MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Classical Cryptographic Techniques
"Ceasar Cipher"

Lab Assignment 1

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

February 19, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement any classical cryptographic technique.

2 Objectives

To conceal the context of some message from all accept the sender and recipient.

3 Theory

3.1 Cryptography

Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries. More generally, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages; various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation are central to modern cryptography.

3.2 Types of Cryptography

3.2.1 Classical Cryptography

Classical cryptography is the study of cryptography before the advent of modern computers. The classical ciphers are those ciphers that were in use before the advent of computers.

Some Classical Cryptographic Techniques are:

1. *Caesar Cipher* - A Caesar cipher is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence.

   ```
   Example:
   Plain Text: ABCDEFGHIJKLMNOPQRSTUVWXYZ
   Key: 3
   Cipher Text: DEFGHIJKLMNOPQRSTUVWXYZABC
   ```

2. *Vigenere Cipher* - A Vigenère cipher is a method of encrypting alphabetic text by using a series of interwoven Caesar ciphers, based on the letters of a keyword. It employs a form of polyalphabetic substitution.

   ```
   Example:
   Input : Plaintext :   GEEKSFORGEEKS
   Keyword :  AYUSH
   Output : Ciphertext :  GCYCZFMLYLEIM
   ```

3. *Hill Cipher* - The Hill cipher is a polygraphic substitution cipher based on linear algebra. Invented by Lester S. Hill in 1929, it was the first polygraphic cipher in which it was practical (though barely) to operate on more than three symbols at once. The matrix used for encryption is known as a key matrix.

```
Example of Hill Cipher:
Input  : Plaintext: ACT
Key: GYBNQKURP
Output : Ciphertext: POH
```

4. *Playfair Cipher* - The Playfair cipher is a manual symmetric encryption technique and was the first literal digram substitution cipher. The scheme was invented in 1854 by Charles Wheatstone, but was named after Lord Playfair for promoting its use.

   ```
   Example:
   Key text: Monarchy
   Plain text: instruments
   Cipher text: gatlmzclrqtx
   ```

5. *Affine Cipher* - In cryptography, an affine cipher is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter. The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing which letter goes to which. As such, it has the weaknesses of all substitution ciphers. Each letter is enciphered with the function (ax + b) mod 26, where b is the magnitude of the shift.

   ```
   Example:
   Encrypted Message is : UBBAHK CAPJKX
   Decrypted Message is: AFFINE CIPHER
   Key: a = 17, b = 20
   ```

6. *DES* - The Data Encryption Standard (DES) is a symmetric-key algorithm for the encryption of electronic data. Although its short key length of 56 bits makes it too insecure for most current applications, it has been highly influential in the advancement of modern cryptography.

7. *AES* - The Advanced Encryption Standard (AES), also known by its original name Rijndael is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is based on the Rijndael cipher developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, who submitted a proposal to NIST during the AES selection process. AES is a subset of Rijndael designed by Daemen and Rijmen to be easy to implement in hardware and software. The AES standard has been adopted by the U.S. government and is now used worldwide to protect classified and sensitive but unclassified information.

8. *Substitution Ciphers* - In cryptography, a substitution cipher is a method of encoding by which units of plaintext are replaced with ciphertext, according to a fixed system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.

   A substitution cipher is a method of encoding by which units of plaintext are replaced with ciphertext, according to a fixed system; the "units" may be single letters (the most common),

pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.

9. *Transposition Ciphers* - In cryptography, a transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext. That is, the plaintext is written out in a certain order, then the ciphertext is formed by reading down the columns going left to right.

3.2.2 Modern Cryptography

Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, electrical engineering, communication science, and physics. Applications of cryptography include electronic commerce, chip-based payment cards, digital currencies, computer passwords, and military communications.

Some Modern Cryptographic Techniques are:

1. *Diffie-Hellman Key Exchange* - In cryptography, the Diffie Hellman key exchange is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as originally conceived by Whitfield Diffie and Martin Hellman. It is one of the earliest practical examples of public key exchange implemented within the field of cryptography. It was the first public key exchange protocol that did not rely on trusted third parties and was the first public key protocol for which polynomial-time algorithms were found. It is also the first public key protocol for which a practical attack was found.

2. *ElGamal Encryption* - In cryptography, ElGamal encryption is an asymmetric key encryption algorithm based on the Diffie Hellman key exchange. It is named after Taher ElGamal, who published it in 1985. ElGamal encryption is a public-key encryption scheme, meaning that a pair of keys, one public and one private, are used. The public key may be known to everyone, while the private key is kept secret. The scheme is based on the difficulty of the discrete logarithm problem in a finite field.

3. *RSA* - RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of the keys can be given to anyone. The other key must be kept private. The algorithm is based on the fact that finding the factors of a large composite number is difficult: when the factors are prime numbers, the problem is called prime factorization. It is also a key pair (public and private) generator.

4. *Digital Signature* - A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, the message has not been altered in transit, and the sender cannot deny having sent the message. Digital signatures are a common method of authentication and data integrity in computer systems and communications.

5. *Hashing* - In cryptography, a hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Pseudo Code or Algorithm

```
// Pseudo Code for Ceasar Cipher
// Input : String, Key
// Output : Ciphered String
// Function to Cipher the String

def Cipher(String, Key):
  Ciphered_String = ""
  for i in String:
    if i.isupper():
      Ciphered_String += chr((ord(i) + Key - 65) % 26 + 65)
    else:
      Ciphered_String += chr((ord(i) + Key - 97) % 26 + 97)
  return Ciphered_String
```

6 Input and Output

```
Welcome to Assignment 1 in Information and CyberSecurity, working with Ceasar Ciphers.
Enter the string that you want to Cipher.
Assignments of Information and Cybersecurity
Enter the key : 9
Applying the Ceasar Cipher to it.
jBBRPWVNWCB XO rWOXAVJCRXW JWM lHKNABNLDARCH
```

7 Code

```
# Assignment 1
# Ceaser Cipher


def get_ascii(some_char):
    if some_char.islower():
        return ord(some_char) - 97
    elif some_char.isupper():
        return ord(some_char) - 65
    else:
        return -1


def ceaser_cipher(plain_text, key):
    cipher_letter = ""
    cipher = []
    for i in plain_text:
```

```
        if i == " ":
            cipher.append(" ")
            continue
        if i.islower():
            cipher_letter = chr(((get_ascii(i) + key) % 26) + 97).upper()
        else:
            cipher_letter = chr(((get_ascii(i) + key) % 26) + 65).lower()

        cipher.append(cipher_letter)
    return cipher


def is_valid(plain_text):
    for i in plain_text.split(" "):
        if not i.isalpha():
            return False
    return True


def main():
    plain_text = "1"
    key = -1

    print(
        "Welcome to Assignment 1 in Information and CyberSecurity, working with Ceaser Ciphers. "
    )
    print("Enter the string that you want to Cipher. ")

    # take inputs from the user.
    plain_text = input()

    while not is_valid(plain_text):
        print("Invalid input, try again!")
        plain_text = input()

    key = int(input("Enter the key : "))

    while key <= 0 or key >= 26:
        print("Key input, try again!")
        key = int(input("Enter the key : "))

    print("Applying the Ceaser Cipher to it. ")
    cipher_text = ceaser_cipher(plain_text, key)
    print("".join(cipher_text))
    return


main()
```

Listing 1: "Ceasar Cipher"

```
import math

ADDED_CHAR = "*"


def rail_transportation(plain_text, key):
    number_of_cols = math.ceil(len(plain_text) / key)
```

```
    matrix = []

    for i in range(number_of_cols * key - len(plain_text)):
        plain_text += "*"

    for _ in range(key):
        col_matrix = []
        for j in range(number_of_cols):
            col_matrix.append(plain_text[j * key + _])
        matrix.append(col_matrix)

    cipher_text = [j for i in matrix for j in i]

    return "".join(cipher_text)


def main():
    # plain_text = input("Enter the Plain text: ")
    # key = int(input("Enter the number of rows as the key:" ))
    plain_text = "GYANENDR"
    key = 3
    if key <= 1:
        print("The key length is smaller than 1, it must be greater! Run again.")
        return

    print("The Plain Text you entered is: ", plain_text)
    print("The key you entered is: ", key)

    cipher_text = rail_transportation(plain_text, key)
    print("The Ciphered Text is: ", cipher_text)

    print("Decrypting now!")

    plain_text = rail_transportation(cipher_text, key)

    plain_text = plain_text.replace(ADDED_CHAR, "")

    print("The Decrypted Plain Text is: ", plain_text)


main()
```

Listing 2: "Rail Transportation Cipher"

8 Conclusion

Thus, learnt about the different kinds of ciphers, classical cryptographic techniques, and how to implement some of them in python.

9 FAQ

1. **What are various classical ciphers?**
   Answer: There are many different kinds of ciphers, some of them are:
   (a) *Caesar Cipher*
   (b) *Vigenere Cipher*
   (c) *Rail Transportation Cipher*
   (d) *Hill Cipher*
   (e) *Playfair Cipher*
   (f) *Autokey Cipher*
   (g) *Columnar Transposition Cipher*
   (h) *Affine Cipher*
   (i) *Monoalphabetic Cipher*
   (j) *Polyalphabetic Cipher*
   (k) *Transposition Cipher*
   (l) *Substitution Cipher*

2. **Compare steganography and Cryptography**

   Answer: **Steganography** is the practice of concealing a file, message, image, or video within another file, message, image, or video.

   Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries. More generally, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages; various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation are central to modern cryptography. Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, electrical engineering, communication science, and physics. Applications of cryptography include electronic commerce, chip-based payment cards, digital currencies, computer passwords, and military communications.

   Steganogrphy is a type of cryptography.

   Steganography comes from the words steganos (meaning covered or hidden) and graphein (meaning writing). It is the practice of concealing a file, message, image, or video within another file, message, image, or video.

   Cryptography however, comes from the Greek words kryptos (meaning hidden) and graphein (meaning writing). It is the practice and study of techniques for secure communication in the presence of third parties called adversaries.

3. **What are the few major applications of cryptography in the modern world?**
   Answer:

(a) *Electronic Commerce* - Cryptography is used to protect the privacy of credit card numbers, bank account numbers, and other sensitive information exchanged over the Internet.

(b) *Chip-based Payment Cards* - Cryptography is used to protect the privacy of credit card numbers, bank account numbers, and other sensitive information exchanged over the Internet.

(c) *Digital Currencies* - Cryptography is used to protect the privacy of credit card numbers, bank account numbers, and other sensitive information exchanged over the Internet.

(d) *Computer Passwords* - Cryptography is used to protect the privacy of credit card numbers, bank account numbers, and other sensitive information exchanged over the Internet.

(e) *Military Communications* - Cryptography is used to protect the privacy of credit card numbers, bank account numbers, and other sensitive information exchanged over the Internet.

4. **How can Caesar cipher be cracked?**

Answer:

(a) *Frequency Analysis* - Frequency analysis is a method of analyzing the frequency of each letter in a message. The frequency of each letter is compared to the frequency of letters in the English language. If the frequency of a letter in the message is close to the frequency of that letter in the English language, then it is likely that the letter is an English letter. This method is not very effective because it is possible to create a message that has the same frequency of letters as the English language.

(b) *Brute Force* - Brute force is a method of trying every possible key until the correct key is found. This method is very effective, but it is very time consuming.

(c) *Cryptanalysis* - Cryptanalysis is a method of analyzing the cipher to find a weakness in the cipher. This method is very effective, but it is very difficult to find a weakness in a cipher.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Classical Cryptographic Techniques - *"Fiestal Cipher"*

Lab Assignment 2

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

February 28, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement Feistal Cipher structure

2 Objectives

To understand the concepts of symmetric key cryptographic system.

3 Theory

3.1 Symmetric Key Cryptography

Symmetric key cryptography is a cryptographic system in which the same key is used for both encryption and decryption. The key is shared between the sender and the receiver. The sender encrypts the message using the key and sends it to the receiver. The receiver decrypts the message using the same key. The key is kept secret and is never sent along with the message.

The most commonly used symmetric key algorithm is the Data Encryption Standard (DES). It uses a 64-bit block size and a 56-bit key. The 64-bit block is divided into two halves of 32-bits each. The key is also divided into two halves of 28-bits each. The first half of the key is used to generate 16 subkeys. Each subkey is 48-bits long. The first 28-bits of the key are shifted left by 1 bit. The first 28-bits of the key are then shifted left by 1 bit. The second 28-bits of the key are shifted left by 1 bit. The second 28-bits of the key are shifted left by 1 bit. This process is repeated for the remaining 16 rounds. The 16 subkeys are then used to encrypt the message.

3.2 Feistal Cipher

Feistel Cipher model is a structure or a design used to develop many block ciphers such as DES. Feistel cipher may have invertible, non-invertible and self invertible components in its design. Same encryption as well as decryption algorithm is used. A separate key is used for each round. However same round keys are used for encryption as well as decryption.

3.3 Fiestal Cipher Algorithm

1. Create a list of all the Plain Text characters.
2. Convert the Plain Text to Ascii and then 8-bit binary format.
3. Divide the binary Plain Text string into two halves: left half (L1) and right half (R1)
4. Generate a random binary keys (K1 and K2) of length equal to the half the length of the Plain Text for the two rounds.

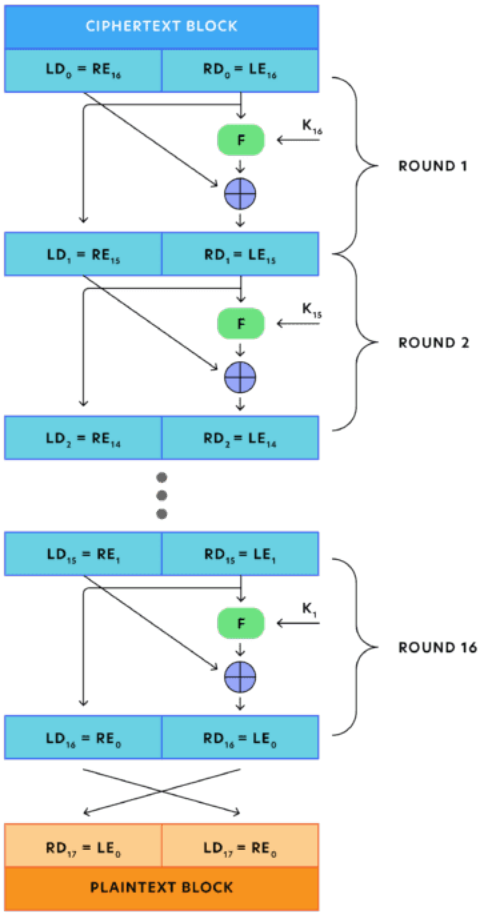


Figure 1: Fiestal Cipher Decryption Method

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

```
The plain text, key
[1, 1, 1, 1, 0, 0, 1, 1] [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
The left and right keys are :  [1, 0, 1, 0, 0, 1, 0, 0] [0, 1, 0, 0, 0, 0, 1, 1]
Starting to cipher.
The cipher text is :  [0, 1, 0, 0, 0, 0, 0, 1]
```

