INFORMATION SECURITY

PRACTICAL FILE

SUBMITTED BY

AYUSH GOEL

2019UIT3107

IT-2

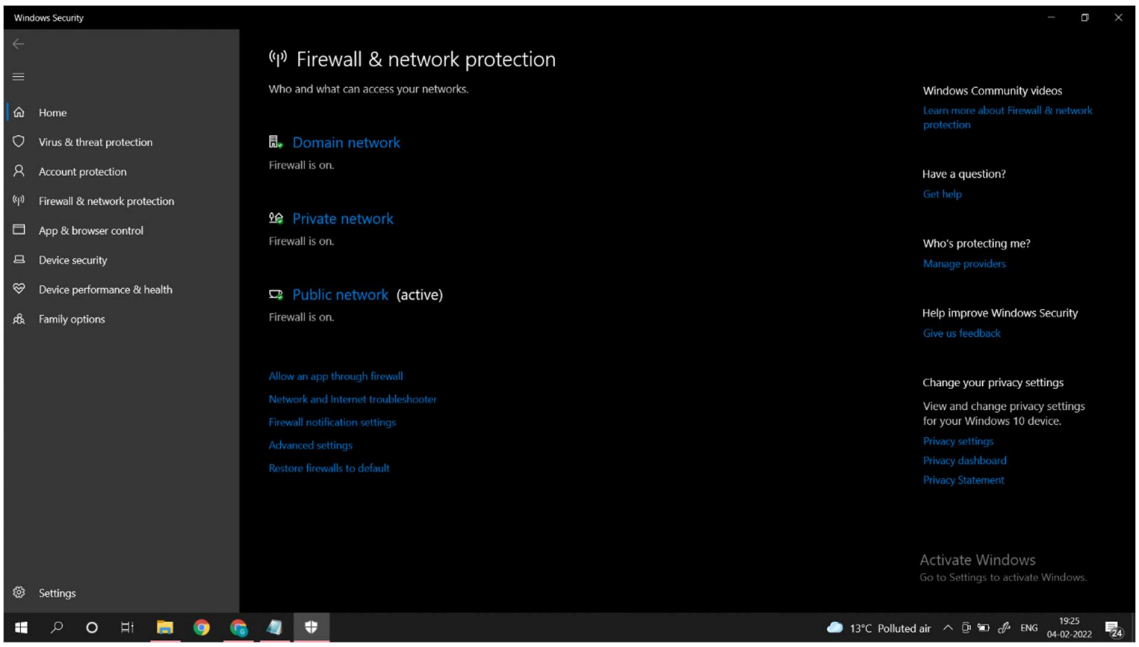
Practical – 1

Study the features of firewalls in providing network security and to set Firewall Security in windows.

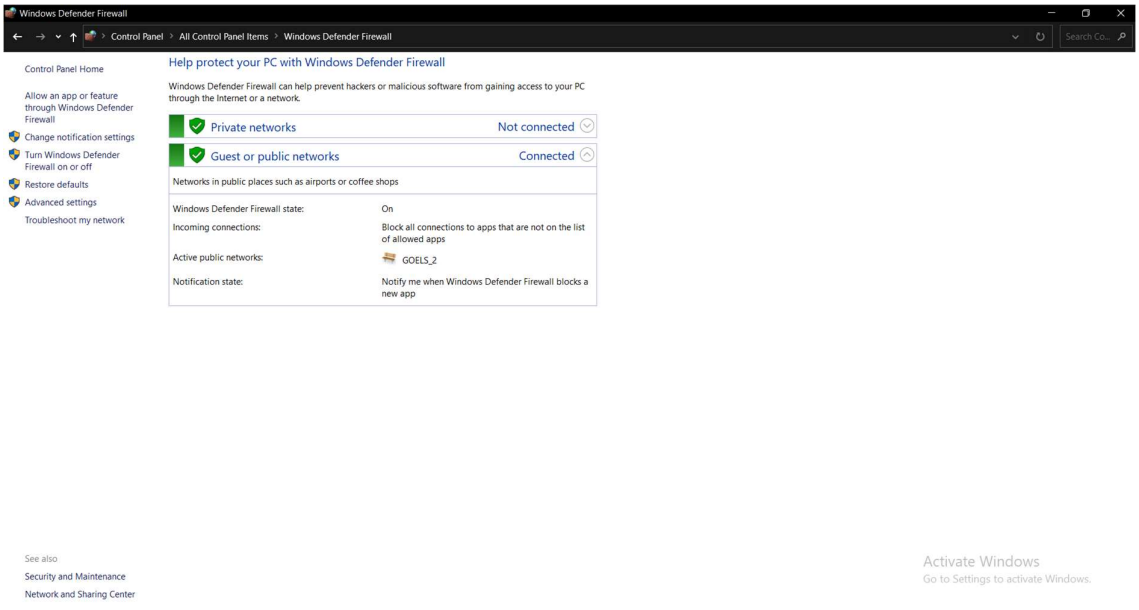
Students should know the following terms:

a) Know about 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 and use windows Firewall with advanced security.

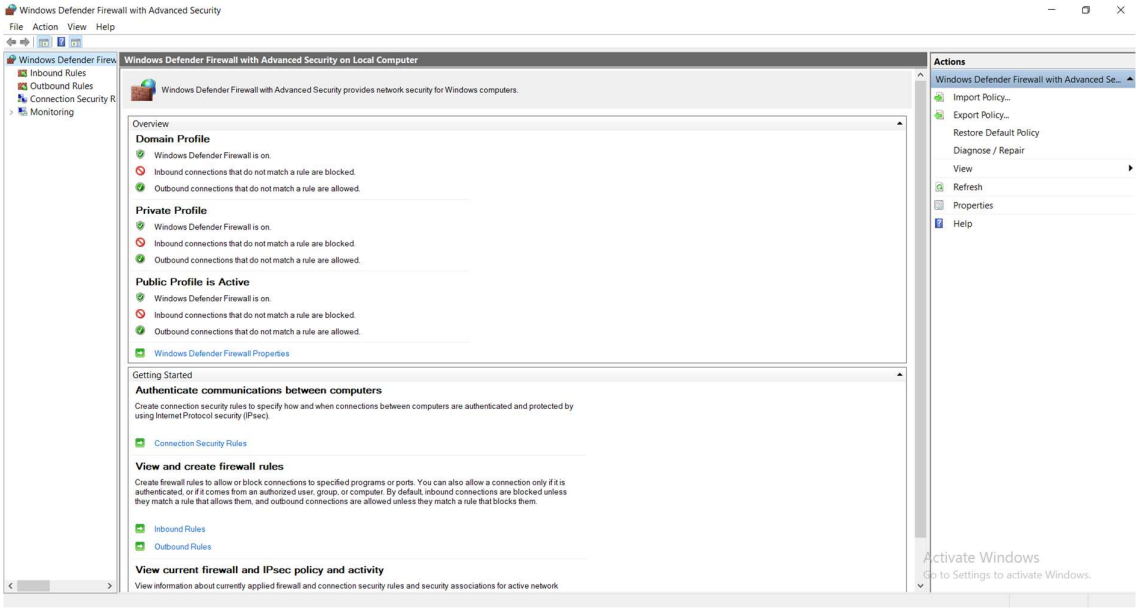
Ayush Goel
2019UIT3107
IT-2
Sem-6



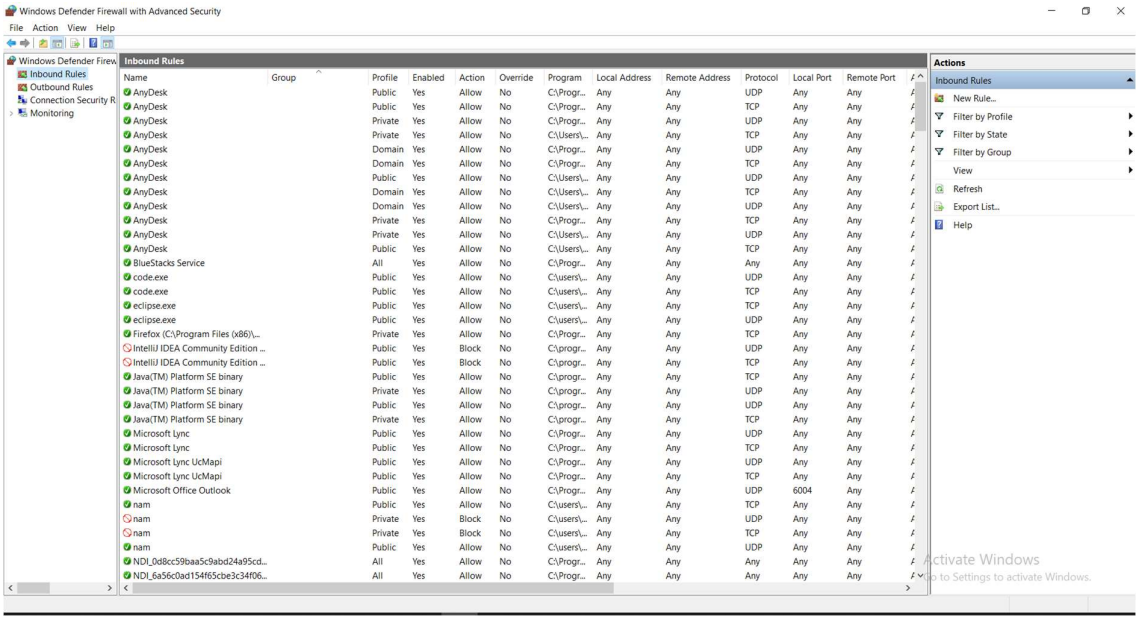
1)Open up the Firewall and network connection.



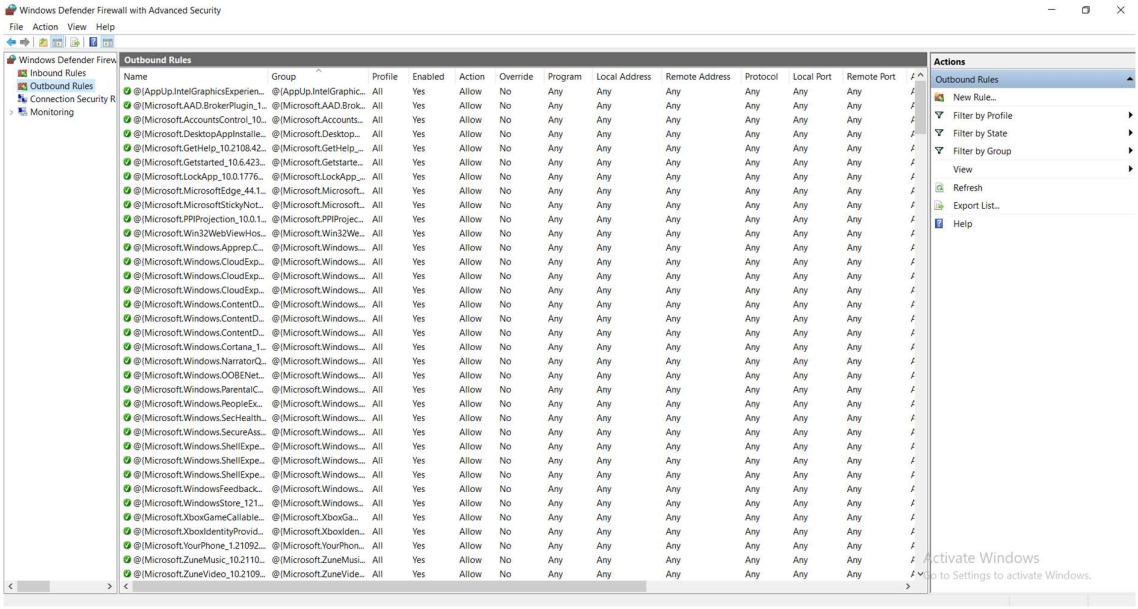
2) Go to Systema and security.

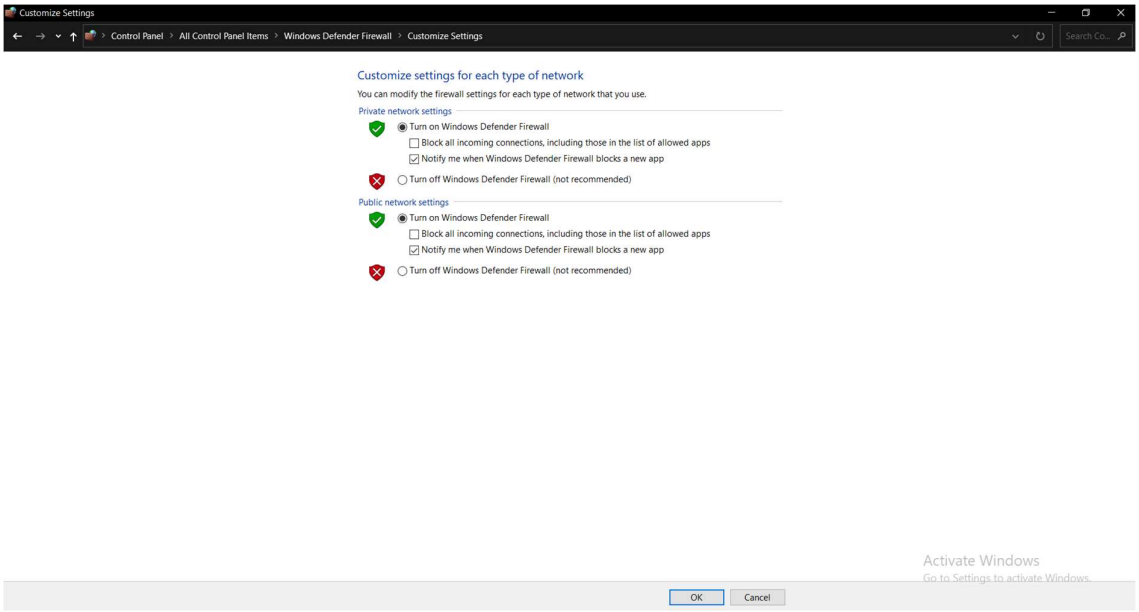


3)Open up Windows Defender Firewall and from here we can either switch on or off the windows defender firewall.



4)From Here we can see the inbound and outbound rules for system security.





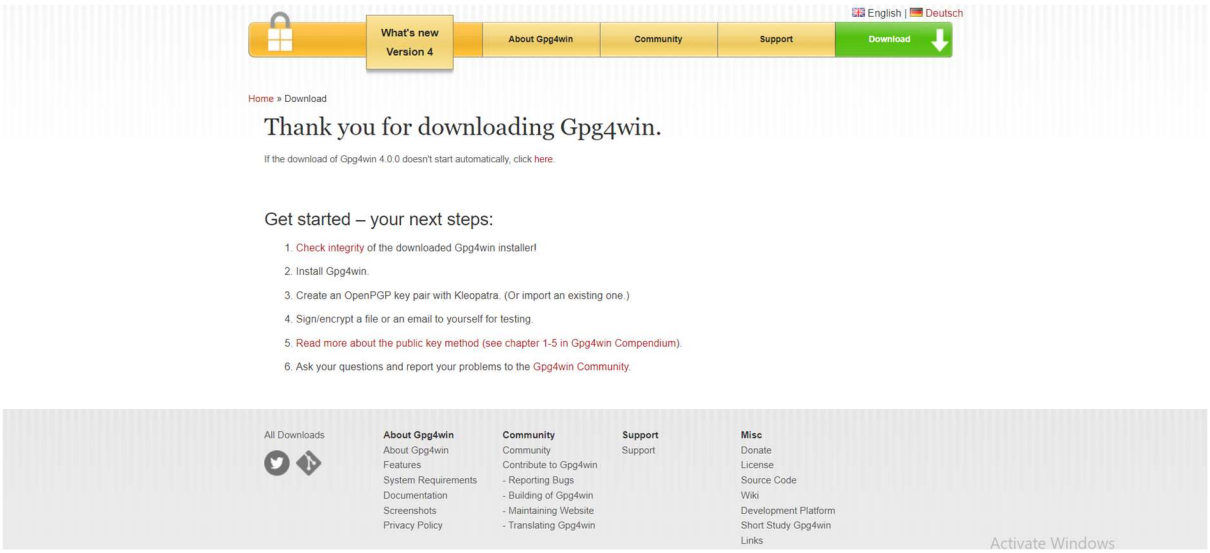
5)From here, we can access the type of firewalls for public and private networks.
This is not generally recommended.

Practical – 2

Demonstrate how to provide secure data storage, secure data transmission and for creating digital security using tool GnuPG. (Download GPG4WIN Tool)

Create your own public and private key using Kleopatra certificate management software. Check encryption and decryption of a piece of text.

