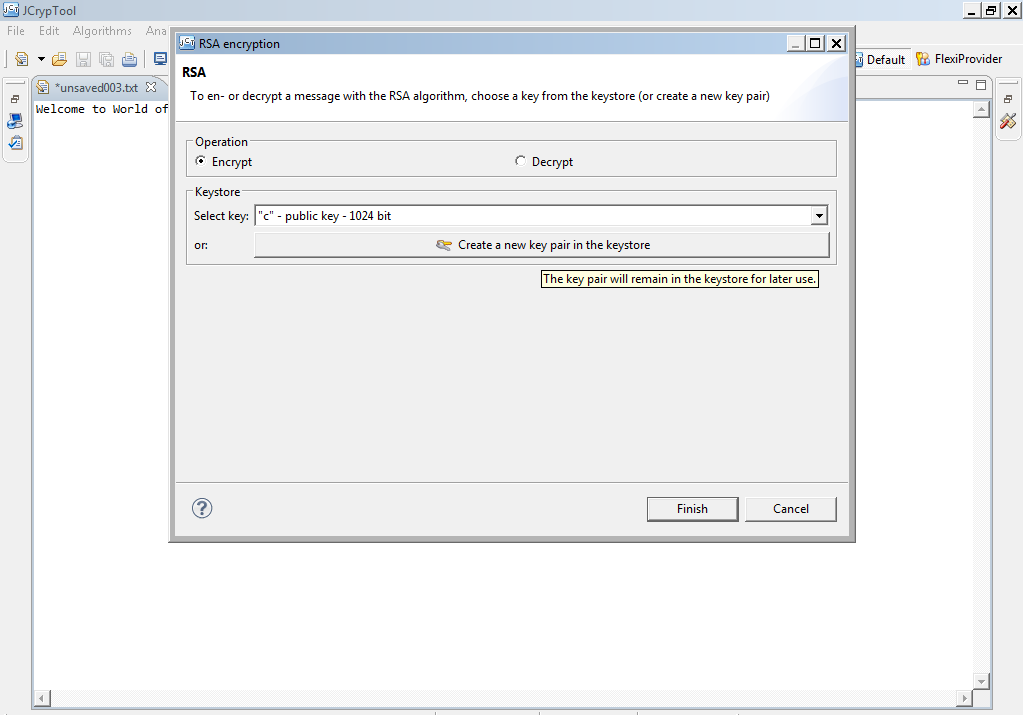
**ASYMMETRIC ALGORITHM**

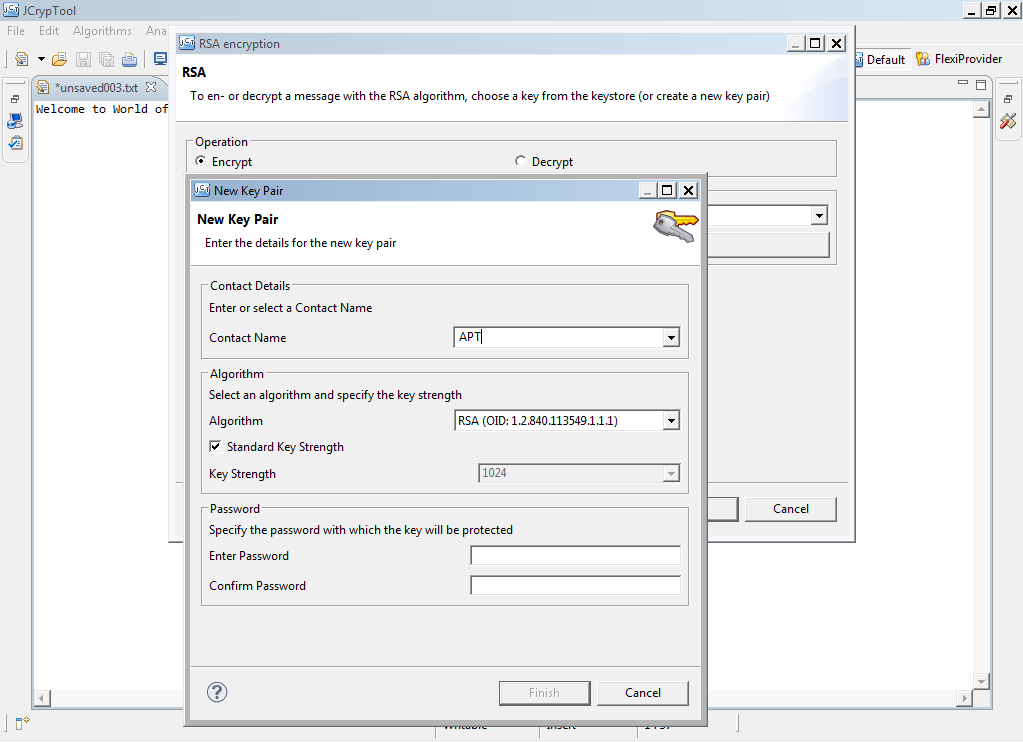
* Download Jcrypt tool from Cryptool Website and Install .
* Open Jcrypt Software and Click on NEW text editior, type the text information into it



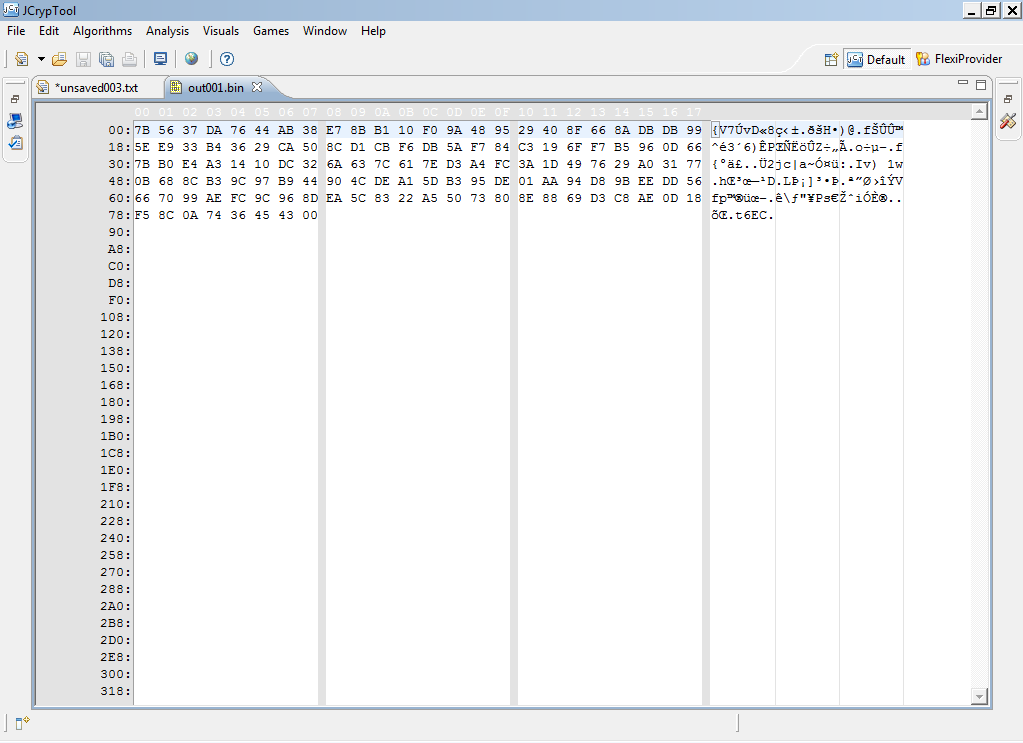


* Click on the Algorithm menu bar and Select Asymmetric algorithm RSA for encryption.
* Click create a New KeyPair and type in the contact name[xxxxx] and enter the password and confirm password, then Click finish again.





* Now you can see RSA output bin file is generated.



* The same output bin file to decrypt select RSA Algorithm and Click on Decrypt, Select keyname you have declared earlier and Click Finish.
* Enter the password to Decrypt and see the output with original Decrypted text on the Screen.

**SYMMETRIC ALGORITHM**

* Click on Algorithm Menu bar Select Symmetric◊AES and Click on it.
* Click on create a new key, type contact name and enter the password and confirm, Click finish◊ Click finish again.
* Enter the password to open the output file.
* To Decrypt Select Algorithms◊ Symmetric◊Select the key which you have created and Click Finish.
* Enter the password and see the result in output bin file with hexadecimal values and plain text.

**IMPLEMENT THE IDENTITY-BASED ENCRYPTION (IBE)**

**USE THE EMAIL ADDRESS OF THE RECIPIENT TO GENERATE THE KEY FOR A DESTINATION.**

**Central Authority Code:**

**PKG.java**

package CentralAuthority;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.math.BigInteger;

import java.util.Scanner;

public class PKG {

    private static BigInteger KEY , n;

    //creating file object from given path

    java.net.URL url1 = getClass().getResource("Data.txt");          // opening user file

    File file = new File(url1.getPath());

    public  void set\_key(String id){

         RSA rsa = new RSA(id);

         KEY = rsa.get\_public\_key();

         n = rsa.getn();

         try(

                 FileWriter fileWriter = new FileWriter(file,true);

                 BufferedWriter bufferFileWriter  = new BufferedWriter(fileWriter);

            ) {

            fileWriter.append(id);

            fileWriter.append("\n");

            fileWriter.append(rsa.get\_public\_key().toString());

            fileWriter.append("\n");

            fileWriter.append(rsa.get\_private\_key().toString());

            fileWriter.append("\n");

            fileWriter.append(n.toString());

            fileWriter.append("\n");

            bufferFileWriter.close();

            fileWriter.close();

        } catch (IOException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

    }

    public BigInteger getn(){

        return n;

    }

    public BigInteger get\_public\_key(String id){                        // Public Key generator

        BigInteger pk = BigInteger.valueOf(-1),sk = BigInteger.valueOf(-1);

        boolean flag = true;

        try {

            Scanner p = new Scanner(file);