6 Code

```
# creating the fiester cipher.
# Assignment 2
# Krishnaraj Thadesar
# 10322108888 Batch A1

############ Defining Constants ############

block_size = 8

binary_to_decimal = {(0, 0): 0, (0, 1): 1, (1, 0): 2, (1, 1): 3}

PT_IP_8 = [2, 6, 3, 1, 4, 8, 5, 7]
PT_IP_8_INV = [4, 1, 3, 5, 7, 2, 8, 6]

S0_MATRIX = [[1, 0, 3, 2],
             [3, 2, 1, 0],
             [0, 2, 1, 3],
             [3, 1, 3, 2]]

S1_MATRIX = [
    [0, 1, 2, 3],
    [2, 0, 1, 3],
    [3, 0, 1, 0],
    [2, 1, 0, 3],
]

############ Defining P Boxes ############

PT_P_10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
PT_P_8 = [6, 3, 7, 4, 8, 5, 10, 9]
PT_P_4 = [2, 4, 3, 1]
PT_EP = [4, 1, 2, 3, 2, 3, 4, 1]


############ Functions ############


def shift_left(list_to_shift):
    """Function to shift bits by 1 to the left

    Args:
        list_to_shift (list): list of the bunch of binary bits that you wanna
    shift to left.

    Returns:
        list: shifted list.
    """
    shifted_list = [i for i in list_to_shift[1:]]
    shifted_list.append(list_to_shift[0])
    return shifted_list


def make_keys(key):
    """Function to Generate 8 bit K1 and 8 bit K2 from given 10 bit key.

    Args:
```

```
        key (list): list of 0's and 1's describing the key.

    Returns:
        (K1, K2): touple containing k1 and k2.
    """
    # make key_p10
    key_P10 = [key[i - 1] for i in PT_P_10]

    # Splitting into lshift and rshift
    key_P10_left = key_P10[: int(len(key) / 2)]
    key_P10_right = key_P10[int(len(key) / 2) :]

    # left shifting the key one time
    key_P10_left_shifted = shift_left(key_P10_left)
    key_P10_right_shifted = shift_left(key_P10_right)

    # temporarily combining the 2 shifted lists.
    temp_key = key_P10_left_shifted + key_P10_right_shifted
    # this gives the first key
    key_1 = [temp_key[i - 1] for i in PT_P_8]

    # now shifting the key 2 times for both left and right.
    key_P10_left_shifted = shift_left(key_P10_left_shifted)
    key_P10_left_shifted = shift_left(key_P10_left_shifted)

    key_P10_right_shifted = shift_left(key_P10_right_shifted)
    key_P10_right_shifted = shift_left(key_P10_right_shifted)

    temp_key = []
    temp_key = key_P10_left_shifted + key_P10_right_shifted

    key_2 = [temp_key[i - 1] for i in PT_P_8]
    # key_1, key_2 = 0, 0
    return (key_1, key_2)


def function_k(input_text, key):

    # splitting the plain text after applying initial permutation on it.
    PT_left_after_ip = input_text[: int(len(input_text) / 2)]
    PT_right_after_ip = input_text[int(len(input_text) / 2) :]

    # Applying Explansion Permutation on the right part of plain text after ip
    PT_right_after_EP = [PT_right_after_ip[i - 1] for i in PT_EP]

    # xoring the right part of pt after ep with key 1
    PT_after_XOR_with_key_1 = [x ^ y for x, y in zip(PT_right_after_EP, key)]

    # splitting the xor output of the right part of the plain text after ep.
    PT_after_XOR_with_key_1_left = PT_after_XOR_with_key_1[
        : int(len(PT_after_XOR_with_key_1) / 2)
    ]
    PT_after_XOR_with_key_1_right = PT_after_XOR_with_key_1[
        int(len(PT_after_XOR_with_key_1) / 2) :
    ]

    # getting the row and column number for S0 matrix.
    row_number_for_S0 = (
        PT_after_XOR_with_key_1_left[0],
```

```
        PT_after_XOR_with_key_1_left[-1],
    )

    col_number_for_S0 = (
        PT_after_XOR_with_key_1_left[1],
        PT_after_XOR_with_key_1_left[2],
    )

    # getting the row and column number for the S1 matrix.
    row_number_for_S1 = (
        PT_after_XOR_with_key_1_right[0],
        PT_after_XOR_with_key_1_right[-1],
    )

    col_number_for_S1 = (
        PT_after_XOR_with_key_1_right[1],
        PT_after_XOR_with_key_1_right[2],
    )

    # Getting the value from the S0 matrix.
    S0_value = S0_MATRIX[binary_to_decimal.get(row_number_for_S0)][
        binary_to_decimal.get(col_number_for_S0)
    ]

    # getting the value from the S1 matrix.
    S1_value = S1_MATRIX[binary_to_decimal.get(row_number_for_S1)][
        binary_to_decimal.get(col_number_for_S1)
    ]

    # converting the decimal numbers from s box output into binary.
    S0_value = list(binary_to_decimal.keys())[
        list(binary_to_decimal.values()).index(S0_value)
    ]
    S1_value = list(binary_to_decimal.keys())[
        list(binary_to_decimal.values()).index(S1_value)
    ]

    s_box_output = list(S0_value + S1_value)

    # applying P4 to s box output.
    s_box_output_after_P4 = [s_box_output[i - 1] for i in PT_P_4]

    # xoring the output of sbox after p4 with the left part of the plain text
    after ip.
    fk_xor_output = [x ^ y for x, y in zip(s_box_output_after_P4, PT_left_after_ip
    )]

    fk_concat_output_8_bit = fk_xor_output + PT_right_after_ip

    return fk_concat_output_8_bit


def encrypt_fiestal_cipher(plain_text, key_1, key_2):
    print("Starting to cipher. ")

    # Initial permutation for the plain text
    plain_text_after_ip = [plain_text[i - 1] for i in PT_IP_8]

    # getting partial output from running f(k) with key 1
```

```
    output_1_function_k = function_k(plain_text_after_ip, key_1)

    # splitting that output.
    output_1_function_k_left = output_1_function_k[:4]
    output_1_function_k_right = output_1_function_k[4:]

    # switching that output.
    temp = output_1_function_k_right + output_1_function_k_left

    # running function again with switched output from running f(k) with key 2
    output_2_function_k = function_k(temp, key_2)

    # running IP Inverse on it.
    cipher_text = [output_2_function_k[i - 1] for i in PT_IP_8_INV]

    return cipher_text


def decrypt_fiestal_cipher(cipher_text, key_1, key_2):
    print("Starting to Decipher. ")

    # Initial permutation for the cipher text
    cipher_text_after_ip = [cipher_text[i - 1] for i in PT_IP_8_INV]

    # getting partial output from running f(k) with key 2
    output_1_function_k = function_k(cipher_text_after_ip, key_2)

    # splitting that output.
    output_1_function_k_left = output_1_function_k[:4]
    output_1_function_k_right = output_1_function_k[4:]

    # switching that output.
    temp = output_1_function_k_right + output_1_function_k_left

    # running function again with switched output from running f(k) with key 1
    output_2_function_k = function_k(temp, key_1)

    # running IP Inverse on it.
    deciphered_plain_text = [output_2_function_k[i - 1] for i in PT_IP_8]

    return deciphered_plain_text


def main():

    # this will make the plaintext a list.
    # plain_text = [int(i) for i in input("Enter the Plain text with spaces: ").
    split()]
    # key = [int(i) for i in input("Enter the Key with spaces: ").split()]
    plain_text = [1, 1, 1, 1, 0, 0, 1, 1]
    key = [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
    print("The plain text, key")
    print(plain_text, key)

    key_1, key_2 = make_keys(key)
    print("The left and right keys are : ", key_1, key_2)

    # Generating the Cipher text.
    cipher_text = encrypt_fiestal_cipher(plain_text, key_1, key_2)
```

```
230     print("The cipher text is : ", cipher_text)
231
232
233 main()
```

Listing 1: "Fiestal Cipher"

7 Conclusion

Thus, learnt about the different kinds of ciphers, classical cryptographic techniques, and how to implement some of them in python.

8 FAQ

1. **Differentiate between stream and block ciphers.**

   (a) Stream ciphers encrypt the data one bit at a time. Block ciphers encrypt the data in blocks of fixed size.

   (b) Stream ciphers are faster than block ciphers.

   (c) Block ciphers are more secure than stream ciphers.

   (d) Stream ciphers are more suitable for real-time applications.

   (e) Block ciphers are more suitable for bulk data encryption.

   (f) Stream ciphers are more suitable for applications where the data is encrypted and decrypted in a single pass.

   (g) Block ciphers are more suitable for applications where the data is encrypted and decrypted in multiple passes.

2. **Write advantages and disadvantages of DES algorithm.**

   Advantages:

   (a) It is a fast, simple, efficient, and secure algorithm.

   (b) The algorithm has been in use since 1977. Technically, no weaknesses have been found in the algorithm. Brute force attacks are still the most efficient attacks against the DES algorithm.

   (c) DES is the standard set by the US Government. The government recertifies DES every five years, and has to ask for its replacement if the need arises.

   (d) The American National Standards Institute (ANSI) and International Organization for Standardization (ISO) have declared DES as a standard as well. This means that the algorithm is open to the public—to learn and implement.

   (e) DES was designed for hardware; it is fast in hardware, but only relatively fast in software.

   Disadvantages:

   (a) Probably the biggest disadvantage of the DES algorithm is the key size of 56-bit. There are chips available that can encrypt and decrypt a million DES operations in a second. A DES cracking machine that can search all the keys in about seven hours is available for 1 million.

   (b) DES can be implemented quickly on hardware. But since it was not designed for software, it is relatively slow on it.

   (c) It has become easier to break the encrypted code in DES as the technology is steadily improving. Nowadays, AES is preferred over DES.

   (d) DES uses a single key for encryption as well as decryption as it is a type of symmetric encryption technique. In case that one key is lost, we will not be able to receive decipherable data at all.

3. **Explain block cipher modes of operations.**

(a) *Electronic Code Book (ECB)*

i. ECB mode stands for Electronic Code Block Mode. It is one of the simplest modes of operation. In this mode, the plain text is divided into a block where each block is 64 bits. Then each block is encrypted separately. The same key is used for the encryption of all blocks. Each block is encrypted using the key and makes the block of ciphertext.

ii. At the receiver side, the data is divided into a block, each of 64 bits. The same key which is used for encryption is used for decryption. It takes the 64-bit ciphertext and, by using the key convert the ciphertext into plain text.

iii. As the same key is used for all blocks' encryption, if the block of plain text is repeated in the original message, then the ciphertext's corresponding block will also repeat. As the same key used for tor all block, to avoid the repetition of block ECB mode is used for an only small message where the repetition of the plain text block is less.

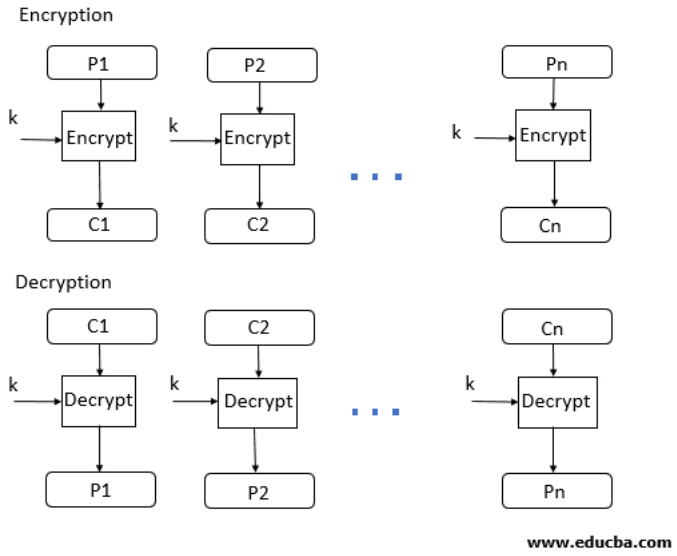


Figure 2: ECB Mode of Operation

(b) *Cipher Block Chaining (CBC)*

i. CBC Mode stands for Cipher block Mode at the sender side; the plain text is divided into blocks. In this mode, IV(Initialization Vector) is used, which can be a random block of text. IV is used to make the ciphertext of each block unique.

ii. The first block of plain text and IV is combined using the XOR operation and then encrypted the resultant message using the key and form the first block of ciphertext. The first block of ciphertext is used as IV for the second block of plain text. The same procedure will be followed for all blocks of plain text.

iii. At the receiver side, the ciphertext is divided into blocks. The first block ciphertext is decrypted using the same key, which is used for encryption. The decrypted result will be XOR with the IV and form the first block of plain text. The second block of ciphertext is also decrypted using the same key, and the result of the decryption will be XOR with the first block of ciphertext and form the second block of plain text. The same procedure is used for all the blocks.

iv. CBC Mode ensures that if the block of plain text is repeated in the original message, it will produce a different ciphertext for corresponding blocks. Note that the key which is used in CBC mode is the same; only the IV is different, which is initialized at a starting point.

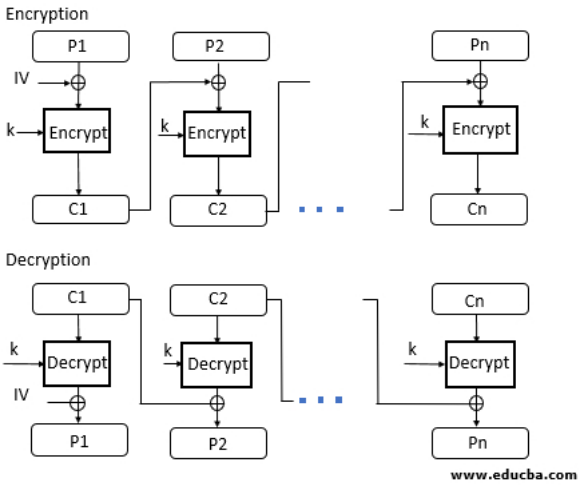


Figure 3: CBC Mode of Operation

(c) *Cipher Feedback (CFB)*

i. CFB mode stands for Cipher Feedback Mode. In this mode, the data is encrypted in the form of units where each unit is of 8 bits.

ii. Like cipher block chaining mode, IV is initialized. The IV is kept in the shift register. It is encrypted using the key and form the ciphertext.

iii. Now the leftmost j bits of the encrypted IV is XOR with the plain text's first j bits. This process will form the first part of the ciphertext, and this ciphertext will be transmitted to the receiver.

iv. Now the bits of IV is shifted left by j bit. Therefore the rightmost j position of the shift register now has unpredictable data. These rightmost j positions are now filed with the ciphertext. The process will be repeated for all plain text units.

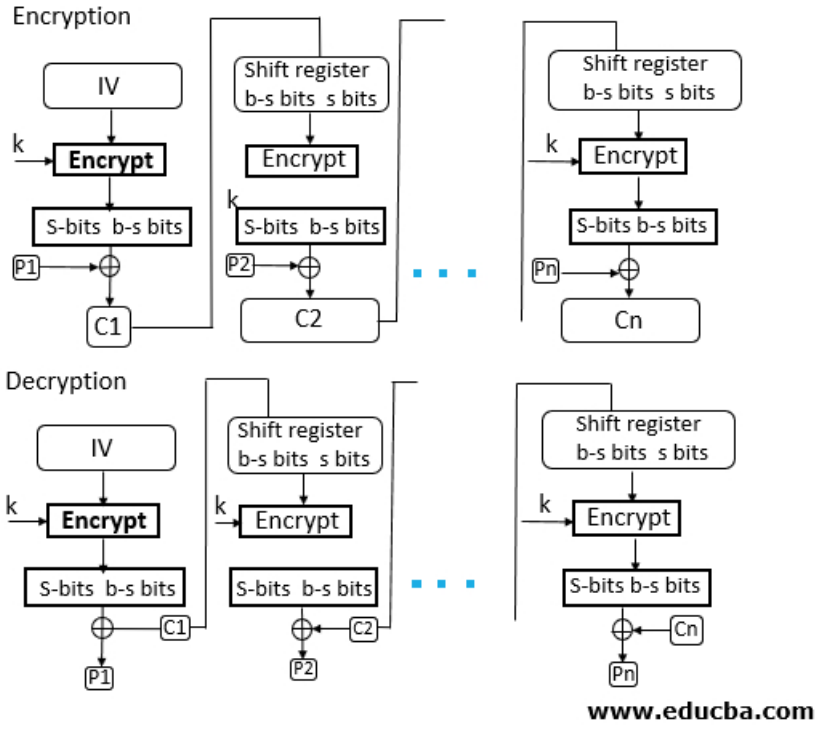


Figure 4: CFB Mode of Operation

(d) *Output Feedback (OFB)*

i. OFB Mode stands for output feedback Mode. OFB mode is similar to CFB mode; the only difference is in CFB, the ciphertext is used for the next stage of the encryption process, whereas in OFB, the output of the IV encryption is used for the next stage of the encryption process.

ii. The IV is encrypted using the key and form encrypted IV. Plain text and leftmost 8 bits of encrypted IV are combined using XOR and produce the ciphertext.

iii. For the next stage, the ciphertext, which is the form in the previous stage, is used as an IV for the next iteration. The same procedure is followed for all blocks.

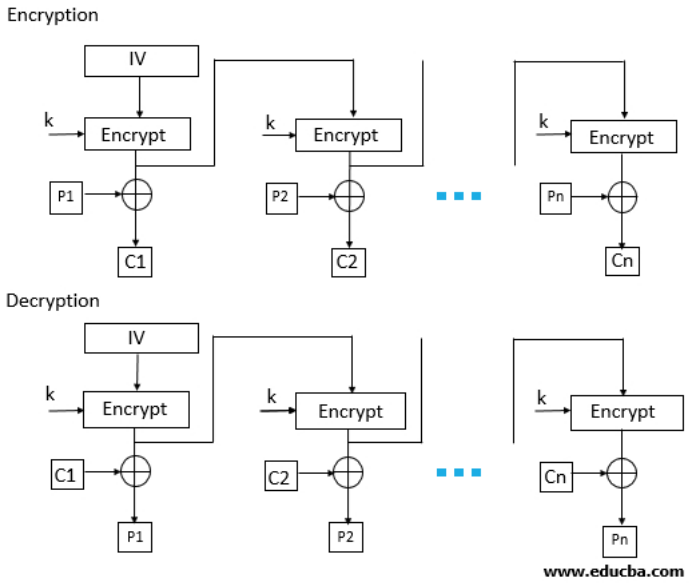


Figure 5: Output Feedback Mode of Operation

(e) *Counter (CTR)*

i. CTR Mode stands for counter mode. As the name is counter, it uses the sequence of numbers as an input for the algorithm. When the block is encrypted, to fill the next register next counter value is used.

ii. For encryption, the first counter is encrypted using a key, and then the plain text is XOR with the encrypted result to form the ciphertext.

iii. The counter will be incremented by 1 for the next stage, and the same procedure will be followed for all blocks. For decryption, the same sequence will be used. Here to convert ciphertext into plain text, each ciphertext is XOR with the encrypted counter. For the next stage, the counter will be incremented by the same will be repeated for all Ciphertext blocks.

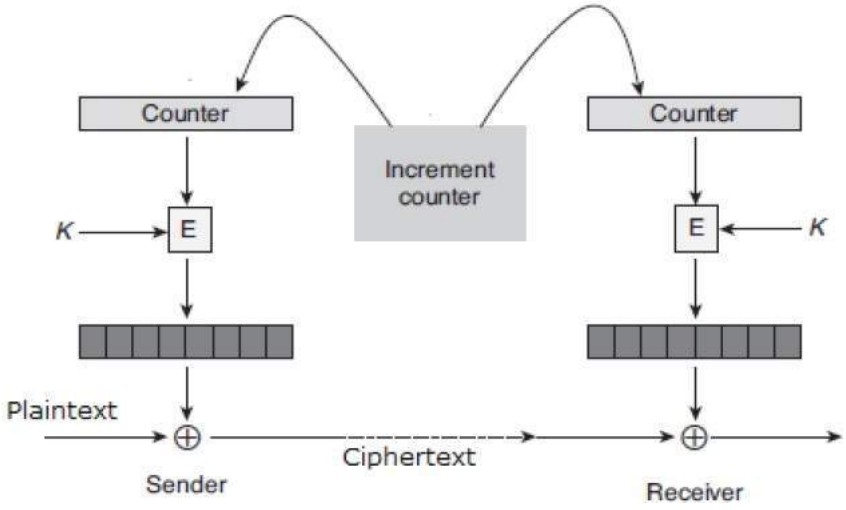


Figure 6:

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Classical Cryptographic Technique Implementations
"Simplified Advaned Encryption Standard"

Lab Assignment 3

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

February 27, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement S-AES symmetric key algorithm.

2 Objectives

To understand the concepts of block cipher and symmetric key cryptographic system.

3 Theory

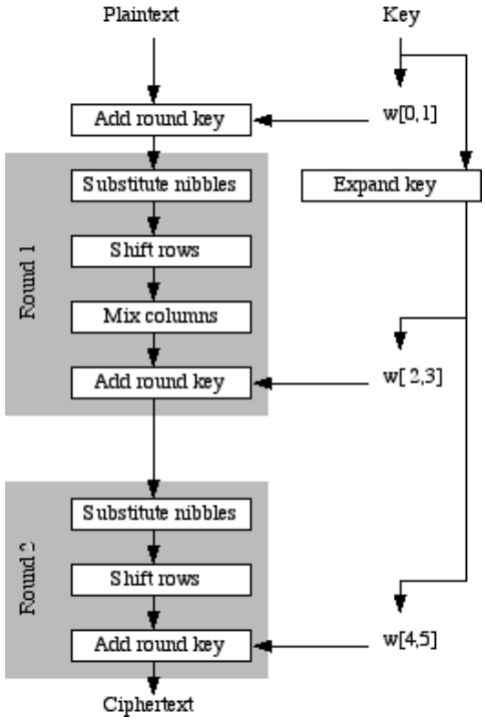
3.1 What is Simplified AES?

S-AES is to AES as S-DES is to DES. In fact, the structure of S-AES is exactly the same as AES. The differences are in the key size (16 bits), the block size (16 bits) and the number of rounds (2 rounds).