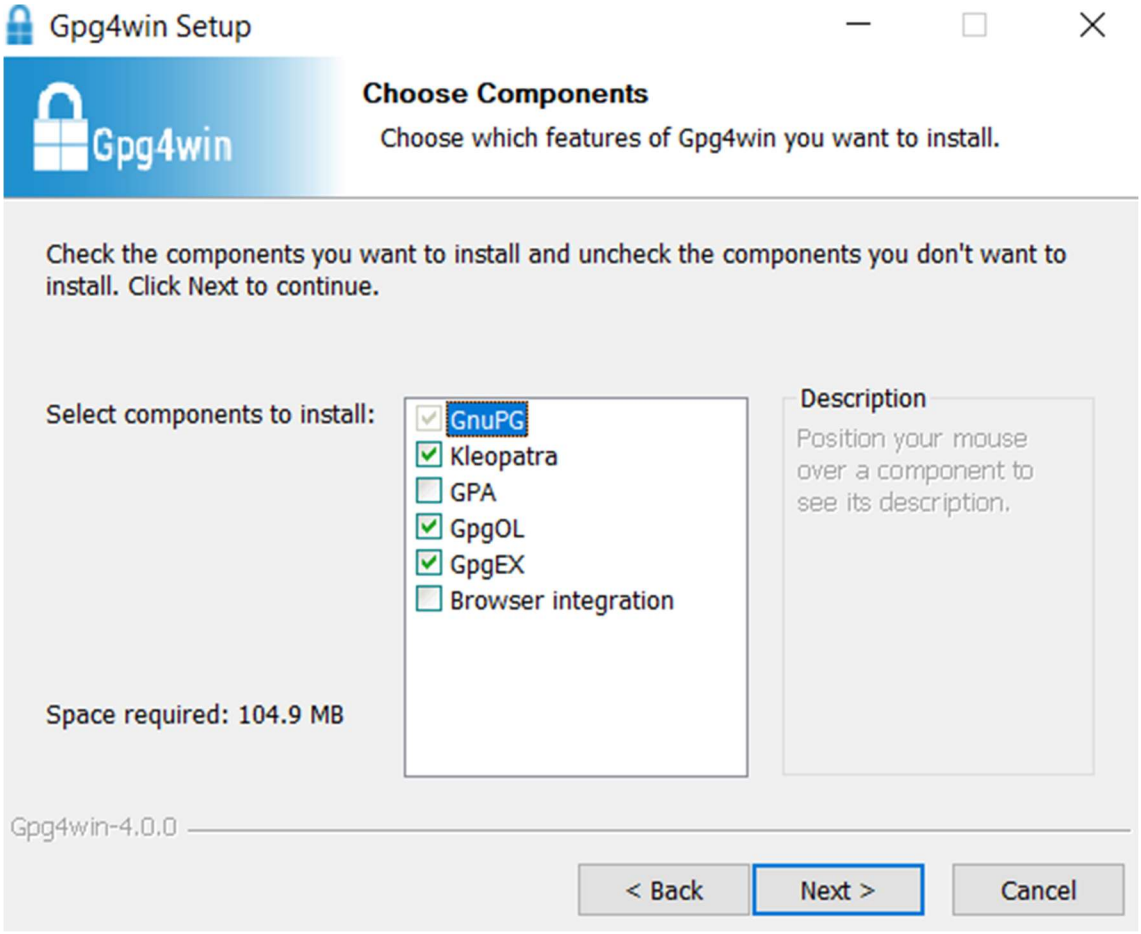
Ayush Goel
2019UIT3107
IT-2
Sem-6

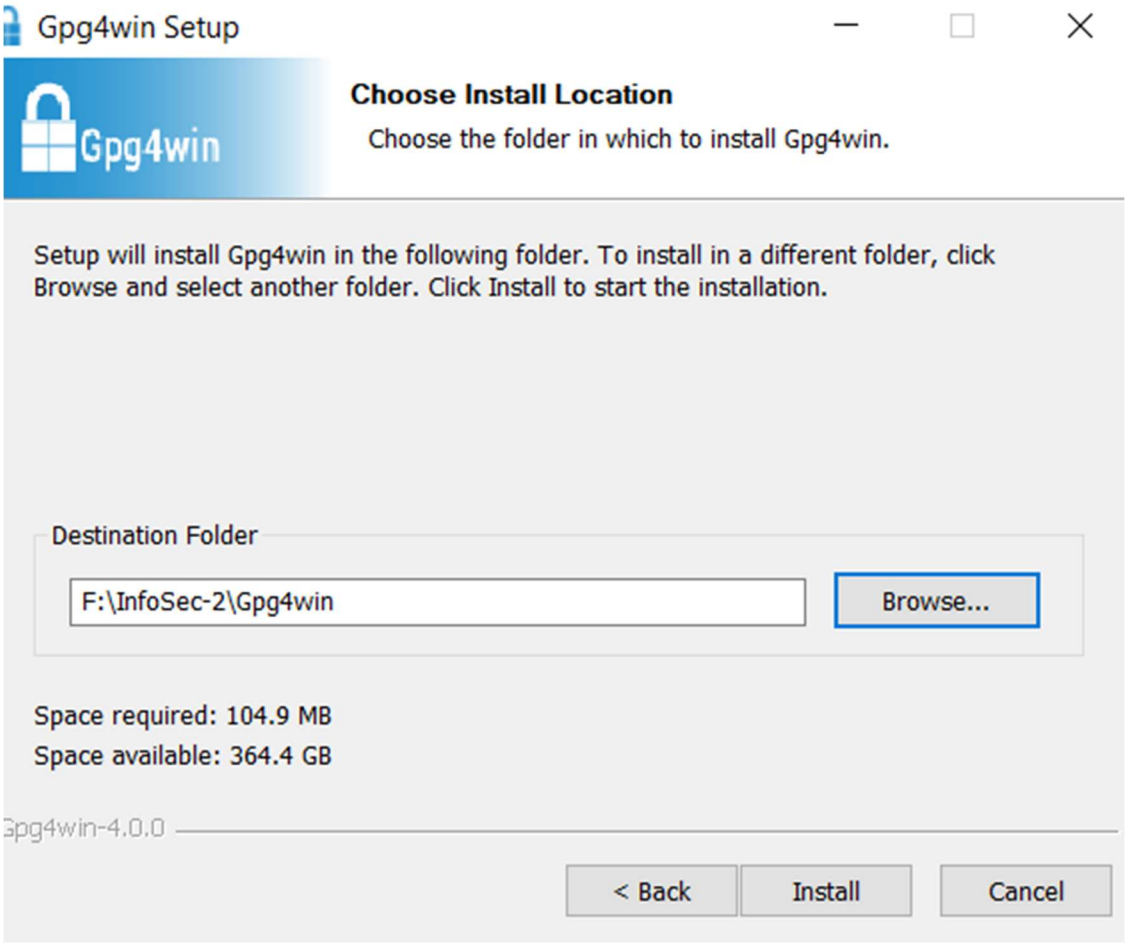


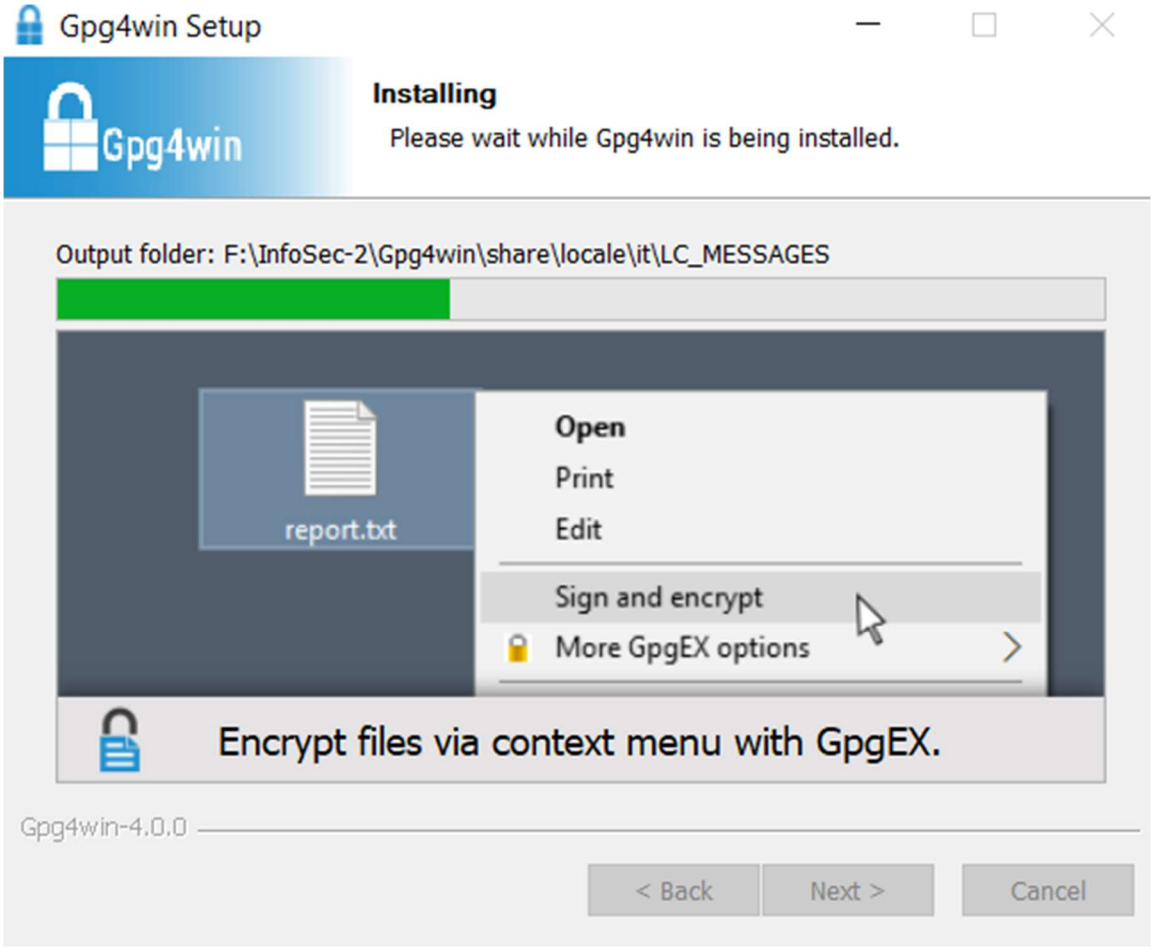
1) Download Gpg4 win.

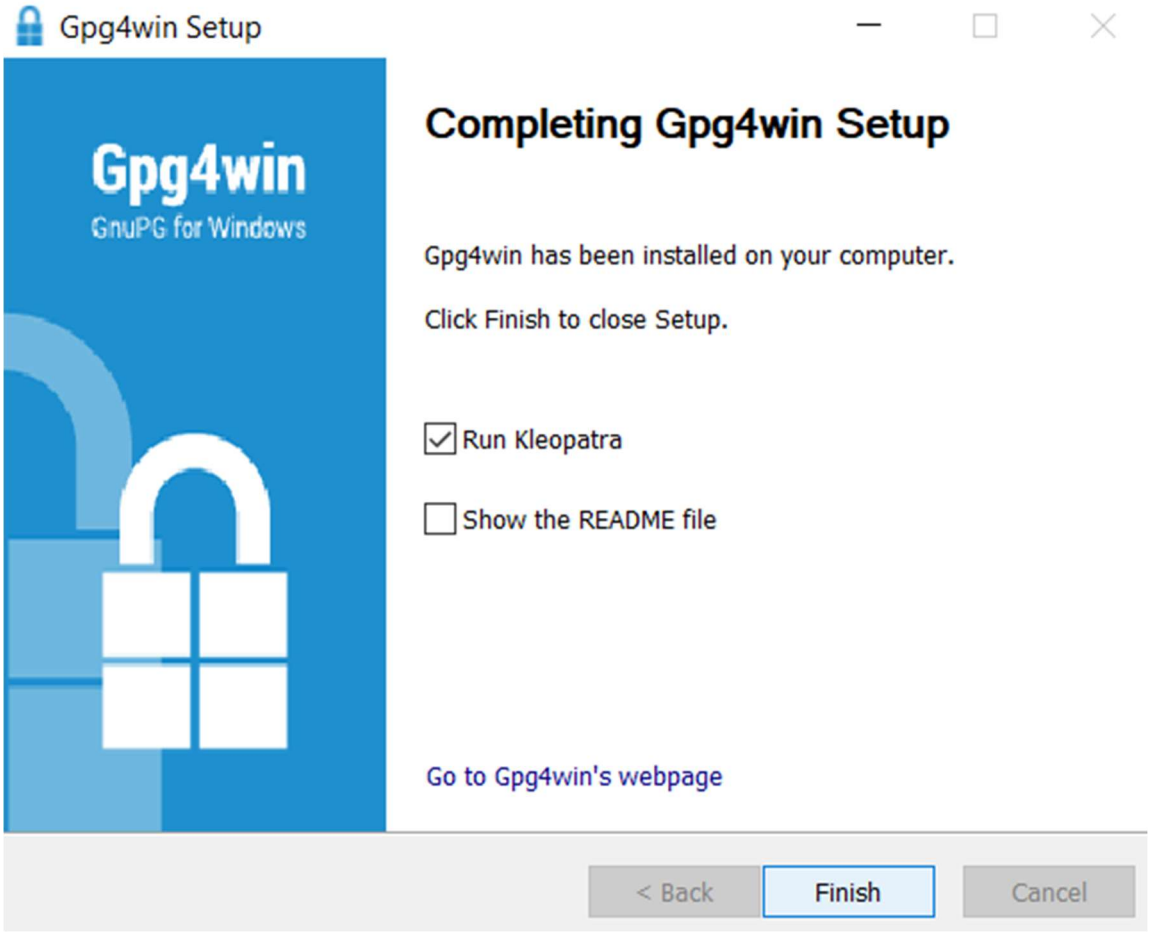
Installation Steps


Gpg4win Setup
Gpg4win
GnuPG for Windows
Welcome to the installation of Gpg4win
Gpg4win is an installer package for Windows for EMail and file encryption using the core component GnuPG for Windows. Both relevant cryptography standards are supported, OpenPGP and S/MIME. Gpg4win and the software included with Gpg4win is Free Software.
Click Next to continue.
This is Gpg4win version 4.0.0
Release date 2021-12-21
Next >
Cancel

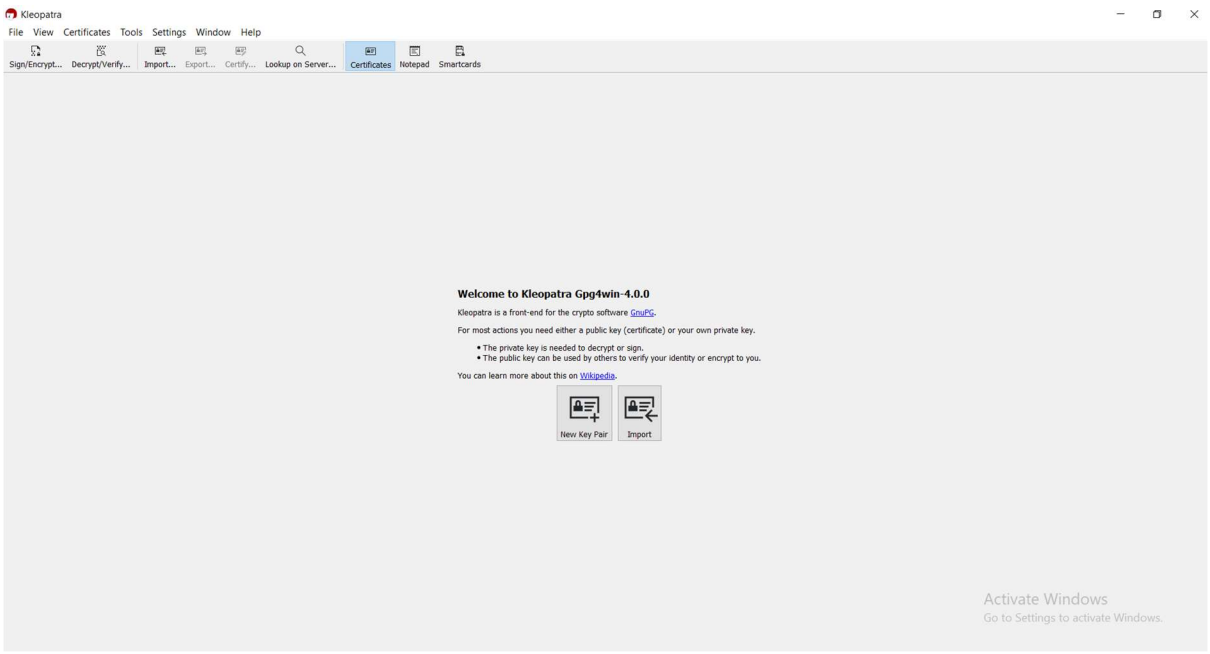

Gpg4win Setup
Gpg4win
Choose Components
Choose which features of Gpg4win you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
GnuPG
Kleopatra
GPA
GpgOL
GpgEX
Browser integration
Description
Position your mouse over a component to see its description.
Space required: 104.9 MB
Gpg4win-4.0.0
< Back
Next >
Cancel


Gpg4win Setup
Gpg4win
Choose Install Location
Choose the folder in which to install Gpg4win.
Setup will install Gpg4win in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
F:\InfoSec-2\Gpg4win
Browse...
Space required: 104.9 MB
Space available: 364.4 GB
Gpg4win-4.0.0
< Back
Install
Cancel

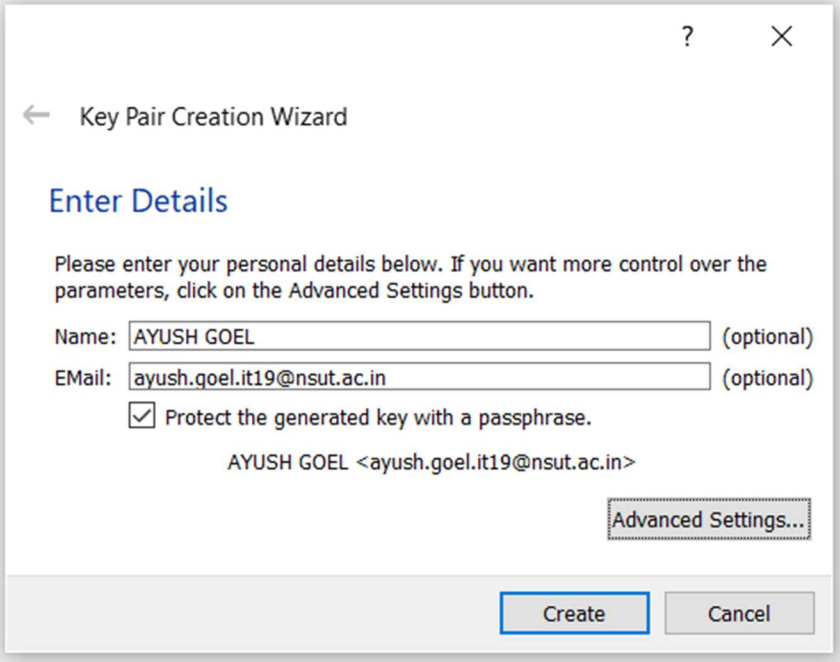

Gpg4win Setup
Gpg4win
Installing
Please wait while Gpg4win is being installed.
Output folder: F:\InfoSec-2\Gpg4win\share\locale\it\LC_MESSAGES
report.txt
Open
Print
Edit
Sign and encrypt
More GpgEX options
Encrypt files via context menu with GpgEX.
Gpg4win-4.0.0
< Back
Next >
Cancel


Gpg4win Setup
Gpg4win
GnuPG for Windows
Completing Gpg4win Setup
Gpg4win has been installed on your computer.
Click Finish to close Setup.
Run Kleopatra
Show the README file
Go to Gpg4win's webpage
< Back
Finish
Cancel

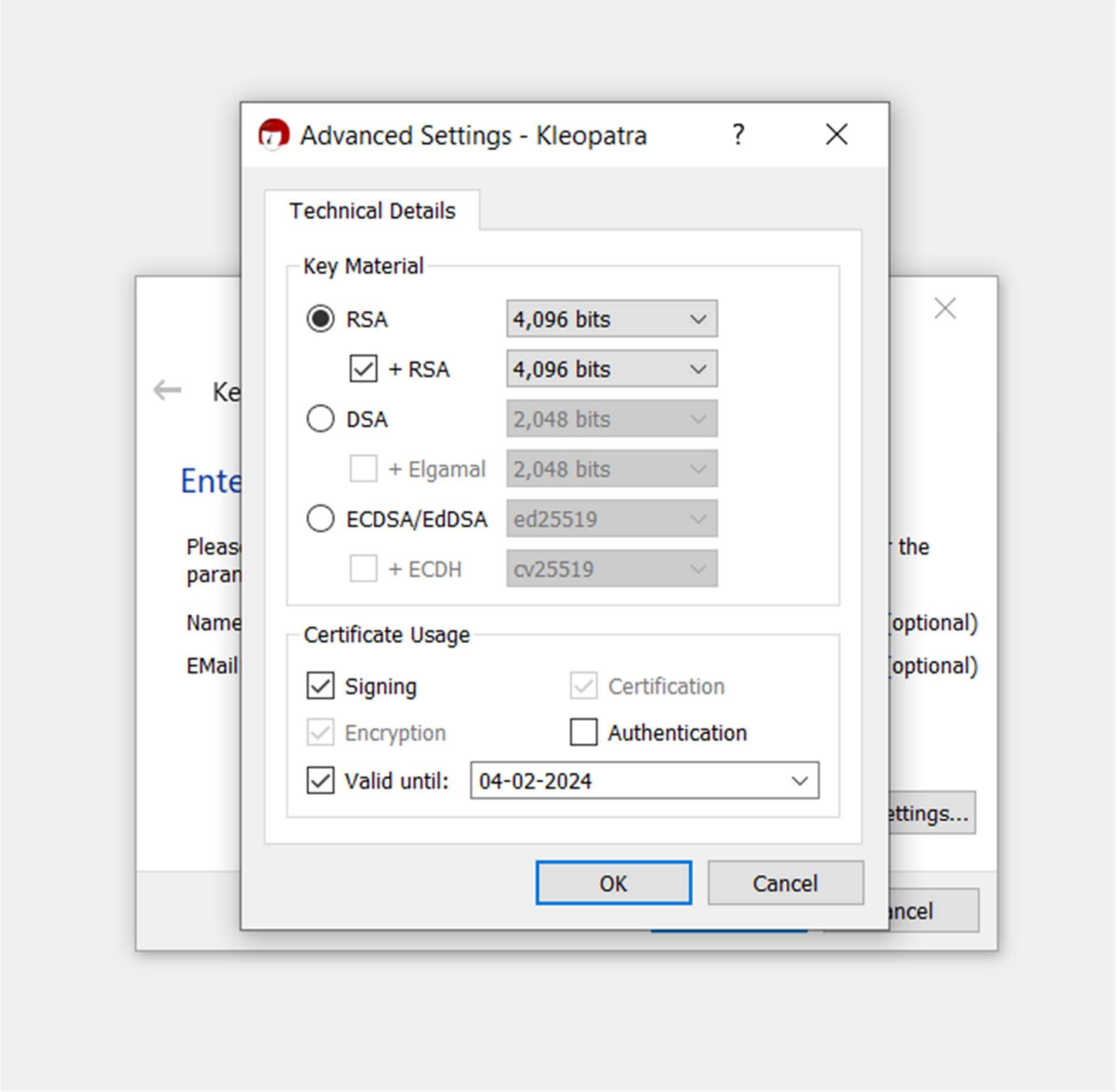
Using Kleopatra to Encrypt and Decrypt Messages step by step.