            String ID;

            while(p.hasNext()){

                ID = p.next();

                pk = p.nextBigInteger();

                sk = p.nextBigInteger();

                n  = p.nextBigInteger();

                if(id.compareTo(ID) == 0){                      // User is already registered

                    flag = false;

                    break;

                }

            }

            p.close();

        }

        catch (FileNotFoundException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

        if(!flag){

            KEY = pk;

            return KEY;

        }

        set\_key(id);                                // User is not registered .... therefore registering

        return KEY;

    }

    public  BigInteger get\_private\_key(String id){              // Private Key generator

        BigInteger x = BigInteger.valueOf(-1);

        boolean flag = true;

        try {

            Scanner p = new Scanner(file);

            String ID;

            while(p.hasNext()){

                ID = p.next();

                x = p.nextBigInteger();

                x = p.nextBigInteger();

                n = p.nextBigInteger();

                if(id.compareTo(ID) == 0){                  // User is already registered

                    flag = false;

                    break;

                }

            }

            p.close();

        }

        catch (FileNotFoundException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

        if(flag){

            set\_key(id);                                        // User is not registered .... therefore registering

            x = get\_private\_key(id);

        }

        return x;

    }

}

**RSA.java**

package CentralAuthority;

import java.io.DataInputStream;

import java.io.IOException;

import java.math.BigInteger;

import java.util.Random;

public class RSA {

    BigInteger public\_key,private\_key;

    private long  public\_key\_temp;

    private BigInteger p;

    private BigInteger q;

    private BigInteger n;

    private BigInteger phi;

    private BigInteger e;

    private BigInteger d;

    private int bitlength = 1024;

    private Random r;

    String ID;

    RSA(String ID){

        this.ID = ID ;

        public\_key\_temp = Math.abs(ID.hashCode());

        r = new Random();

        p = BigInteger.probablePrime(bitlength, r);

        q = BigInteger.probablePrime(bitlength, r);

        n = p.multiply(q);

    }

    public BigInteger get\_public\_key(){                     // generating public key

        phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

        e =  BigInteger.valueOf(public\_key\_temp);

        while (phi.gcd(e).compareTo(BigInteger.valueOf(1)) != 0 ) {

             e = e.divide(phi.gcd(e));

        }

        public\_key = e;

        get\_private\_key();

        return public\_key;

    }

    public BigInteger get\_private\_key(){                // generating private key

        d = public\_key.modInverse(phi);

        private\_key = extendedEuclid(public\_key,(this.p.subtract(BigInteger.ONE)).multiply(this.q.subtract(BigInteger.ONE)));

        return private\_key;

    }

    public BigInteger getn(){

        return n;

    }

    public BigInteger extendedEuclid(BigInteger a, BigInteger b) {

        BigInteger  x = BigInteger.valueOf(1), y =  BigInteger.valueOf(0);

        BigInteger xLast = BigInteger.valueOf(0), yLast = BigInteger.valueOf(0);

        BigInteger q, r, m, n;

        while(a.compareTo(BigInteger.valueOf(0)) != 0) {

            q = b.divide(a);

            r = b.remainder(a);

            m = xLast.subtract(q.multiply(x));

            n = yLast.subtract(q.multiply(y));

            xLast = x;

            yLast = y;

            x = m;

            y = n;

            b = a;

            a = r;

        }

        if(xLast.compareTo(BigInteger.valueOf(0))<0)

                xLast = xLast.add((this.p.subtract(BigInteger.ONE)).multiply(this.q.subtract(BigInteger.ONE)));

        return xLast;

    }

    public long power(long a, long b,long p) {                          // power funation a^b%p

        long  r = 1;

        while(b!=0) {

            if((b & 1) != 0) r = r \* a % p;

            a = (a \* a)% p;

            b >>= 1;

        }

        return r;

    }

    public BigInteger gcd1(BigInteger x,BigInteger y){              // computing gcd

        if(y.compareTo(BigInteger.valueOf(0)) == 0) return x;

        return gcd1(y,x.remainder(y));

    }

}

**Client code:**

**Client.java**

package Client;

import java.math.BigInteger;

import java.util.Scanner;

import CentralAuthority.PKG;

public class Client {

    public static void main(String args[]){

        String message,ID,temp,ID1;

        Scanner p = new Scanner (System.in);

        System.out.println("Hi ! ...Client");

        System.out.println("Enter the your ID  (for eg: xyz@gmail.com)");

        ID1 = p.next();

        System.out.println("Enter the User ID of Server (for eg: xyz@gmail.com)");

        ID = p.next();

        temp = p.nextLine();

        System.out.println("Enter the Message");

        temp = p.nextLine();

        message = ID1;

        message +=" has sent you a message:\n";

        message += temp;

        PKG pkg = new PKG();

        BigInteger Public\_key = pkg.get\_public\_key(ID1);

        Public\_key = pkg.get\_public\_key(ID);

        BigInteger n = pkg.getn();

        System.out.println("\npublic key of Server is : " + Public\_key);

        System.out.println("\n --------------------Encrypted message is -------------------------\n");

        Encrypt encrypt1 = new Encrypt(ID,message,n,Public\_key);

        encrypt1.encrypt();

    }

}

**Encrypt.java**

package Client;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.math.BigInteger;

import CentralAuthority.PKG;

public class Encrypt {

    private String Message,ID;

    BigInteger n,Public\_key;

    java.net.URL url = getClass().getResource("../Server/EncryptedMessage.txt");

    File file = new File(url.getPath());

    public Encrypt(String ID,String Message , BigInteger n , BigInteger Public\_key){

        this.ID = ID;

        this.Message = Message;

        this.n = n;

        this.Public\_key = Public\_key;

    }

 //Encrypt message

     public byte[] encrypt() {                          // Encrypting message

         byte[] message = Message.getBytes();

         System.out.println(bytesToString((new BigInteger(message)).modPow(Public\_key, n).toByteArray()));

         byte [] encryptedMessage = messageEncrypt(message);

         try(

                 FileWriter fileWriter = new FileWriter(file,true);

                 BufferedWriter bufferFileWriter  = new BufferedWriter(fileWriter);

            ) {

             fileWriter.append(ID);                                                 // Sending Message to server

             fileWriter.append(" ");

             fileWriter.append(Integer.toString(encryptedMessage.length));

             fileWriter.append(" ");

             for(byte e\_message : encryptedMessage){

                 fileWriter.append(Byte.toString(e\_message));

                 fileWriter.append(" ");

             }

            fileWriter.append("\n");

            bufferFileWriter.close();

            fileWriter.close();

        } catch (IOException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

        return (new BigInteger(message)).modPow(Public\_key, n).toByteArray();

    }

     private byte [] messageEncrypt(byte [] message){               // Actual message Encryption

         return (new BigInteger(message)).modPow(Public\_key, n).toByteArray();

     }

     private static String bytesToString(byte[] encrypted) {

         String test = "";

         for (byte b : encrypted) {

             test += Byte.toString(b);

             //test +=" ";

         }

         return test;

     }

}

**Server code:**

**Server.java**

package Server;

import java.math.BigInteger;

import java.util.Scanner;

import CentralAuthority.PKG;

public class Server {

public static void main(String args[]){

        String ID;

        Scanner p = new Scanner (System.in);

        System.out.println("Enter your User ID (for eg: abc@gmail.com)");

        ID = p.next();

        System.out.println("Hi... I am Server,\nSearching for messages\n");

        PKG pkg = new PKG();

        BigInteger Private\_key =pkg.get\_private\_key(ID);

        BigInteger n = pkg.getn();

        System.out.println("\nMy Private Key is :- " +   Private\_key );

        Decrypt decryptMessage = new Decrypt(ID,n,Private\_key);

        System.out.println("\n --------------------Decrypted message is -------------------------\n");

        decryptMessage.decrypt();

    }

}

**Decrypt.java**

package Server;

import java.io.File;

import java.io.FileNotFoundException;

import java.math.BigInteger;

import java.util.Scanner;

import Client.\*;

public class Decrypt {

    BigInteger n , private\_key;

    String ID;

    java.net.URL url = getClass().getResource("EncryptedMessage.txt");

    File file = new File(url.getPath());

    public Decrypt( String ID,BigInteger n , BigInteger private\_Key){

        this.ID = ID;

        this.private\_key = private\_Key;

        this.n = n;

    }

    public String decrypt() {                           // First Reading Message then crypting

        byte [] message = new byte[0];

        int k;

        boolean flag = false;

        try {

            Scanner p = new Scanner(file);

            while(p.hasNext()){

                    if(p.next().compareTo(ID) == 0){            // Message Found for this server

                        k = p.nextInt();

                        message  = new byte[k];

                        for(int i=0;i<k;i++){

                            message[i] = p.nextByte();

                        }

                        flag = true;

                        System.out.println(new String(decryptMessage(message)) + "\n");

                    }

                    else{

                        k = p.nextInt();

                        for(int i=0;i<k;i++){

                            p.nextByte();

                        }

                    }

            }

            p.close();

            if(flag)    return new String(decryptMessage(message));

        }

        catch (FileNotFoundException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

        System.out.println(new String("There is no message for you !!!!!!!"));

        return new String("There is no message for you !!!!!!!");   // No Message Found for this server

    }

    private static String bytesToString(byte[] encrypted) {

        String test = "";

        for (byte b : encrypted) {

            test += Byte.toString(b);

        }

        return test;

    }

    public byte[] decryptMessage(byte[] message) {          // Actual Message Decryption