The Advanced Encryption Standard (AES) is a widely-used symmetric-key encryption algorithm that is used to encrypt and decrypt data. The simplified AES algorithm is a simplified version of the AES algorithm that is often used as a teaching tool to help people understand how AES works.

The simplified AES algorithm operates on a 4x4 matrix of bytes called a "state." The algorithm consists of several rounds, each of which performs a series of operations on the state. The number of rounds depends on the key size: 10 rounds for a 128-bit key, 12 rounds for a 192-bit key, and 14 rounds for a 256-bit key.

The simplified AES algorithm is a simplified version of the AES algorithm that is often used as a teaching tool to help people understand how AES works.



S-AES Encryption Overview

Figure 1:

3.2 Key Expansion

The key expansion function is the same as AES. The key is expanded to 32 bits and then split into two 16-bit keys. The first key is used in the first round and the second key is used in the second round.

3.3 Constants

3.4 Substitution

In this step, each byte in the state is replaced by another byte using a substitution table called the S-box. The S-box is a fixed table that maps each possible byte value to another byte value. The byte value is used to look up a corresponding value in the S-box. The substitution is done in a byte-wise manner.

3.5 Shift Rows

Here are the steps of one round of the simplified AES algorithm:

1. SubBytes: In this step, each byte in the state is replaced by another byte using a substitution table called the S-box.
2. ShiftRows: In this step, the bytes in each row of the state are shifted to the left. The first row is not shifted, the second row is shifted by one byte to the left, the third row is shifted by two bytes to the left, and the fourth row is shifted by three bytes to the left.
3. MixColumns: In this step, each column of the state is multiplied by a fixed matrix. This is a bit more complex than the other steps, but it essentially "mixes" the bytes in each column.
4. AddRoundKey: In this step, each byte in the state is XORed with a byte from the key schedule. The key schedule is derived from the original key using a key expansion algorithm.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

```
Enter Text to be encrypted via S-AES:AES is much better than DES
Enter 4 digit Key to be used for encryption:9087
Your Cipher Text is:
                      HéWõëd¸½KùÐ´^#EGã’
The decrypted plain text is:  AES is much better than DES
```

6 Code

```
binary_to_decimal = {(0, 0): 0, (0, 1): 1, (1, 0): 2, (1, 1): 3}

s_box = [
    [0x9, 0x4, 0xA, 0xB],
    [0xD, 0x1, 0x8, 0x5],
    [0x6, 0x2, 0x0, 0x3],
    [0xC, 0xE, 0xF, 0x7],
]

inv_s_box = [
    [0xA, 0x5, 0x9, 0xB],
    [0x1, 0x7, 0x8, 0xF],
    [0x6, 0x0, 0x2, 0x3],
    [0xC, 0x4, 0xD, 0xE],
]

R_CON = [
    [1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 1],
]

MIX_COLUMN_TABLE = {
    1: [0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE,
     0xF],
    2: [0x0, 0x2, 0x4, 0x6, 0x8, 0xA, 0xC, 0xE, 0x3, 0x1, 0x7, 0x5, 0xB, 0x9, 0xF,
     0xD],
    4: [0x0, 0x4, 0x8, 0xC, 0x3, 0x7, 0xB, 0xF, 0x6, 0x2, 0xE, 0xA, 0x5, 0x1, 0xD,
     0x9],
    9: [0x0, 0x9, 0x1, 0x8, 0x2, 0xB, 0x3, 0xA, 0x4, 0xD, 0x5, 0xC, 0x6, 0xF, 0x7,
     0xE],
}

MIX_COLUMN_MATRIX = [[1, 4], [4, 1]]
MIX_COLUMN_MATRIX_DECRYPT = [[9, 2], [2, 9]]


def ceaser_cipher(plain_text, key):
    """Function to encrypt plain text using Ceaser Cipher.

    Args:
        plain_text (string): plain text to be encrypted.
        key (int): key to be used for encryption.
    """

    def get_ascii(some_char):
        if some_char.islower():
            return ord(some_char) - 97
        elif some_char.isupper():
            return ord(some_char) - 65
        else:
            return -1

    cipher_letter = ""
    cipher = []
```

```
53
54      for i in plain_text:
55          if i == " " or not i.isalpha():
56              cipher.append(i)
57              continue
58          if i.islower():
59              cipher_letter = chr(((get_ascii(i) + key) % 26) + 97).upper()
60          else:
61              cipher_letter = chr(((get_ascii(i) + key) % 26) + 65).lower()
62
63          cipher.append(cipher_letter)
64      return cipher
65
66
67  def decrypt_ceaser_cipher(cipher_text, ceaser_key):
68      """Function to decrypt cipher text using Ceaser Cipher.
69
70      Args:
71          cipher_text (string): cipher text to be decrypted.
72          ceaser_key (int): key to be used for decryption.
73      """
74
75      def get_ascii(some_char):
76          if some_char.islower():
77              return ord(some_char) - 97
78          elif some_char.isupper():
79              return ord(some_char) - 65
80          else:
81              return -1
82
83      plain_letter = ""
84      plain_text = []
85
86      for i in cipher_text:
87          if i == " " or not i.isalpha():
88              plain_text.append(i)
89              continue
90          if i.islower():
91              plain_letter = chr(((get_ascii(i) - ceaser_key) % 26) + 97).upper()
92          else:
93              plain_letter = chr(((get_ascii(i) - ceaser_key) % 26) + 65).lower()
94
95          plain_text.append(plain_letter)
96      return "".join(plain_text)
97
98
99  def decimal_to_binary(ip_val, reqBits):
100     """Function to convert decimal to binary. Returns a list that has integers 0
        and 1 represented in binary.
101
102     Args:
103         ip_val (_type_): input_value in decimal.
104         reqBits (_type_: required number of bits in the output. 4, 8, etc.
105     """
106
107     def decimalToBinary_rec(ip_val, list):
108         if ip_val >= 1:
109             # recursive function call
110             decimalToBinary_rec(ip_val // 2, list)
```

```
        list.append(ip_val % 2)

    list = []
    decimalToBinary_rec(ip_val, list)
    if len(list) < reqBits:
        while len(list) < reqBits:
            list.insert(0, 0)
    if len(list) > reqBits:
        list.pop(0)
    return list


def nibble_substitution_encrypt(nibble):
    """Performs and returns substitution of nibble using S-Box.

    Args:
        nibble (list of integers 0 and 1): nibble to be substituted.
    """

    s_box_row_num = binary_to_decimal.get((nibble[0], nibble[1]))
    s_box_col_num = binary_to_decimal.get((nibble[2], nibble[3]))

    nibble_after_s_box = s_box[s_box_row_num][s_box_col_num]
    nibble_after_s_box = decimal_to_binary(nibble_after_s_box, 4)

    return nibble_after_s_box


def nibble_substitution_decrypt(nibble):
    """Performs and returns substitution of nibble using S-Box.

    Args:
        nibble (list of integers 0 and 1): nibble to be substituted.
    """

    s_box_row_num = binary_to_decimal.get((nibble[0], nibble[1]))
    s_box_col_num = binary_to_decimal.get((nibble[2], nibble[3]))

    nibble_after_s_box = inv_s_box[s_box_row_num][s_box_col_num]
    nibble_after_s_box = decimal_to_binary(nibble_after_s_box, 4)

    return nibble_after_s_box


def key_expansion_function_g(key_w, round_number):

    # divide into 2 parts. N0, and N1
    n_0 = key_w[:4]
    n_1 = key_w[4:]

    # Perform nibble substitution on N0 and N1
    n_0_after_s_box = nibble_substitution_encrypt(n_0)
    n_1_after_s_box = nibble_substitution_encrypt(n_1)

    # XOR N0 and N1 with RCON
    sub_nib = n_1_after_s_box + n_0_after_s_box

    return [x ^ y for x, y in zip(sub_nib, R_CON[round_number])]

```

```
170
171 def make_keys(key):
172     """
173     key = 16 bits.
174     """
175     key_w0, key_w1, key_w2, key_w3, key_w4, key_w5 = (0, 0, 0, 0, 0, 0)
176
177     # divide the key into 2 parts. key_w0 and key_w1
178     key_w0 = key[:8]
179     key_w1 = key[8:]
180
181     key_w1_after_g = key_expansion_function_g(key_w1, 0)
182
183     key_w2 = [x ^ y for x, y in zip(key_w0, key_w1_after_g)]
184     key_w3 = [x ^ y for x, y in zip(key_w1, key_w2)]
185
186     key_w3_after_g = key_expansion_function_g(key_w3, 1)
187
188     key_w4 = [x ^ y for x, y in zip(key_w2, key_w3_after_g)]
189     key_w5 = [x ^ y for x, y in zip(key_w3, key_w4)]
190
191     return key_w0 + key_w1, key_w2 + key_w3, key_w4 + key_w5
192
193
194 def col_matrix_table_lookup(x, y):
195     """Returns the result of multiplication of x and y in GF(2^8) using
    MIX_COLUMN_TABLE.
196
197     Args:
198         x (int): first number to be multiplied.
199         y (int): second number to be multiplied.
200     """
201     answer = MIX_COLUMN_TABLE.get(y)[x]
202     return decimal_to_binary(int(answer), 4)
203
204
205 def mix_columns(s_matrix, mix_column_matrix):
206     # returns a 16 bit answer.
207     result_matrix = [
208         [[0, 0, 0, 0], [0, 0, 0, 0]],
209         [[0, 0, 0, 0], [0, 0, 0, 0]],
210     ]
211     # clearly, multiplication by another 2d matrix while seemingly easy, doesnt
    work for some reason.
212     # So we will take advantage of the fact that this is a SIMPLIFIED AES cipher,
    and do it manually.
213
214     # multiply 2 dimensional matrices
215
216     # for k in range(len(mix_column_matrix)):
217     #     for i in range(len(mix_column_matrix[0])):
218     #         for j in range(len(mix_column_matrix[0])):
219     #             table_lookup = col_matrix_table_lookup(
220     #                 int("".join([str(i) for i in s_matrix[k][j]]), base=2),
221     #                 mix_column_matrix[i][k],
222     #             )
223     #             result_matrix[i][j] = [
224     #                 x ^ y for x, y in zip(result_matrix[i][j], table_lookup)
225     #             ]
```

```
    # 1st row, 1st column
    # table_lookup(value, mat[0][0]) ^ table_lookup(s[0][1], mat[1][0])
    table_lookup_left = col_matrix_table_lookup(
        int("".join([str(i) for i in s_matrix[0][0]]), base=2),
        mix_column_matrix[0][0],
    )
    table_lookup_right = col_matrix_table_lookup(
        int("".join([str(i) for i in s_matrix[1][0]]), base=2),
        mix_column_matrix[0][1],
    )
    result_matrix[0][0] = [x ^ y for x, y in zip(table_lookup_left,
    table_lookup_right)]

    # 1st row, 1st column
    # table_lookup(value, mat[0][0]) ^ table_lookup(s[0][1], mat[1][0])
    table_lookup_left = col_matrix_table_lookup(
        int("".join([str(i) for i in s_matrix[0][1]]), base=2),
        mix_column_matrix[0][0],
    )
    table_lookup_right = col_matrix_table_lookup(
        int("".join([str(i) for i in s_matrix[1][1]]), base=2),
        mix_column_matrix[0][1],
    )
    result_matrix[0][1] = [x ^ y for x, y in zip(table_lookup_left,
    table_lookup_right)]

    # 1st row, 1st column
    # table_lookup(value, mat[0][0]) ^ table_lookup(s[0][1], mat[1][0])
    table_lookup_left = col_matrix_table_lookup(
        int("".join([str(i) for i in s_matrix[0][0]]), base=2),
        mix_column_matrix[1][0],
    )
    table_lookup_right = col_matrix_table_lookup(
        int("".join([str(i) for i in s_matrix[1][0]]), base=2),
        mix_column_matrix[1][1],
    )
    result_matrix[1][0] = [x ^ y for x, y in zip(table_lookup_left,
    table_lookup_right)]

    # 1st row, 1st column
    # table_lookup(value, mat[0][0]) ^ table_lookup(s[0][1], mat[1][0])
    table_lookup_left = col_matrix_table_lookup(
        int("".join([str(i) for i in s_matrix[0][1]]), base=2),
        mix_column_matrix[1][0],
    )
    table_lookup_right = col_matrix_table_lookup(
        int("".join([str(i) for i in s_matrix[1][1]]), base=2),
        mix_column_matrix[1][1],
    )
    result_matrix[1][1] = [x ^ y for x, y in zip(table_lookup_left,
    table_lookup_right)]

    return (
        result_matrix[0][0]
        + result_matrix[1][0]  # no idea why im shifting this and the next line
        + result_matrix[0][1]
        + result_matrix[1][1]
    )

```

```
281
282 def encrypt_SAES_cipher(plain_text, key):
283
284     key_0, key_1, key_2 = make_keys(key)
285     # round 0 - Only Add round key
286     round_0 = [x ^ y for x, y in zip(plain_text, key_0)]
287
288     # STARTING ROUND 1
289
290     # Making nibbles
291     s_0, s_1, s_2, s_3 = (round_0[:4], round_0[4:8], round_0[8:12], round_0[12:])
292     s_0_after_sub = nibble_substitution_encrypt(s_0)
293     s_1_after_sub = nibble_substitution_encrypt(s_1)
294     s_2_after_sub = nibble_substitution_encrypt(s_2)
295     s_3_after_sub = nibble_substitution_encrypt(s_3)
296
297     # Shifting Rows, exchanging s1 ands s3
298     s_1_after_sub, s_3_after_sub = s_3_after_sub, s_1_after_sub
299
300     # Mixing Columns
301     s_matrix = [[s_0_after_sub, s_2_after_sub], [s_1_after_sub, s_3_after_sub]]
302
303     mix_col_result = mix_columns(s_matrix, MIX_COLUMN_MATRIX)
304     round_1 = [x ^ y for x, y in zip(mix_col_result, key_1)]
305
306     # STARTING ROUND 2
307     s_0, s_1, s_2, s_3 = (round_1[:4], round_1[4:8], round_1[8:12], round_1[12:])
308     s_0_after_sub = nibble_substitution_encrypt(s_0)
309     s_1_after_sub = nibble_substitution_encrypt(s_1)
310     s_2_after_sub = nibble_substitution_encrypt(s_2)
311     s_3_after_sub = nibble_substitution_encrypt(s_3)
312
313     # Shifting Rows, exchanging s1 ands s3
314     s_1_after_sub, s_3_after_sub = s_3_after_sub, s_1_after_sub
315
316     s_box = s_0_after_sub + s_1_after_sub + s_2_after_sub + s_3_after_sub
317
318     round_2 = [x ^ y for x, y in zip(s_box, key_2)]
319
320     return round_2
321
322
323 def decrypt_SAES_cipher(cipher_text, key):
324
325     key_0, key_1, key_2 = make_keys(key)
326     # round 0 - Only Add round key
327     round_0 = [x ^ y for x, y in zip(cipher_text, key_2)]
328
329     # STARTING ROUND 1
330
331     # Inverse nibbles substitution
332     s_0, s_1, s_2, s_3 = (round_0[:4], round_0[4:8], round_0[8:12], round_0[12:])
333     s_0_after_sub = nibble_substitution_decrypt(s_0)
334     s_1_after_sub = nibble_substitution_decrypt(s_1)
335     s_2_after_sub = nibble_substitution_decrypt(s_2)
336     s_3_after_sub = nibble_substitution_decrypt(s_3)
337
338     # Inverse Shifting Rows, exchanging s1 ands s3
339     s_1_after_sub, s_3_after_sub = s_3_after_sub, s_1_after_sub
```

```
340
341     nib_sub = s_0_after_sub + s_1_after_sub + s_2_after_sub + s_3_after_sub
342
343     # Add Round key
344     round_1 = [x ^ y for x, y in zip(nib_sub, key_1)]
345
346     s_0, s_1, s_2, s_3 = (round_1[:4], round_1[4:8], round_1[8:12], round_1[12:])
347
348     # Inverse Mixing Columns
349     s_matrix = [[s_0, s_2], [s_1, s_3]]
350
351     round_1 = mix_columns(s_matrix, MIX_COLUMN_MATRIX_DECRYPT)
352
353     # STARTING ROUND 2
354     # making nibbles
355     s_0, s_1, s_2, s_3 = (round_1[:4], round_1[4:8], round_1[8:12], round_1[12:])
356
357     # Inverse Shifting Rows, exchanging s1 ands s3
358     s_1, s_3 = s_3, s_1
359
360     # Inverse nibbles substitution
361     s_0_after_sub = nibble_substitution_decrypt(s_0)
362     s_1_after_sub = nibble_substitution_decrypt(s_1)
363     s_2_after_sub = nibble_substitution_decrypt(s_2)
364     s_3_after_sub = nibble_substitution_decrypt(s_3)
365
366     s_box = s_0_after_sub + s_1_after_sub + s_2_after_sub + s_3_after_sub
367
368     round_2 = [x ^ y for x, y in zip(s_box, key_0)]
369
370     return round_2
371
372
373 def main():
374
375     plain_text = input("Enter Text to be encrypted via S-AES:")
376     key = input("Enter 4 digit Key to be used for encryption:")
377
378     # Make keys
379     ceaser_key = 0
380     for i in key[:2]:
381         ceaser_key += int(i)
382     key = [decimal_to_binary(int(i), 4) for i in key]
383     key = [j for i in key for j in i]
384
385     ceaser_ciphered_text = ceaser_cipher(plain_text, ceaser_key)
386
387     # make plain_text list of 16 bits
388     plain_text = [decimal_to_binary(ord(i), 8) for i in ceaser_ciphered_text]
389     plain_text = [j for i in plain_text for j in i]
390     plain_texts = [plain_text[i : i + 16] for i in range(0, len(plain_text), 16)]
391     for i in plain_texts:
392         if len(i) < 16:
393             i += [0 for i in range(16 - len(i))]
394
395     ciphers = []
396     for plain_text in plain_texts:
397         cipher_text = encrypt_SAES_cipher(plain_text, key)
398         ciphers.append(cipher_text)
```

```
399
400     final_cipher_text = ""
401
402     # decrypting
403     for cipher in ciphers:
404         cipher = [str(i) for i in cipher]
405         cipher = [
406             "".join(cipher[i : i + 8]) for i in range(0, len(cipher), 8)
407         ]
408         cipher = [chr(int(i, base=2)) for i in cipher if i != "00000000"]
409         cipher = "".join(cipher)
410         final_cipher_text += cipher
411
412     print("Your Cipher Text is: ", final_cipher_text)
413     final_decrypted_text = ""
414
415     # decrypting
416     for cipher in ciphers:
417         plain_text = decrypt_SAES_cipher(cipher, key)
418         plain_text = [str(i) for i in plain_text]
419         plain_text = [
420             "".join(plain_text[i : i + 8]) for i in range(0, len(plain_text), 8)
421         ]
422         plain_text = [chr(int(i, base=2)) for i in plain_text if i != "00000000"]
423         plain_text = "".join(plain_text)
424         final_decrypted_text += decrypt_ceaser_cipher(plain_text, ceaser_key)
425
426     print("The decrypted plain text is: ", final_decrypted_text)
427
428     # plain_text = [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
429
430     # key = [0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1]
431
432     # print("The plain text is: ", plain_text)
433     # print("The key is: ", key)
434
435     # # till here we are good. now we need to encrypt the plain text.
436
437     # cipher_text = encrypt_SAES_cipher(plain_text, key)
438
439     # print("The cipher text is: ", cipher_text)
440
441     # # DECRYPTING
442     # plain_text = decrypt_SAES_cipher(cipher_text, key)
443     # print("The decrypted plain text is: ", plain_text)
444
445
446 main()
```

Listing 1: "Fiestal Cipher"

7 Conclusion

Thus, learnt about the different kinds of ciphers, classical cryptographic techniques, and how to implement some of them in python.