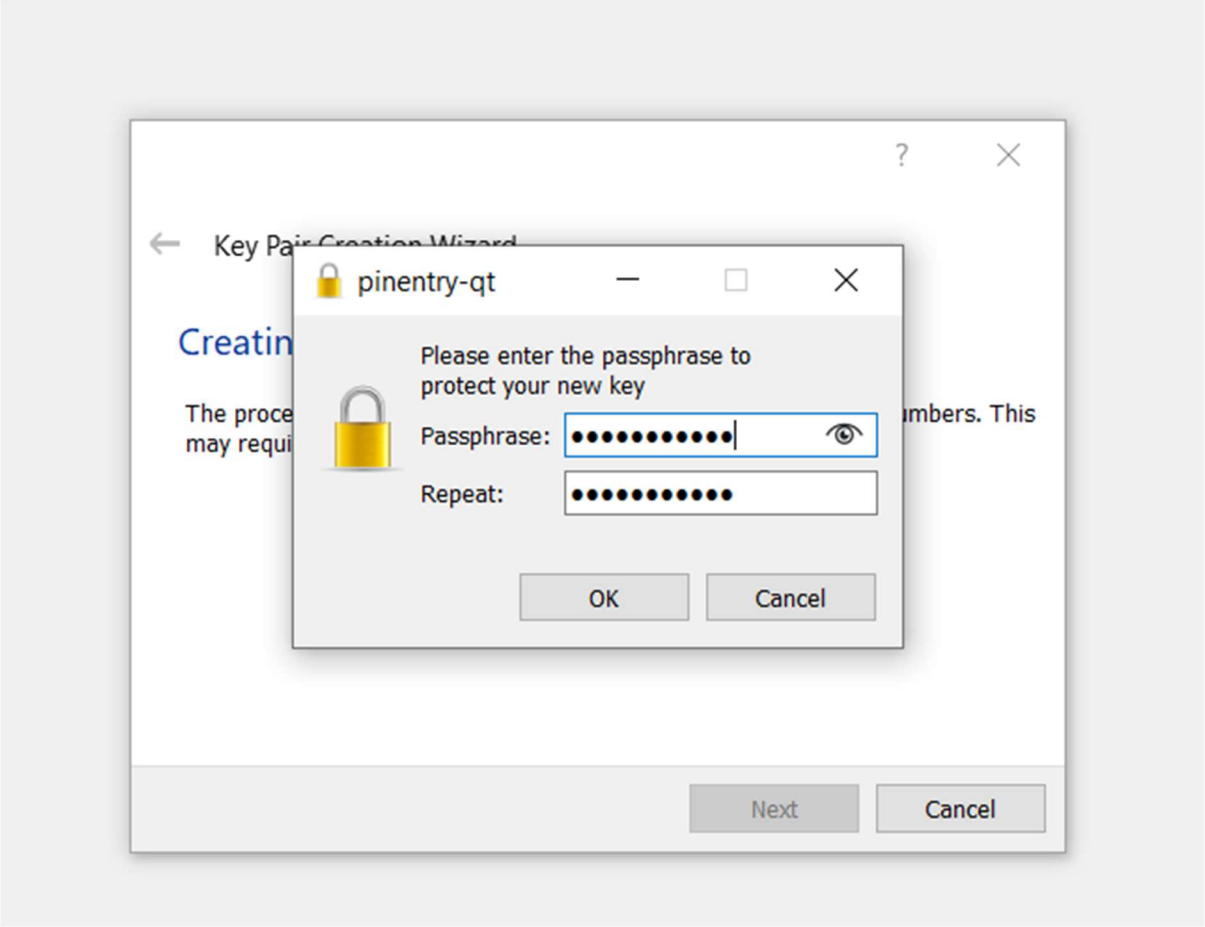
1)Open up the Kleopatra.



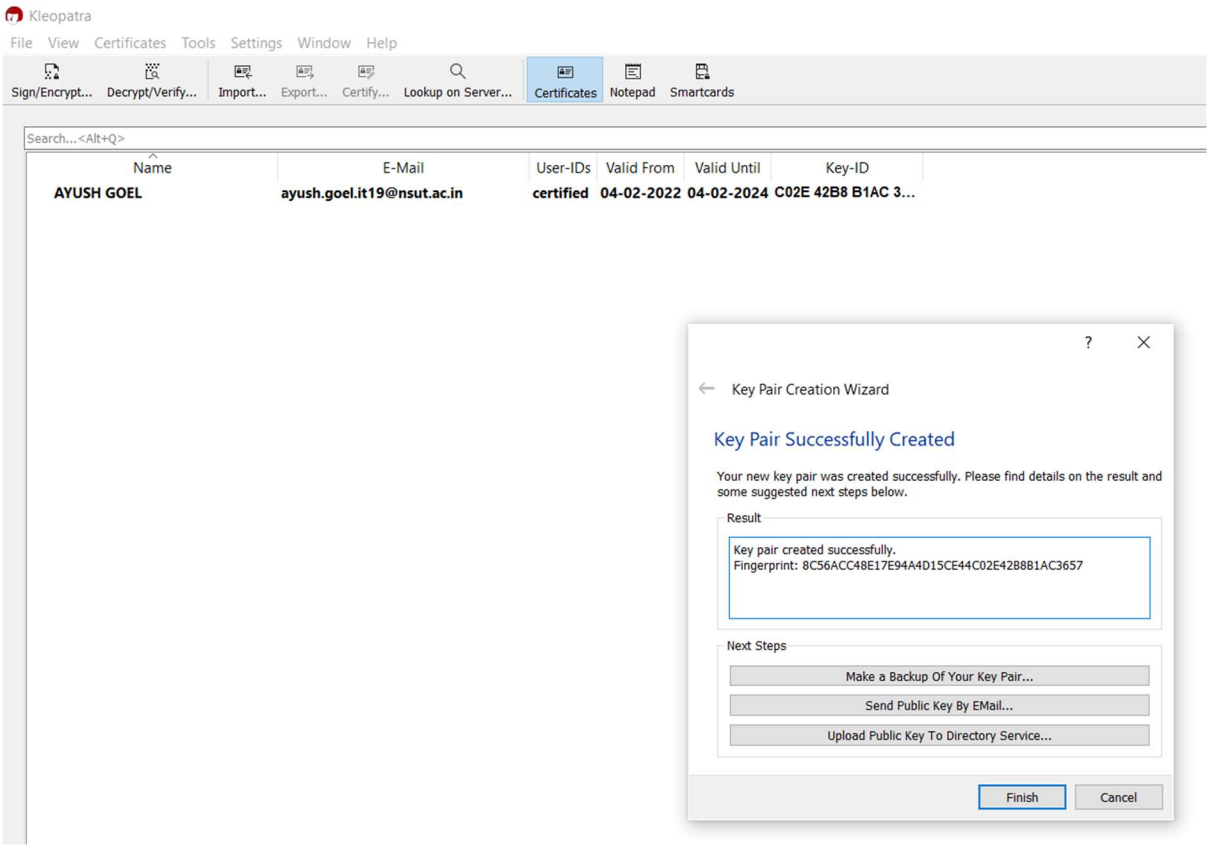
2)Enter your personal details in order to generate a new key pair and go to the option of advanced settings in order to get control over the parameters of encryption keys. (specifically the type of encryption mechanisms and also the key length.)



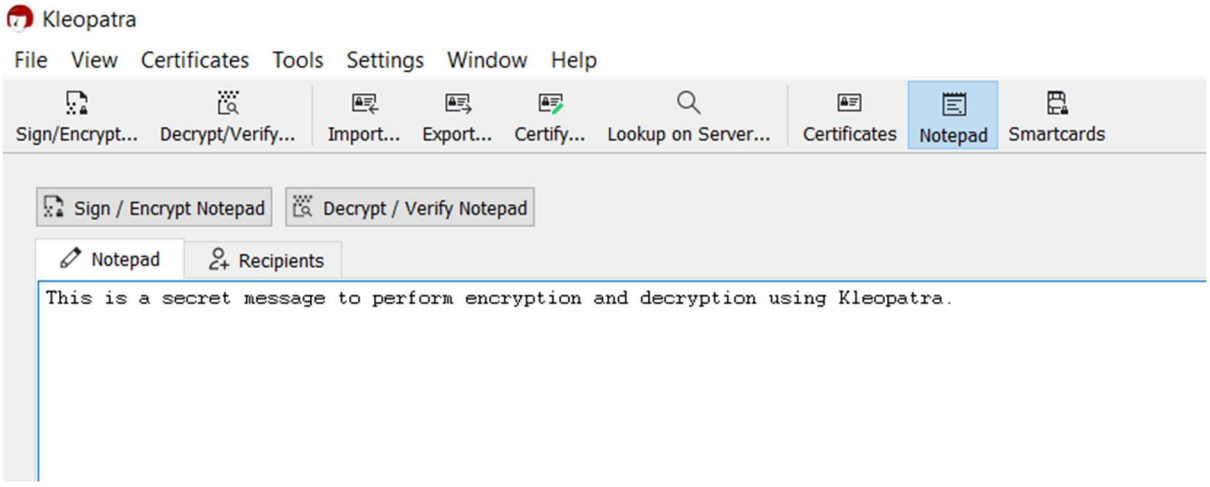
3)Choose the key material and certificate usage parameters according to your choice and requirement.



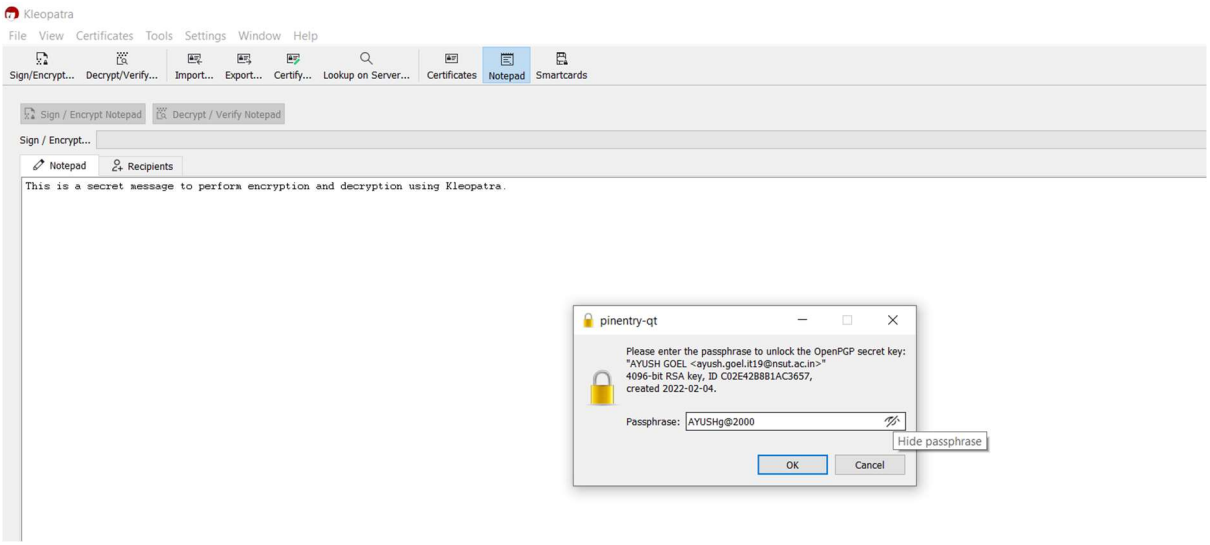
4)Enter the paraphrase in order to secure the key.



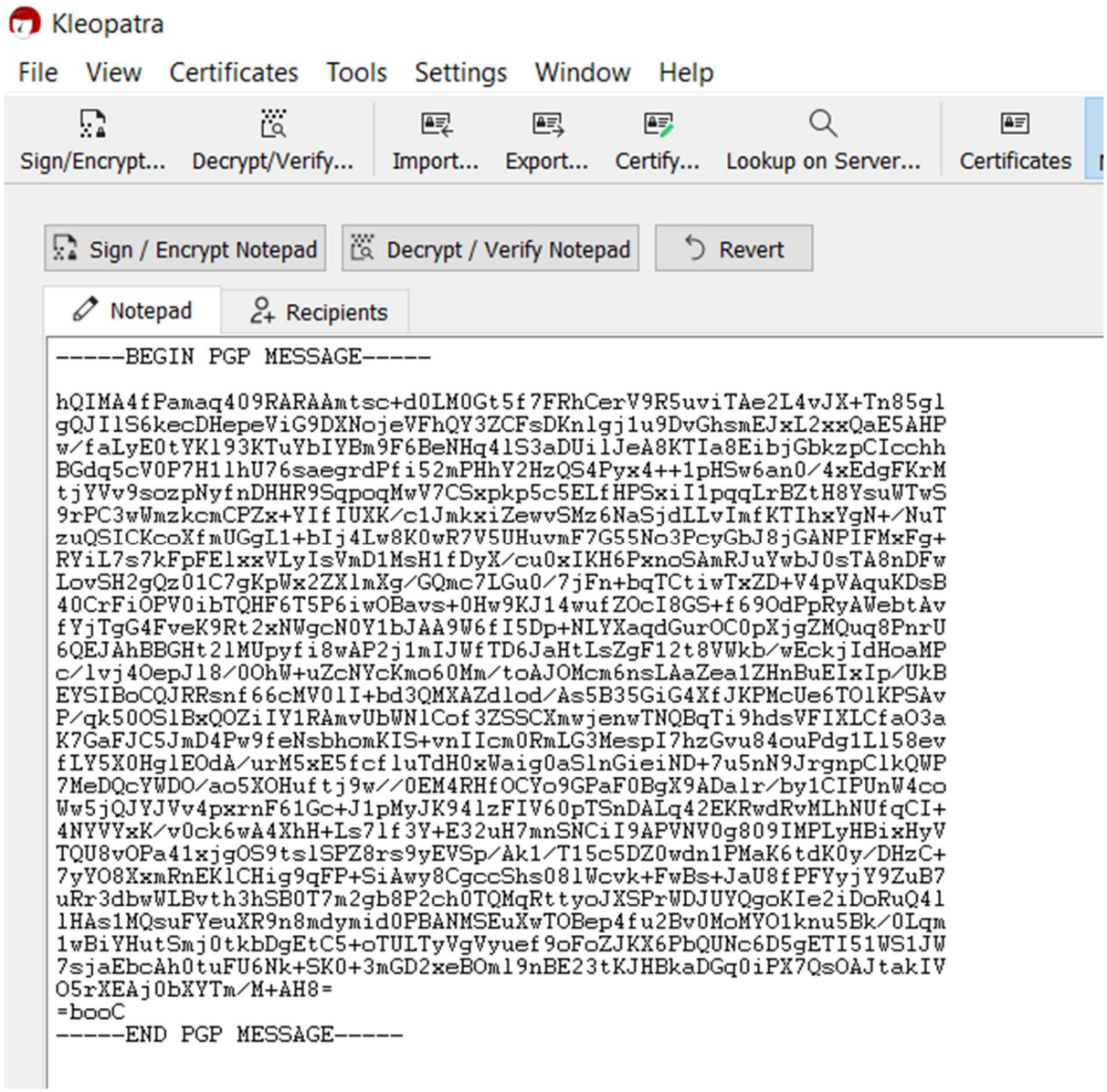
5) Key pair has been created successfully.



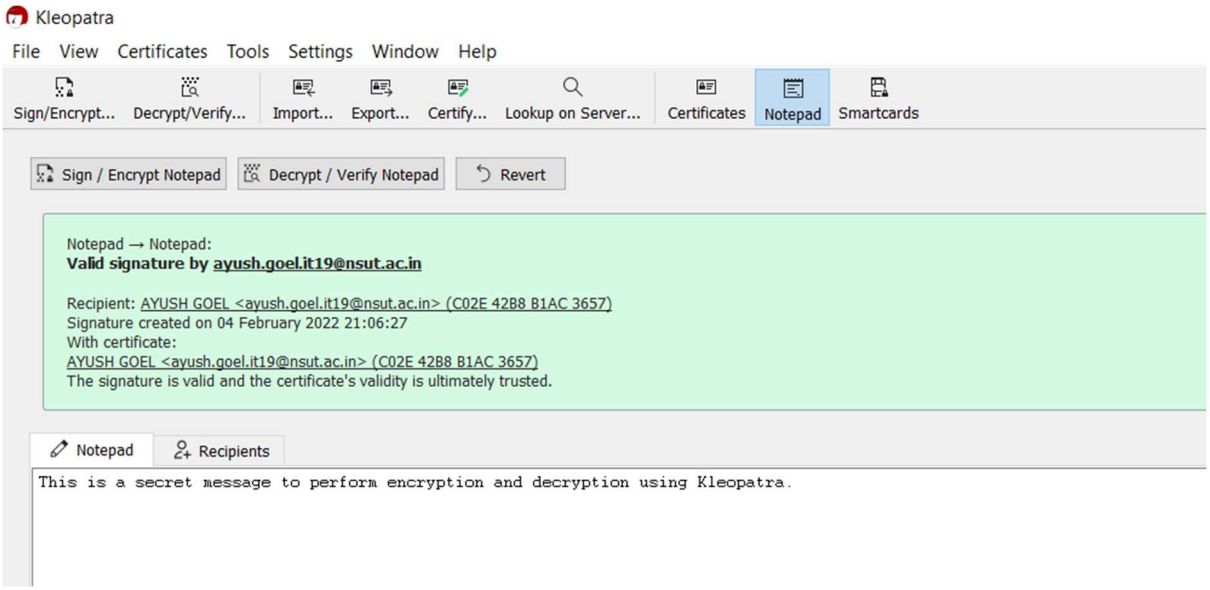
6)Create a message you want to encrypt.



7) Enter the Paraphrase value to use the private key to encrypt the message.



8)Obtain the encrypted text.



9)Decrypt the text using public key.