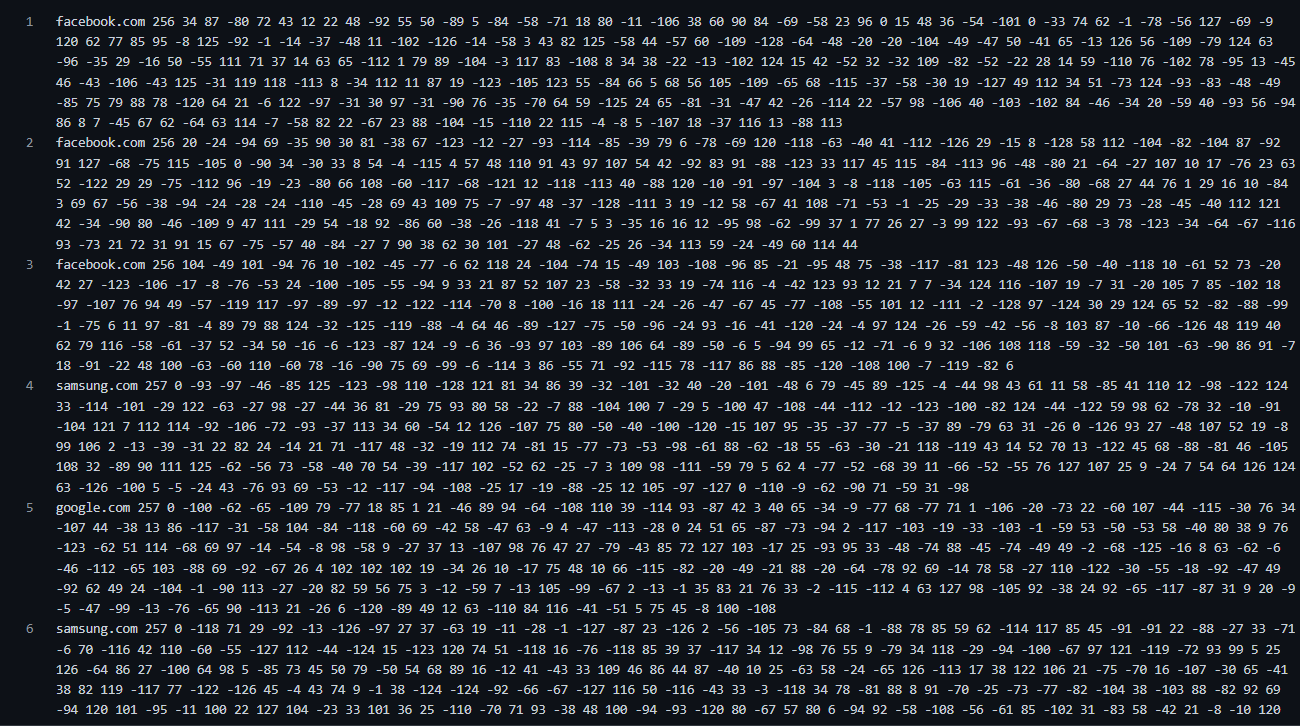
        return (new BigInteger(message)).modPow(private\_key, n).toByteArray();

    }

}

**OUTPUT**

**Encrypted message text**



**TO STUDY AND WORK WITH KF SENSOR INTRUSION DETECTION TOOL. SETUP A HONEYPOT AND MONITOR THE HONEYPOT ON THE NETWORK.**

Honey Pot is a device placed on Computer Network specifically designed to capture malicious network traffic. KF Sensor is the tool to setup as honeypot when KF Sensor is running it places a siren icon in the windows system tray in the bottom right of the screen. If there are no alerts then green icon is displayed.

**HONEY POT:**

A honeypot is a computer system that is set up to act as a decoy to lure cyber attackers, and to detect, deflect or study attempts to gain unauthorized access to information systems. Generally, it consists of a computer, applications, and data that simulate the behavior of a real system that appears to be part of a network but is actually isolated and closely monitored. All communications with a honeypot are considered hostile, as there's no reason for legitimate users to access a honeypot. Viewing and logging this activity can provide an insight into the level and types of threat a network infrastructure faces while distracting attackers away from assets of real value.

Honeypots can be classified based on their deployment (use/action) and based on their level of involvement. Based on deployment, honeypots may be classified as:

* Production honeypots
* 2. Research honeypots

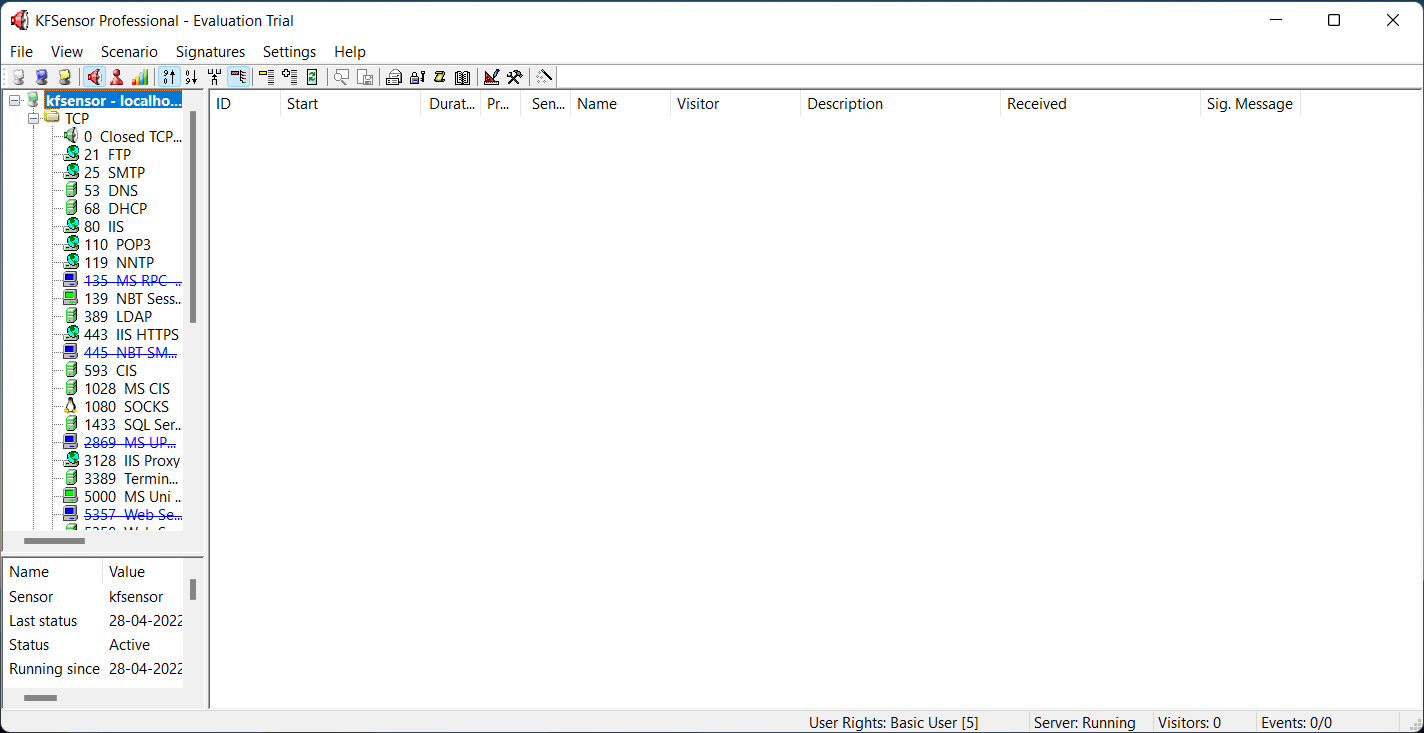
**Production honeypots** are easy to use, capture only limited information, and are used primarily by companies or corporations. Production honeypots are placed inside the production network with other production servers by an organization to improve their overall state of security. Normally, production honeypots are low-interaction honeypots, which are easier to deploy. They give less information about the attacks or attackers than research honeypots.

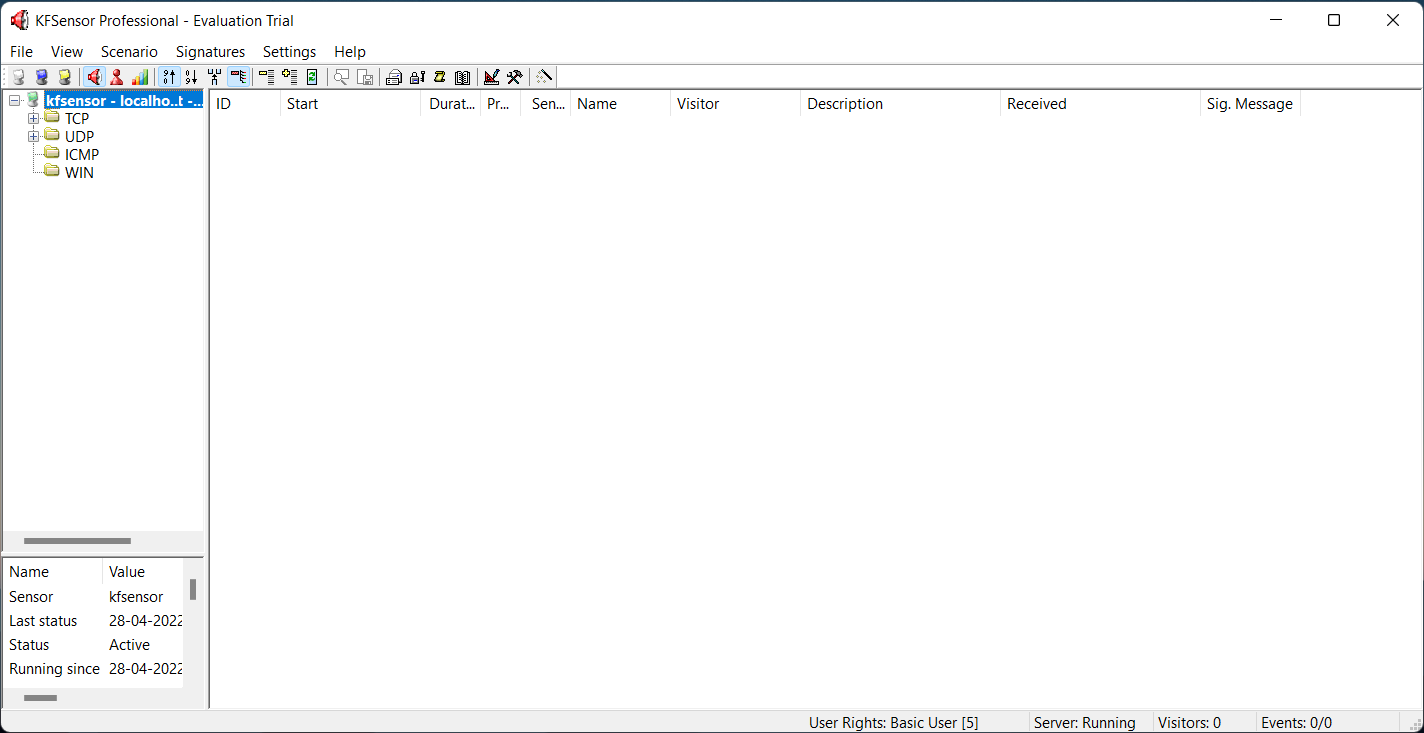
**Research honeypots** are run to gather information about the motives and tactics of the Black hat community targeting different networks. These honeypots do not add direct value to a specific organization; instead, they are used to research the threats that organizations face and to learn how to better protect against those threats.