8 FAQ

1. **Differentiate between DES and AES.**
   AES:

   (a) AES stands for advanced encryption standard.

   (b) The key length can be 128 bits, 192 bits, or 256 bits.

   (c) The rounds of operations per key length are as follows: 128 bits: 10 192 bits: 12 256 bits: 14

   (d) AES is based on a substitution and permutation network.

   (e) AES is considered the standard encryption algorithm in the world and is more secure than DES.

   (f) Key Addition, Mix Column, Byte Substitution, and Shift Row.

   (g) AES can encrypt plaintext of 128 bits.

   (h) AES was derived from the Square Cipher.

   (i) AES was designed by Vincent Rijmen and Joan Daemen.

   (j) There are no known attacks for AES.

   DES:

   (a) DES stands for data encryption standard.

   (b) The key length is 56 bits.

   (c) There are 16 identical rounds of operations.

   (d) DES is based on the Feistel network.

   (e) DES is considered to be a weak encryption algorithm; triple DES is a more secure encryption algorithm.

   (f) Substitution, XOR Operation, Permutation, and Expansion.

   (g) DES can encrypt plaintext of 64 bits.

   (h) DES was derived from the Lucifer Cipher.

   (i) DES was designed by IBM.

   (j) Brute force attacks, differential cryptanalysis, and linear cryptanalysis.

2. **What are the different advantages and Limitations of AES?**
   Advantages:

   (a) Following are the benefits or advantages of AES:

   (b) As it is implemented in both hardware and software, it is most robust security protocol.

   (c) It uses higher length key sizes such as 128, 192 and 256 bits for encryption. Hence it makes AES algorithm more robust against hacking.

   (d) It is most common security protocol used for wide various of applications such as wireless communication, financial transactions, e-business, encrypted data storage etc.

   (e) It is one of the most spread commercial and open source solutions used all over the world.

(f) No one can hack your personal information.

(g) For 128 bit, about 2128 attempts are needed to break. This makes it very difficult to hack it as a result it is very safe protocol.

Limitations:

(a) It uses too simple algebraic structure.

(b) Every block is always encrypted in the same way.

(c) Hard to implement with software.

(d) AES in counter mode is complex to implement in software taking both performance and security into considerations.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Public Key Cryptographic Techniques
"Rivest, Shamir, Adleman's Algorithm (RSA)"

Lab Assignment 4

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 14, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement RSA asymmetric key algorithm.

2 Objectives

To understand the concepts of public key and private key

3 Theory

3.1 Euler Totient Function

Euler's Totient function, denoted as $\varphi(n)$, is a number theoretic function that counts the number of positive integers less than or equal to n that are relatively prime to n. In other words, $\varphi(n)$ gives the number of integers in the range $1 \leq k \leq n$ such that $\gcd(k,n) = 1$.

Here is an example of how to calculate Euler's Totient function for a specific integer n:

Let's take $n = 12$. The prime factorization of n is $n = 2^2 \cdot 3^1$, so we can use the formula for calculating $\varphi(n)$ in terms of the prime factorization of n:

$$\varphi(n) = n \cdot \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

where the product is taken over all distinct prime factors of n. Plugging in the prime factorization of $n = 12$, we get:

$$\begin{aligned} \varphi(12) &= 12 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) \\ &= 12 \cdot \frac{1}{2} \cdot \frac{2}{3} = 4 \end{aligned}$$

Therefore, the value of Euler's Totient function for $n = 12$ is $\varphi(12) = 4$. This means that there are 4 positive integers less than or equal to 12 that are relatively prime to 12: 1, 5, 7, and 11.

3.2 Euclidean Algorithm

The Euclidean Algorithm is a method for finding the greatest common divisor (GCD) of two integers. It works by repeatedly finding the remainder when one integer is divided by the other, and then replacing the larger integer with the remainder. This process is repeated until the remainder is zero, at which point the GCD is the last non-zero remainder.

Here is an example of the Euclidean Algorithm applied to finding the GCD of 54 and 24:

$$\begin{aligned} 54 &= 2 \cdot 24 + 6 \\ 24 &= 4 \cdot 6 + 0 \end{aligned}$$

We start by dividing 54 by 24 and finding the remainder, which is 6. We then replace 54 with 24 and 24 with 6, and repeat the process by dividing 24 by 6 and finding the remainder, which is 0. Since the remainder is now zero, the GCD is the last non-zero remainder, which in this case is 6. Therefore, the GCD of 54 and 24 is 6.

3.3 Extended Euclidean Algorithm

The Extended Euclidean Algorithm is an extension of the Euclidean Algorithm that not only finds the GCD of two integers, but also finds two coefficients that can be used to express the GCD as a linear combination of the two integers. Specifically, given two integers a and b, the Extended Euclidean Algorithm finds integers x and y such that:

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

Here is an example of the Extended Euclidean Algorithm applied to finding the GCD of 54 and 24, along with the coefficients x and y:

$$54 = 2 \cdot 24 + 6$$
$$24 = 4 \cdot 6 + 0$$

To start, we apply the Euclidean Algorithm as we did before to find the GCD of 54 and 24, which is 6. We then work backwards through the steps of the algorithm to find the coefficients x and y.

Starting from the second-to-last step:

$$6 = 54 - 2 \cdot 24$$

We can rearrange this equation to isolate 6:

$$\begin{aligned} 6 &= 54 - 2 \cdot 24 \\ &= 54 - 2(54 - 2 \cdot 24) \\ &= 5 \cdot 54 - 2 \cdot 24 \end{aligned}$$

So, we have found that $x = 5$ and $y = -2$. Substituting these values into the original equation, we get:

$$54(5) + 24(-2) = 6$$

Therefore, the GCD of 54 and 24 is 6, and it can be expressed as a linear combination of 54 and 24 with coefficients 5 and -2, respectively.

3.4 RSA Algorithm

3.4.1 Key Generation

1. Selecting two large primes at random: p and q.
2. Computing their system modulus: $n = (p * q)$.
3. Compute: $\phi(n) = (p-1) * (q-1)$.
4. selecting at random the encryption key (public key): e such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$.
5. To find decryption key d such that $d * e \equiv 1 \pmod{\phi(n)}$.
6. Public Encryption key: $PU = e, n$
7. Private Decryption key: $PR = d, n$

3.4.2 RSA Encryption

Encrypt the plain text M using the public key PU.

1. Computes Cipher text: $C = M^e \pmod{n}$, where 0 <= M < n.

3.4.3 RSA Decryption

Decrypt the cipher text C using the private key PR.

1. Computes Plaintext: $M = C^d \pmod{n}$

3.4.4 Example of RSA Encryption

1. Select two large primes at random: $p = 3$ and $q = 11$.
2. Compute their system modulus: $n = (p * q) = 33$.
3. Compute: $\phi(n) = (p-1) * (q-1) = 20$.
4. Select at random the encryption key (public key): $e = 7$ such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$.
5. To find decryption key d such that $d * e \equiv 1 \pmod{\phi(n)}$.
6. Public Encryption key: $PU = e, n = 7, 33$
7. Private Decryption key: $PR = d, n = 3, 33$
8. Encrypt the plain text $M = 30$ using the public key PU.
9. Computes Cipher text: $C = M^e \pmod{n} = 5^7 \pmod{33} = 24$.
10. Decrypt the cipher text $C = 24$ using the private key PR.
11. Computes Plaintext: $M = C^d \pmod{n} = 24^3 \pmod{33} = 30$.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

```
Enter the Message to be encrypted:
This Assignment's Due date is very near
private key is:  (14633, 31373)
public key is:  (89, 31373)
<encrypted text>
This Assignment's Due date is very near
```

6 Code

```
import math
import random

# Pre generated primes
first_primes_list = [
2,
3,
5,
7,
11,
13,
17,
19,
23,
29,
31,
37,
41,
43,
47,
53,
59,
61,
67,
71,
73,
79,
83,
89,
97,
101,
103,
107,
109,
113,
127,
131,
137,
139,
149,
151,
157,
163,
167,
173,
179,
181,
191,
193,
197,
199,
211,
223,
227,
229,
233,
```

```
239,
241,
251,
257,
263,
269,
271,
277,
281,
283,
293,
307,
311,
313,
317,
331,
337,
347,
349,
]

# Iterative Function to calculate
# (a^n)%p in O(logy)


def power(a, n, p):

    # Initialize result
    res = 1

    # Update 'a' if 'a' >= p
    a = a % p

    while n > 0:

        # If n is odd, multiply
        # 'a' with result
        if n % 2:
            res = (res * a) % p
            n = n - 1
        else:
            a = (a**2) % p

            # n must be even now
            n = n // 2

    return res % p


def nBitRandom(n):
    return random.randrange(2 ** (n - 1) + 1, 2**n - 1)


def getLowLevelPrime(n):
    """Generate a prime candidate divisible
    by first primes"""
    while True:
        # Obtain a random number
        pc = nBitRandom(n)
```

```
116
117         # Test divisibility by pre-generated
118         # primes
119         for divisor in first_primes_list:
120             if pc % divisor == 0 and divisor**2 <= pc:
121                 break
122         else:
123             return pc
124
125
126 def isPrime(n, k):
127     """
128     If n is prime, then always returns true, If n is composite than returns false
        with
129     high probability Higher value of k increases probability of correct result
130
131     works on Primality Test by Fermat's Little Theorem:
132     If n is a prime number, then for every a, 1 < a < n-1,
133
134     a^(n-1) = 1 (mod n)
135     OR
136     a^n-1 % n = 1
137
138     """
139     # Corner cases
140     if n == 1 or n == 4:
141         return False
142     elif n == 2 or n == 3:
143         return True
144
145     # Try k times
146     else:
147         for i in range(k):
148
149             # Pick a random number
150             # in [2..n-2]
151             # Above corner cases make
152             # sure that n > 4
153             a = random.randint(2, n - 2)
154
155             # Fermat's little theorem
156             if power(a, n - 1, n) != 1:
157                 return False
158
159     return True
160
161
162 def eucleadean_gcd(x, y):
163     if y == 0:
164         return x
165     if x == 0:
166         return y
167
168     else:
169         return eucleadean_gcd(y, x % y)
170
171
172 def extended_eucleadean(x, y):
173     # y is smaller than x
```

```
    # we need to return g, a, b such that
    # g = gcd(x, y) = ax + by

    if y == 0:
        return (x, 1, 0)
    else:
        g, a, b = extended_eucleadean(y, x % y)
        return (g, b, a - (x // y) * b)


def make_keys(prime_no_bits=1024):
    """
    Return a public and private key pair.
    returns: tuple (private_key, public_key)
    """
    # while True:
    #     p = int(input("Enter the value of p: "))
    #     q = int(input("Enter the value of q: "))
    #     check_p = isPrime(p, 10)
    #     check_q = isPrime(q, 10)
    #     if check_p and check_q:
    #         break
    #     print("Primality: p: ", check_p)
    #     print("Primality: q: ", check_q)

    while True:
        p = getLowLevelPrime(prime_no_bits)
        if isPrime(p, 20):
            break

    while True:
        q = getLowLevelPrime(prime_no_bits)
        if isPrime(q, 20):
            break

    # computing their system mod:
    n = p * q

    # computing phi n
    phi_n = (p - 1) * (q - 1)

    # computing random key e
    list_of_ees = []
    for i in range(2, phi_n):
        if eucleadean_gcd(i, phi_n) == 1:
            if len(list_of_ees) < 50:
                list_of_ees.append(i)
            else:
                break

    e = random.choice(list_of_ees)
    public_key = (e, n)

    # remainder 1 = a * phi_n + b * e
    g, a, b = extended_eucleadean(phi_n, e)

    if b < 0:
        b = b + phi_n
    d = b
```

```
    private_key = (d, n)

    return (private_key, public_key)


def rsa_encryption(plain_text, key):
    """
    Algorithm to encrypt a integer via RSA.
    """
    e, n = key

    cipher_text = pow(plain_text, e) % n
    return cipher_text


def rsa_decryption(cipher_text, key):
    """
    Decrypts the cipher_text using key.
    """
    d, n = key
    plain_text = pow(cipher_text, d) % n
    return plain_text


if __name__ == "__main__":
    # making keys first
    private_key, public_key = make_keys(8)

    messages = []
    cipher_texts = []
    plain_texts = []

    print("Enter the Message to be encrypted: ")
    message = input()
    for i in message:
        messages.append(ord(i))

    # print("The message to be encrypted is: ", messages)

    print("private key is: ", private_key)
    print("public key is: ", public_key)

    # this will be done by some one else who has my public key
    for i in messages:
        cipher_text = rsa_encryption(i, public_key)
        cipher_texts.append(cipher_text)

    # print(cipher_texts)
    cipher_text = "".join([chr(i) for i in cipher_texts])
    print(cipher_text)

    # once I get cipher_text, I would then decrypt it using my private key.

    cipher_texts = [ord(i) for i in cipher_text]
    for i in cipher_texts:
        plain_text = rsa_encryption(i, private_key)
        plain_texts.append(plain_text)

    plain_texts = [chr(i) for i in plain_texts]
```

```
    plain_text = "".join(plain_texts)
    print(plain_text)
```

Listing 1: "RSA Algorithm"

7 Conclusion

Thus, learnt about the different kinds of public key cryptography works. Also, learnt about the RSA algorithm and its implementation in Python in depth. We also tried to implement RSA on a dummy client server model using sockets in python.

8 FAQ

1. *Compare symmetric key cryptography and asymmetric key cryptography*

   (a) *Symmetric Key Cryptography*
   - *Advantages*
     - *Fast*
     - *Easy to implement*
     - *Easy to share the key*
   - *Disadvantages*
     - *Key Distribution is a problem*
     - *Key Management is a problem*

   (b) *Asymmetric Key Cryptography*
   - *Advantages*
     - *Easy to share the key*
     - *Easy to manage the key*
   - *Disadvantages*
     - *Slow*
     - *Difficult to implement*

2. *Write advantages and disadvantages of RSA algorithm.*

   Advantages

   - *RSA is a public key algorithm* : The public key is available to everyone. The private key is kept secret. The public key is used for encryption and the private key is used for decryption.
   - *RSA is a secure algorithm* : The security of RSA is based on the difficulty of factoring large integers. The security of RSA depends on the fact that the factoring of the product of two large prime numbers is difficult. This is to say that it is a very difficult problem to find the two prime factors of a large composite number, and therefore it makes RSA very secure.
   - *RSA is a widely used algorithm* : RSA is used in many applications such as secure email, digital signatures, file encryption, etc.

   Disadvantages

   - *RSA is a slow algorithm* : The RSA algorithm is slow because of the large number of multiplications and modular exponentiations that are required.
   - *RSA is not suitable for bulk data encryption* : RSA is not suitable for bulk data encryption because of its slowness. It is suitable for encrypting small amounts of data.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Message Integrity Checks
(Message Digest Method - 5) MD5 and (Secure Hashing Algorithm) SHA-256

Lab Assignment 5

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 19, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement integrity of message using MD5 or SHA

2 Objectives

To use of hashing algorithm to check message integrity.

3 Theory

3.1 Secure Hashing Algorithm (SHA-256)

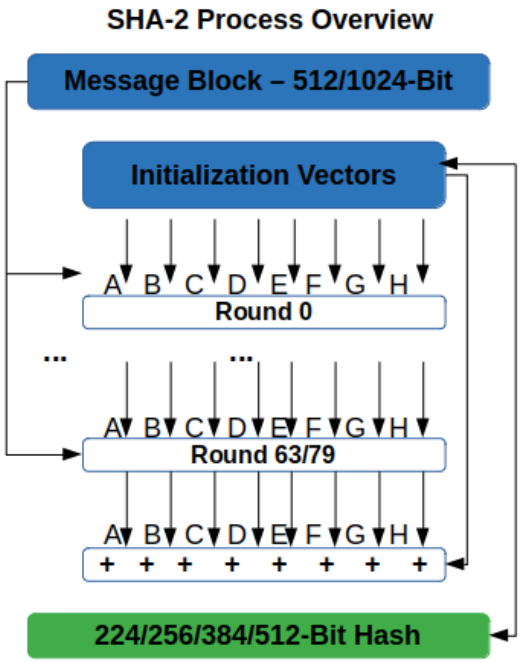


Figure 1: Working of SHA-256

SHA-256 is a widely used cryptographic hash function that generates a 256-bit (32-byte) digest or hash value from an input message.

Here are the steps involved in the SHA-256 algorithm:

1. **Message padding**: The input message is padded so that its length is a multiple of 512 bits, which is the block size used by SHA-256. The padding scheme used by SHA-256 is similar to the one used by SHA-1, but with some modifications.

2. **Initialize hash values**: SHA-256 uses eight initial hash values, represented as 32-bit words. These values are specified in the SHA-256 specification and are typically denoted as H0, H1, H2, H3, H4, H5, H6, and H7.

3. **Process message in 512-bit blocks**: The padded message is divided into 512-bit blocks, and each block is processed by the SHA-256 algorithm.

4. **Initialize working variables**: SHA-256 uses a set of working variables, represented as 32-bit words, to store intermediate values during the hashing process. These working variables are typically denoted as a, b, c, d, e, f, g, and h.

5. **Compute the hash**: For each 512-bit block, SHA-256 performs 64 rounds of computation to update the working variables and generate the hash value. Each round involves several operations, including bitwise operations, logical operations, and modular addition. The specific operations used in SHA-256 are designed to provide strong cryptographic properties, such as collision resistance and preimage resistance.