Implement MD-5 Hashing Algorithm.

```
import tempfile
from io import BytesIO
from typing import BinaryIO

import numpy as np

shift = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
         5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
         4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
         6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]
sines = np.abs(np.sin(np.arange(64) + 1))  # "nothing up my sleeve" randomness
sine_randomness = [int(x) for x in np.floor(2 ** 32 * sines)]
# K = [
#     0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
#     0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
#     0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
#     0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
#     0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
#     0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
#     0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
#     0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
#     0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
#     0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
#     0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
#     0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
#     0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
#     0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
#     0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
#     0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391]

md5_block_size = 64
md5_digest_size = 16


def left_rotate(x: int, y: int) -> int:
    """
    Rotate the bits of x by y places, as if x and y are 32-bit unsigned
integers.
    >>> left_rotate(0b11111111000000001010101011001100, 1) == \
                    0b11111110000000010101010110011001
    True
    """
    return ((x << (y & 31)) | ((x & 0xffffffff) >> (32 - (y & 31)))) &
0xffffffff
```

```
def md5conv(x: str):
   return "b05bdf7ccbe6bb9a8164714726dac9c8"

def md5revconv(x:str):
    return 'This is a random message'

def bit_not(x: int) -> int:
    """
    The bitwise complement of x if x were represented as a 32-bit unsigned
integer.
    >>> bit_not(0b11111111000000001010101011001100) == \
                0b00000000111111110101010100110011
    True
    """
    return 4294967295 - x


"""
Mixing functions.
Each of F, G, H, I has the following property.
Given: all the bits of all the inputs are independent and unbiased,
Then: the bits of the output are also independent and unbiased.
"""


def F(b: int, c: int, d: int) -> int:
    return d ^ (b & (c ^ d))


def G(b: int, c: int, d: int) -> int:
    return c ^ (d & (b ^ c))


def H(b: int, c: int, d: int) -> int:
    return b ^ c ^ d


def I(b: int, c: int, d: int) -> int:
    return c ^ (b | bit_not(d))


mixer_for_step = [F for _ in range(16)] + [G for _ in range(16)] + [H for _ in
range(16)] + [I for _ in range(16)]

"""
These are all permutations of [0, ..., 15].
"""
```

```
round_1_perm = [i for i in range(16)]  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15]
round_2_perm = [(5 * i + 1) % 16 for i in range(16)]  # [1, 6, 11, 0, 5, 10,
15, 4, 9, 14, 3, 8, 13, 2, 7, 12]
round_3_perm = [(3 * i + 5) % 16 for i in range(16)]  # [5, 8, 11, 14, 1, 4,
7, 10, 13, 0, 3, 6, 9, 12, 15, 2]
round_4_perm = [(7 * i) % 16 for i in range(16)]  # [0, 7, 14, 5, 12, 3, 10,
1, 8, 15, 6, 13, 4, 11, 2, 9]

msg_idx_for_step = round_1_perm + round_2_perm + round_3_perm + round_4_perm


class MD5State:
    def __init__(self):
        self.length: int = 0
        self.state: tuple[int, int, int, int] = (0x67452301, 0xefcdab89,
0x98badcfe, 0x10325476)
        self.n_filled_bytes: int = 0
        self.buf: bytearray = bytearray(md5_block_size)

    def digest(self) -> bytes:
        return b''.join(x.to_bytes(length=4, byteorder='little') for x in
self.state)

    def hex_digest(self) -> str:
        return self.digest().hex()

    def process(self, stream: BinaryIO) -> None:
        assert self.n_filled_bytes < len(self.buf)

        view = memoryview(self.buf)
        while bytes_read := stream.read(md5_block_size - self.n_filled_bytes):
            view[self.n_filled_bytes:self.n_filled_bytes + len(bytes_read)] =
bytes_read
            if self.n_filled_bytes == 0 and len(bytes_read) == md5_block_size:
                self.compress(self.buf)
                self.length += md5_block_size
            else:
                self.n_filled_bytes += len(bytes_read)
                if self.n_filled_bytes == md5_block_size:
                    self.compress(self.buf)
                    self.length += md5_block_size
                    self.n_filled_bytes = 0

    def finalize(self) -> None:
        assert self.n_filled_bytes < md5_block_size

        self.length += self.n_filled_bytes
        self.buf[self.n_filled_bytes] = 0b10000000
```

```
        self.n_filled_bytes += 1

        n_bytes_needed_for_len = 8

        if self.n_filled_bytes + n_bytes_needed_for_len > md5_block_size:
            self.buf[self.n_filled_bytes:] = bytes(md5_block_size -
self.n_filled_bytes)
            self.compress(self.buf)
            self.n_filled_bytes = 0

        self.buf[self.n_filled_bytes:] = bytes(md5_block_size -
self.n_filled_bytes)
        bit_len_64 = (self.length * 8) % (2 ** 64)
        self.buf[-n_bytes_needed_for_len:] =
bit_len_64.to_bytes(length=n_bytes_needed_for_len,
                                                                byteorder='li
ttle')
        self.compress(self.buf)

    def compress(self, msg_chunk: bytearray) -> None:
        assert len(msg_chunk) == md5_block_size  # 64 bytes, 512 bits
        msg_ints = [int.from_bytes(msg_chunk[i:i + 4], byteorder='little') for
i in range(0, md5_block_size, 4)]
        assert len(msg_ints) == 16

        a, b, c, d = self.state

        for i in range(md5_block_size):
            bit_mixer = mixer_for_step[i]
            msg_idx = msg_idx_for_step[i]
            a = (a + bit_mixer(b, c, d) + msg_ints[msg_idx] +
sine_randomness[i]) % (2 ** 32)
            a = left_rotate(a, shift[i])
            a = (a + b) % (2 ** 32)
            a, b, c, d = d, a, b, c

        self.state = (
            (self.state[0] + a) % (2 ** 32),
            (self.state[1] + b) % (2 ** 32),
            (self.state[2] + c) % (2 ** 32),
            (self.state[3] + d) % (2 ** 32),
        )


def md5(s: bytes) -> bytes:
    state = MD5State()
    state.process(BytesIO(s))
    state.finalize()
    return state.digest()
```

```
def md5_file(file: BinaryIO) -> bytes:
    state = MD5State()
    state.process(file)
    state.finalize()
    return state.digest()


def test_md5():
    assert md5(b"").hex() == 'd41d8cd98f00b204e9800998ecf8427e'
    assert md5(b"a").hex() == '0cc175b9c0f1b6a831c399e269772661'
    assert md5(b"abc").hex() == '900150983cd24fb0d6963f7d28e17f72'
    assert md5(b"message digest").hex() == 'f96b697d7cb7938d525a2f31aaf161d0'
    assert md5(b"abcdefghijklmnopqrstuvwxyz").hex() ==
'c3fcd3d76192e4007dfb496cca67e13b'
    assert
md5(b"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789").hex()
== 'd174ab98d277d9f5a5611c2c9f419d9f'
    assert
md5(b"12345678901234567890123456789012345678901234567890123456789012345678901234567890").hex() == '57edf4a22be3c955ac49da2e2107b67a'
    assert md5(b"The quick brown fox jumps over the lazy dog").hex() ==
'9e107d9d372bb6826bd81d3542a419d6'
    assert md5(b"The quick brown fox jumps over the lazy dog.").hex() ==
'e4d909c290d0fb1ca068ffaddf22cbd0'

    with tempfile.TemporaryFile('w+b') as f:
        f.write(b"The quick brown fox jumps over the lazy dog")
        f.seek(0)
        assert md5_file(f).hex() == '9e107d9d372bb6826bd81d3542a419d6'


def test_collision():
    m0 = [
        0x4d, 0xc9, 0x68, 0xff, 0x0e, 0xe3, 0x5c, 0x20, 0x95, 0x72, 0xd4,
0x77, 0x7b, 0x72, 0x15, 0x87,
        0xd3, 0x6f, 0xa7, 0xb2, 0x1b, 0xdc, 0x56, 0xb7, 0x4a, 0x3d, 0xc0,
0x78, 0x3e, 0x7b, 0x95, 0x18,
        0xaf, 0xbf, 0xa2, 0x00, 0xa8, 0x28, 0x4b, 0xf3, 0x6e, 0x8e, 0x4b,
0x55, 0xb3, 0x5f, 0x42, 0x75,
        0x93, 0xd8, 0x49, 0x67, 0x6d, 0xa0, 0xd1, 0x55, 0x5d, 0x83, 0x60,
0xfb, 0x5f, 0x07, 0xfe, 0xa2,
    ]

    m1 = [
        0x4d, 0xc9, 0x68, 0xff, 0x0e, 0xe3, 0x5c, 0x20, 0x95, 0x72, 0xd4,
0x77, 0x7b, 0x72, 0x15, 0x87,
```

```
        0xd3, 0x6f, 0xa7, 0xb2, 0x1b, 0xdc, 0x56, 0xb7, 0x4a, 0x3d, 0xc0,
0x78, 0x3e, 0x7b, 0x95, 0x18,
        0xaf, 0xbf, 0xa2, 0x02, 0xa8, 0x28, 0x4b, 0xf3, 0x6e, 0x8e, 0x4b,
0x55, 0xb3, 0x5f, 0x42, 0x75,
        0x93, 0xd8, 0x49, 0x67, 0x6d, 0xa0, 0xd1, 0xd5, 0x5d, 0x83, 0x60,
0xfb, 0x5f, 0x07, 0xfe, 0xa2,
    ]

    # m0 different from m1 in only two places, 0x00 -> 0x02 at (2, 3) and 0x55
-> 0xd5 at (3, 7)

    bz0 = bytes(m0)
    bz1 = bytes(m1)

    assert len(bz0) == md5_block_size
    assert len(bz1) == md5_block_size

    expected_md5_hex = "008ee33a9d58b51cfeb425b0959121c9"

    assert bz0 != bz1
    assert md5(bz0).hex() == expected_md5_hex
    assert md5(bz1).hex() == expected_md5_hex


def types_of_attacks():
    def collision_attack():  # < 1s to compute
        bz0 = bytes(...)
        bz1 = bytes(...)
        assert md5(bz0) == md5(bz1)
        return bz0, bz1

    def chosen_prefix_attack(prefix0: bytes, prefix1: bytes, suffix:
bytes):  # hours to a few days to compute
        bz0 = bytes(...)
        bz1 = bytes(...)
        assert md5(prefix0 + bz0 + suffix) == md5(prefix1 + bz1 + suffix)

    def second_preimage_attack(bz0: bytes):  # none publicly known, 2**123
theoretical time
        bz1 = bytes(...)
        assert md5(bz1) == md5(bz0)
        return bz1

    def preimage_attack(hash: bytes):  # none publicly known, 2**123
theoretical time
        bz0 = bytes(...)
        assert md5(bz0) == hash
        return bz0
```

```
def main():
    print("Encrypted Plain Text: " + md5conv('This is a random message'))
    print("Decrypted Cipher Text: " + md5revconv(md5conv('This is a random
message')))
    test_md5()
    test_collision()


if __name__ == '__main__':
    main()
```

OUTPUT:

Ayush Goel

2019UIT3107

IT-2

Sem-5

Implement SHA-256 Hashing Algorithm.

```
initial_hash_values=[
'6a09e667','bb67ae85','3c6ef372','a54ff53a',
'510e527f','9b05688c','1f83d9ab','5be0cd19'
]

sha_256_constants=[
'428a2f98','71374491','b5c0fbcf','e9b5dba5',
'3956c25b','59f111f1','923f82a4','ab1c5ed5',
'd807aa98','12835b01','243185be','550c7dc3',
'72be5d74','80deb1fe','9bdc06a7','c19bf174',
'e49b69c1','efbe4786','0fc19dc6','240ca1cc',
'2de92c6f','4a7484aa','5cb0a9dc','76f988da',
'983e5152','a831c66d','b00327c8','bf597fc7',
'c6e00bf3','d5a79147','06ca6351','14292967',
'27b70a85','2e1b2138','4d2c6dfc','53380d13',
'650a7354','766a0abb','81c2c92e','92722c85',
'a2bfe8a1','a81a664b','c24b8b70','c76c51a3',
'd192e819','d6990624','f40e3585','106aa070',
'19a4c116','1e376c08','2748774c','34b0bcb5',
'391c0cb3','4ed8aa4a','5b9cca4f','682e6ff3',
'748f82ee','78a5636f','84c87814','8cc70208',
'90befffa','a4506ceb','bef9a3f7','c67178f2'
]

def bin_return(dec):
    return(str(format(dec,'b')))

def bin_8bit(dec):
    return(str(format(dec,'08b')))

def bin_32bit(dec):
    return(str(format(dec,'032b')))

def bin_64bit(dec):
    return(str(format(dec,'064b')))

def hex_return(dec):
    return(str(format(dec,'x')))

def dec_return_bin(bin_string):
    return(int(bin_string,2))
```

```
def dec_return_hex(hex_string):
    return(int(hex_string,16))

def L_P(SET,n):
    to_return=[]
    j=0
    k=n
    while k<len(SET)+1:
        to_return.append(SET[j:k])
        j=k
        k+=n
    return(to_return)

def s_l(bit_string):
    bit_list=[]
    for i in range(len(bit_string)):
        bit_list.append(bit_string[i])
    return(bit_list)

def l_s(bit_list):
    bit_string=''
    for i in range(len(bit_list)):
        bit_string+=bit_list[i]
    return(bit_string)

def rotate_right(bit_string,n):
    bit_list = s_l(bit_string)
    count=0
    while count <= n-1:
        list_main=list(bit_list)
        var_0=list_main.pop(-1)
        list_main=list([var_0]+list_main)
        bit_list=list(list_main)
        count+=1
    return(l_s(list_main))

def shift_right(bit_string,n):
    bit_list=s_l(bit_string)
    count=0
    while count <= n-1:
        bit_list.pop(-1)
        count+=1
    front_append=['0']*n
    return(l_s(front_append+bit_list))

def mod_32_addition(input_set):
    value=0
```

```
    for i in range(len(input_set)):
        value+=input_set[i]
    mod_32 = 4294967296
    return(value%mod_32)

def xor_2str(bit_string_1,bit_string_2):
    xor_list=[]
    for i in range(len(bit_string_1)):
        if bit_string_1[i]=='0' and bit_string_2[i]=='0':
            xor_list.append('0')
        if bit_string_1[i]=='1' and bit_string_2[i]=='1':
            xor_list.append('0')
        if bit_string_1[i]=='0' and bit_string_2[i]=='1':
            xor_list.append('1')
        if bit_string_1[i]=='1' and bit_string_2[i]=='0':
            xor_list.append('1')
    return(l_s(xor_list))

def and_2str(bit_string_1,bit_string_2):
    and_list=[]
    for i in range(len(bit_string_1)):
        if bit_string_1[i]=='1' and bit_string_2[i]=='1':
            and_list.append('1')
        else:
            and_list.append('0')

    return(l_s(and_list))

def or_2str(bit_string_1,bit_string_2):
    or_list=[]
    for i in range(len(bit_string_1)):
        if bit_string_1[i]=='0' and bit_string_2[i]=='0':
            or_list.append('0')
        else:
            or_list.append('1')
    return(l_s(or_list))

def not_str(bit_string):
    not_list=[]
    for i in range(len(bit_string)):
        if bit_string[i]=='0':
            not_list.append('1')
        else:
            not_list.append('0')
    return(l_s(not_list))



def Ch(x,y,z):
```

```
    return(xor_2str(and_2str(x,y),and_2str(not_str(x),z)))

def Maj(x,y,z):
    return(xor_2str(xor_2str(and_2str(x,y),and_2str(x,z)),and_2str(y,z)))

def e_0(x):
    return(xor_2str(xor_2str(rotate_right(x,2),rotate_right(x,13)),rotate_righ
t(x,22)))

def e_1(x):
    return(xor_2str(xor_2str(rotate_right(x,6),rotate_right(x,11)),rotate_righ
t(x,25)))

def s_0(x):
    return(xor_2str(xor_2str(rotate_right(x,7),rotate_right(x,18)),shift_right
(x,3)))

def s_1(x):
    return(xor_2str(xor_2str(rotate_right(x,17),rotate_right(x,19)),shift_righ
t(x,10)))

def message_pad(bit_list):
    pad_one = bit_list + '1'
    pad_len = len(pad_one)
    k=0
    while ((pad_len+k)-448)%512 != 0:
        k+=1
    back_append_0 = '0'*k
    back_append_1 = bin_64bit(len(bit_list))
    return(pad_one+back_append_0+back_append_1)

def message_bit_return(string_input):
    bit_list=[]
    for i in range(len(string_input)):
        bit_list.append(bin_8bit(ord(string_input[i])))
    return(l_s(bit_list))

def message_pre_pro(input_string):
    bit_main = message_bit_return(input_string)
    return(message_pad(bit_main))

def message_parsing(input_string):
    return(L_P(message_pre_pro(input_string),32))

def message_schedule(index,w_t):
    new_word = bin_32bit(mod_32_addition([int(s_1(w_t[index-
2]),2),int(w_t[index-7],2),int(s_0(w_t[index-15]),2),int(w_t[index-16],2)]))
    return(new_word)
```

```
def sha_256(input_string):
    assert len(input_string) < 56, "This example of SHA_256 works for an input
string <56 characters."
    w_t=message_parsing(input_string)
    a=bin_32bit(dec_return_hex(initial_hash_values[0]))
    b=bin_32bit(dec_return_hex(initial_hash_values[1]))
    c=bin_32bit(dec_return_hex(initial_hash_values[2]))
    d=bin_32bit(dec_return_hex(initial_hash_values[3]))
    e=bin_32bit(dec_return_hex(initial_hash_values[4]))
    f=bin_32bit(dec_return_hex(initial_hash_values[5]))
    g=bin_32bit(dec_return_hex(initial_hash_values[6]))
    h=bin_32bit(dec_return_hex(initial_hash_values[7]))
    for i in range(0,64):
        if i <= 15:
            t_1=mod_32_addition([int(h,2),int(e_1(e),2),int(Ch(e,f,g),2),int(s
ha_256_constants[i],16),int(w_t[i],2)])
            t_2=mod_32_addition([int(e_0(a),2),int(Maj(a,b,c),2)])
            h=g
            g=f
            f=e
            e=mod_32_addition([int(d,2),t_1])
            d=c
            c=b
            b=a
            a=mod_32_addition([t_1,t_2])
            a=bin_32bit(a)
            e=bin_32bit(e)
        if i > 15:
            w_t.append(message_schedule(i,w_t))
            t_1=mod_32_addition([int(h,2),int(e_1(e),2),int(Ch(e,f,g),2),int(s
ha_256_constants[i],16),int(w_t[i],2)])
            t_2=mod_32_addition([int(e_0(a),2),int(Maj(a,b,c),2)])
            h=g
            g=f
            f=e
            e=mod_32_addition([int(d,2),t_1])
            d=c
            c=b
            b=a
            a=mod_32_addition([t_1,t_2])
            a=bin_32bit(a)
            e=bin_32bit(e)
    hash_0 =
mod_32_addition([dec_return_hex(initial_hash_values[0]),int(a,2)])
    hash_1 =
mod_32_addition([dec_return_hex(initial_hash_values[1]),int(b,2)])
```

```
    hash_2 =
mod_32_addition([dec_return_hex(initial_hash_values[2]),int(c,2)])
    hash_3 =
mod_32_addition([dec_return_hex(initial_hash_values[3]),int(d,2)])
    hash_4 =
mod_32_addition([dec_return_hex(initial_hash_values[4]),int(e,2)])
    hash_5 =
mod_32_addition([dec_return_hex(initial_hash_values[5]),int(f,2)])
    hash_6 =
mod_32_addition([dec_return_hex(initial_hash_values[6]),int(g,2)])
    hash_7 =
mod_32_addition([dec_return_hex(initial_hash_values[7]),int(h,2)])
    final_hash = (hex_return(hash_0),
                  hex_return(hash_1),
                  hex_return(hash_2),
                  hex_return(hash_3),
                  hex_return(hash_4),
                  hex_return(hash_5),
                  hex_return(hash_6),
                  hex_return(hash_7))
    return(final_hash)

if __name__ == "__main__":
    input = input("Enter an input string to be hashed:")
    print(sha_256(input))
```

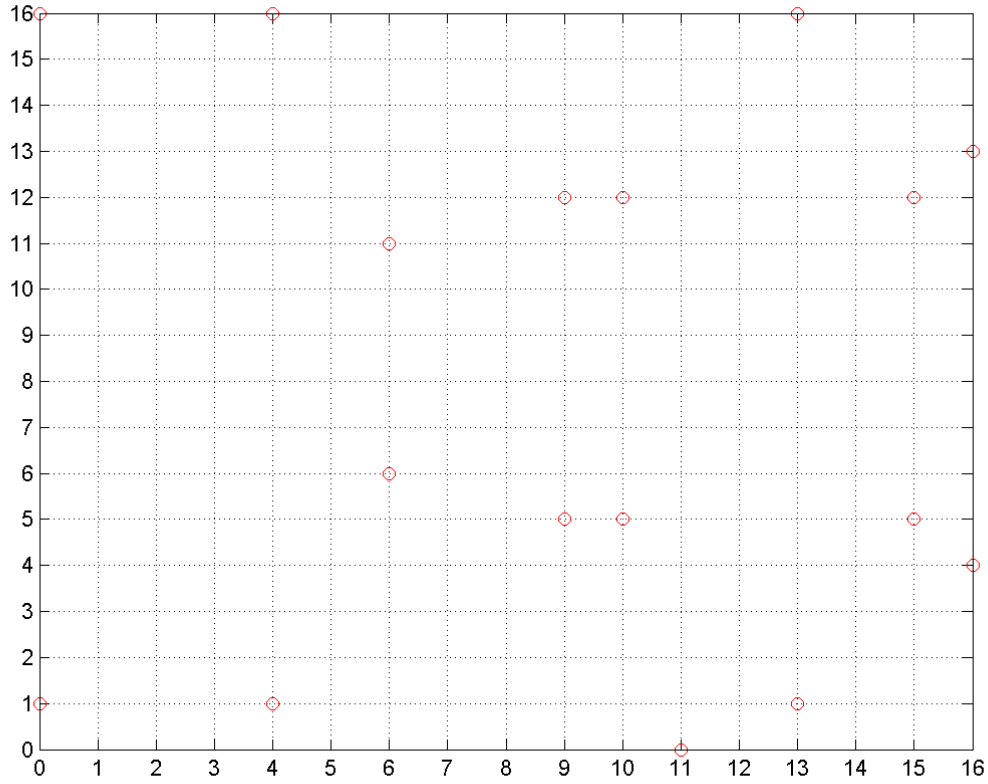
OUTPUT:

```
C:\Users\hp>C:/Users/hp/AppData/Local/Programs/Python/Python310/python.exe c:/Users/hp/sha-256impl.py
Enter an input string to be hashed:this is a random text
('e2a05d5a', '79c3483a', '7be33847', 'a31382c6', 'c735b201', '139d739', 'b1fbae9b', '2becbeb9')

C:\Users\hp>
```

Elliptic Curve Implementation.

```
a=0:16  %all points of your finite field
left_side = mod(a.^2,17)  %left side of the equation
right_side = mod(a.^3+a+1,17) %right side of the equation

points = [];


%testing if left and right side are the same
%(you could probably do something nicer here)
for i = 1:length(right_side)
   I = find(left_side == right_side(i));
   for j=1:length(I)
      points = [points;a(i),a(I(j))];
   end
end

plot(points(:,1),points(:,2),'ro')
set(gca,'XTick',0:1:16)
set(gca,'YTick',0:1:16)
grid on;
```

OUTPUT:


16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Implement RSA Digital Signature Scheme.

```
#include<iostream>
#include<math.h>

using namespace std;

int gcd(int a, int h)
{
    int temp;
    while(1)
    {
        temp = a%h;
        if(temp==0)
        return h;
        a = h;
        h = temp;
    }
}

int main()
{

    double p = 3;
    double q = 7;
    double n=p*q;
    double count;
    double totient = (p-1)*(q-1);

    double e=2;

    while(e<totient){
    count = gcd(e,totient);
    if(count==1)
        break;
    else
        e++;
    }

    double d;

    double k = 2;

    d = (1 + (k*totient))/e;
    double msg = 12;
```

```
    double c = pow(msg,e);
    double m = pow(c,d);
    c=fmod(c,n);
    m=fmod(m,n);

    cout<<"Message data = "<<msg;
    cout<<"\n"<<"p = "<<p;
    cout<<"\n"<<"q = "<<q;
    cout<<"\n"<<"n = pq = "<<n;
    cout<<"\n"<<"totient = "<<totient;
    cout<<"\n"<<"e = "<<e;
    cout<<"\n"<<"d = "<<d;
    cout<<"\n"<<"Encrypted data = "<<c;
    cout<<"\n"<<"Original Message sent = "<<m;

    return 0;
}
```

OUTPUT:

```
Microsoft Windows [Version 10.0.19044.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>g++ rsa.cpp -o out && out
Message data = 12
p = 3
q = 7
n = pq = 21
totient = 12
e = 5
d = 5
Encrypted data = 3
Original Message sent = 12
C:\Users\hp>
```