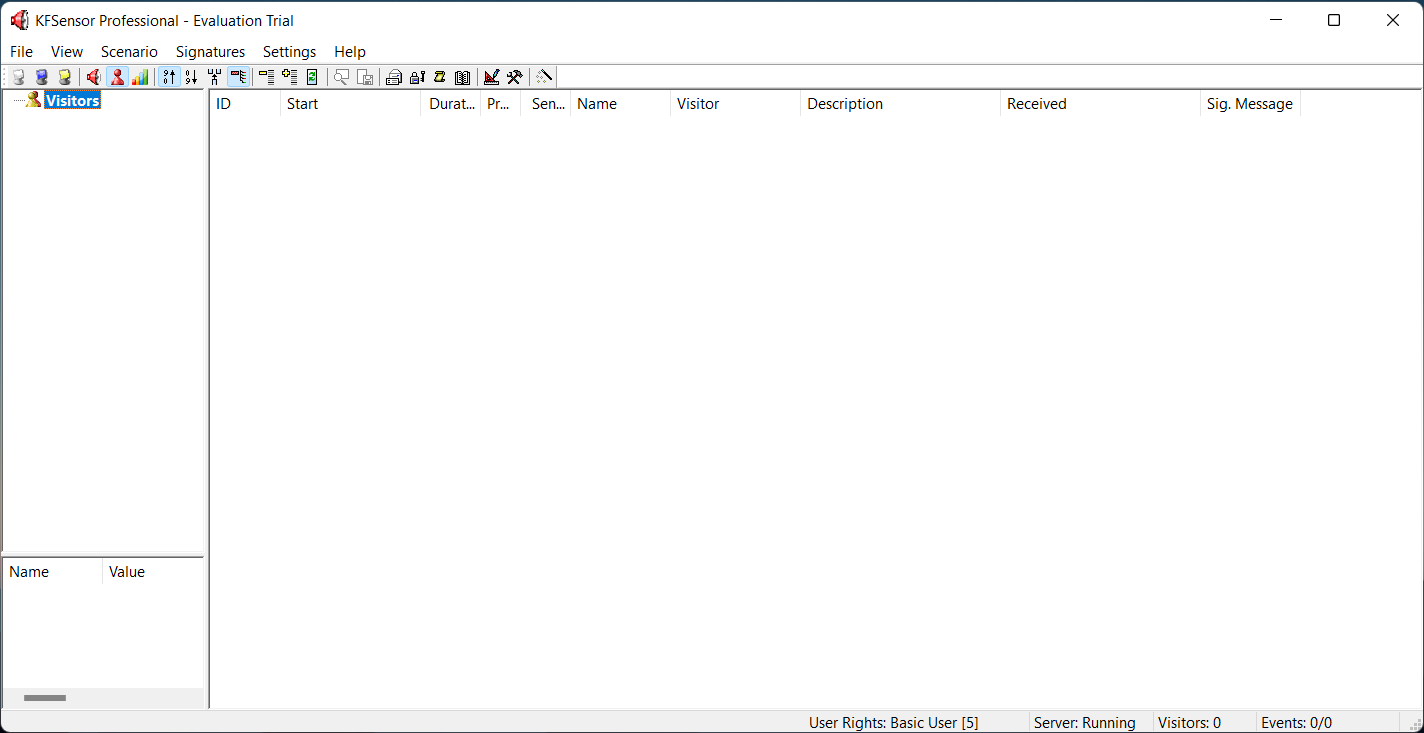
**KF SENSOR:**

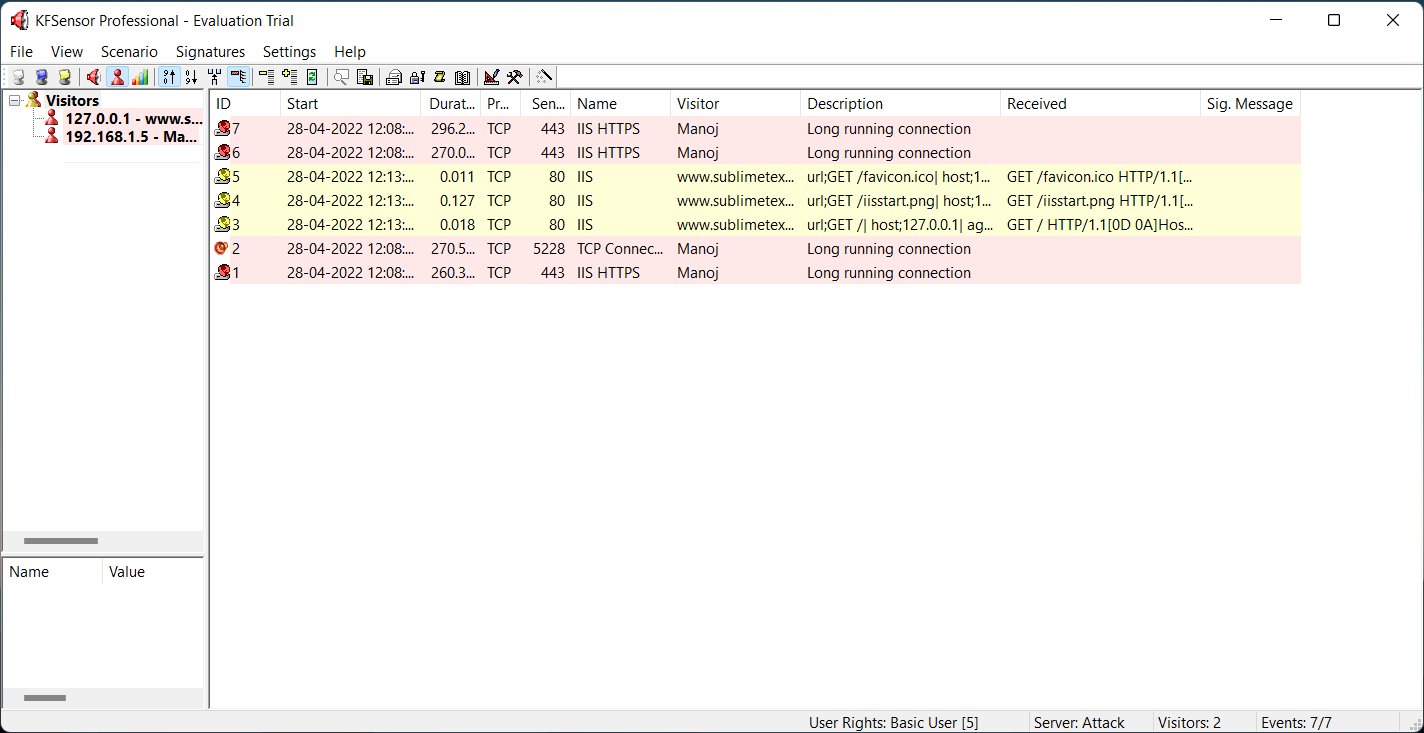
KFSensor is a Windows based honeypot Intrusion Detection System (IDS). It acts as a honeypot to attract and detect hackers and worms by simulating vulnerable system services and trojans. By acting as a decoy server it can divert attacks from critical systems and provide a higher level of information than can be achieved by using firewalls and NIDS alone. KFSensor is a system installed in a network in order to divert and study an attacker’s behavior. This is a new technique that is very effective in detecting attacks.