6. **Combine hash values**: After processing all blocks in the message, the final hash value is computed by combining the eight 32-bit hash values computed during the process. This is typically done by concatenating the hash values in the order H0, H1, H2, H3, H4, H5, H6, and H7 to obtain the 256-bit SHA-256 digest.

3.2 Example

1. here is an example of how SHA-256 can be used to generate a hash value from a sample input message "Hello, world!":

2. **Message Padding**: The ASCII encoding of the message is "48656c6c6f2c20776f726c6421". To pad the message to a length that is a multiple of 512 bits, the message is first appended with a single "1" bit, followed by as many "0" bits as necessary to make the length of the message 448 bits (56 bytes less than a multiple of 512). Then, the original length of the message in bits (80 bits in this case) is added as a 64-bit big-endian integer. Therefore, the padded message is:

   48656c6c6f2c20776f726c6421000000000000000000000000000000000000000000000000
   0000000000000000000005000

3. **Initialize Hash Values**: The initial hash values for SHA-256 are specified in the SHA-256 specification and are typically denoted as H0, H1, H2, H3, H4, H5, H6, and H7:

   H0 = 6a09e667 H1 = bb67ae85 H2 = 3c6ef372 H3 = a54ff53a H4 = 510e527f H5 = 9b05688c H6 = 1f83d9ab H7 = 5be0cd19

4. **Process Message in 512-bit Blocks**: The padded message is divided into 512-bit blocks, and each block is processed by the SHA-256 algorithm. In this case, there is only one block since the message is short enough to fit into a single block.

5. **Initialize Working Variables**: SHA-256 uses a set of working variables, represented as 32-bit words, to store intermediate values during the hashing process. These working variables are typically denoted as a, b, c, d, e, f, g, and h:

   a = H0 b = H1 c = H2 d = H3 e = H4 f = H5 g = H6 h = H7

6. **Compute the Hash**: For each 512-bit block, SHA-256 performs 64 rounds of computation to update the working variables and generate the hash value. Each round involves several operations, including bitwise operations, logical operations, and modular addition. Here is a summary of the operations for the first block:

7. **Divide the block into 16 32-bit words, denoted as W[0] through W[15]**: Compute W[i] for i from 16 to 63 using the formula: sigma1(W[i-2]) + W[i-7] + sigma0(W[i-15]) + W[i-16],

where sigma0 and sigma1 are specific bitwise operations used in SHA-256. Initialize temporary variables T1 and T2 to a and e, respectively. Perform 64 rounds of computation using the formulae: T1 = h + Sigma1(e) + Ch(e, f, g) + K[i] + W[i] T2 = Sigma0(a) + Maj(a, b, c)

8. **Finalize Hash Value**: After processing all of the blocks, the final hash value is computed by concatenating the values of a, b, c, d, e, f, g, and h, in that order, and converting each 32-bit word to a 4-byte big-endian integer. In this example, the final hash value is:

   0x7cf5d5e440b5d760b5a22d9b759f4419f544e1e7525ccae037ba20a8d1e26c35

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

Image Before Changing

Figure 2: Tony Stark Just before Showcasing the Jericho Missile

Image After Changing

Figure 3: Tony Stark Just before Showcasing the Jericho Missile, but with a hidden message

```
Hash of the file is:
b91bf2a1dd825725f788a7e204cfd38d0a4badd1bf3745ec2efceeb4e3cab591
Modified the File, added hidden data to it.
Data: I am Iron Man and I love by daughter 3000.
Hash of the file now is:
d02d021cc9c7771339327216c82540b1796ce1f2d715797907c97ee7c24788e8
```

6 Code

```python
from hashlib import sha256
import os

file_location = os.path.join(os.path.dirname(__file__), "tony.jpg")
# Read image file

print("Hash of the file is: ")
# Convert to byte array
with open(file_location, "rb") as image_file:
    f = image_file.read()
    b = bytearray(f)

# Print the result
print(sha256(b).hexdigest())

```

```
print("Modified the File, added hidden data to it.")
print("Hash of the file now is: ")

# run this command to modify the file

file_location = os.path.join(os.path.dirname(__file__), "tony_changed.jpg")
# Read image file

# Convert to byte array
with open(file_location, "rb") as image_file:
    f = image_file.read()
    b = bytearray(f)

# Print the result
print(sha256(b).hexdigest())
```

Listing 1: "SHA Integrity Check"

7 Conclusion

Thus, learnt about the different types of message integrity checks and how to implement them in Python. MD5 and SHA are the most commonly used message integrity checks, and were implemented in this assignment using the hashlib library.

8 FAQ

1. *What is MAC? What is the difference between MAC and message digest.*

   Definition:

   MAC stands for *Message Authentication Code*, and it is *a type of cryptographic technique used to provide integrity and authenticity of a message.* A MAC is generated by taking a secret key and combining it with the message using a cryptographic hash function or a symmetric encryption algorithm. The resulting MAC is appended to the message and sent along with it. The receiver can then use the same secret key and cryptographic algorithm to compute the MAC and verify that it matches the MAC sent with the message, thus ensuring the integrity and authenticity of the message.

   A **message digest**, on the other hand, is *a fixed-length output generated by applying a cryptographic hash function to a message.* The resulting digest can be used to verify that the message has not been tampered with, but it does not provide authentication. A message digest can be thought of as a digital fingerprint of a message, which is unique to that message and can be used to identify any changes made to the message.

   Differences:

   (a) **Purpose**: The purpose of MAC is to provide both integrity and authenticity of a message, while the purpose of a message digest is to provide integrity only.

   (b) **Technique**: MAC uses a secret key and a cryptographic algorithm such as a hash function or a symmetric encryption algorithm to create a unique code that authenticates the message. In contrast, message digest only uses a cryptographic hash function to generate a fixed-length output that verifies the integrity of the message.

   (c) **Key Requirements**: MAC requires a secret key that is shared between the sender and receiver to generate the code, while message digest only requires a publicly available cryptographic hash function.

   (d) **Strength**: MAC provides stronger authentication than message digest because it uses a secret key to create a unique code that cannot be forged. Message digest, on the other hand, only provides integrity verification and can be more susceptible to attacks.

   (e) **Resource Requirements**: MAC requires more computational resources than message digest because it uses a cryptographic algorithm that requires more processing power. Additionally, MAC requires a secure key distribution channel, which can add to the resource requirements.

   (f) **Usage**: MAC is commonly used in secure communication protocols such as SSL/TLS, IPSec, and SSH, while message digest is commonly used for verifying file integrity, password storage, and digital signatures.

2. *Compare MD5 and SHA-1.*
   MD5 and SHA1 are both cryptographic hash functions that are used to generate a fixed-length output, or digest, from a message. Here are some differences between MD5 and SHA1:

(a) **Output size**: MD5 generates a 128-bit digest, while SHA1 generates a 160-bit digest. This means that SHA1 is slightly stronger than MD5 in terms of the number of possible output values.

(b) **Collision resistance**: MD5 is considered to be weak in terms of collision resistance, which means that it is possible to generate two different messages that produce the same MD5 digest. SHA1 is also vulnerable to collision attacks, but it is considered to be stronger than MD5.

(c) **Speed**: MD5 is generally faster than SHA1 because it uses a simpler algorithm. However, because of its weaknesses, it is no longer recommended for use in security-sensitive applications. SHA1 is slightly slower than MD5 but is still considered to be a fast and efficient hash function.

(d) **Usage**: MD5 was once widely used in security applications such as digital signatures and password storage, but it is now considered to be insecure and should be avoided. SHA1 is still commonly used for data integrity and message authentication, but it is gradually being phased out in favor of stronger hash functions such as SHA-256 and SHA-3.

(e) **Security**: Both MD5 and SHA1 are considered to be weak and vulnerable to attacks by modern computing resources, which can compromise their security in practical applications. As a result, it is recommended to use stronger cryptographic hash functions, such as SHA-256 or SHA-3, for security-sensitive applications.

3. *Compare various versions of SHA.*

There are several versions of the SHA (Secure Hash Algorithm) family of cryptographic hash functions, each with different strengths and output sizes. Here is a comparison of some of the most widely used SHA versions:

(a) **SHA-1**: SHA-1 is a 160-bit hash function that was widely used for data integrity and message authentication. However, it is now considered to be vulnerable to collision attacks and has been deprecated in favor of stronger hash functions.

(b) **SHA-2**: SHA-2 is a family of hash functions that includes SHA-224, SHA-256, SHA-384, and SHA-512, each with different output sizes. SHA-256 is a 256-bit hash function that is widely used for digital signatures, file verification, and other security applications. SHA-384 and SHA-512 are designed for use in applications that require a higher level of security, such as secure communications and financial transactions.

(c) **SHA-3**: SHA-3 is the latest addition to the SHA family of hash functions, and was developed as a result of the NIST hash function competition. It is a family of hash functions that includes SHA3-224, SHA3-256, SHA3-384, and SHA3-512, each with different output sizes. SHA-3 is designed to be more secure and efficient than previous versions of SHA.

(a) **Output size**: SHA-1 generates a 160-bit digest, SHA-2 generates a range of digest sizes from 224 to 512 bits, while SHA-3 also generates a range of digest sizes from 224 to 512 bits. Generally, larger output sizes provide stronger security and are more resistant to collision attacks.

(b) **Collision resistance**: SHA-1 has been found to be vulnerable to collision attacks, while SHA-2 and SHA-3 are designed to be more resistant to such attacks. SHA-3 is particularly strong in this regard due to its sponge construction.

(c) **Algorithm**: SHA-1, SHA-2, and SHA-3 all use different algorithms to generate their digests. SHA-1 and SHA-2 use a Merkle-Damgård construction, while SHA-3 uses a sponge construction. The sponge construction provides better resistance against attacks, such as length extension attacks, that can affect Merkle-Damgård construction.

(d) **Performance**: SHA-2 and SHA-3 generally require more processing power than SHA-1 due to their larger digest sizes and more complex algorithms. However, the performance difference may be insignificant in practice, and the security benefits of using SHA-2 or SHA-3 may outweigh the performance costs.

(e) **Standardization**: SHA-1 and SHA-2 are widely standardized and have been widely used in various security applications, while SHA-3 is a relatively new standard and has yet to gain widespread adoption.

(f) **Security**: SHA-1 is now considered insecure and should not be used in security-sensitive applications. SHA-2 and SHA-3 are considered to be secure and are recommended for use in security-sensitive applications, with SHA-3 being the stronger choice.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Implementation of Diffie - Hellman Key Exchange

Lab Assignment 6

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 19, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement Diffie-Hellman Key Exchange Algorithm

2 Objectives

To learn how to distribute the key.

3 Theory

3.1 What is the Diffie-Hellman Key Exchange Algorithm?

Diffie-Hellman key exchange is a cryptographic protocol that allows two parties to establish a shared secret key over an insecure communication channel without any prior knowledge of each other. The protocol is based on the discrete logarithm problem and modular arithmetic, and it is widely used in secure communication systems such as SSL/TLS, VPNs, and SSH.

3.2 Working of the Algorithm

Here's how the Diffie-Hellman key exchange works, using a simple example:

1. Alice and Bob agree on a large prime number p and a primitive root of p, g. These values are public and can be shared over an insecure channel.
2. Alice chooses a random secret number a and computes $A = g^a$ mod p. She sends A to Bob over the insecure channel.
3. Bob chooses a random secret number b and computes $B = g^b$ mod p. He sends B to Alice over the insecure channel.
4. Alice computes the shared secret key as $K = B^a$ mod p.
5. Bob computes the shared secret key as $K = A^b$ mod p.

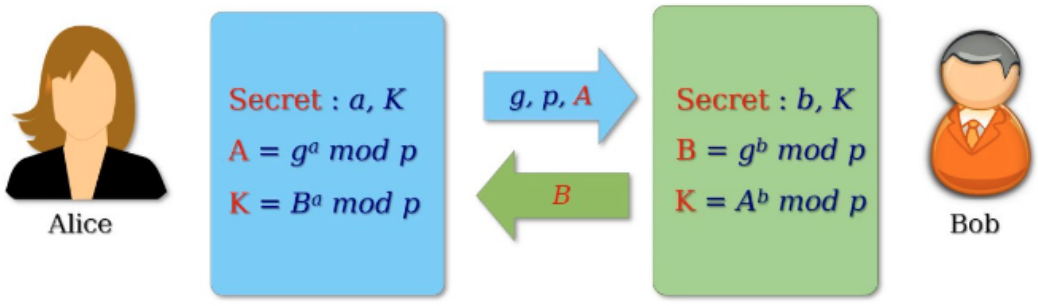


Figure 1: Diffie Hellman Protocol

Both Alice and Bob end up with the same shared secret key K, which can be used for further encryption and decryption of messages.

3.3 Example

Alice and Bob agree on a large prime number $p = 11$ and a primitive root of p, $g = 2$. These values are public and can be shared over an insecure channel.

1. Alice chooses a random secret number $a = 7$ and computes $A = g^a \bmod p = 2^7 \bmod 11 = 7$. She sends $A = 7$ to Bob over the insecure channel.
2. Bob chooses a random secret number $b = 5$ and computes $B = g^b \bmod p = 2^5 \bmod 11 = 10$. He sends $B = 10$ to Alice over the insecure channel.
3. Alice receives $B = 10$ from Bob and computes the shared secret key as $K = B^a \bmod p = 10^7 \bmod 11 = 7$.
4. Bob receives $A = 7$ from Alice and computes the shared secret key as $K = A^b \bmod p = 7^5 \bmod 11 = 10$.

Now Alice and Bob both have the same shared secret key $K = 7 = 10$, which they can use to encrypt and decrypt their messages using a symmetric encryption algorithm such as AES.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

```
p:  239
g:  7
Shared secret key from A 95
Shared secret key from B 95
```

6 Code

```python
# Diffie Hellman algorithm.
from pathlib import Path
import sys, os
ROOT_DIR = str(Path(__file__).parent.parent)
sys.path.insert(1, os.path.join(ROOT_DIR, "Assignment_4"))
from rsa import isPrime, getLowLevelPrime
import random

def diffie_hellman(p, g):
    x_a = random.randint(2, p - 1)
    x_b = random.randint(2, p - 1)
    p_a = user_a(p, g, x_a)
    p_b = user_b(p, g, x_b)
    shared_secret_key = pow(p_a, x_b) % p
    print("Shared secret key from A", shared_secret_key)
```

```
    shared_secret_key = pow(p_b, x_a) % p
    print("Shared secret key from B", shared_secret_key)

    return shared_secret_key

def user_a(p, g, x):
    p_b = pow(g, x) % p
    return p_b

def user_b(p, g, x):
    p_b = pow(g, x) % p
    return p_b

def check_primitive_root(p):
    for i in range(2, p):
        temp_list = []
        for j in range(0, p-2):
            temp_list.append(pow(i, j) % p)
        # print(temp_list)
        if len(set(temp_list)) == len(temp_list):
            return i

# print("Enter the value of p: ")
# p = int(input())

p = getLowLevelPrime(8)
print("p: ", p)
# p = 5
g = check_primitive_root(p)
print("g: ", g)

shared_secret_key = diffie_hellman(p, g)
```

Listing 1: "SHA Integrity Check"

7 Conclusion

Thus, we have successfully implemented the Diffie-Hellman Key Exchange Algorithm. We learnt about the working of the algorithm and how it is used to distribute any key between 2 parties.

8 FAQ

1. **What are other key exchange protocols, other than DH algorithm?**

   (a) **RSA Key Exchange**: This protocol uses the RSA algorithm to exchange keys securely between two parties.

   (b) **Elliptic Curve Diffie-Hellman (ECDH)**: This is a variant of the DH algorithm that uses elliptic curve cryptography to provide stronger security and more efficient key exchange.

   (c) **Kerberos**: This is a network authentication protocol that uses a trusted third party (a Key Distribution Center or KDC) to distribute secret keys between two parties.

   (d) **Secure Remote Password (SRP)**: This is a password-based key exchange protocol that allows two parties to establish a shared secret key without revealing their passwords to each other or to an eavesdropper.

   (e) **Signal Protocol**: This is a modern and widely used protocol for secure messaging that uses a combination of the Double Ratchet Algorithm and the DH algorithm to perform key exchange.

   (f) **Station-to-Station (STS)**: This is a protocol that combines elements of the DH and RSA key exchange protocols to provide stronger security and more efficient key exchange.

2. **Explain the different types of keys.**

   In cryptography, keys refer to the secret values used for encryption and decryption of messages.

   (a) **Symmetric Keys**: Also known as shared keys, these are secret keys that are used for both encryption and decryption of messages. The same key is used by both the sender and the receiver. Examples of symmetric key algorithms include AES, DES, and Blowfish.

   (b) **Public Keys**: Also known as asymmetric keys, these are key pairs consisting of a public key and a private key. The public key is widely distributed and is used for encryption, while the private key is kept secret and is used for decryption. Examples of public key algorithms include RSA, Diffie-Hellman, and elliptic curve cryptography.

   (c) **Session Keys**: These are temporary symmetric keys that are generated for a single session of communication between two parties. They are used to encrypt and decrypt messages exchanged during the session and are discarded once the session is over. Session keys are often used to provide forward secrecy, which means that compromising one session's key does not compromise the security of past or future sessions.

   (d) **Key Exchange Keys**: These are public keys used specifically for exchanging symmetric keys between two parties. Key exchange algorithms like Diffie-Hellman and Elliptic Curve Diffie-Hellman are used to establish a shared secret key between two parties without actually transmitting the key over the communication channel.

3. **Explain different key management issues.**

   Key management is the process of securely generating, storing, distributing, and revoking cryptographic keys. Here are some of the most common key management issues in cryptography:

(a) **Key Generation**: The process of generating strong cryptographic keys is essential to ensuring the security of cryptographic systems. However, generating keys that are both random and unpredictable can be difficult. Key generation must be done securely, and the keys must be protected from disclosure or compromise.

(b) **Key Storage**: Cryptographic keys must be securely stored to prevent unauthorized access or disclosure. The storage of keys is often the weakest link in key management. Keys must be stored in a secure environment, and access to the keys must be tightly controlled.

(c) **Key Distribution**: Cryptographic keys must be securely distributed to all parties involved in the communication. Key distribution can be challenging, especially when there are multiple parties involved. Key exchange protocols like Diffie-Hellman and RSA can be used to securely exchange keys between parties.

(d) **Key Revocation**: Keys must be revoked when they are no longer needed or when they have been compromised. Revocation is necessary to prevent unauthorized access to data that was encrypted using the compromised key. Revocation can be challenging, especially in large systems where there are many keys in use.