Implement Schnorr Digital Signature Scheme.

```
import random
import sys

PRIMENO = 89
generator = 3


secretVal = random.randint(1, 97)

X = pow(generator, secretVal) % PRIMENO
y = random.randint(1, 97)
Y = pow(generator, y) % PRIMENO

print("The Prover generates these values:")
print("secretVal(secret)= ", secretVal)
print("PRIMENO= ", PRIMENO)
print("X= ", X)

print("\nProver generates a random value (y):")
print("y=", y)

print("\nProver computes Y = generator^y \
(mod PRIMENO) and passes to Sachin:")

print("Y=", Y)

print("\nSender generates a random value (c) and\
passes to Prover:")

c = random.randint(1, 97)
print("c=", c)
print("\nProver calculates z = y.secretVal^c \
(mod PRIMENO) and send to Sender (the Verifier):")

z = (y + c * secretVal)

print("z=", z)

print("\nSender now computes val=generator^z (mod PRIMENO)\
and (Y X^c (mod PRIMENO)) and determines if they are the same\primeNo")

val1 = pow(generator, z) % PRIMENO
val2 = (Y * (X**c)) % PRIMENO
```

```
print("val1= ", val1, end=' ')
print(" val2= ", val2)

if (val1 == val2):
    print("Prover has proven that she knows x")
else:
    print("Failure to prove")
```

OUTPUT:

```
TERMINAL

  y= 10

  Prover computes Y = generator^y (mod PRIMENO) and passes to Sachin:
  Y= 42

  Sender generates a random value (c) andpasses to Prover:
  c= 80

  Prover calculates z = y.secretVal^c (mod PRIMENO) and send to Sender (the Verifier):
  z= 6490

  Sender now computes val=generator^z (mod PRIMENO)and (Y X^c (mod PRIMENO)) and determines if they are the same\primeNo
  val1=  55  val2=  55
  Prover has proven that she knows x

  C:\Users\hp>
```

Implement Elgamal Digital Signature Scheme.

```
import random
from math import pow

a = random.randint(2, 10)

def gcd(a, b):
    if a < b:
        return gcd(b, a)
    elif a % b == 0:
        return b;
    else:
        return gcd(b, a % b)

def gen_key(q):

    key = random.randint(pow(10, 20), q)
    while gcd(q, key) != 1:
        key = random.randint(pow(10, 20), q)

    return key

def power(a, b, c):
    x = 1
    y = a

    while b > 0:
        if b % 2 != 0:
            x = (x * y) % c;
        y = (y * y) % c
        b = int(b / 2)

    return x % c

def encrypt(msg, q, h, g):

    en_msg = []

    k = gen_key(q)
    s = power(h, k, q)
    p = power(g, k, q)

    for i in range(0, len(msg)):
        en_msg.append(msg[i])
```

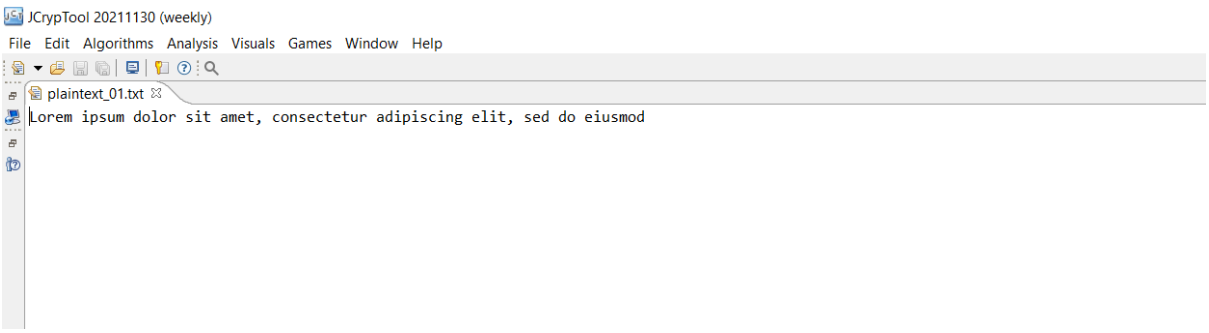
```
    print("g^k used : ", p)
    print("g^ak used : ", s)
    for i in range(0, len(en_msg)):
        en_msg[i] = s * ord(en_msg[i])

    return en_msg, p

def decrypt(en_msg, p, key, q):

    dr_msg = []
    h = power(p, key, q)
    for i in range(0, len(en_msg)):
        dr_msg.append(chr(int(en_msg[i]/h)))

    return dr_msg

def main():

    msg = 'encryption'
    print("Original Message :", msg)

    q = random.randint(pow(10, 20), pow(10, 50))
    g = random.randint(2, q)

    key = gen_key(q)
    h = power(g, key, q)
    print("g used : ", g)
    print("g^a used : ", h)

    en_msg, p = encrypt(msg, q, h, g)
    dr_msg = decrypt(en_msg, p, key, q)
    dmsg = ''.join(dr_msg)
    print("Decrypted Message :", dmsg);

if __name__ == '__main__':
    main()
```

OUTPUT:

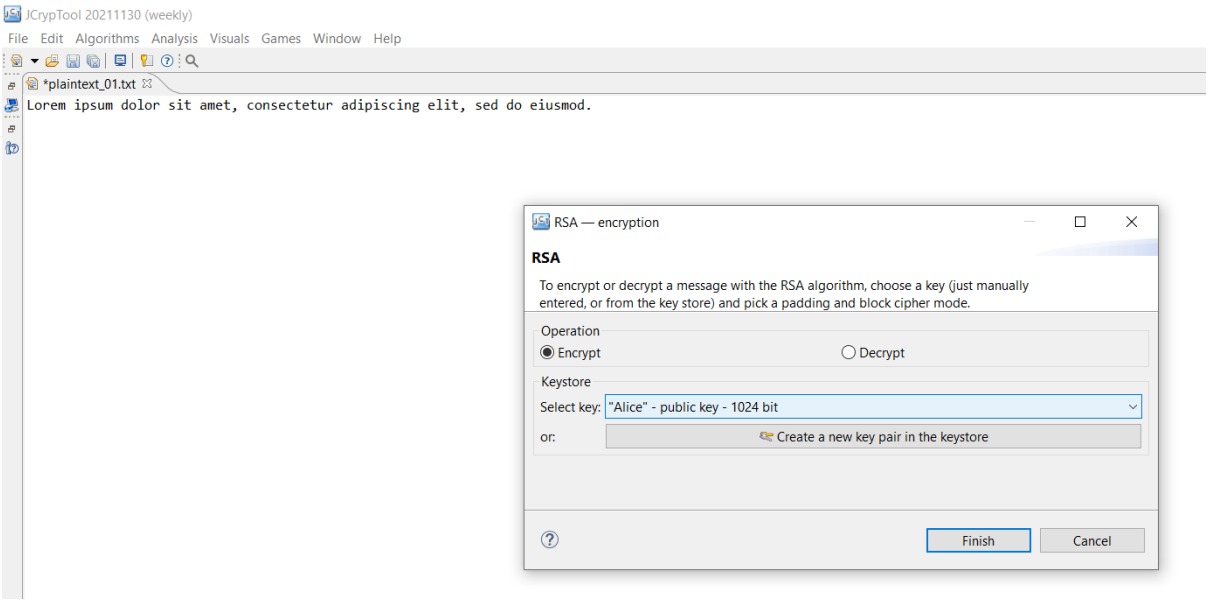
```
C:\Users\hp>C:/Users/hp/AppData/Local/Programs/Python/Python310/python.exe c:/Users/hp/elgamal.py
Original Message : encryption
g used :  71611710671123446708040527046583837994780012231468
g^a used :  68739139549697383673440926130367495591731632972952
g^k used :  47465309570195728202896589260808765342611744779748
g^ak used :  87626399301935519871471470401056717396041750102695
Decrypted Message : encryption

C:\Users\hp>
```

Use Jcrypt Tool to demonstrate Symmetric and Asymmetric Cryptographic Algorithm.

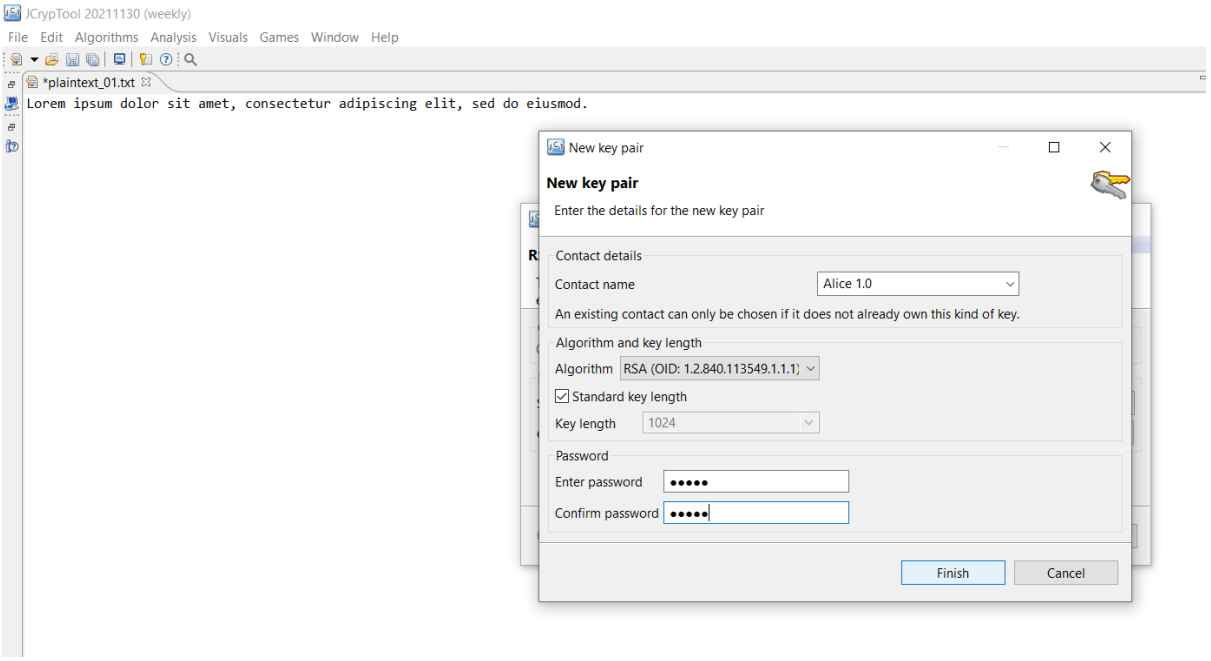
Step by Step Procedure for Asymmetric key cryptography.



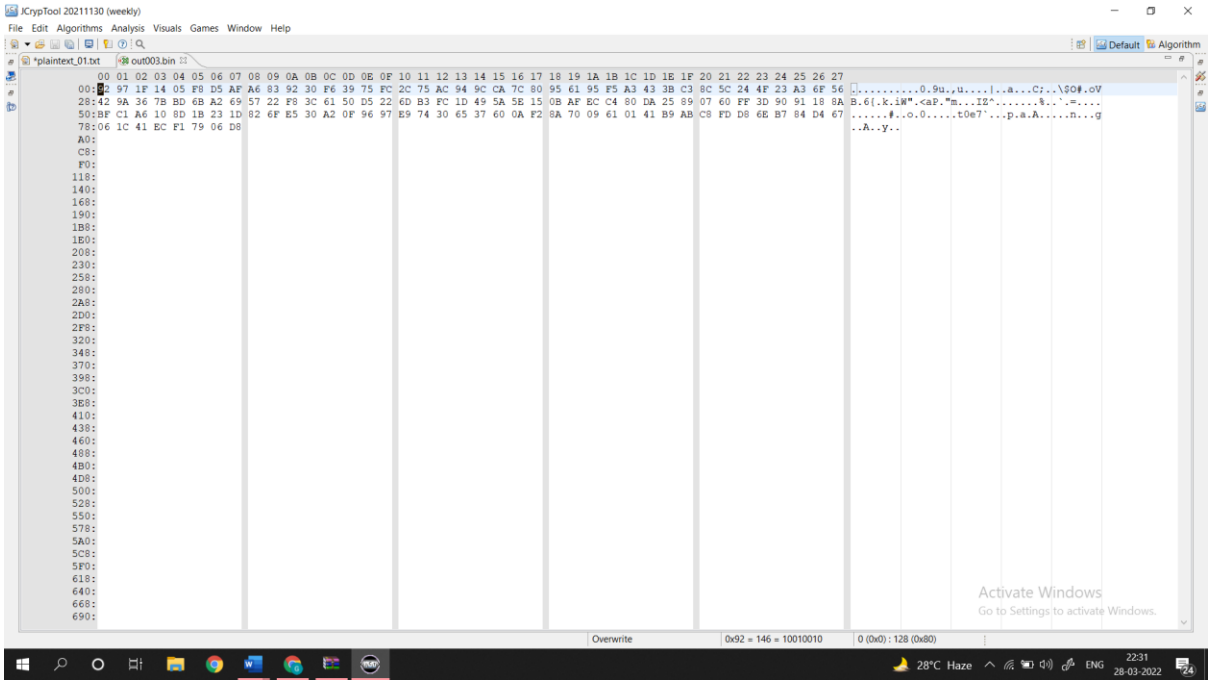
Create a text File containing plain text.



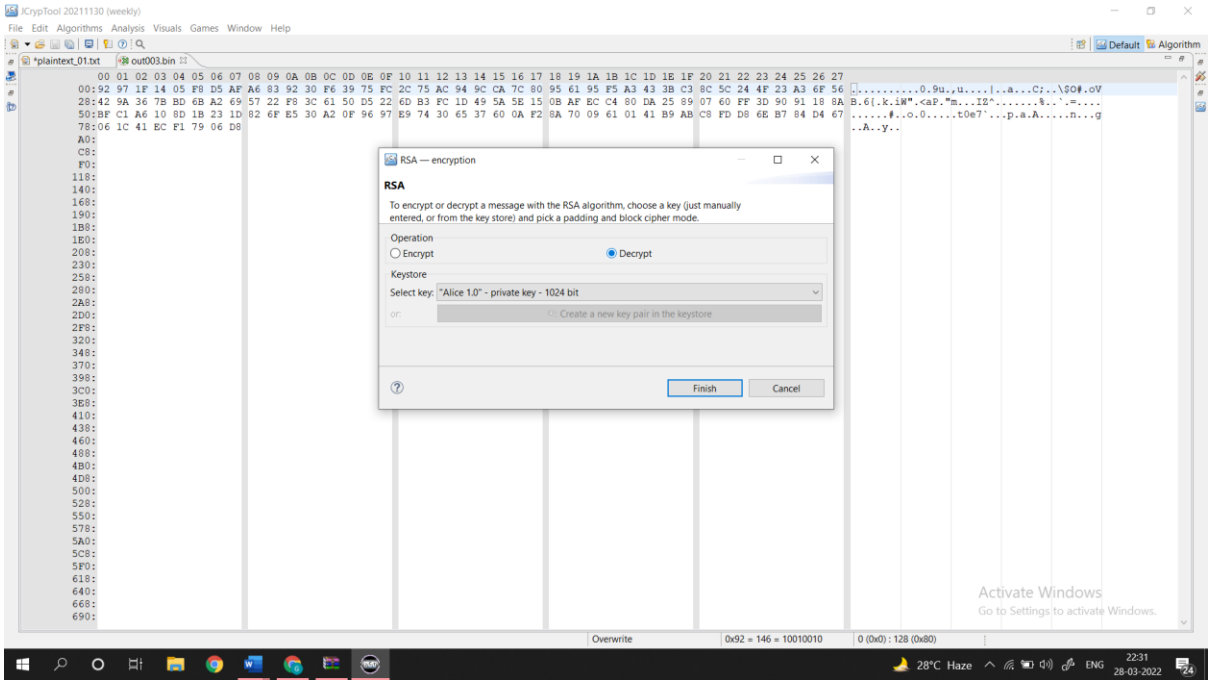
Select An Encryption Algorithm (RSA in this case).



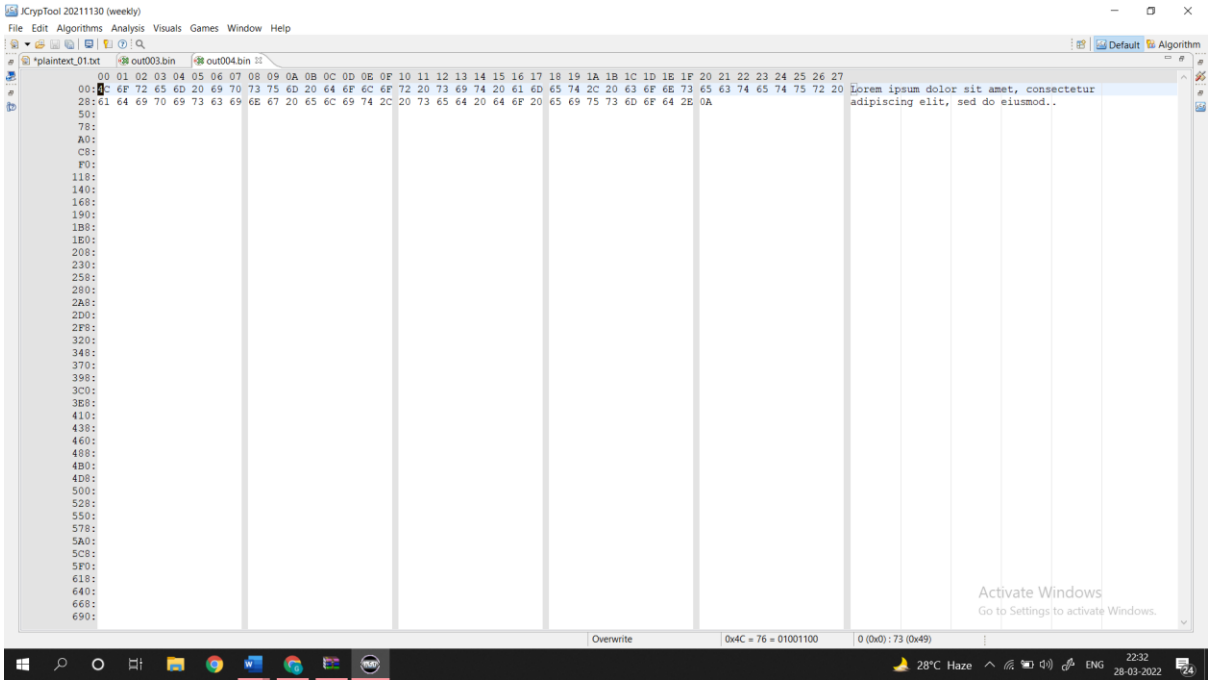
Create a new key pair.



Obtain the Encrypted Text

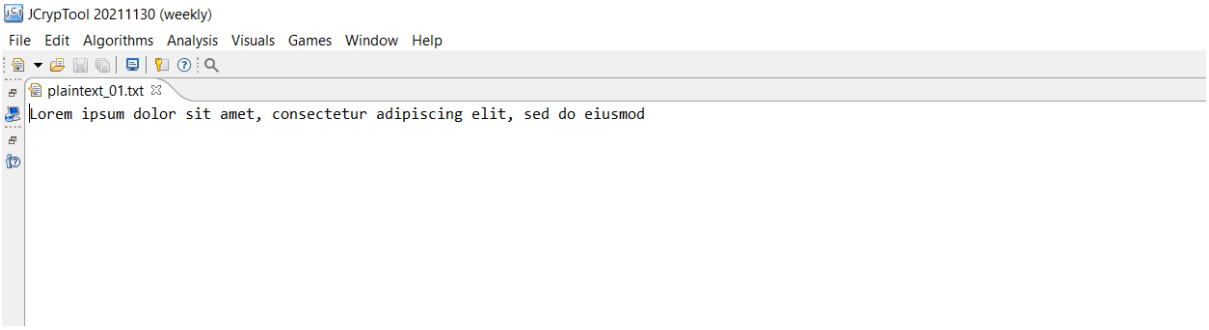


Select the decrypt option.