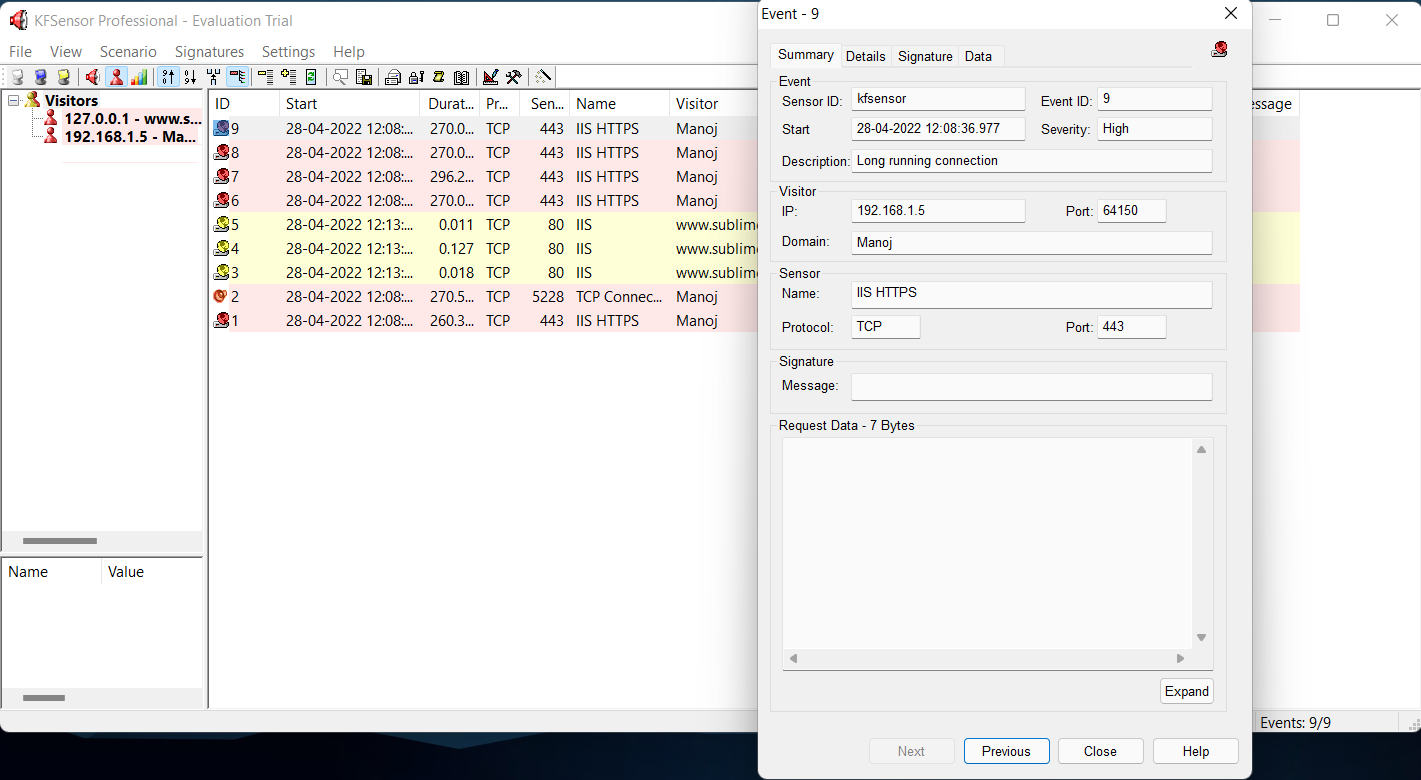
The main feature of KFSensor is that every connection it receives is a suspect hence it results in very few false alerts. At the heart of KFSensor sits a powerful internet daemon service that is built to handle multiple ports and IP addresses. It is written to resist denial of service and buffer overflow attacks. Building on this flexibility KFSensor can respond to connections in a variety of ways, from simple port listening and basic services (such as echo), to complex simulations of standard system services. For the HTTP protocol KFSensor accurately simulates the way Microsoft’s web server (IIS) responds to both valid and invalid requests. As well as being able to host a website it also handles complexities such as range requests and client side cache negotiations. This makes it extremely difficult for an attacker to fingerprint, or identify KFSensor as a honeypot.









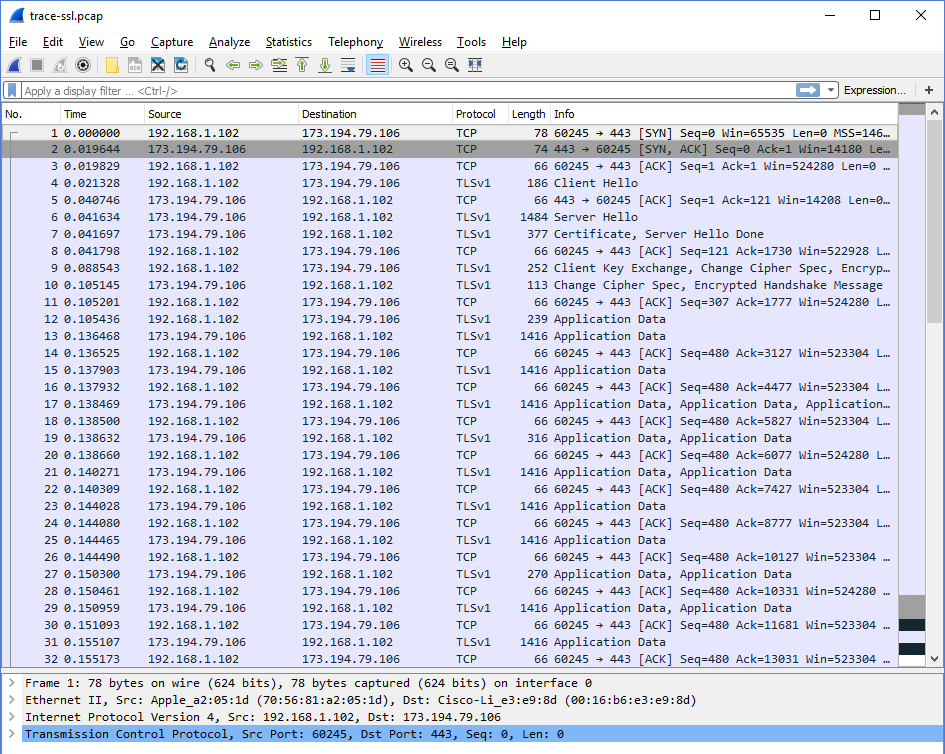


**CONFIGURE WIRESHARK WITH A KEY TO LET YOU LOOK INSIDE ENCRYPTED SSL MESSAGES. YOU CAN READ ON THE WEB HOW TO DO THIS. ONCE DECRYPTED, YOU WILL BE ABLE TO OBSERVE THE HTTP PROTOCOL RUNNING ON TOP OF SSL, AS WELL AS THE DETAILS OF OTHER SSL MESSAGES SUCH AS ALERTS.**

Step 1: Open a Trace

1. Open the Wireshark trace <https://kevincurran.org/com320/labs/wireshark/trace-ssl.pcap>

You should see the following trace.

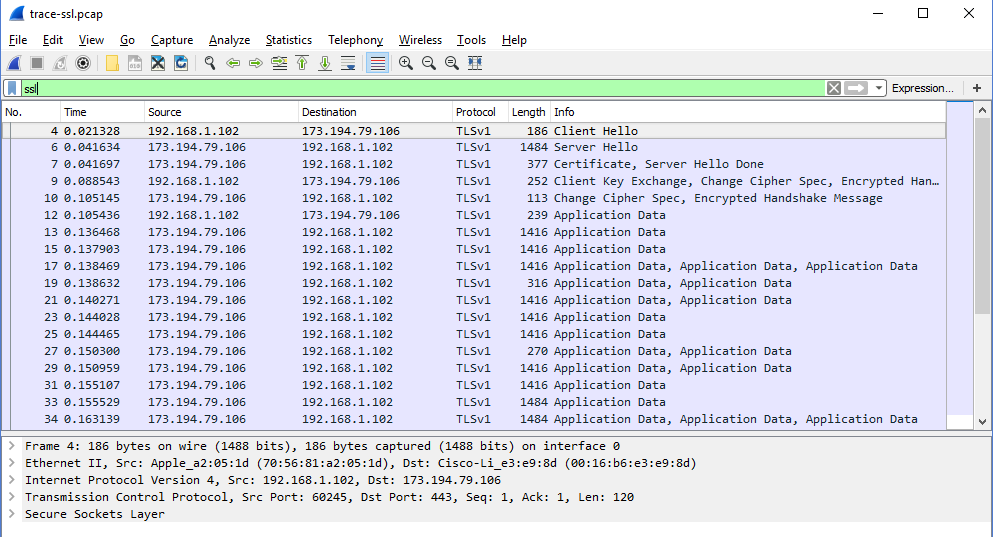


Step 2: Inspect the Trace

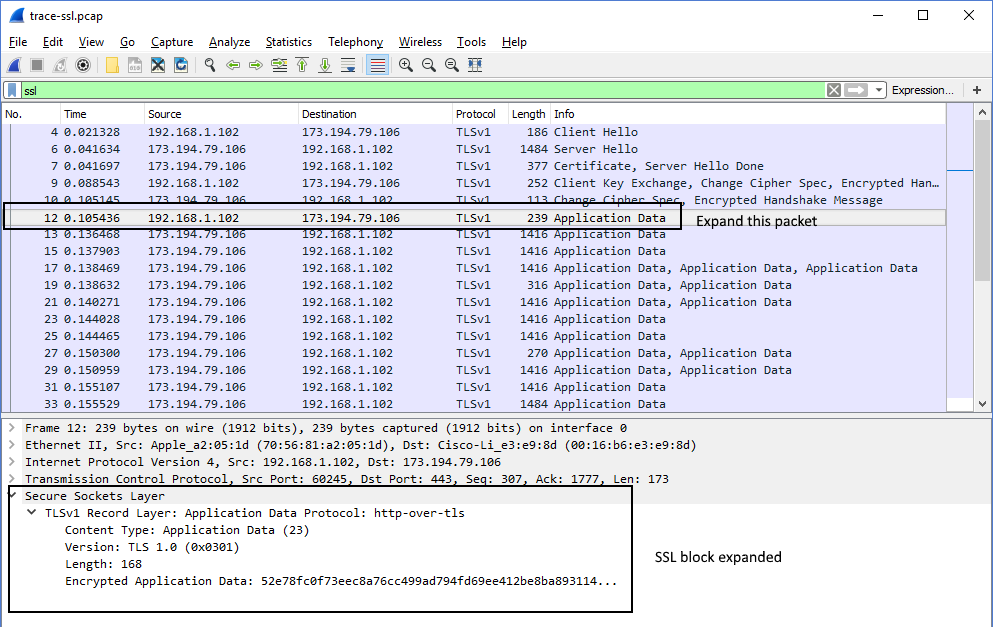
Now we are ready to look at the details of some “SSL” messages.

2. To begin, enter and apply a display filter of “ssl”. (see below)

This filter will help to simplify the display by showing only SSL and TLS messages. It will exclude other TCP segments that are part of the trace, such as Acks and connection open/close



3. Select a TLS message somewhere in the middle of your trace for which the Info reads “Application Data” & expand its Secure Sockets Layer block (by using the “+” expander or icon). For in- stance, packet #12 (see below).



Application Data is a generic TLS message carrying contents for the application, such as the web page. It is a good place for us to start looking at TLS messages.

The lower layer protocol blocks are TCP and IP because SSL runs on top of TCP/IP. The SSL layer contains a “TLS Record Layer”. This is the foundational sublayer for TLS. All messages contain records. Expand this block to see its details. Each record starts with a Content Type field. This tells us what is in the contents of the record. Then comes a Version identifier. It will be a constant value for the SSL connection. It is followed by a Length field giving the length of the record. Last comes the contents of the record. Application Data records are sent after SSL has secured the connection, so the contents will show up as encrypted data. To see within this block, we could configure Wireshark with the decryption key. This is possible, but outside of our scope. Note that, unlike other protocols we will see such as DNS, there may be multiple records in a single message. Each record will show up as its own block. Look at the Info column, and you will see messages with more than one block.