(e) **Key Expiration**: Cryptographic keys have a limited lifespan, and must be periodically updated or replaced to maintain their security. Key expiration policies must be carefully designed to balance security and usability.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Implementation of Digital Signatures

Lab Assignment 7

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 29, 2023

Contents

1 Aim

Write a program using JAVA or Python or C++ to implement Digital signature using DSA

2 Objectives

To learn authentication technique for access control

3 Theory

3.1 The Digital Signature Algorithm (DSA)

The Digital Signature Algorithm (DSA) is a public key cryptographic algorithm that is used to create and verify digital signatures. It was developed by the US National Institute of Standards and Technology (NIST) and is based on the mathematical concept of modular exponentiation.

The DSA algorithm consists of three main components: key generation, signature generation, and signature verification.

3.2 Algorithm

3.2.1 Key Generation:

1. Choose a prime number p such that p is 1024 bits or longer, and $p-1$ is divisible by a 160-bit prime number q.
2. Choose an integer g such that $1 < g < p$ and $g^{(p-1)/q} \bmod p \neq 1$.
3. Choose a random integer x such that $0 < x < q$.
4. Compute $y = g^x \bmod p$.
5. The public key is (p, q, g, y), and the private key is x.

3.2.2 Signature Generation:

1. Choose a random integer k such that $0 < k < q$.
2. Compute
$$r = (g^k \bmod p) \bmod q$$
3. Compute
$$s = k^{-1}(H(m) + xr) \bmod q$$
where $H(m)$ is the hash of the message m.
4. The signature is (r, s).

3.2.3 Signature Verification:

1. Verify that $0 < r < q$ and $0 < s < q$.
2. Compute $w = s^{-1} \bmod q$.
3. Compute
$$u_1 = (H(m)w) \bmod q$$
and
$$u_2 = (rw) \bmod q$$
4. Compute
$$v = ((g^{u_1} y^{u_2}) \bmod p) \bmod q$$
5. If $v = r$, the signature is valid. Otherwise, it is invalid.

3.3 Example

Here is an example of how the DSA algorithm works:

Suppose Alice wants to send a message to Bob and sign it using DSA. Alice and Bob have already generated their public and private keys.

3.3.1 Signature Generation:

1. Alice chooses a random integer $k = 123$.
2. Alice computes
$$r = (g^k \bmod p) \bmod q = (2^{123} \bmod 467) \bmod 61 = 8$$
3. Alice computes
$$s = k^{-1}(H(m) + xr) \bmod q = 123^{-1}(H(m) + 22 \cdot 8) \bmod 61$$
where $H(m)$ is the hash of the message m.
4. Alice sends the message and the signature (r, s) to Bob.

3.3.2 Signature Verification:

1. Bob receives the message and the signature (r, s) from Alice.
2. Bob verifies that $0 < r < q$ and $0 < s < q$.
3. Bob computes
$$w = s^{-1} \bmod q = 123^{-1} \bmod 61 = 31$$
4. Bob computes
$$u_1 = (H(m)w) \bmod q$$
and
$$u_2 = (rw) \bmod q$$
where $H(m)$ is the hash of the message m.

5. Bob computes

$$v = ((g^{u_1} y^{u_2}) \bmod p) \bmod q = ((2^{39} \cdot 22^{43}) \bmod 467) \bmod 61 = 8$$

6. Since $v = r$, the signature is valid, and Bob knows that the message was sent by Alice and has not been altered.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

```
The Given Signature is Valid
```

6 Code

```python
import hashlib
import Crypto.PublicKey.RSA as RSA

# Generating keys
key = RSA.generate(2048)
private_key = key.exportKey('PEM')
public_key = key.publickey().exportKey('PEM')


# Message to sign
message = b'This is a message to be signed'

# Hashing the message
hash = hashlib.sha256(message).digest()

# Signing the hash
signature = key.sign(hash, '')

# Verifying the signature
if key.publickey().verify(hash, signature):
    print("Signature is valid")
else:
    print("Signature is not valid")
```

Listing 1: "DSA Signature Validity using PyCrypto Library"

7 Conclusion

Thus, we have seen how to implement digital signatures using DSA algorithm.

8 FAQ

1. **What are various digital signatures algorithms?**

   (a) RSA (Rivest-Shamir-Adleman): The RSA algorithm is one of the most widely used public-key cryptographic algorithms. It is used for both encryption and digital signatures. RSA signatures are computed using the signer's private key and verified using their public key.

   (b) DSA (Digital Signature Algorithm): The DSA algorithm is a public-key cryptographic algorithm used to create and verify digital signatures. It was developed by the US National Institute of Standards and Technology (NIST).

   (c) ECDSA (Elliptic Curve Digital Signature Algorithm): The ECDSA algorithm is a variant of the DSA algorithm that uses elliptic curve cryptography instead of modular arithmetic. It is commonly used in mobile and IoT devices because it requires less computational power than other signature algorithms.

   (d) EdDSA (Edwards-curve Digital Signature Algorithm): The EdDSA algorithm is another variant of the DSA algorithm that uses Edwards curves instead of elliptic curves. It is designed to be faster and more secure than other signature algorithms.

2. **Draw the diagrams of digital signature generation and verification.**

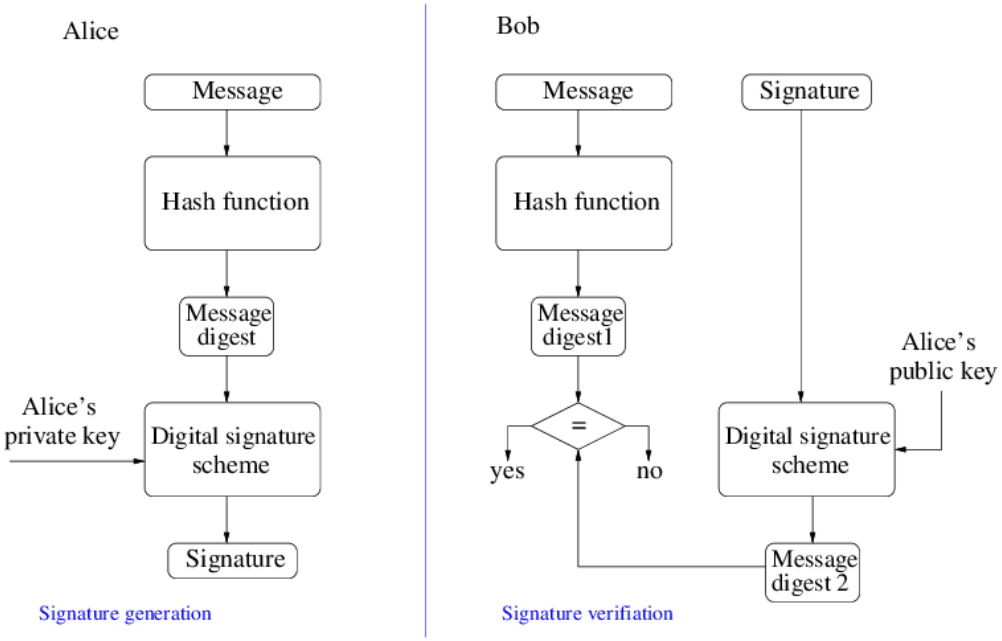


Figure 1: The General Process of Digital Signature Generation and Verification

3. **Which government agencies are involved to issue the digital signature? What is the validity of digital signature**

Government Agencies:

The government agencies involved in issuing digital signatures may vary depending on the country. In general, it is common for a government agency responsible for electronic signatures or digital certificates to issue digital signatures.

For example, in the United States, the *National Institute of Standards and Technology (NIST)* provides guidelines and standards for digital signatures and certificates.

Agencies in India

- In India, digital signatures are issued by *Certifying Authorities (CAs)* that are licensed by the Controller of Certifying Authorities (CCA). The CCA is a government agency under the Ministry of Electronics and Information Technology, responsible for the regulation and licensing of CAs in India.

- The CCA is responsible for implementing the provisions of the Information Technology (IT) Act, 2000, which provides for the legal recognition of electronic records and digital signatures in India. The CCA issues licenses to CAs for issuing digital certificates, and also maintains a national repository of digital certificates that can be used to verify the authenticity of digital signatures.
- In addition to the CCA, the Ministry of Electronics and Information Technology and the Ministry of Law and Justice also play a role in the regulation and promotion of digital signatures and electronic transactions in India.

Validity of Digital Signature:

The validity of a digital signature also varies depending on the country and the laws governing electronic signatures. In general, a digital signature is considered to be legally binding and valid if it meets certain criteria, such as:

(a) The signature is created using a valid digital certificate issued by a trusted certification authority.

(b) The signer's private key is kept secure and not accessible to others.

(c) The signature was created at the time the document was signed and has not been altered since then.

(d) The certificate used to create the signature has not expired or been revoked.

(e) In most countries, digital signatures are considered to be as legally binding as traditional signatures.

(f) The validity of a digital signature can be verified by using the signer's public key to decrypt and verify the signature, and by checking the certificate used to create the signature to ensure that it is valid and has not been revoked.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Email Security using - PGP

Lab Assignment 8

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 30, 2023

Contents

1 Aim

Demonstrate Email Security using - PGP or S/MIME for Confidentiality, Authenticity and Integrity.

2 Objectives

To learn authentication technique for access control

3 Theory

3.1 PGP

1. PGP (Pretty Good Privacy) is a data encryption and decryption computer program that provides cryptographic privacy and authentication for data communication. PGP is used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications.
2. Phil Zimmermann developed PGP in 1991. PGP and similar software follow the OpenPGP standard (RFC 4880) for encrypting and decrypting data.
3. PGP encryption uses a serial combination of hashing, data compression, symmetric-key cryptography, and, finally, public-key cryptography; each step uses one of several supported algorithms. Each public key is bound to a user name and/or an e-mail address. The first version of this system was generally known as a web of trust to contrast with the X.509 system, which uses a hierarchical approach based on certificate authority and which was added to PGP implementations later. Current versions of PGP encryption include both options through an automated key management server.
4. PGP encryption should only be used with data that is transferred via file transfer applications that use secure connections. PGP should not be used with email applications that send and receive data in plain text. PGP encryption is not compatible wi

3.2 Steps to Send an EMail using PGP

3.2.1 Generate a key Pair

The first step is to generate a key pair consisting of a private key and a public key. The private key is kept secret and is used for decrypting messages that are encrypted using the corresponding public key. The public key is shared with others so they can encrypt messages that only the owner of the private key can decrypt.

3.2.2 Share public keys

In order to exchange encrypted messages, each person needs to share their public key with the other person. This can be done by sharing the key through a key server, sending the key as an email attachment, or sharing it in person.

3.2.3 Encrypt the message

Once the public keys have been exchanged, the sender can encrypt the message using the recipient's public key. The encrypted message can only be decrypted by the recipient using their private key.

3.2.4 Sign the message

The sender can optionally sign the message using their private key. This adds a digital signature to the message, which provides a way for the recipient to verify that the message was actually sent by the claimed sender, and that it has not been altered in transit.

3.2.5 Send the message

The encrypted and optionally signed message can now be sent to the recipient through email or another messaging service.

3.2.6 Decrypt the message

Upon receiving the message, the recipient can decrypt it using their private key. If the message was also signed, the recipient can verify the digital signature using the sender's public key.

These steps are shown below in Input and outputs.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code

5 Input and Output

5.1 Generated Keys

```
-----BEGIN PGP PRIVATE KEY BLOCK-----

lQIGBGQ3mhsBBAC00HioVaOXhVbOpo7Gw9/LlM4K9LvmiOdjRaQdJs8kTagJGJkP
yQk2Dpt0oxN3FY4/HeutrKnn/EYDnuTmglXvo7Vbessfbn9h6NQ9ZyCJmln/SAQT
Vz14PtaXymiii+puG7QC0rIYA4acNc2kWiGoC0kyqtljAL0+5K8qxrGKhwARAQAB
/gcDAnqziXIUqkxp+emZF2o8uw7JRWCyLUedfNztqU+Jy8XcQJeSEhqobqM3p8X6
zOlJ9QHm6lr6BduvzulxBnkjuqPyCOUmJz2C7368uq2mNgYwb7w9tfYhI6O8wWVs
LCBa/f0mLVhAqCBhj26HNYkPYkt1PcKgYU/Qx2R0bDhMavNMwWxmE9IAkySQpQjb
mOtcYva53lj3jnq2LmWinvvD88P2uHvd4fXGmomqIzeRK2rADIcY4z8s4o72BLVL
zCGITg4YLChjJttxtUlJSobApla//ckJFNdnuJbsjNColrN+J8XYPMsI/C/aTwxR
X21wJcwf7tVn1Ps9BXY3IGZurrbGbwg9I8V/091hb5od3J5IP3CJrs0i8CTwu7/l
FaKWTDSl3aIpFbqdeClCReJF4fMBbe1WwECdAgYEZDeaGwEEALeH73HJ8lyR6pN1
I7pT+FNgutYL49S5OXPyPh4jcQ3EqV4Sm96Xy6YBpFEsHvokWHSBrmiMe3OEGKdY
go8g0tKBMG1sByiPSmE/ktLChWae2EV/taWkynhNwisrAonLrYCVo/FRR/PIBaT8
qWC8P6w4XIRUpKIi2QEhd+1sIGvbYUKUbkeadnDunDgyg/7gb/H2OTXaL6XvMyzT
QjWlWBy+GMuL1fSCB3Kh1cRVHVsjdjj1V2c1zx0lilToO8PNV2X6NvDNI/6yfuRX
NeNUq//u3WJd1Ayq/2SE91b0M0LLKg5/42ZUoQetc6VEIcjL7c2/XQ9rH6vgy/BP
6A378Y1ZZo3Z0zvCyVh5NMhgKYi2BBgBCAAgFiEEseSJ53V8cfAaI9i3JiEHouMD
/UIFAmQ3mhsCGwwACgkQJiEHouMD/UImEgP8D/hMV0JBlSWERGuGqthbsslrWtKB
uFnGobSJuHCjRtUlzsRLcKixulMXyJiuTMtqpjEl1i2BqhpZqz6IDojFn8Q0KsWw
g9Hw/cFGwTPNPgujchRWZFkOqu+BBzzgl/bS1+EpbkAD7oxkJZxZI7yjQGNTrJJ5
3rJ6wrgXJLQyuJs=
=hg+7
-----END PGP PRIVATE KEY BLOCK-----
```

Listing 1: "Krish Private Key"