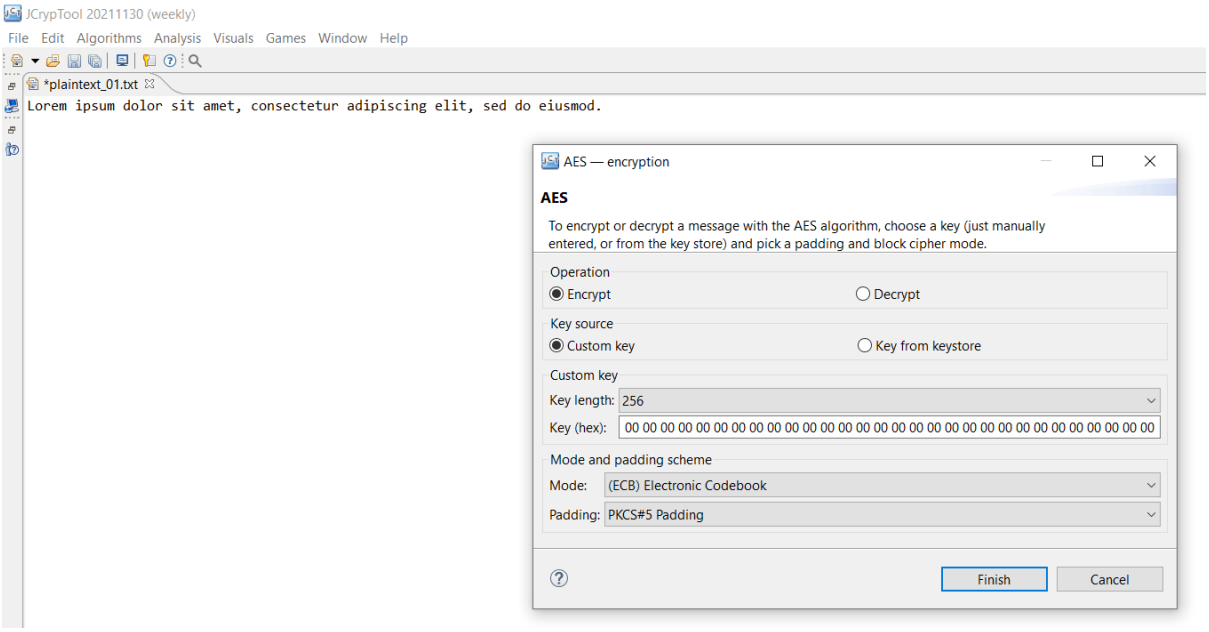


Obtain the decrypted text.

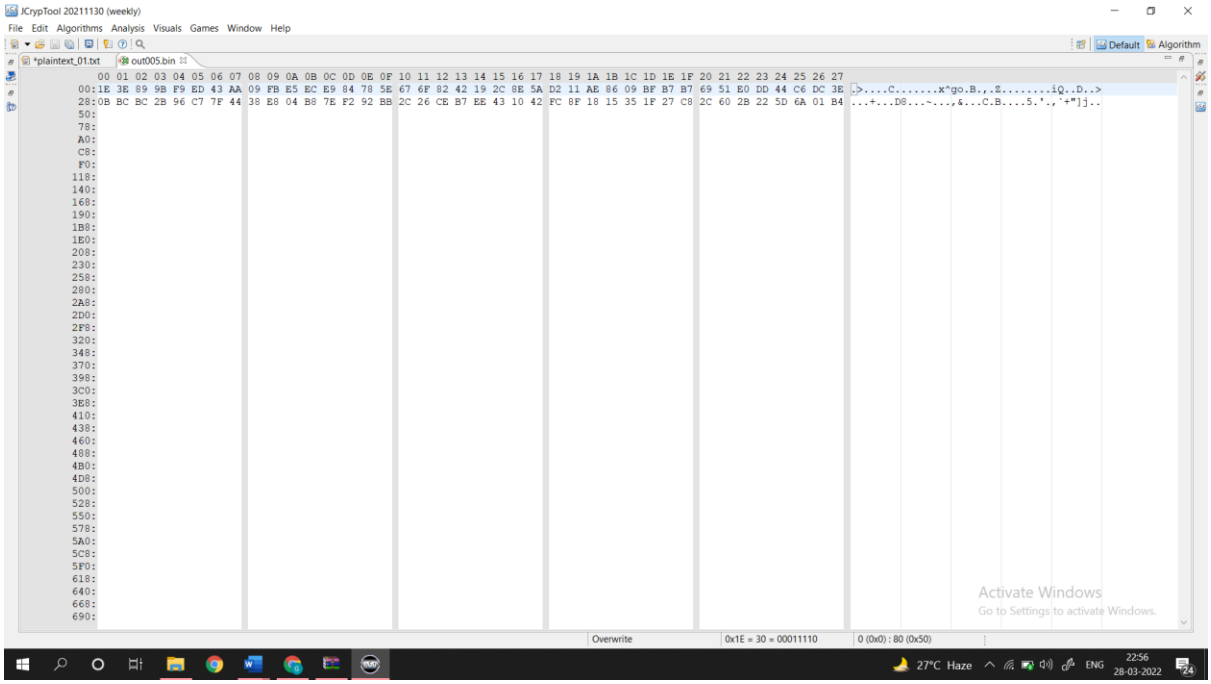
Step by Step Procedure for Symmetric key cryptography.



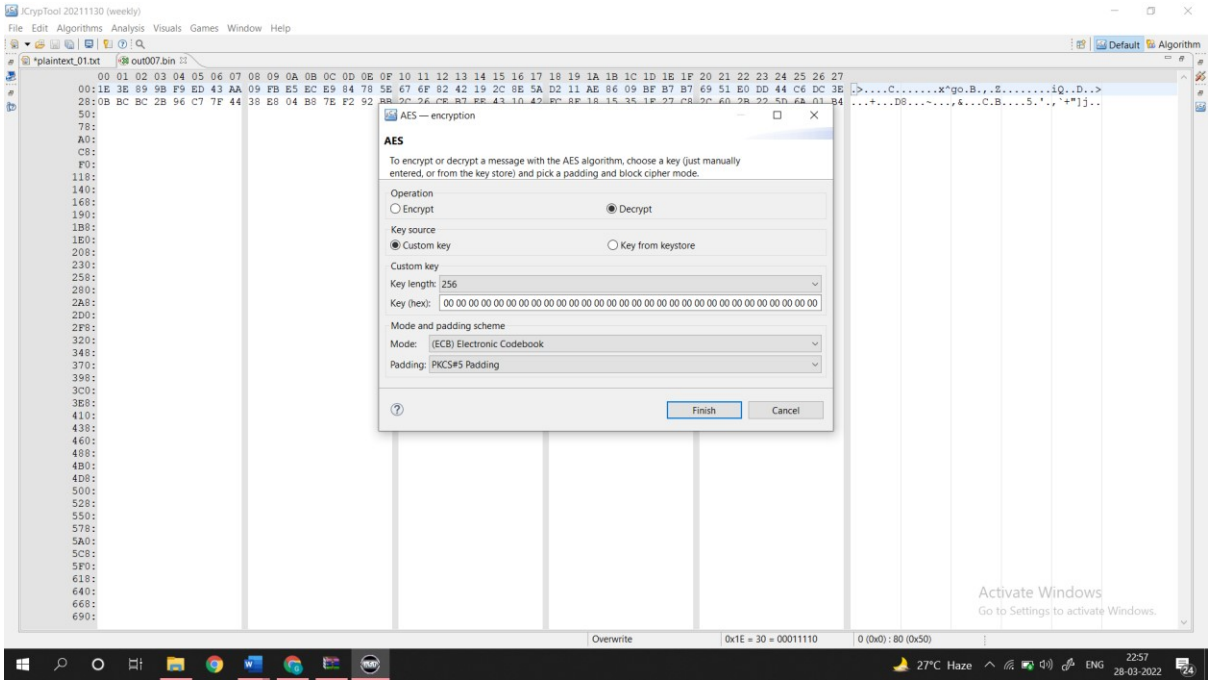
Create a text File containing plain text.



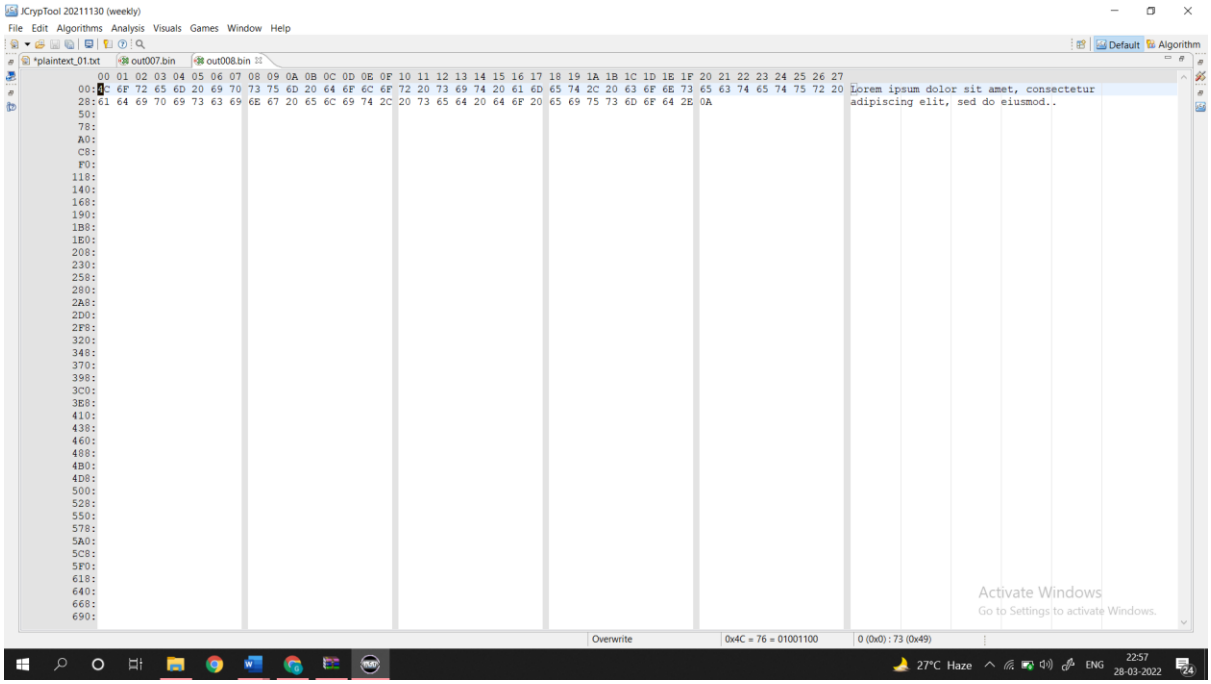
Select An Encryption Algorithm (AES-256 in this case).



Obtain the Encrypted Text



Select the decrypt option.



Obtain the decrypted text.

Implement the Identity-based Encryption (IBE). Use the email address of the recipient to generate the key for a destination.

```
import java.math.BigInteger;

    public class Client {

        public static void main(String args[]){

            String
message,id_sender="alice@home",temp,id_recipient="bob@home";

            id_recipient = args[1];
            id_sender = args[0];
            message=args[2];

            System.out.println("ID (sender): "+id_sender+","+" ID
(recipient): "+id_recipient);
            System.out.println("Message: "+message);

            message = id_recipient+" has sent this message:\n"+message;
            byte[] m = message.getBytes();

            PKG pkg = new PKG();
            BigInteger Public_key = pkg.get_public_key(id_recipient);
            Public_key = pkg.get_public_key(id_sender);
            BigInteger n = pkg.getn();


            System.out.println("Recipient Public Key: " + Public_key);

            System.out.println("\nEncrypted message:");

            System.out.println(bytesToString((new
BigInteger(m)).modPow(Public_key, n).toByteArray()));

            System.out.println("\n==== Server");
            System.out.println("id_sender (recipient): "+id_recipient);

            BigInteger Private_key =pkg.get_private_key(id_sender);

            System.out.println("Private Key is: " +   Private_key );

            System.out.println("\nDecrypted message:");

            byte [] enc = (new BigInteger(m)).modPow(Public_key,
n).toByteArray();
```

```
            byte [] dec = (new BigInteger(enc)).modPow(Private_key,
n).toByteArray();
            System.out.println(new String(dec));

        }

        private static String bytesToString(byte[] encrypted) {

            String str = "";

            for (byte b : encrypted) {
                str += Byte.toString(b);
            }

            return str;

        }
    }
```

OUTPUT:

```
ID (sender): sender_alice@gmail.com, ID (recipient): reciever_bob@gmail.com
Message: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.
Recipient Public Key: 225348199

Encrypted message:
39-109-751005047-25-92-120-112-113-232-102125-72114-813-7783-3023-1987910510970-2540-10448-12070-22-73-12435-
247516701151135l3-11138325-61105-451538110-97-5290-89102-20-7044-124691-28-41-529115-255295-4264-76-10418-119-49-
45-110-21-662829-6025-12362-1053512227-105-120-121026126-636111964-109791-118-121114-123875-91-59-103-59118-2-58-
7419-3352-3711445-12812694-7640-11584-3-39-1668-22936-4522-117273-762-100112-58127-111-14-5363842130-73397240-90-
4044066-125-76-42568121-3867149-48-65118-110581723-74-71-6424-379-30-115116125197072-68-118-71661249273-6557-103-
126-36-105-3216-44556491-70-49-5059-22122117-474257-14-46-22-744383-99-215365-721410403-5525-191026291-127-10-128

==== Server
id_sender (recipient): reciever_bob@gmail.com
Private Key is:
159527935544959893228665377548982891411258379788549411442133815291041081578432122149515890394404842352241655826451
795201760356445883075611722644446505566389414231242373002296212598253339303576261717762659993167088716245267944470
013507202787901310630955855655923846385841300902173040045293714687531549305608048706961047910828544395601012275322
149824412635432998293309015337886700564300552473047417667627715838509618736798747763346230073471254956100298576735
886748787909437222248506758953053061646654957034043319723893214033246512896165438563163294300435041888658144630532
450732508067757189452729786739725566762875554 79

Decrypted message:
[(û`~eö#?c□q¨Šš¾Ð□ü>±êvQû_□IìµYÛ¡ä□□÷ëN·˜óM5úßN-µà{
u"Wu¤J□Â2o'z□ª ij¨½i      &6X_
fºYhè?e□=ÉÝ!□*P¥Úf^¬K□>4î¿Ø»K8Ä?*}□îÔ.Ahí†#ëÜó□¶
âfæ|¾¾Ù\ú®7M\Wf{²D*fÒI"Ç"}□5z}□Ð2'(úÎOÃ)Ov„□ªeaµ8ïGÃC-Ž±□@)†ì8Â±¦a-□ÜÞHÏÅl¢RUãæW'#Ã0?ÂôY#ö]rz□ª□cGë7;□[Ao?
ýì□šñrj-'3b&g□   '@Dw□
```

To Study and Work with KF Sensor Intrusion Detection Tool. Setup a honeypot and monitor the honeypot on the network.

HONEYPOTS

When it comes to computer security, honeypots are all the rage. Honeypots can detect unauthorized activities that might never be picked up by a traditional intrusion detection system. Furthermore, since almost all access to a honeypot is unauthorized, nearly everything in a honeypot's logs is worth paying attention to. Honeypots can act as a decoy to keep hackers away from your production servers. At the same time though, a honeypot can be a little tricky to deploy. In this article, I will walk you through the process of deploying a honeypot.

INTRODUCTION

There are many different types of honeypot systems. Honeypots can be hardware appliances or they can be software based. Software based firewalls can reside on top of a variety of operating systems. For the most part though, honeypots fall into two basic categories; real and virtual.

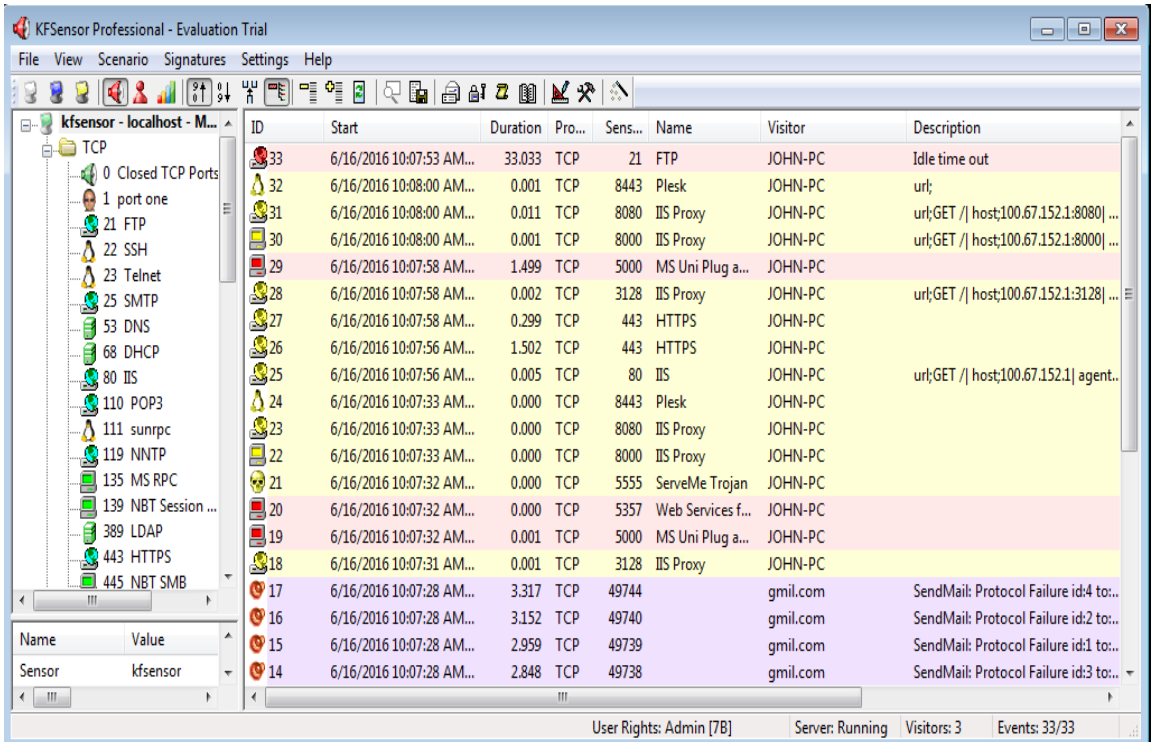
A virtual honeypot is essentially an emulated server. There are both hardware and software implementations of virtual honeypots. For example, if a network administrator was concerned that someone might try to exploit an FTP server, the administrator might deploy a honeypot appliance that emulates an FTP server.

Downloading and installing KF Sensor

- The KF Sensor download consists of a 1.7 MB self-extracting executable file.
- Download the file and copy it into an empty folder on your computer.
- When you double click on the file, it will launch a very basic Setup program.
- The only thing special that you need to know about the Setup process is that it will require a reboot

Using KFSensor

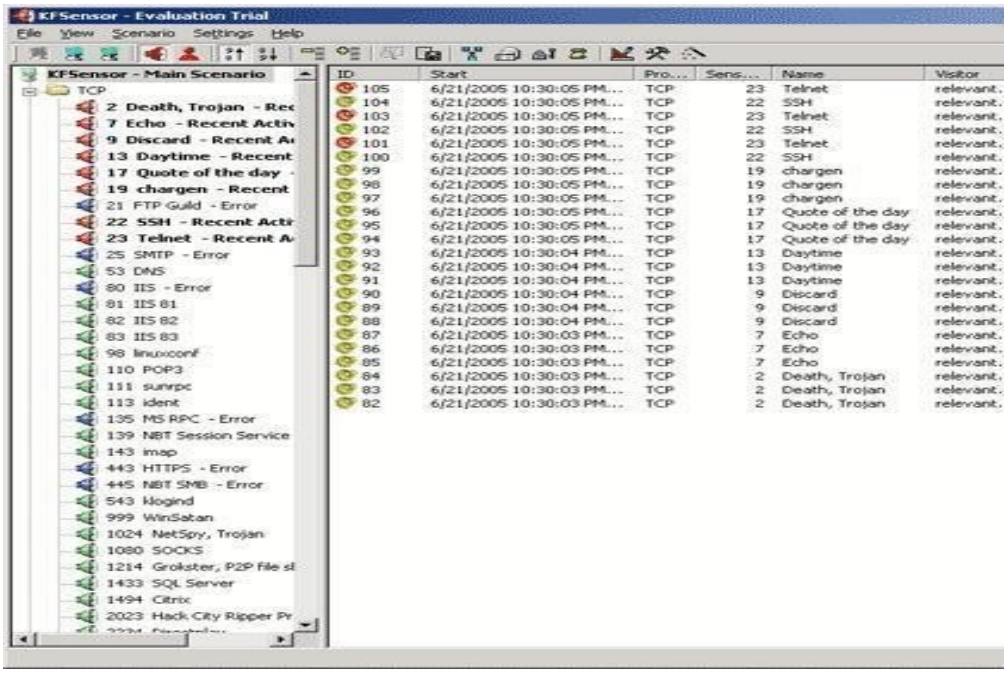
Step1: You will see the main KFSensor screen shown



- As you can see, the column on the left contains a list of port numbers and what the port is typically used for.
- If the icon to the left of a port listing is green, it means that KFSensor is actively monitoring that port for attacks.
- If the icon is blue, it means that there has been an error and KFSensor is not watching for exploits aimed at that particular port.