The Content-Type for a record containing “Application Data” is 23. The version constant used in this trace is 0x0301 which represents TLS 1.0. The Length covers only the payload of the Record Layer.

Step 3: The SSL Handshake

An important part of SSL is the initial handshake that establishes a secure connection. The handshake proceeds in several phases. There are slight differences for different versions of TLS and depending on the encryption scheme that is in use. The usual outline for a brand-new connection is:

a. Client (the browser) and Server (the web server) both send their Hellos

b. Server sends its certificate to Client to authenticate (and optionally asks for Client Certificate)

c. Client sends keying information and signals a switch to encrypted data.

d. Server signals a switch to encrypted data.

e. Both Client and Server send encrypted data.

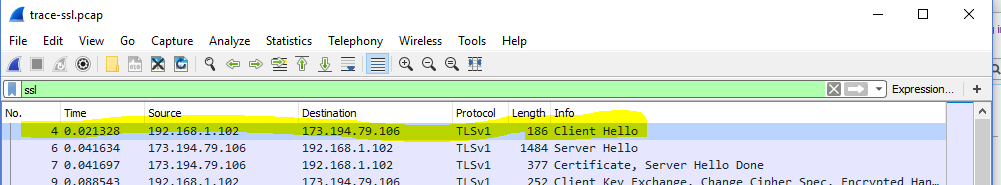
f. An Alert is used to tell the other party that the connection is closing.

Note that there is also a mechanism to resume sessions for repeat connections between the same client and server to skip most of steps b and c. However, we will not study session resumption.

**Hello Messages**

Next we will find and inspect the details of the Client Hello and Server Hello messages, including expanding the Handshake protocol block within the TLS Record. For these initial messages, an encryption scheme is not yet established so the contents of the record are visible to us. They contain details of the secure connection setup in a Handshake protocol format.

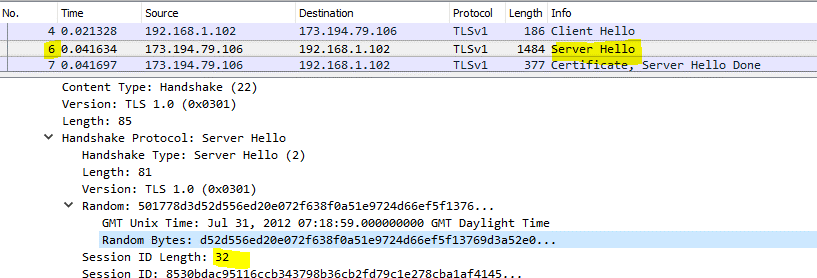
4. Select packet #4, which is a TLS Client Hello message



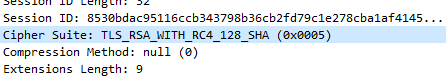
We can see several important fields here worth mentioning. First, the time (GMT seconds since midnight Jan 1, 1970) and random bytes (size 28) are included. This will be used later in the protocol to generate our symmetric encryption key. The client can send an optional session ID to quickly resume a previous TLS connection and skip portions of the TLS handshake. Arguably the most important part of the ClientHello message is the list of cipher suites, which dictate the key exchange algorithm, bulk encryption algorithm (with key length), MAC, and a psuedo-random function. The list should be ordered by client preference. The collection of these choices is a “cipher suite”, and the server is responsible for choosing a secure one it supports or return an error if it doesn’t support any. The final field specified in the specification is for compression methods. However, secure clients will advertise that they do not support compression (by passing “null” as the only algorithm) to avoid the CRIME attack. Finally, the ClientHello can have a number of different extensions. A common one is server\_name, which specifies the host- name the connection is meant for, so webservers hosting multiple sites can present the correct certificate.

5. Select packet #6, which is a TLS Server Hello message

The session ID sent by the server is 32 bytes long. This identifier allows later resumption of the session with an abbreviated handshake when both the client and server indicate the same value. In our case, the client likely sent no session ID as there was nothing to resume (see below)

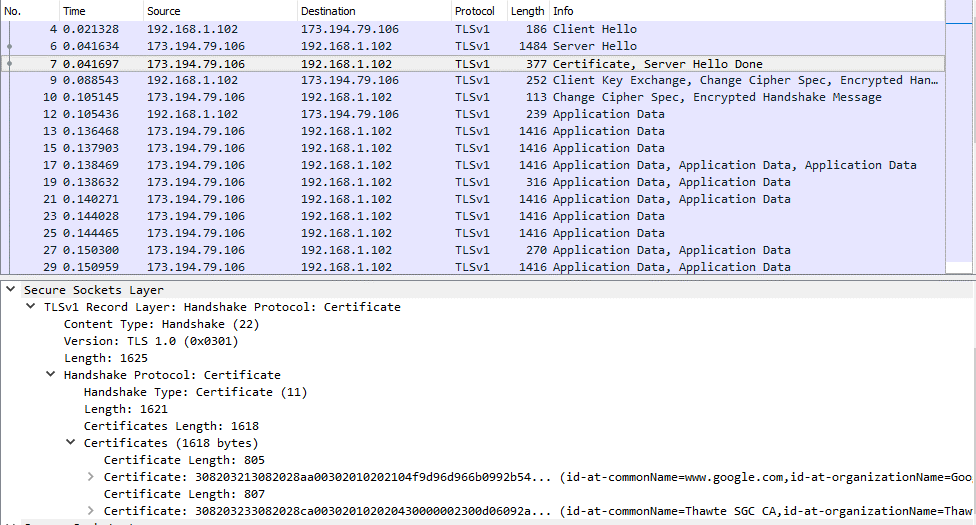


The Cipher method chosen by the Server is TLS\_RSA\_WITH\_RC4\_128\_SHA (0x0005). The Client will list the different cipher methods it supports, and the Server will pick one of these methods to use



Certificate Messages

6. Next, find and inspect the details of the Certificate message including expanding the Handshake protocol block within the TLS Record (see below for expansion of packet #7).



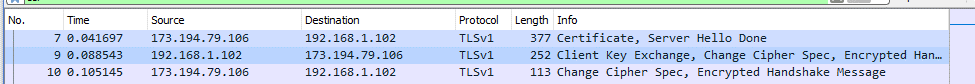
As with the Hellos, the contents of the Certificate message are visible because an encryption scheme is not yet established. It should come after the Hello messages.

Note it is the server that sends a certificate to the client, since it is the browser that wants to verify the identity of the server. It is also possible for the server to request certificates from the client, but this behavior is not normally used by web applications.

A Certificate message will contain one or more certificates, as needed for one party to verify the identity of the other party from its roots of trust certificates. You can inspect those certificates in your browser

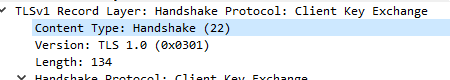
Client Key Exchange and Change Cipher Messages

7. Find and inspect the details of the Client Key Exchange and Change Cipher messages i.e. packet #9 (see below)

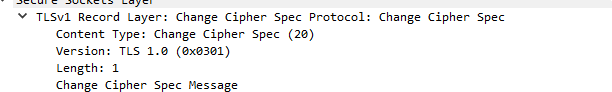


The key exchange message is sent to pass keying information so that both sides will have the same se- cret session key. The change cipher message signal a switch to a new encryption scheme to the other party. This means that it is the last unencrypted message sent by the party.

Note how the Client Key Exchange has a Content-Type of 22, indicating the Handshake protocol. This is the same as for the Hello and Certificate messages, as they are part of the Handshake protocol.



The Change Cipher Spec message has a Content-Type of 20, indicating the Change Cipher Spec protocol (see packet #10 – see below).



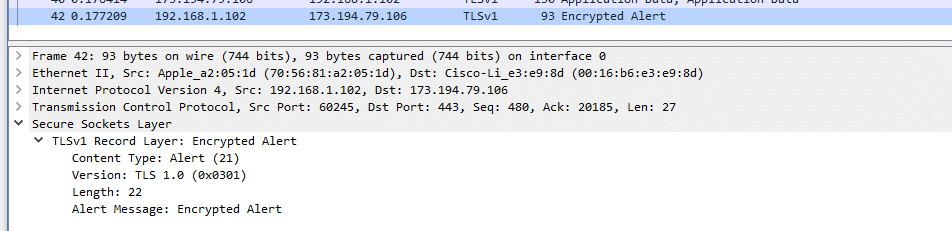
That is, this message is part of its own protocol and not the Handshake protocol.

Both sides send the Change Cipher Spec message immediately before they switch to sending encrypted contents. The message is an indication to the other side. The contents of the Change Cipher Spec message are simply the value 1 as a single byte. Actually, it is the value “1” encrypted under the current scheme, which uses no encryption for the handshake so that we can see it.

Alert Message

8. Finally, find and inspect the details of an Alert message at the end of the trace (packet #42).

The Alert message is sent to signal a condition, such as notification that one party is closing the connection. You should find an Alert after the Application Data messages that make up the secure web fetch.



Note, the Content-Type value is 21 for Alert. This is a new protocol, different from the Handshake, Change Cipher Spec and Application Data values that we have already seen.

The alert is encrypted; we cannot see its contents. Wireshark also describes the message as an “Encrypted Alert”. Presumably is it a “close\_notify” alert to signal that the connection is ending, but we cannot be certain.

**TO BUILD A TROJAN AND KNOW THE HARMNESS OF THE TROJAN MALWARES IN A COMPUTER SYSTEM. WHEN THE TROJAN CODE EXECUTES, IT WILL OPEN MS-PAINT, NOTEPAD, COMMAND PROMPT, EXPLORER, CALCULATOR, INFINITELY. NOTE: USE VMWARE TO PERFORM THIS EXPERIMENT.**

A Trojan horse, or Trojan, is a type of malicious code or software that looks legitimate but can take control of your computer. A Trojan is designed to damage, disrupt, steal, or in general inflict some other harmful action on your data or network.

A Trojan acts like a bona fide application or file to trick you. It seeks to deceive you into loading and executing the malware on your device. Once installed, a Trojan can perform the action it was designed for.

A Trojan is sometimes called a Trojan virus or a Trojan horse virus, but that’s a misnomer. Viruses can execute and replicate themselves. A Trojan cannot. A user has to execute Trojans. Even so, Trojan malware and Trojan virus are often used interchangeably.

Whether you prefer calling it Trojan malware or a Trojan virus, it’s smart to know how this infiltrator works and what you can do to keep your devices safe.

**How do Trojans work?**

Here’s a Trojan malware example to show how it works. You might think you’ve received an email from someone you know and click on what looks like a legitimate attachment. But you’ve been fooled. The email is from a cybercriminal, and the file you clicked on — and downloaded and opened — has gone on to install malware on your device.

When you execute the program, the malware can spread to other files and damage your computer.

How? It varies. Trojans are designed to dodifferent things. But you’ll probably wish they weren’t doing any of them on your device.

**Types of Trojan malware, from A to Z**

Here’s a look at some of the most common types of Trojan malware, including their names and what they do on your computer:

**Backdoor Trojan** This Trojan can create a “backdoor” on your computer.

It lets an attacker access your computer and control it. Your data can be downloaded by a third party and stolen. Or more malware can be uploaded to your device.

**Distributed Denial of Service (DDoS) attack Trojan** This Trojan performs DDoS attacks. The idea is to take down a network by flooding it with traffic. That traffic comes from your infected computer and others.

**Downloader Trojan** This Trojan targets your already-infected computer. It downloads and installs new versions of malicious programs. These can include Trojans and adware.

**Fake AV Trojan**: This Trojan behaves like antivirus software, but demands money from you to detect and remove threats, whether they’re real or fake.

**Game-thief Trojan** The losers here may be online gamers. This Trojan seeks to steal their account information.

**Infostealer Trojan** As it sounds, this Trojan is after data on your infected computer.

**Mailfinder Trojan** This Trojan seeks to steal the email addresses you’ve accumulated on your device.

**Ransom Trojan** This Trojan seeks a ransom to undo damage it has done to your computer. This can include blocking your data or impairing your computer’s performance.

**Remote Access Trojan** This Trojan can give an attacker full control over your computer via a remote network connection. Its uses include stealing your information or spying on you.

**Rootkit Trojan** A rootkit aims to hide or obscure an object on your infected computer. The idea? To extend the time a malicious program runs on your device.

**SMS Trojan** This type of Trojan infects your mobile device and can send and intercept text messages. Texts to premium-rate numbers can drive up your phone costs.

**TO WORK WITH SNORT TOOL TO DEMONSTRATE INTRUSION DETECTION SYSTEM. DOWNLOAD SNORT FROM SNORT.ORG.**

Snort is an open-source network intrusion detection system (NIDS) and it is a packet sniffer that monitors network traffic in real time.

**INTRUSION DETECTION SYSTEM**:

Intrusion detection is a set of techniques and methods that are used to detect suspicious activity both at the network and host level.

Intrusion detection systems fall into two basic categories:

* Signature-based intrusion detection systems
* Anomaly detection systems.

Intruders have signatures, like computer viruses, that can be detected using software. You try to find data packets that contain any known intrusion-related signatures or anomalies related to Internet protocols. Based upon a set of signatures and rules, the detection system is able to find and log suspicious activity and generate alerts.

Anomaly-based intrusion detection usually depends on packet anomalies present in protocol header parts. In some cases these methods produce better results compared to signature-based IDS. Usually an intrusion detection system captures data from the network and applies its rules to that data or detects anomalies in it. Snort is primarily a rule-based IDS, however input plug-ins are present to detect anomalies in protocol headers.

**SNORT TOOL:** Snort is based on libpcap (for library packet capture), a tool that is widely used in TCP/IPtraffic sniffers and analyzers. Through protocolanalysis and content searching and matching, Snort detects attack methods, including denial of service, buffer overflow, CGI attacks, stealthport scans, and SMB probes. When suspicious behavior is detected, Snort sends a real-time alert to syslog, a separate 'alerts' file, or to apop-up window. Snort is currently the most popular free network intrusion detection software.

The advantages of Snort are numerous. According to the snort web site, “It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflow, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more” (Caswell).

One of the advantages of Snort is its ease of configuration. Rules are very flexible, easily written, and easily inserted into the rule base. If a new exploit or attack is found a rule for the attack can be added to the rule base in a matter of seconds. Another advantage of snort is that it allows for raw packet data analysis.

**SNORT can be configured to run in three modes:**

1. Sniffer mode

2. Packet Logger mode

3. Network Intrusion Detection System mode

**1. Sniffer mode**

* Snort –v Print out the TCP/IP packets header on the screen
* Snort –vd show the TCP/IP ICMP header with application data in transmit

**2. Packet Logger mode**

* snort –dev –l c:\log [create this directory in the C drive] and snort will

automatically know to go into packet logger mode, it collects every packet it

sees and places it in log directory.

* snort –dev –l c:\log –h ipaddress/24:This rule tells snort that you want to

print out the data link and TCP/IP headers as well as application data into the

log directory. snort –l c:\log –b This is binary mode logs everything into a

single file.

**3. Network Intrusion Detection System mode**

* snort –d c:\log –h ipaddress/24 –c snort.conf This is a configuration file

applies rule to each packet to decide it an action based upon the rule type in

the file.

* Snort –d –h ipaddress/24 –l c:\log –c snort.conf This will cnfigure snort to

run in its most basic NIDS form, logging packets that trigger rules specifies in

the snort.conf.

**PROCEDURE:**

STEP-1: Sniffer mode snort –v Print out the TCP/IP packets header on the screen.

STEP-2: Snort –vd Show the TCP/IP ICMP header with application data in transit.

STEP-3: Packet Logger mode snort –dev –l c:\log [create this directory in the C drive] and snort will automatically know to go into packet logger mode, it collects every packet it sees and places it in log directory.

STEP-4: snort –dev –l c:\log –h ipaddress/24 This rule tells snort that you want to print out the data link and TCP/IP headers as well as application data into the log

directory.

STEP-5: snort –l c:\log –b this binary mode logs everything into a single file.

STEP-6: Network Intrusion Detection System mode snort –d c:\log –h ipaddress/24 –c snort.conf This is a configuration file that applies rule to each packet to decide it an action based upon the rule type in the file.

STEP-7: snort –d –h ip address/24 –l c:\log –c snort.conf This will configure snort to run in its most basic NIDS form, logging packets that trigger rules specifies in the

snort.conf.

STEP-8: Download SNORT from snort.org. Install snort with or without database support.

STEP-9: Select all the components and Click Next. Install and Close.

STEP-10: Skip the WinPcap driver installation.

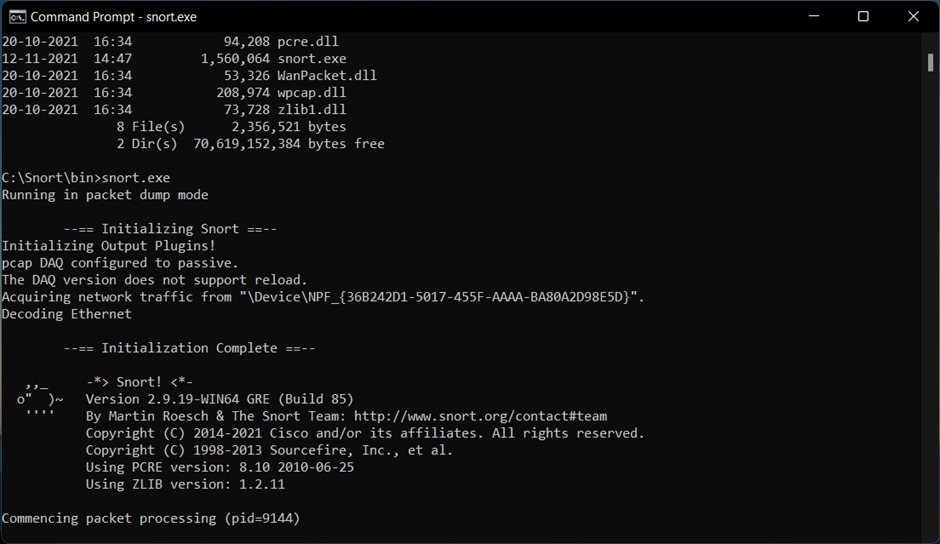
STEP-11: Add the path variable in windows environment variable by selecting new

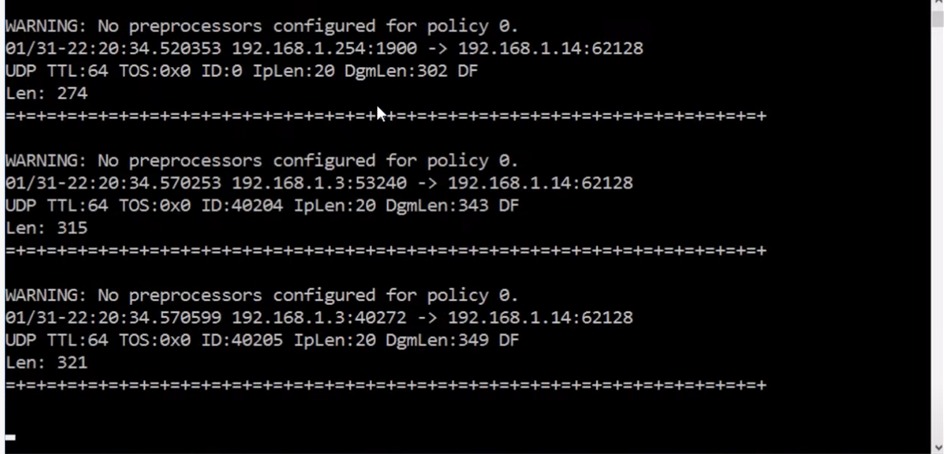
classpath.