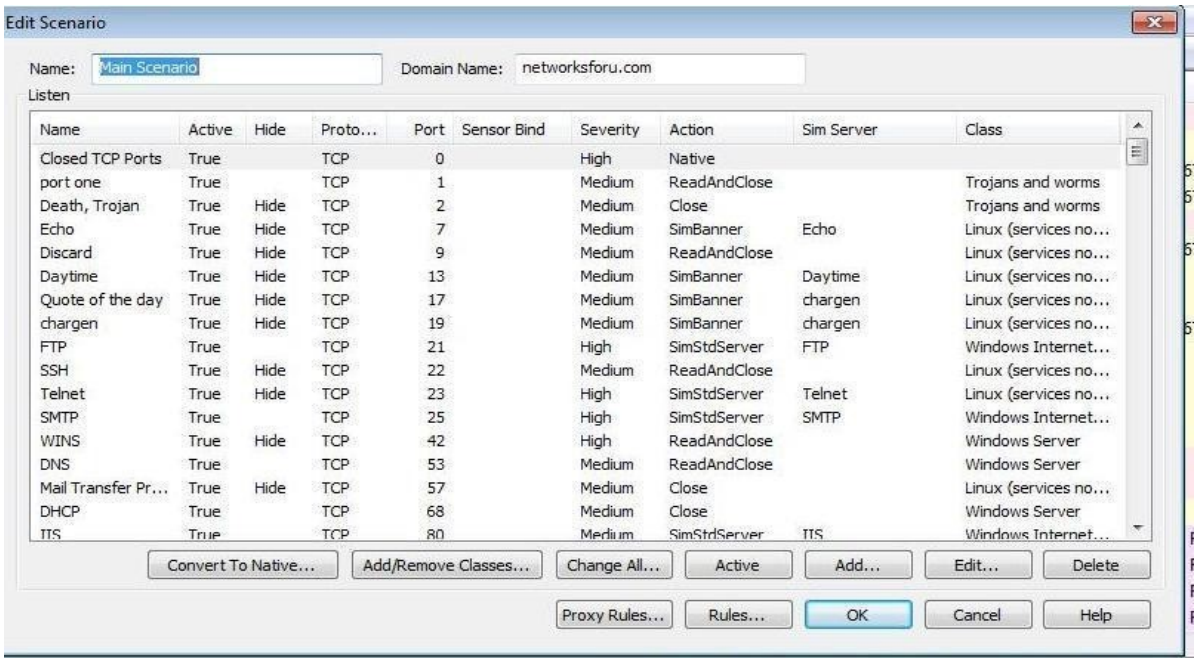
Testing the software

- Once you've got the software up and running, one of the best things that you can do is to test the software by launching a port scan against the machine that's running KFSensor.
  - For the port scan, we using the HostScan.
- It simply scans a block of IP addresses, looking for open ports. Figure B shows how the KFSensor reacts to a partial port scan.
- If you look at Figure B, you will notice that the icons next to ports that were scanned turn red to indicate recent activity.



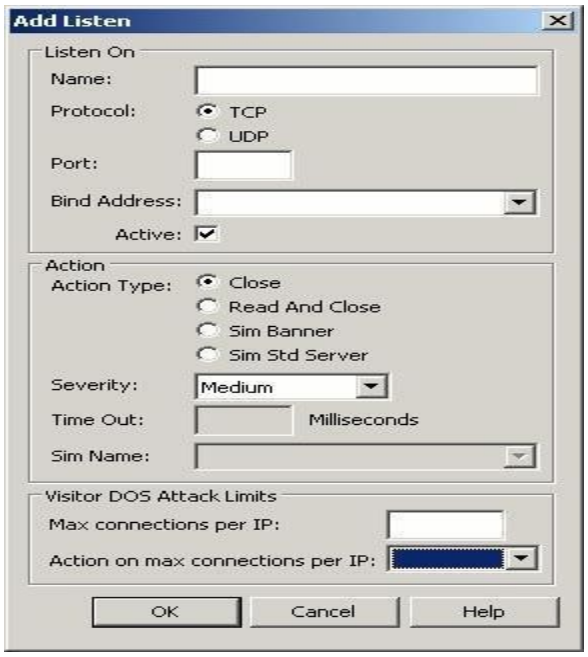
Modifying the Honeypot's behavior

- To create or modify rules, select the Edit Active Scenario command from the scenario menu.
- When you do, you will see a dialog box which contains a summary of all of the existing rules.
- You can either select a rule and click the Edit button to edit a rule, or you can click the Add button to create a new rule.
- Both procedures work similarly.

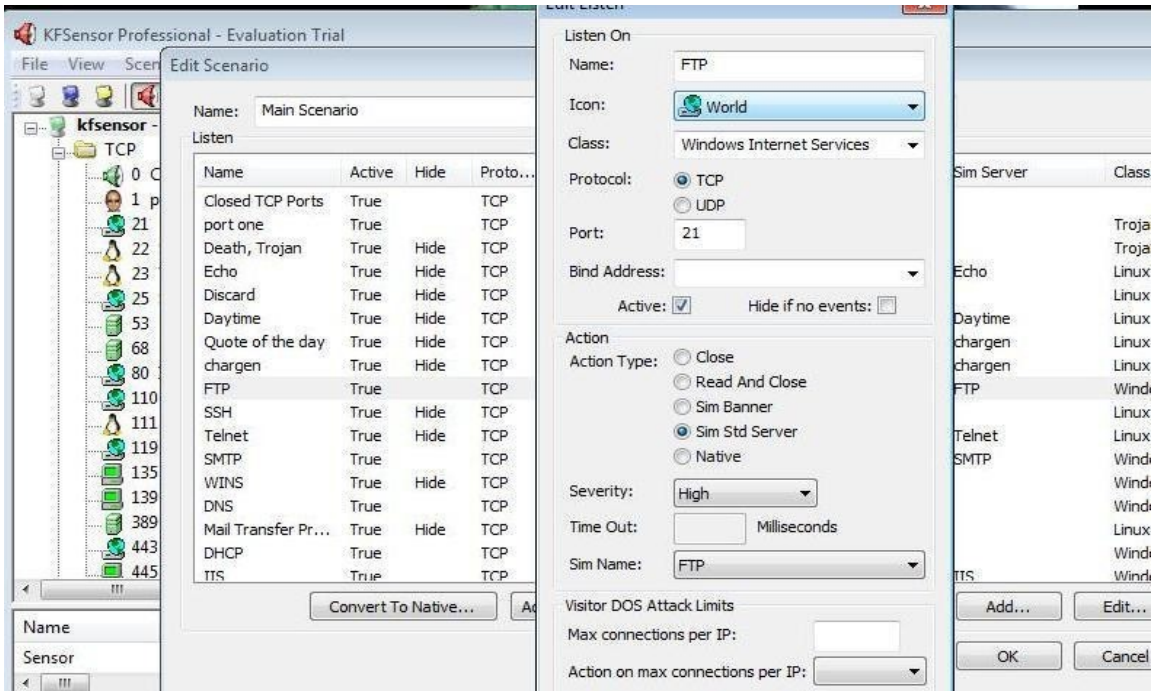


Click the Add button and you will see the Add Listen dialog box, shown in Figure D.

- The first thing that this dialog box asks for is a name. This is just a name for the rule.
- Pick something descriptive though, because the name that you enter is what will show up in the logs whenever the rule is triggered.



Click on Edit Button



- The next few fields are protocol, port, and Bind Address. These fields allow you to

choose what the rule is listening for. For example, you could configure the rule to listen to TCP port 1023 on IP address 192.168.1.100. The bind address portion of the rule is optional though. If you leave the bind address blank, the rule will listen across all of the machine's NICs.

- Now that you have defined the listener, it's time to configure the action that the rule takes when traffic is detected on the specified port. Your options are close, read and close, Sim Banner, and SimStd Server.

- The close option tells the rule to just terminate the connection. Read and close logs the information and then terminates the connection. The SimStd Server and Sim Banner options

  ation, such as what you might use to emulate an FTP server.

- The Sim STD Server option allows you to emulate a more complex server, such as an IIS server.

- If you choose to use one of the sim options, you will have to fill in the simulator's name just below the Time Out field.

- The other part of the Action section that's worth mentioning is the severity section. KFSensor treated some events as severe and other events as a more moderate threat. The dialog box's Severity drop down list allows you to determine what level of severity should be associated with the event that you are logging.

- The final portion of the Add Listen dialog box is the Visitor DOS Attack Limits section. This section allows you to prevent denial of service attacks against KFSensor. You can determine the maximum number of connections to the machine per IP address (remember that this applies on a per rule basis).

- If your threshold is exceeded, you can choose to either ignore the excessive connections or you can lock out the offending IP address.

- Now that you have configured the new rule, select the Active Button to Enable/Disable. The new rule should now be in effect.

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.

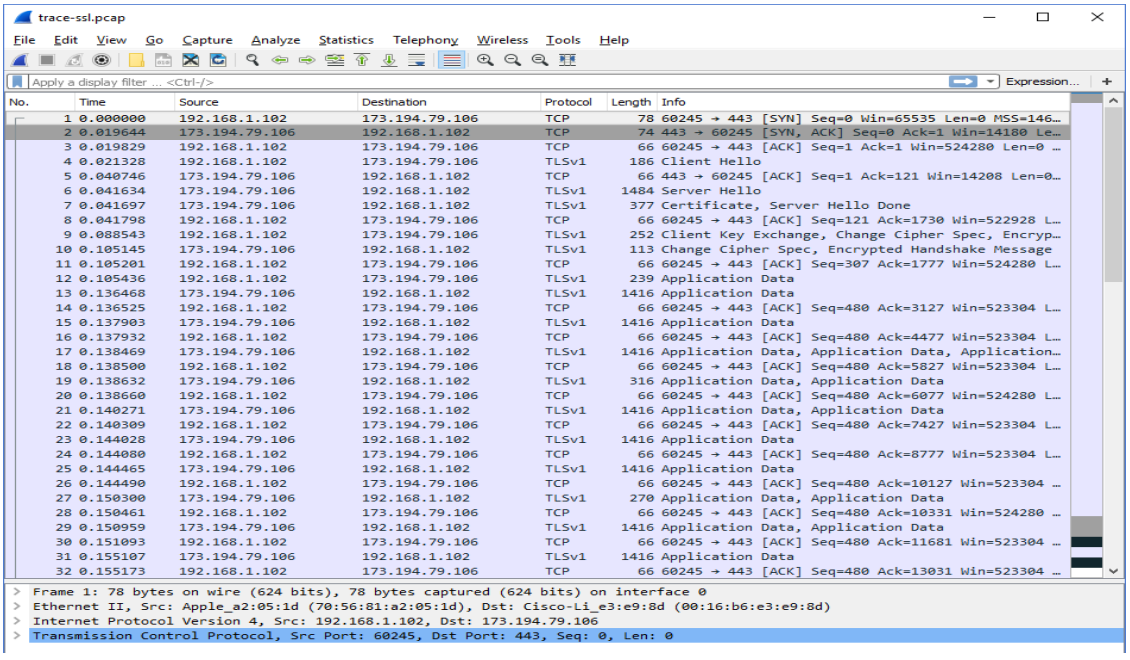


Figure 1: Trace of "HTTPS" traffic

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.

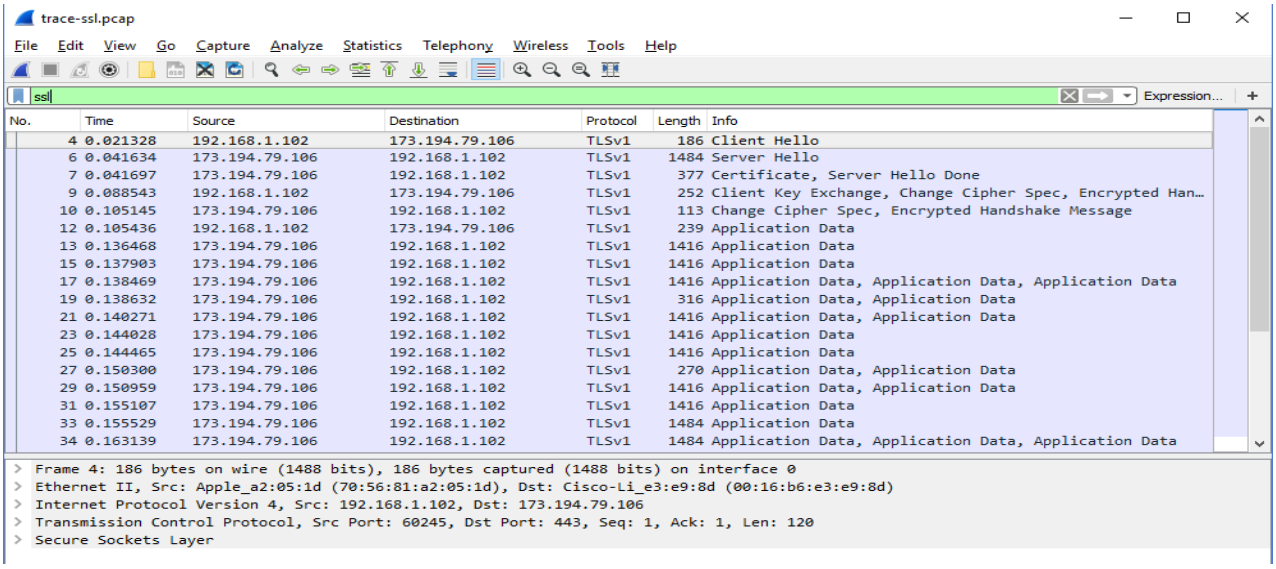
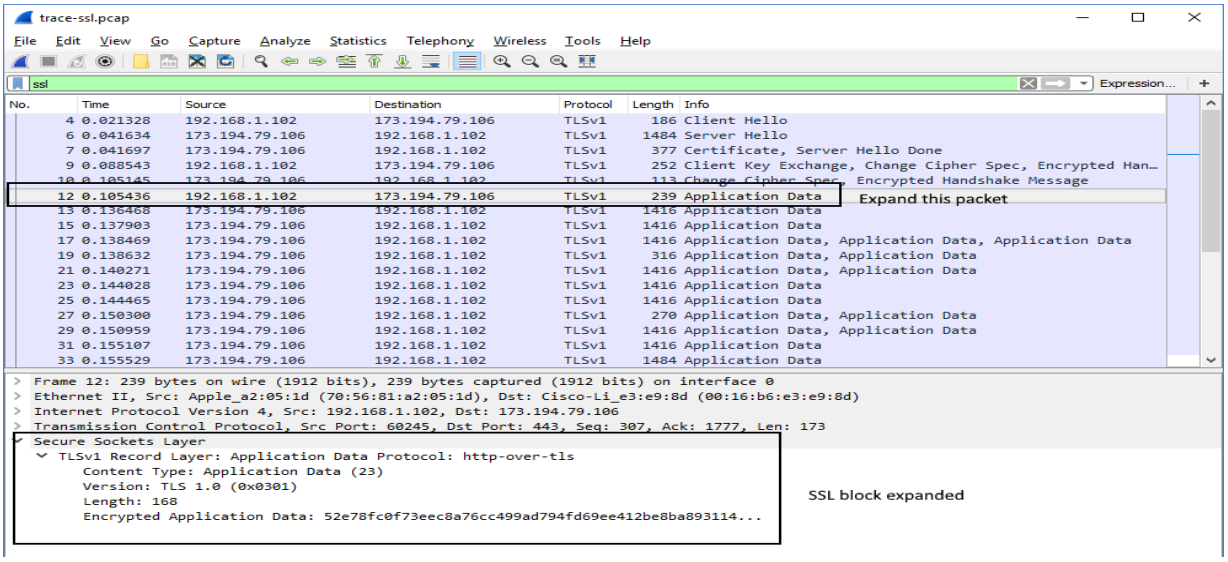


Figure 2: Trace of "SSL" traffic showing the details of the SSL header

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 fol- lowed by a Length field giving the length of the record. Last comes the contents of the record. Applica- tion Data records are sent after SSL has secured the connection, so the contents will show up as en- crypted data. To see within this block, we could configure Wireshark with the decryption key. This is pos- sible, 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:

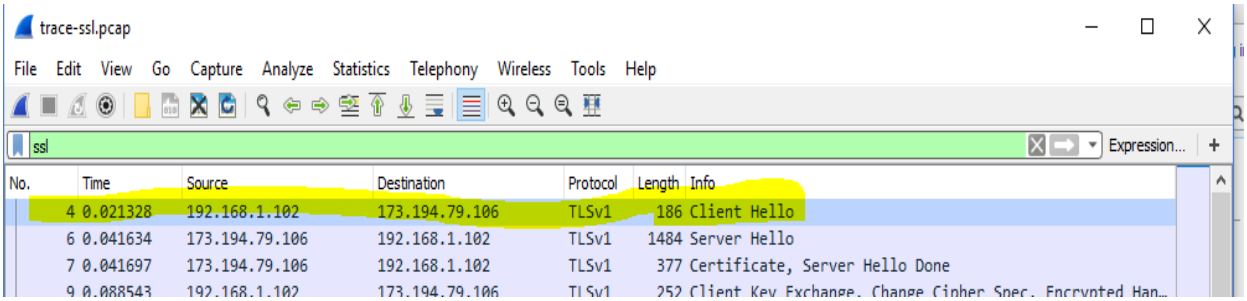
a. Client (the browser) and Server (the web server) both send their Hellos
b. Server sends its certificate to Client to authenticate (and optionally asks for Client Certificate)
c. Client sends keying information and signals a switch to encrypted data.
d. Server signals a switch to encrypted data.
e. 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.

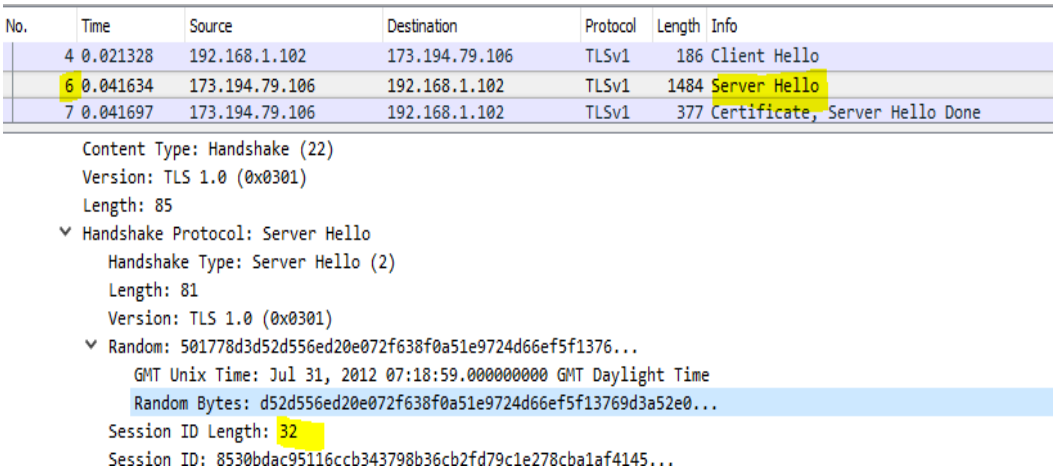
4. *Select packet #4, which is a TLS Client Hello message*



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 Cli- entHello 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 speci- fication 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 certifi- cate.

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)

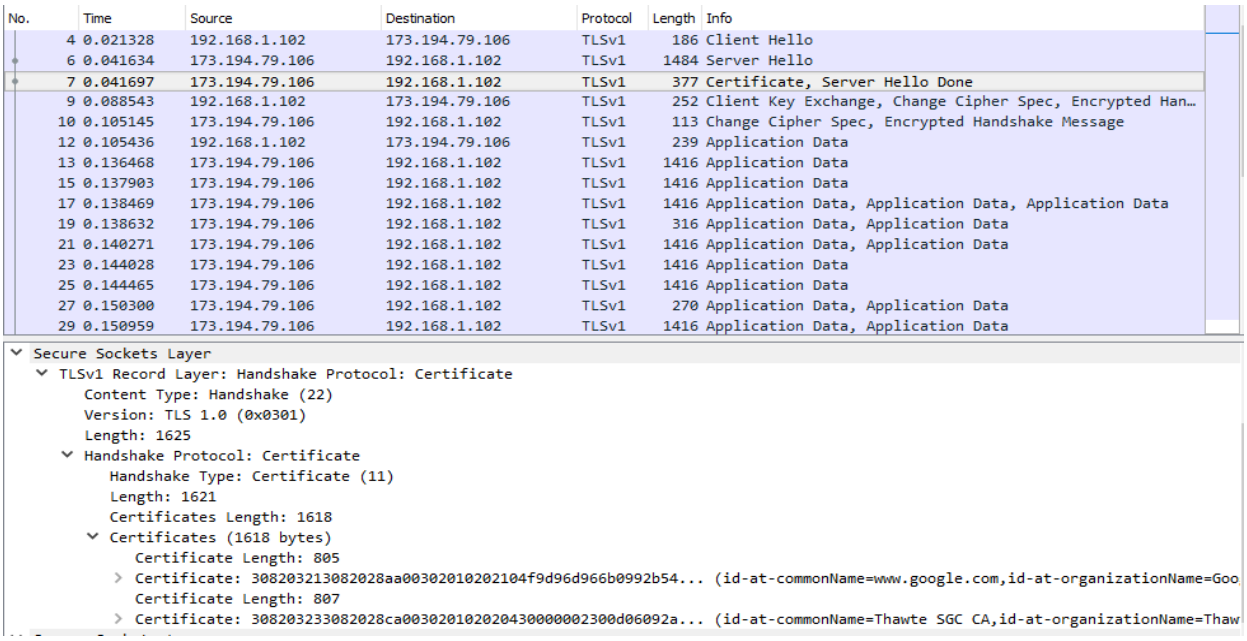


The Cipher method chosen by the Server is *TLS_RSA_WITH_RC4_128_SHA (0x0005)*. The Client will list the different cipher methods it supports, and the Server will pick one of these methods to use.

```
Session ID Length: 32
Session ID: 8530bdac95116ccb343798b36cb2fd79c1e278cba1af4145...
Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
Compression Method: null (0)
Extensions Length: 9
```

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).*



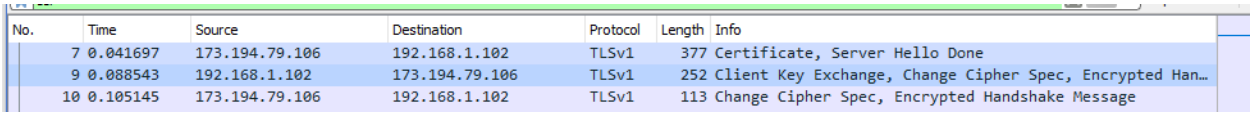
As with the Hellos, the contents of the Certificate message are visible because an encryption scheme is not yet established. It should come after the Hello messages.