STEP-12: Create a path variable and point it at snort.exe variable name path and variable value c:\snort\bin.

STEP-13: Click OK button and then close all dialog boxes. Open command prompt and type the following commands

OUTPUT





**IMPLEMENT A CODE TO SIMULATE BUFFER OVERFLOW ATTACK.**

#include<bits/stdc++.h>

int main(void)

{

    char buff[15];

    int pass = 0;

    printf("\n Enter the password :");

    gets(buff);

    if(strcmp(buff, "mypassword"))

    {

        printf ("Wrong Password \n");

    }

    else

    {

        printf ("Correct Password \n");

        pass = 1;

    }

    if(pass)

    {

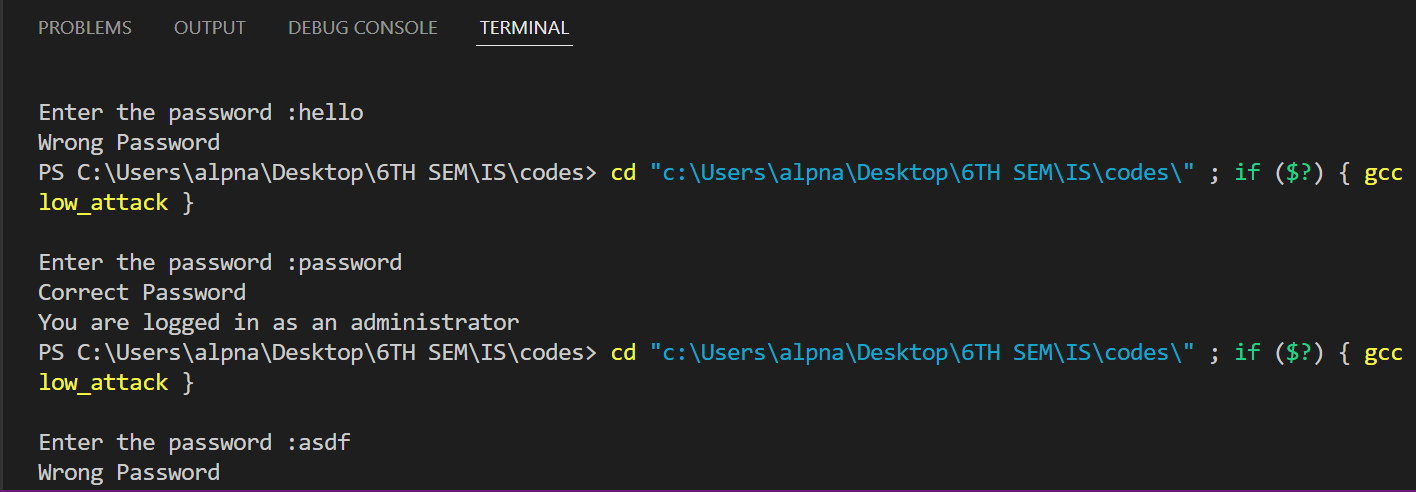
        printf ("You are logged in as an administrator\n");

    }

    return 0;

}

**OUTPUT**



**USE THE NESSUS TOOL TO SCAN THE NETWORK FOR VULNERABILITIES**

Step 1: Creating a Scan

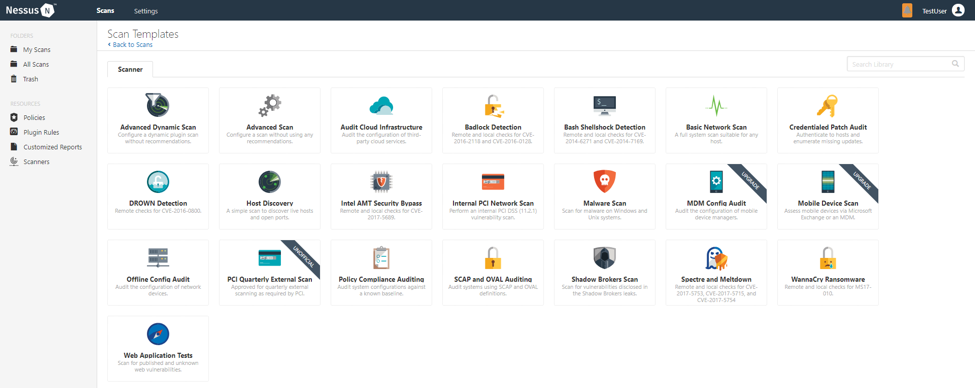
Once you have installed and launched Nessus, you’re ready to start scanning.

First, you have to create a scan. To create your scan:

• In the top navigation bar, click Scans.

• In the upper-right corner of the My Scans page, click the New Scan button.

Step 2: Choose a Scan Template



Next, click the scan template you want to use. Scan templates simplify the process by determining which settings are configurable and how they can be set. For a detailed explanation of all the options available, refer to Scan and Policy Settings in the Nessus User Guide.

A scan policy is a set of predefined configuration options related to performing a scan. After you create a policy, you can select it as a template in the User Defined tab when you create a scan. For more information, see Create a Policy in the Nessus User Guide.

The Nessus interface provides brief explanations of each template in the product. Some templates are only available when you purchase a fully licensed copy of Nessus Professional.

To see a full list of the types of templates available in Nessus, see Scan and Policy Templates. To quickly get started with Nessus, use the Basic Network Scan template

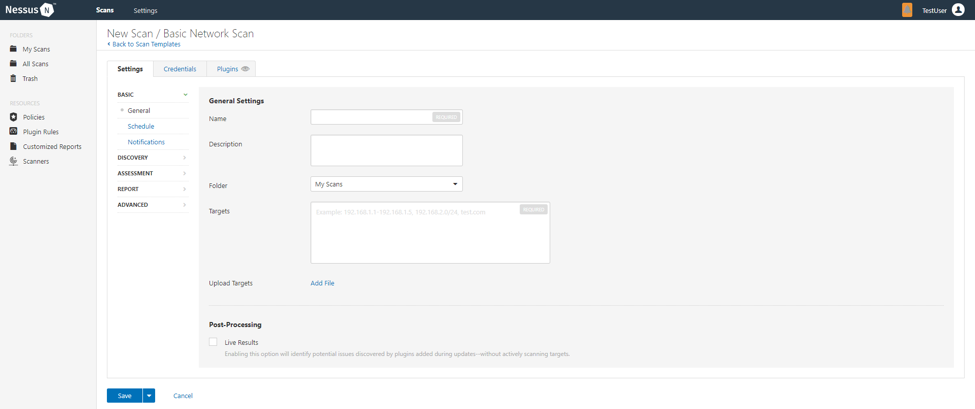
Step 3: Configure Scan Settings

Prepare your scan by configuring the settings available for your chosen template.

The Basic Network Scan template has several default settings preconfigured, which allows you to quickly perform your first scan and view results without a lot of effort.

Follow these steps to run a basic scan:

1. Configure the settings in the Basic Settings section.

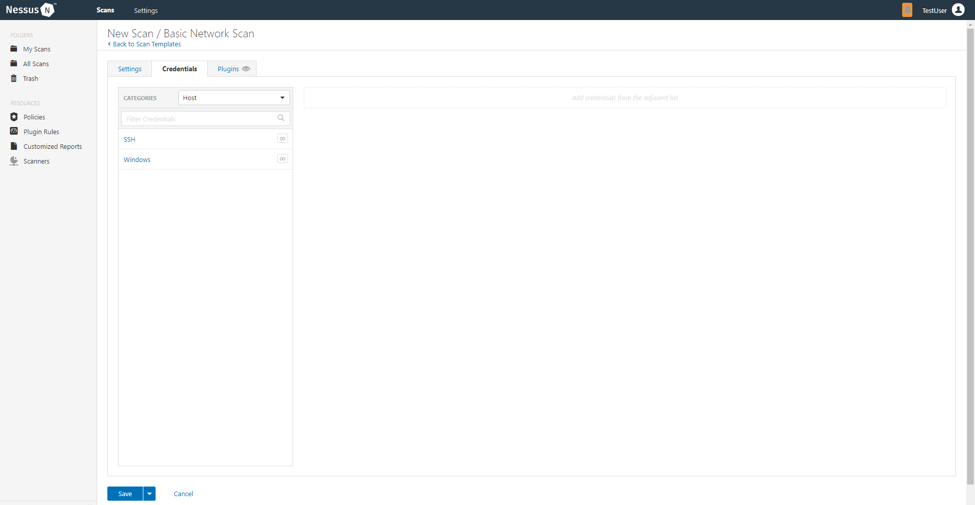


2. Configure remaining settings

Although you can leave the remaining settings at their pre-configured default, Tenable recommends reviewing the Discovery, Assessment, Report and Advanced settings to ensure they are appropriate for your environment.

For more information, see the Scan Settings documentation in the Nessus User Guide.

3. Configure Credentials Optionally, you can configure Credentials for a scan. This allows credentialed scans to run, which can provide much more complete results and a more thorough evaluation of the vulnerabilities in your environment



4. Launch Scan

After you have configured all your settings, you can either click the Save button to launch the scan later, or launch the scan immediately

If you want to launch the scan immediately, click the button, and then click Launch. Launching the scan will also save it.

The time it takes to complete a scan involves many factors, such as network speed and congestion, so the scan may take some time to run.

Step 4: Viewing Your Results

Viewing scan results can help you understand your organization’s security posture and vulnerabilities. Color-coded indicators and customizable viewing options allow you to tailor how you view your scan’s data.

You can view scan results in one of several



Viewing scan results by vulnerabilities gives you a view into potential risks on your assets

**To view vulnerabilities:**

1. In the top navigation bar, click Scans.

2. Click the scan for which you want to view results.

3. Do one of the following:

* Click a specific host to view vulnerabilities found on that host.
* Click the Vulnerabilities tab to view all vulnerabilities.

4. (Optional) To sort the vulnerabilities, click an attribute in the table header row to sort by that attribute.

5. Clicking on the vulnerability row will open the vulnerability details page, displaying plugin information and output for each instance on a host.