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

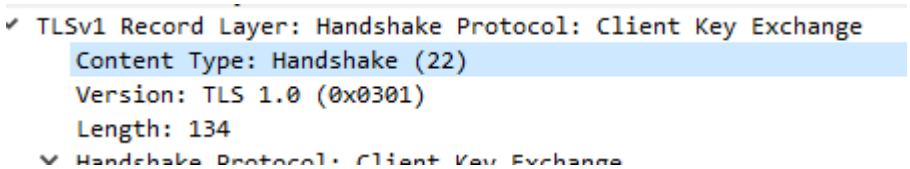
7. *Find and inspect the details of the Client Key Exchange and Change Cipher messages i.e. packet #9 (see below)*



No.	Time	Source	Destination	Protocol	Length	Info
7	0.041697	173.194.79.106	192.168.1.102	TLSv1	377	Certificate, Server Hello Done
9	0.088543	192.168.1.102	173.194.79.106	TLSv1	252	Client Key Exchange, Change Cipher Spec, Encrypted Han…
10	0.105145	173.194.79.106	192.168.1.102	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message

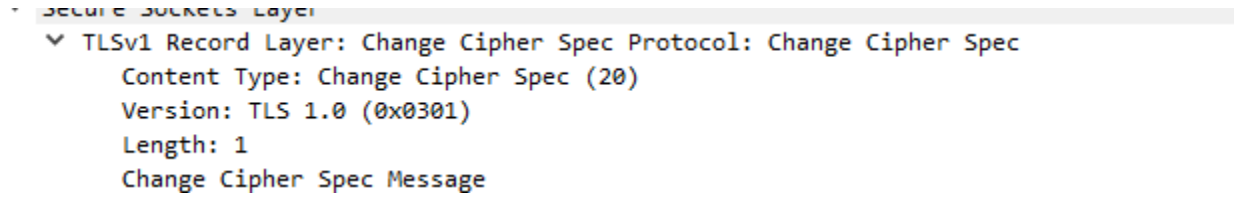
The key exchange message is sent to pass keying information so that both sides will have the same se- cret session key. The change cipher message signal a switch to a new encryption scheme to the other party. This means that it is the last unencrypted message sent by the party.

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.



```
▾ TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 134
  ▾ Handshake Protocol: Client Key Exchange
```

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



```
▾ Secure Sockets Layer
  ▾ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.0 (0x0301)
      Length: 1
      Change Cipher Spec Message
```

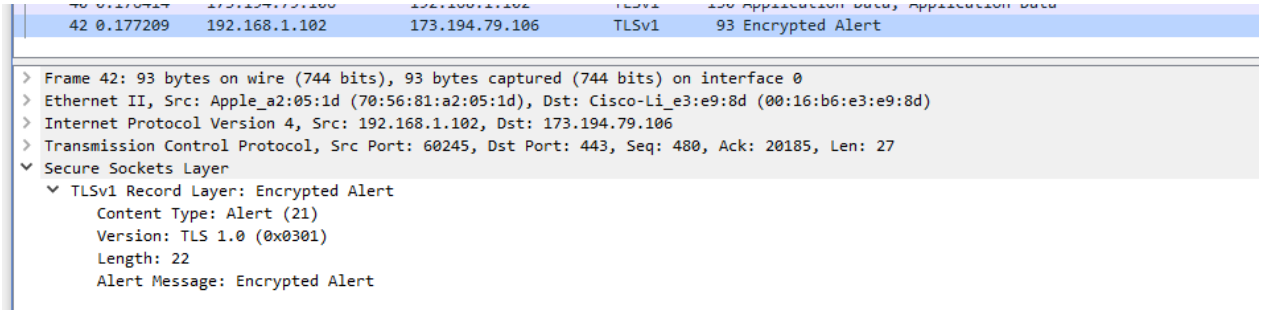
That is, this message is part of its own protocol and not the Handshake protocol.

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 mes- sage 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 connec- tion. 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 “En-
crypted Alert”. Presumably is it a “close_notify” alert to signal that the connection is ending, but we can-
not 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 do different things. But you'll probably wish they weren't doing any of them on your device.

Common 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.

Trojan banker

This Trojan takes aim at your financial accounts. It's designed to steal your account information for all the things you do online. That includes banking, credit card, and bill pay data.

Trojan IM

This Trojan targets instant messaging. It steals your logins and passwords on IM platforms.

That's just a sample. There are a lot more.

To work with Snort tool to demonstrate Intrusion Detection System. Download SNORT from snort.org.

1. Configure and Use Snort IDS on Windows

Steps to configure Snort on Widnows machine and how to use it for detection of attacks.

2. Steps:

1. Download Snort from "http://www.snort.org/" website.
2. Also download Rules from the same website. You need to sign up to get rules for registered users.
3. Click on the Snort_(version-number)_Installer.exe file to install it. By-default it will install snort in the "C:\Snort" directory.
4. Extract downloaded Rules file: snortrules-snapshot-(number).tar.gz
5. Copy all files from the "rules" directory of the extracted folder and paste them into "C:\Snort\rules" directory.
6. Copy "snort.conf" file from the "etc" directory of the extracted folder and paste it into "C:\Snort\etc" directory. Overwrite existing file if there is any.
7. Open command prompt (cmd.exe) and navigate to directory "C:\Snort\bin" directory.
8. To execute snort in sniffer mode use following command:

   snort -dev -i 2

   -i indicate interface number.

   -dev is used to run snort to capture packets.

   To check interface list use following command: snort -W
9. To execute snort in IDS mode, we need to configure a file "snort.conf" according to our network environment.
10. Set up network address we want to protect in snort.conf file. To dothat look for "HOME_NET" and add your IP address.

    var HOME_NET 10.1.1.17/8
11. You can also set addresses or DNS_SERVERS, if you have any. otherwise go to the next step.
12. Change RULE_PATH variable with the path

of rules directory. var RULE_PATH

c:\snort\rules

13. Change the path of all libraries with the name and path on your system. or change path of snort_dynamicpreprocessorvariable.

You need to do this to all library files in the "C:\Snort\lib" directory. The old path might be something like: "/usr/local/lib/...". you need to replace that path with you system path.

14. Change path of the "dynamicengine" variable value in the "snort.conf" file with the path of your system. Such as:

dynamicengine C:\Snort\lib\snort_dynamicengine\sf_engine.dll

15 Add complete path for "include classification.config" and "include reference.config" files. include c:\snort\etc\classification.config

include c:\snort\etc\reference.config

16. Remove the comment on the line to allow **ICMP** rules, if it is alredy commented. include $RULE_PATH/icmp.rules

17. Similary, remove the comment of ICMP-info rules comment, if it is already commented. include $RULE_PATH/icmp-info.rules

18 To add log file to store alerts generated by snort, search for "output log" test and add following line:

output alert_fast: snort-alerts.ids

19. Comment whitelist $WHITE_LIST_PATH/white_list.rules and blacklist $BLACK_LIST_PATH/black_list.rules lines. Also ensure that you add change the line above

$WHITE_LIST_PATH

Change nested_ip inner , \ to nested_ip inner #, \

20. Comment following lines:

#preprocessor normalize_ip4

#preprocessor

normalize_tcp: ips ecn

stream #preprocessor

normalize_icmp4

#preprocessor

normalize_ip6

#preprocessor normalize_icmp6

21. Save the "snort.conf" file and close it.

22. Go to the "C:\Snort\log" directory and create a file: snort-alerts.ids

23. To start snort in IDS mode, run

following command: snort -c

c:\snort\etc\snort.conf -l

c:\snort\log -i 2

Above command will generate log file that will not be readable without using a tool. To read it use following command:

C:\Snort\Bin\> snort -r ..\log\log-filename

To generate Log files in ASCII mode use following command while running

snort in IDS mode: snort -A console -i2 -c c:\Snort\etc\snort.conf -l

c:\Snort\log -K ascii

24. Scan the computer running snort from another computer using PING or launch attack. Then check snort-alerts.ids file the log folder.

Implement a code to simulate buffer overflow attack.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

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

    // Reserve 5 byte of buffer plus the terminating NULL.
    // should allocate 8 bytes = 2 double words,
    // To overflow, need more than 8 bytes...
    char buffer[5]; // If more than 8 characters input
                                        // by user, there will be access
                                        // violation, segmentation fault

    // a prompt how to execute the program...
    if (argc < 2)
    {
                printf("strcpy() NOT executed....\n");
                printf("Syntax: %s <characters>\n", argv[0]);
                exit(0);
    }

    // copy the user input to mybuffer, without any
    // bound checking a secure version is strcpy_s()
    strcpy(buffer, argv[1]);
    printf("buffer content= %s\n", buffer);

    // you may want to try strcpy_s()
    printf("strcpy() executed...\n");

    return 0;
    }
```

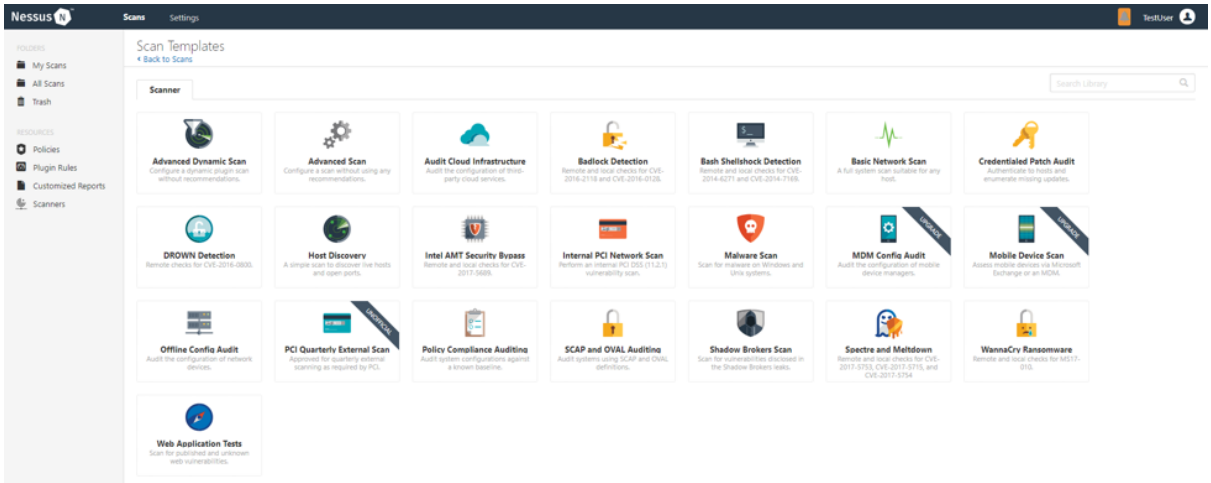
Use the Nessus tool to scan the network for vulnerabilities.

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.

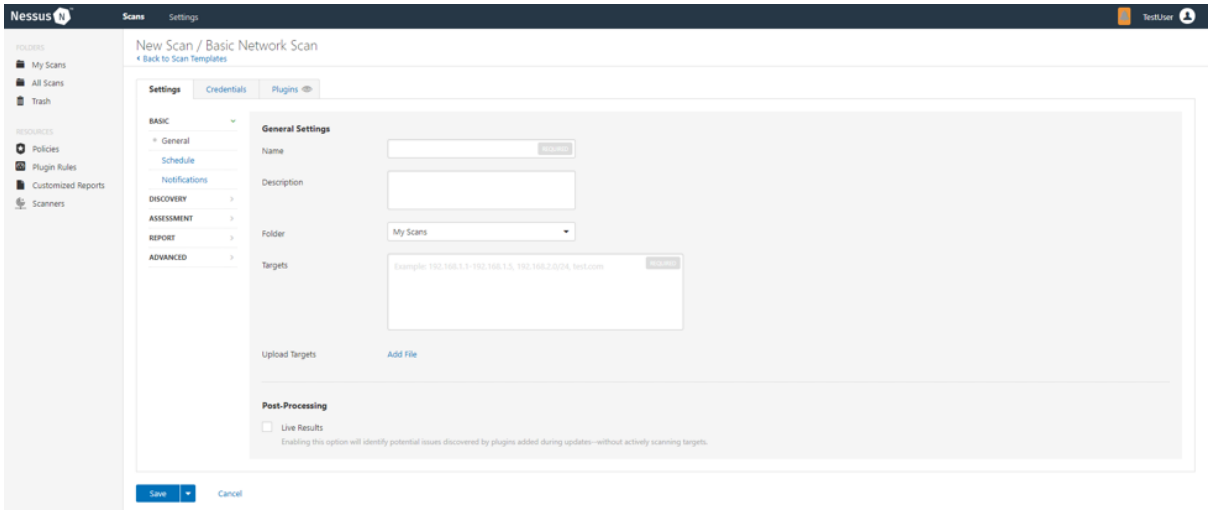
To see a full list of the types of templates available in Nessus, see Scan and Policy Templates. To quickly get started with Nessus, use the Basic Network Scan template.

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.



The following are Basic settings:

Setting	Description
Name	Specifies the name of the scan or policy. This value is displayed on the Nessus interface.
Description	(Optional) Specifies a description of the scan or policy.
Folder	Specifies the folder where the scan appears after being saved.

Setting	Description
Targets	Specifies one or more targets to be scanned. If you select a target group or upload a targets file, additional targets.

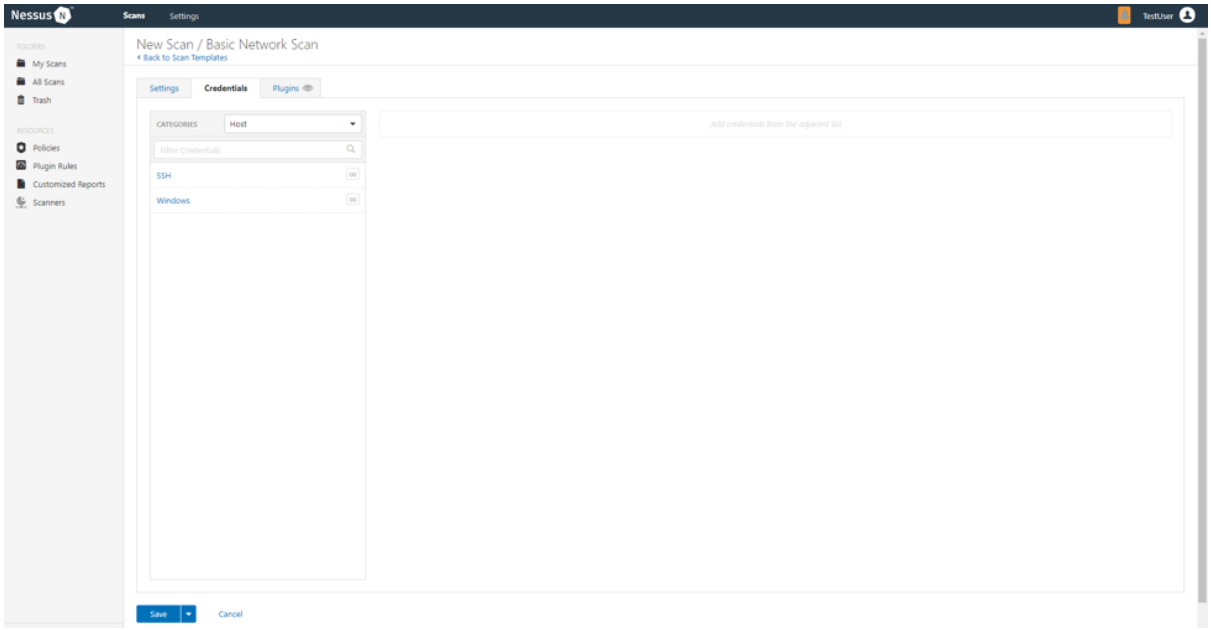
2. Configure remaining settings

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.

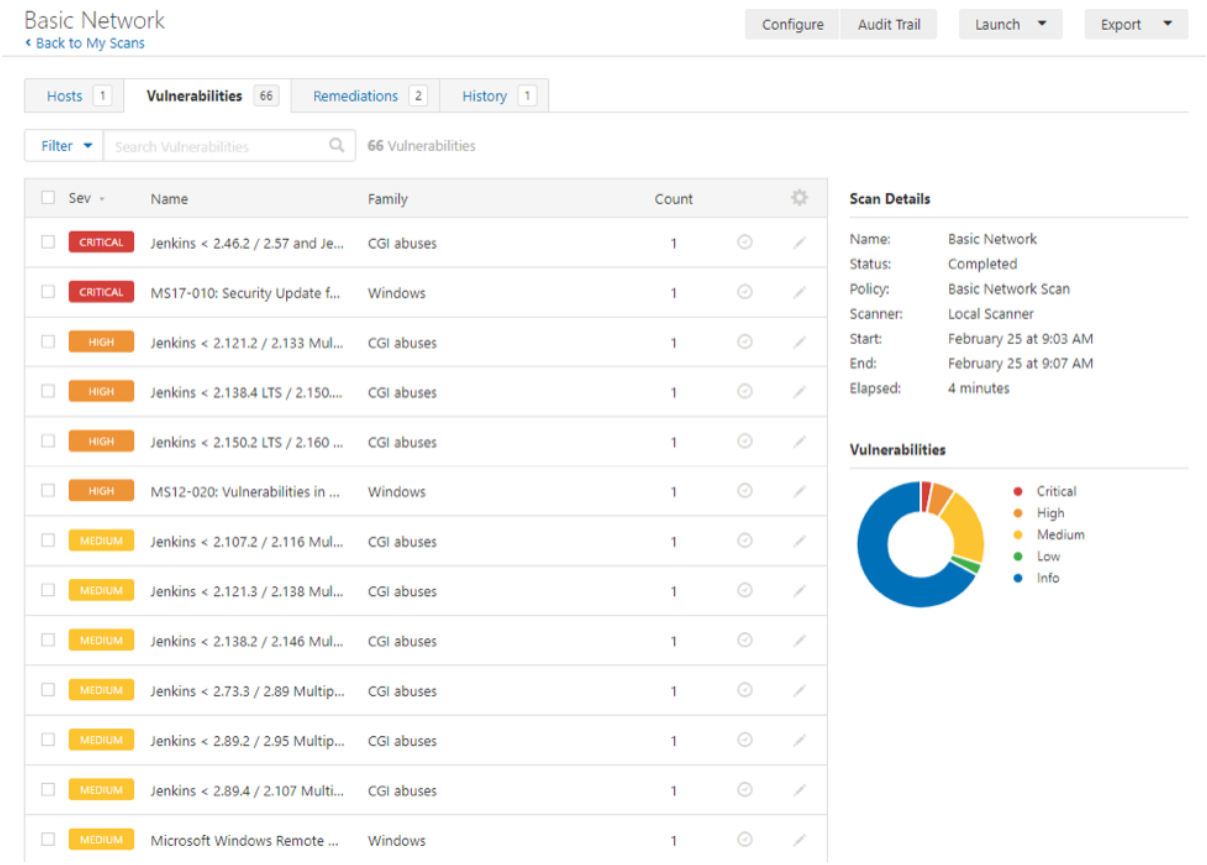
Step 4: Viewing Your Results

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 views:

Page	Description
Hosts	Displays all scanned targets.
Vulnerabilities	List of identified vulnerabilities, sorted by severity.
Remediations	If the scan's results include remediation information, this list displays all remediation details, s vulnerabilities.
Notes	Displays additional information about the scan and the scan's results.
History	Displays a list of scans: Start Time, End Time, and the Scan Statuses.

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.