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mI0EZDeoGgEEANBxkECodUnki0ouPXtisjdTOb6Z6ASGjaaYltwlNp1PjuIE1E1E
/U+FXB1yPnzQXYJNzZinyKALznNHPA5u1q3kfkxHx0Bsz2Jr79Ly8VcFOXi621c0
Lw50mFDBYCotKHqrMQa4V4ovoG1toJn2RDEbYYo5zpj/cVIS4R9M+3qFABEBAAG0
NktyaXNobmFyYWogKGljcyBhc3NpZ25tZW50KSA8a3B0LmtyaXNobmFyYWpAZ21h
aWwuY29tPojOBBMBCAA4FiEE00HVO8avaij3dBuGrN2zFzRsyHgFAmQ3qBoCGwMF
CwkIBwIGFQoJCAsCBBYCAwECHgECF4AACgkQrN2zFzRsyHgE0AQAg+kQmV7DKOX1
gOzyW0oDcbyLhnTodytDT2RZnNi96d+cFjRfXDB3ET26gBKVzn7b8QG5JO7jSZw2
6noliMFlbM7JFJDP881Dr5SeSqwQZnB2Mozc0gqrlFUYmlShppoJ0wPY5y2ME8U8
g2u1rx3EUsu4GxyQUguyO7S+u4DDLO24jQRkN6gaAQQArO87+003qyBMi88xSx9u
ktouHOso25kAR3tl3tjB2hCstRZoQgGJCbKnt6k0fhRmEqYFwgUKULYvODs+GnWM
ah/E2shSlcSPpOdAjovHTSncOUJmU6qZDLUk78j7NMiFwf3M1vLU2KkBg6Jqw5QT
rpyXDIYMV2RQ3cO/nS7bMd8AEQEAAYi2BBgBCAAgFiEE00HVO8avaij3dBuGrN2z
FzRsyHgFAmQ3qBoCGwwACgkQrN2zFzRsyHgE4AP/WNEAzJuQLh/eyPNW/CzOwkQK
REeu1xj0pFGdXfPytoe1GT+xZ6oDd0WHPzMb3oa7NDC4DbzbRw7AfLJj2Qo4PdYG
vmEHmlON7NY4Wv5W73bxiR8HJGThL6SS4TJXF/etW/Jxs/LYaEnifa77quKeX0Sh
hI+UmhljipkecYXakA4=
=Qz5/
-----END PGP PUBLIC KEY BLOCK-----
```

Listing 2: "Krish Public Key"

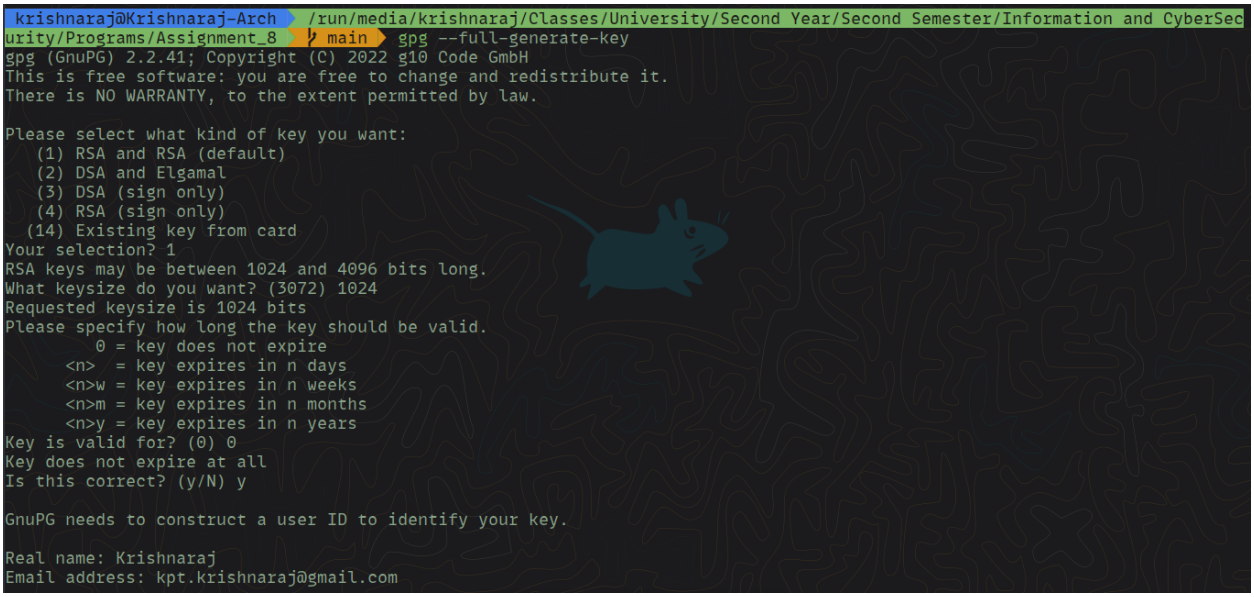


Figure 1: Generating Key Pairs

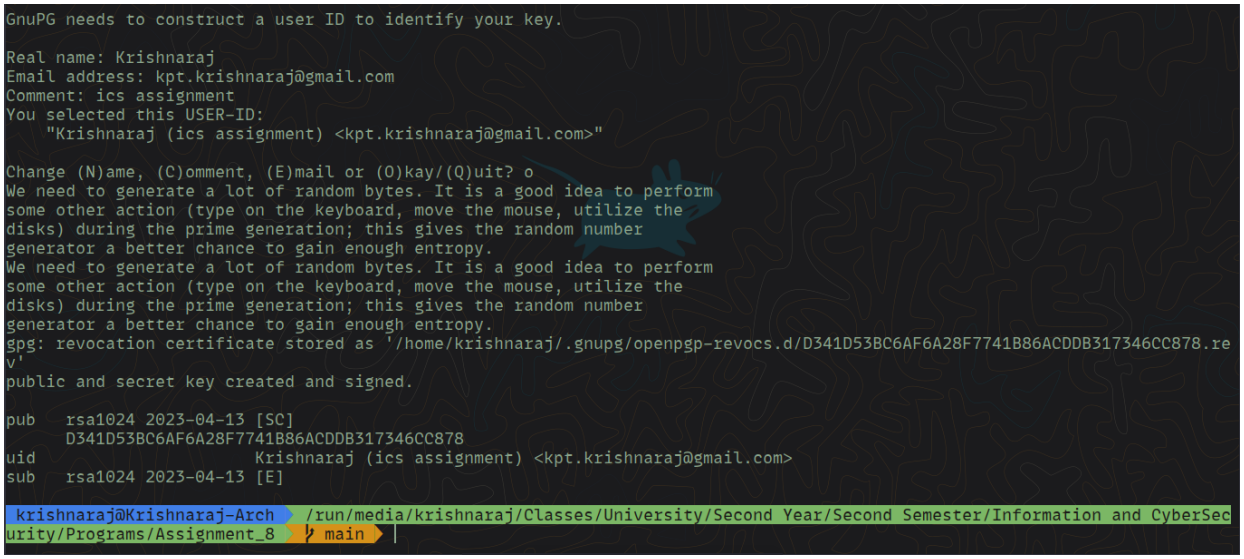


Figure 2: Generating Key Pairs continued

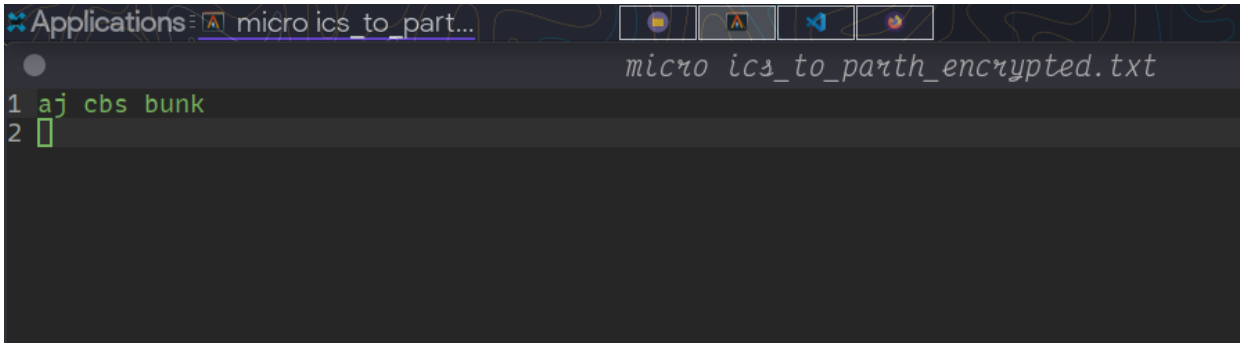


Figure 3: Secret Message to Parth - Recipient - "Aj CBS Bunk"

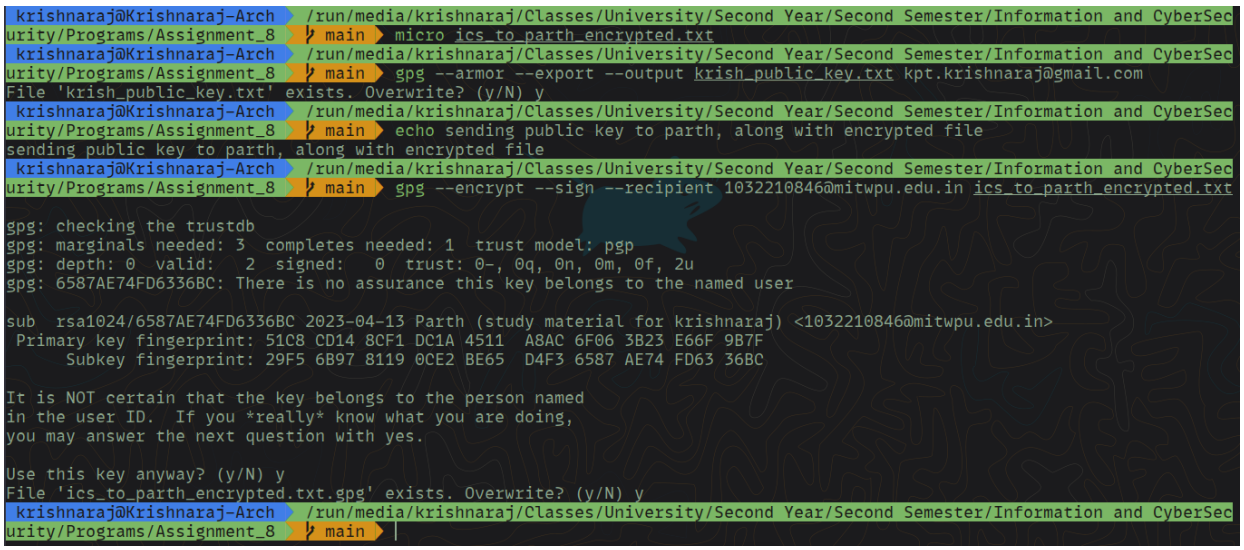


Figure 4: Signing Message to Send Parth using Parth's Public Key

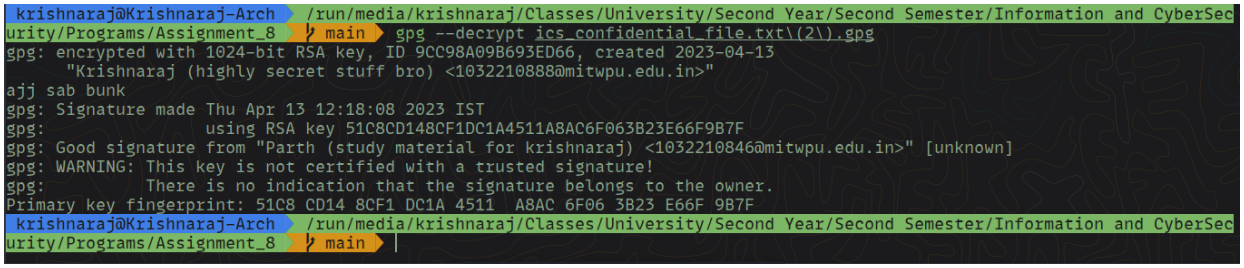


Figure 5: Decrypting Message from Parth using his Public key - "Ajj Sab Bunk"

6 Conclusion

Thus, we have successfully implemented Email Security using - PGP or S/MIME for Confidentiality, Authenticity and Integrity.

7 FAQ

1. **How email security is provided through PGP?**

   (a) PGP (Pretty Good Privacy) provides email security through a combination of encryption, digital signatures, and compression. When a user sends an email using PGP, the message is encrypted using a symmetric key algorithm.

   (b) The symmetric key is then encrypted using the recipient's public key, which is obtained from a key server or a public key directory. The encrypted message and the encrypted symmetric key are then sent to the recipient, who can decrypt the message using their private key.

   (c) PGP also allows users to sign their emails digitally using their private key. The digital signature provides a way for the recipient to verify that the email was actually sent by the claimed sender, and that it has not been altered in transit.

   (d) In addition, PGP can compress the message before encryption, which can reduce the size of the message and make it easier to send over a slow or unreliable connection.

2. **What type of encryption is PGP?**

   PGP uses a combination of symmetric and asymmetric encryption. The symmetric encryption algorithm is used to encrypt the message itself, while the asymmetric encryption algorithm is used to encrypt the symmetric key.

   The symmetric encryption algorithm used in PGP is typically AES (Advanced Encryption Standard), which is a widely used and highly secure algorithm. The asymmetric encryption algorithm used in PGP is typically RSA (Rivest–Shamir–Adleman), which is also widely used and highly secure.

3. **What is the key size allowed in PGP**

   PGP supports a wide range of key sizes, from 512 bits to 4096 bits. The key size determines the level of security provided by the encryption algorithm.

   In general, larger key sizes provide stronger security, but they also require more processing power to encrypt and decrypt the data. For most purposes, a key size of 2048 bits is considered to be sufficient, but some applications may require larger key sizes for enhanced security.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Secured web applications

Lab Assignment 9

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

May 6, 2023

Contents

1 Aim

Demonstration of secured web applications system using SSL certificates and its deployment in Apache tomcat server

2 Objectives

To learn different vulnerability scanning.

3 Theory

3.1 SSL Certificate

An SSL (Secure Sockets Layer) certificate is a digital certificate that verifies the authenticity of a website and encrypts data transmitted between the website and the user's web browser. SSL certificates ensure that all sensitive information, such as usernames, passwords, and credit card numbers, are transmitted securely over the internet.

3.2 How does an SSL certificate work?

When a user visits a website that has an SSL certificate, the website's server sends the user's web browser a copy of the certificate. The browser then verifies the certificate with the Certificate Authority (CA) that issued it. If the certificate is valid, the browser establishes a secure connection with the website using SSL/TLS encryption.

3.3 Types of SSL certificates

There are different types of SSL certificates based on the number of domains or subdomains they cover, the level of validation, and the warranty offered by the Certificate Authority (CA). Some common types of SSL certificates include:

1. Domain Validated (DV) SSL Certificate: This type of SSL certificate only validates the domain name of the website, ensuring that it belongs to the entity requesting the certificate. DV SSL certificates are the most common type of SSL certificate and are suitable for personal websites or blogs.

2. Organization Validated (OV) SSL Certificate: This type of SSL certificate validates both the domain name and the identity of the organization or business requesting the certificate. OV SSL certificates are suitable for e-commerce websites and other online businesses.

3. Extended Validation (EV) SSL Certificate: This type of SSL certificate provides the highest level of validation and requires extensive documentation to prove the identity of the organization requesting the certificate. EV SSL certificates are suitable for large businesses and financial institutions.

4. Wildcard SSL Certificate: This type of SSL certificate covers multiple subdomains of a single domain name. For example, a wildcard SSL certificate for the domain example.com would cover subdomains such as blog.example.com and shop.example.com.

3.4 Benefits of SSL certificates

There are several benefits of using an SSL certificate for a website, including:

1. Data encryption: SSL certificates encrypt all data transmitted between the website and the user's web browser, ensuring that sensitive information is secure.
2. Authentication: SSL certificates provide authentication, verifying that the website is legitimate and belongs to the entity requesting the certificate.
3. Trust and credibility: SSL certificates display trust indicators, such as the padlock icon in the web browser's address bar, which can increase a website's credibility and reputation.
4. SEO benefits: SSL certificates can improve a website's search engine ranking, as search engines prefer secure websites.
5.

3.5 Example

Suppose you want to create an e-commerce website where users can purchase products and enter their personal information, such as name, address, and credit card details. To ensure that the website is secure and trustworthy, you decide to obtain an SSL certificate.

You choose to purchase an OV SSL certificate, which will validate your domain name and your business's identity. You submit your company's legal documents and undergo a validation process to prove your identity.

Once the CA verifies your documents and identity, they issue the SSL certificate, which you install on your website's server. Now, when users visit your website, their web browsers will establish a secure connection using SSL/TLS encryption, ensuring that their personal information is protected.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code

5 Input and Output

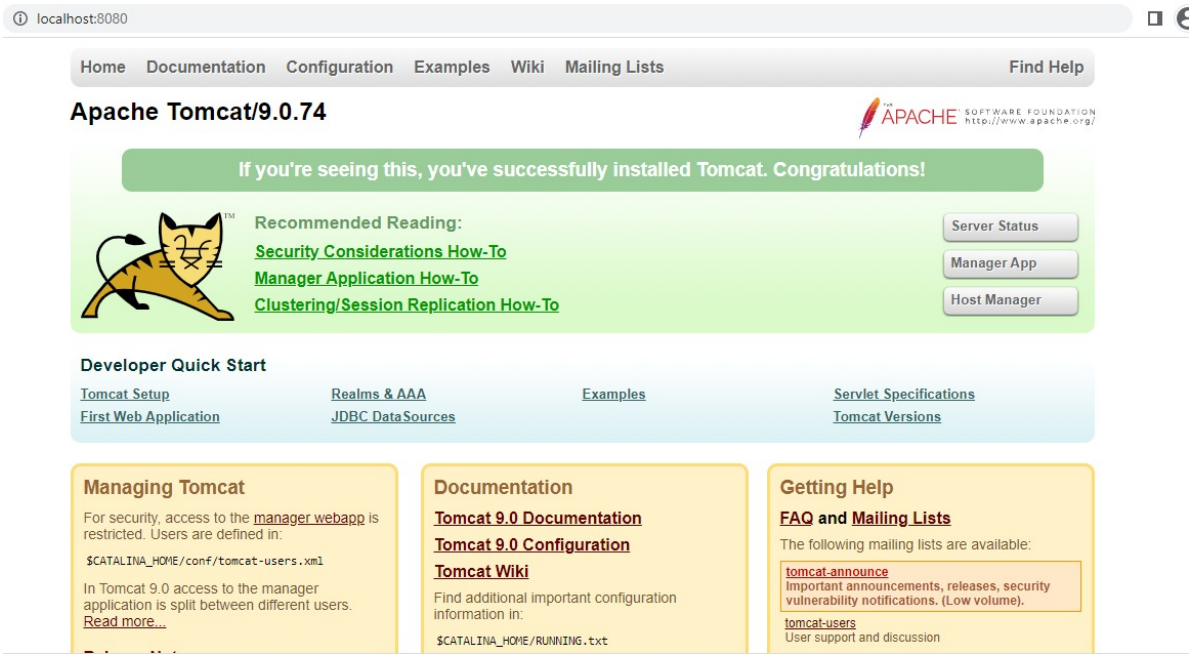


Figure 1:

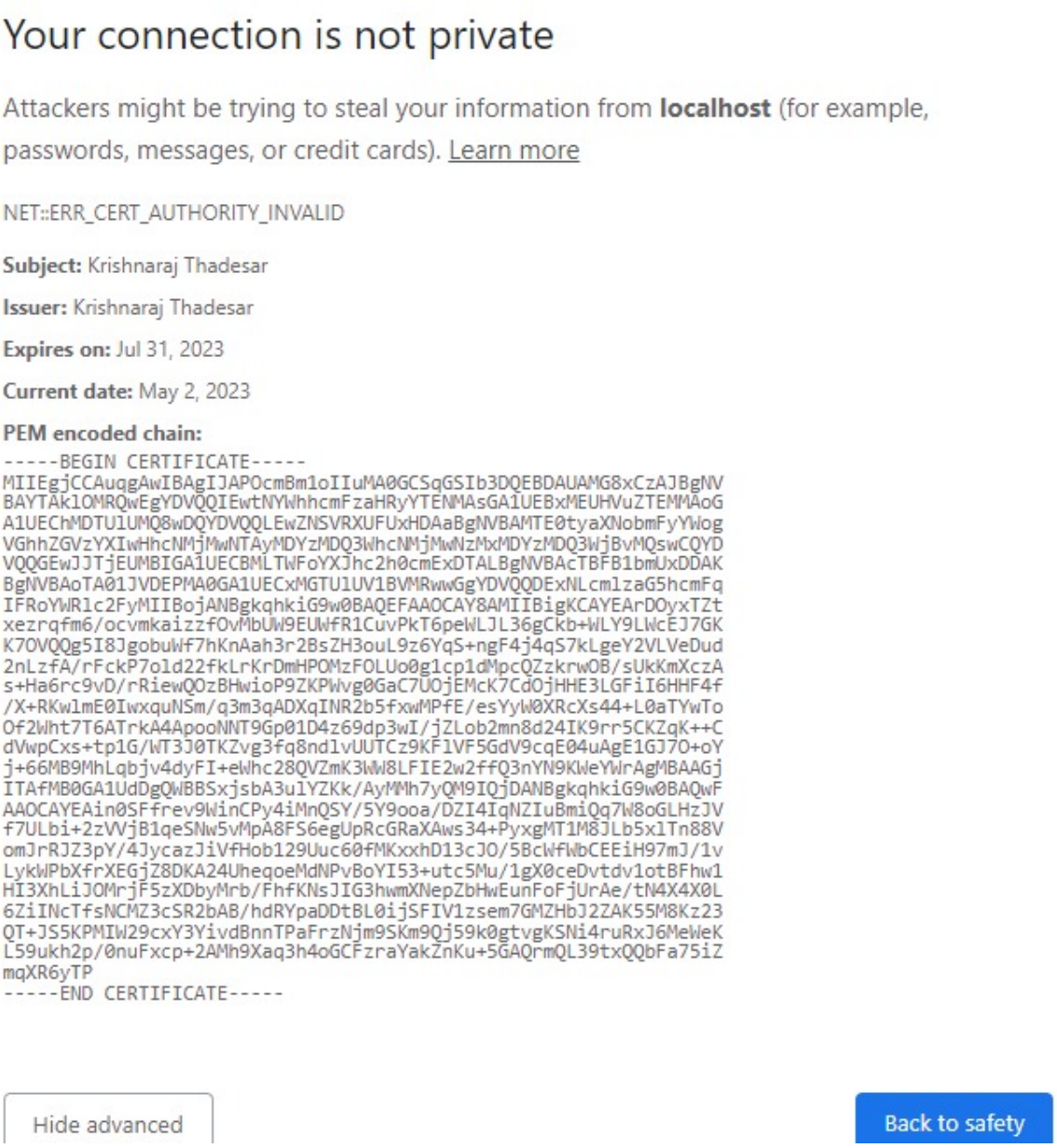

Your connection is not private
Attackers might be trying to steal your information from **localhost** (for example, passwords, messages, or credit cards). Learn more
NET::ERR_CERT_AUTHORITY_INVALID
Subject: Krishnaraj Thadesar
Issuer: Krishnaraj Thadesar
Expires on: Jul 31, 2023
Current date: May 2, 2023
PEM encoded chain:
-----BEGIN CERTIFICATE-----
MIIEgjCCAuqgAwIBAgIJAPOcmBm1oIIuMA0GCSqGSIb3DQEBDAUAMG8xCzAJBgNV
BAYTAklOMRQwEgYDVQQIEwtNYWhhcmFzaHRyYTENMAsGA1UEBxMEUHVuZTEMMAoG
A1UEChMDTUlUMQ8wDQYDVQQLEwZNSVRXUFUxHDAaBgNVBAMTE0tyaXNobmFyYWog
VGhhZGVzYXIwHhcNMjMwNTAyMDYzMDQ3WhcNMjMwNzMxMDYzMDQ3WjBvMQswCQYD
VQQGEwJJTjEUMBIGA1UECBMLTWFoYXJhc2h0cmExDTALBgNVBAcTBFB1bmUxDDAK
BgNVBAoTA01JVDEPMA0GA1UECxMGTUlUV1BVMRwwGgYDVQQDExNLcmlzaG5hcmFq
IFRoYWR1c2FyMIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEArDOyxTZt
xezrqfm6/ocvmkaizzfOvMbUW9EUWfR1CuvPkT6peWLJL36gCkb+WLY9LWcEJ7GK
K7OVQQg5I8JgobuWf7hKnAah3r2BsZH3ouL9z6YqS+ngF4j4qS7kLgeY2VLVeDud
2nLzfA/rFckP7old22fkLrKrDmHPOMzFOLUo0g1cp1dMpcQZzkrwOB/sUkKmXczA
s+Ha6rc9vD/rRiewQOzBHwioP9ZKPWvg0GaC7UOjEMcK7CdOjHHE3LGFiI6HHF4f
/X+RKwlmE0IwxquNSm/q3m3qADXqINR2b5fxwMPfE/esYyW0XRcXs44+L0aTYwTo
Of2Wht7T6ATrkA4ApooNNT9Gp01D4z69dp3wI/jZLob2mn8d24IK9rr5CKZqK++C
dVwpCxs+tp1G/WT3J0TKZvg3fq8nd1vUUTCz9KF1VF5GdV9cqE04uAgE1GJ7O+oY
j+66MB9MhLqbjv4dyFI+eWhc28QVZmK3WW8LFIE2w2ffQ3nYN9KWeYWrAgMBAAGj
ITAfMB0GA1UdDgQWBBSxjsbA3ulYZKk/AyMMh7yQM9IQjDANBgkqhkiG9w0BAQwF
AAOCAYEAin0SFfrev9WinCPy4iMnQSY/5Y9ooa/DZI4IqNZIuBmiQq7W8oGLHzJV
f7ULbi+2zVVjB1qeSNw5vMpA8FS6egUpRcGRaXAws34+PyxgMT1M8JLb5xlTn88V
omJrRJZ3pY/4JycazJiVfHob129Uuc60fMKxxhD13cJO/5BcWfWbCEEiH97mJ/1v
LykWPbXfrXEGjZ8DKA24UheqoeMdNPvBoYI53+utc5Mu/1gX0ceDvtdv1otBFhw1
HI3XhLiJOMrjF5zXDbyMrb/FhfKNsJIG3hwmXNepZbHwEunFoFjUrAe/tN4X4X0L
6ZiINcTfsNCMZ3cSR2bAB/hdRYpaDDtBL0ijSFIV1zsem7GMZHbJ2ZAK55M8Kz23
QT+JS5KPMIW29cxY3YivdBnnTPaFrzNjm9SKm9Qj59k0gtvgKSNi4ruRxJ6MeWeK
L59ukh2p/0nuFxcp+2AMh9Xaq3h4oGCFzraYakZnKu+5GAQrmQL39txQQbFa75iZ
mqXR6yTP
-----END CERTIFICATE-----
Hide advanced
Back to safety

Figure 2:

Figure 3:

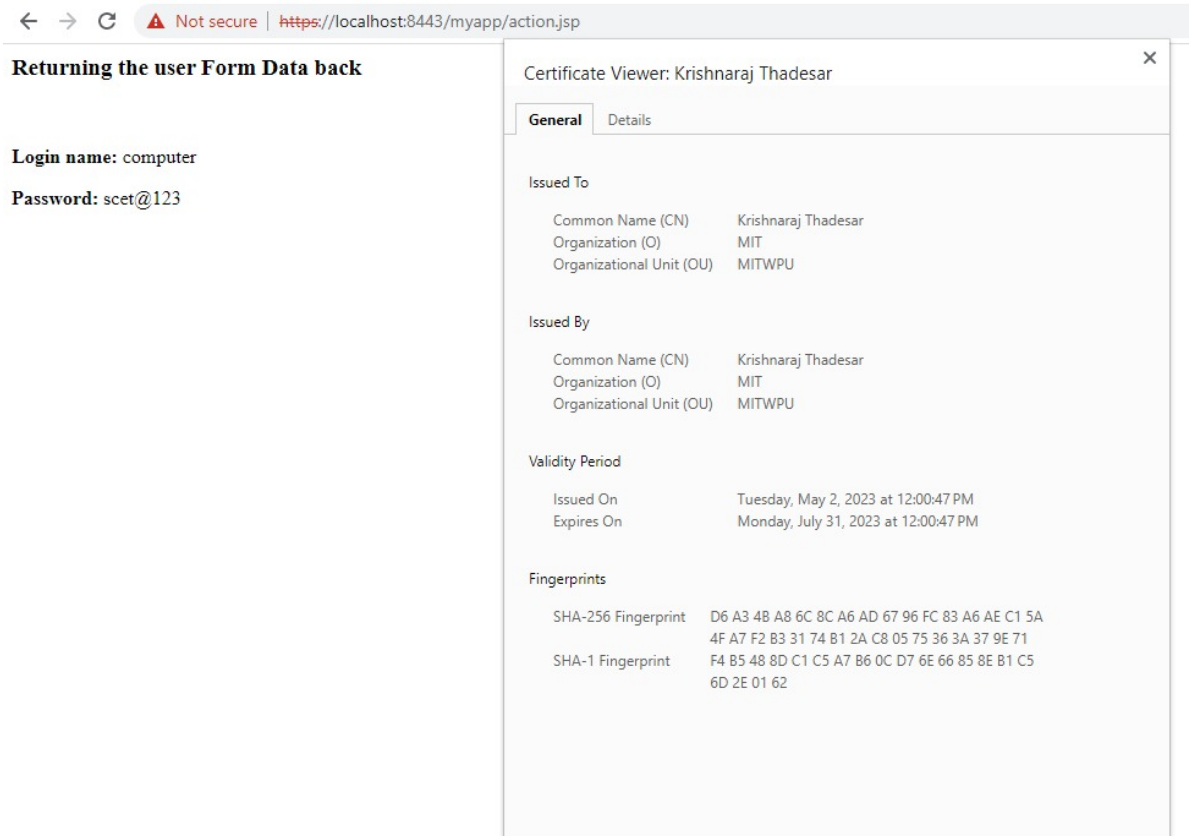


Figure 4:

```
The Given Signature is Valid
```

6 Conclusion

Thus, we have successfully implemented

7 FAQ

1. **What type of encryption does SSL use?**

   Secure Socket Layer (SSL) uses a combination of symmetric and asymmetric encryption to secure data transmitted over a network. SSL uses asymmetric encryption during the initial handshake process, where the client and server exchange public keys to establish a secure communication channel. Once the secure channel is established, SSL uses symmetric encryption to encrypt the data being transmitted between the client and server.

   Symmetric encryption uses a single secret key to encrypt and decrypt data, while asymmetric encryption uses a pair of keys, one private and one public. The public key can be freely shared, while the private key is kept secret. In SSL, the public key is used to encrypt the data, and the private key is used to decrypt the data.

   SSL supports a variety of symmetric encryption algorithms, including AES, DES, and 3DES. It also supports a variety of asymmetric encryption algorithms, including RSA, DSA, and ECDSA.

2. **How safe is SSL?**

   SSL is generally considered to be a secure protocol for transmitting sensitive data over a network. SSL has undergone multiple revisions over the years, with the latest version being Transport Layer Security (TLS). TLS version 1.3 is the latest and most secure version of SSL/TLS, which has been designed to provide strong encryption and better security features.

   However, SSL can be vulnerable to various types of attacks, such as Man-in-the-Middle (MITM) attacks, where an attacker intercepts the communication between the client and server and eavesdrops on the conversation or alters the data being transmitted. SSL is also vulnerable to attacks that exploit weaknesses in the encryption algorithms or the SSL protocol itself.

   To mitigate these risks, SSL implementations must be properly configured and maintained to ensure they are up-to-date with the latest security patches and best practices. It is also recommended to use SSL/TLS certificates from trusted Certificate Authorities (CA) and to use strong passwords and multi-factor authentication to protect the private keys used for encryption.

3. **What are the benefits of SSL?**

   The benefits of SSL include:

   (a) Data encryption: SSL encrypts the data transmitted between the client and server, which helps to protect the data from unauthorized access and eavesdropping.

   (b) Authentication: SSL uses digital certificates to authenticate the identity of the server and ensure that the client is communicating with the intended server.

   (c) Trust: SSL/TLS certificates are issued by trusted Certificate Authorities (CA) that verify the identity of the server and ensure that the SSL/TLS certificate is valid.

(d) Compliance: SSL is required by many industry regulations and compliance standards, such as the Payment Card Industry Data Security Standard (PCI DSS).

(e) Improved search engine ranking: Google has confirmed that HTTPS is a ranking factor, and using SSL/TLS can help to improve search engine rankings.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Intrusion Detection Systems

Lab Assignment 10

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

May 6, 2023

Contents

1 Aim

Configuration and demonstration of Intrusion Detection System using Snort

2 Objectives

To learn authentication techniques for Access Control

3 Theory

Host-based IDS (HIDS) and Network-based IDS (NIDS) are two types of intrusion detection systems that are used to monitor and detect potential security threats.

3.1 HIDS - Host-based IDS

A Host-based IDS (HIDS) is an IDS system that is installed on individual hosts or endpoints to monitor the activity on that host. It uses system logs, file integrity checks, and other methods to identify potential security threats. The HIDS can detect attacks that are not visible on the network, such as local privilege escalation, malware infections, and unauthorized access attempts.

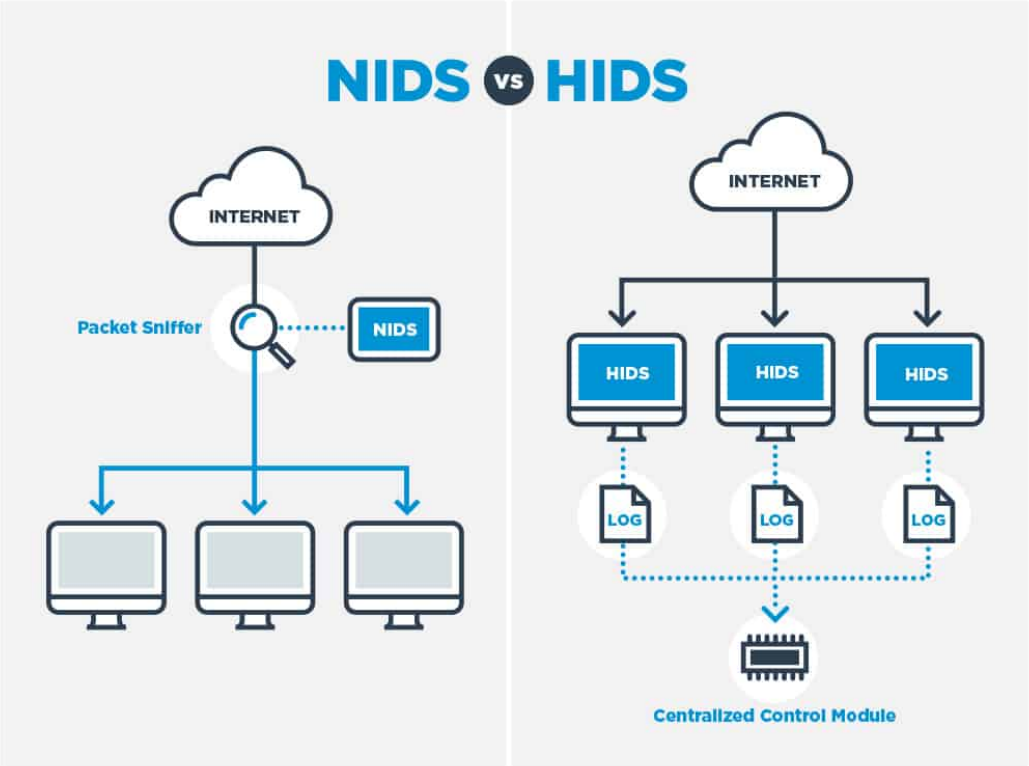
Example: OSSEC is an open-source HIDS that can detect various host-based attacks, including rootkits, file changes, and unauthorized access attempts. It uses a combination of signature-based, anomaly-based, and heuristic-based detection methods to increase the accuracy of attack detection and reduce false positives.

3.2 NIDS - Network-based IDS

A Network-based IDS (NIDS) is an IDS system that is installed on the network to monitor traffic passing through it. It analyzes packets passing through the network to identify suspicious behavior, such as unusual traffic patterns, unauthorized access attempts, and malware infections. NIDS can detect attacks that are visible on the network, but may not detect attacks that are targeted to a specific host or endpoint.

Example: Snort is an open-source NIDS that can detect a wide range of attacks and anomalies in network traffic. It uses a combination of signature-based, anomaly-based, and heuristic-based detection methods to increase the accuracy of attack detection and reduce false positives. Suricata is another open-source NIDS that can detect various network-based attacks, including DDoS, malware infections, and suspicious traffic patterns.

HIDS and NIDS can be used together to provide a comprehensive security solution. While HIDS is good at detecting attacks on a specific host or endpoint, NIDS can detect attacks that are not visible on the host or endpoint. Using both HIDS and NIDS can help provide a layered security approach to detect and respond to potential security threats.



4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters: Python 3.10.1

5 Input and Output

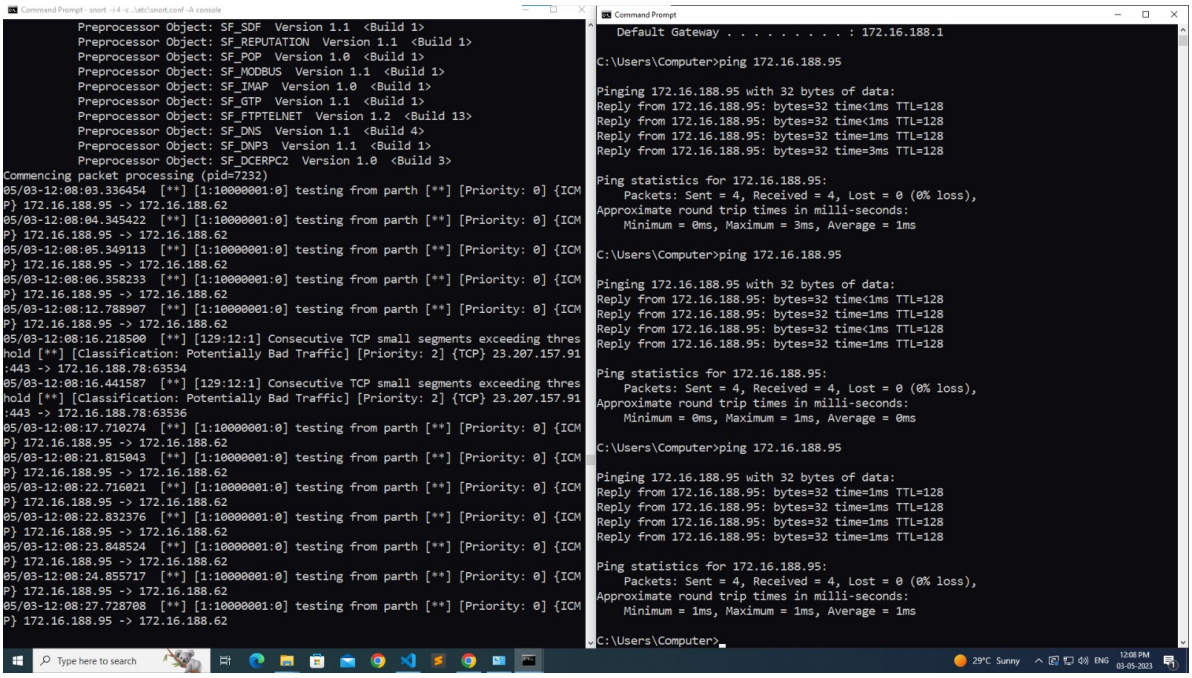


Figure 1:

:443 -> 172.16.188.78:63536
05/03-12:08:17.710274 [**] [1:10000001:0] testing from parth [**] [Priority: 0] {ICMP} 172.16.188.95 -> 172.16.188.62
05/03-12:08:21.815043 [**] [1:10000001:0] testing from parth [**] [Priority: 0] {ICMP} 172.16.188.95 -> 172.16.188.62
05/03-12:08:22.716021 [**] [1:10000001:0] testing from parth [**] [Priority: 0] {ICMP} 172.16.188.95 -> 172.16.188.62
05/03-12:08:22.832376 [**] [1:10000001:0] testing from parth [**] [Priority: 0] {ICMP} 172.16.188.95 -> 172.16.188.62
05/03-12:08:23.848524 [**] [1:10000001:0] testing from parth [**] [Priority: 0] {ICMP} 172.16.188.95 -> 172.16.188.62
05/03-12:08:24.855717 [**] [1:10000001:0] testing from parth [**] [Priority: 0] {ICMP} 172.16.188.95 -> 172.16.188.62
05/03-12:08:27.728708 [**] [1:10000001:0] testing from parth [**] [Priority: 0] {ICMP} 172.16.188.95 -> 172.16.188.62

Figure 2:

```
#-------------
# LOCAL RULES
#-------------

alert icmp any any -> any any (msg:"testing icmp"; sid:10000001;)
alert icmp 172.16.188.95 any -> any any (msg:"testing from parth"; sid:10000001;)
```

Figure 3:

```
% The Given Signature is Valid
```

6 Conclusion

Thus, We have learnt IDS systems and their types. We have also learnt about various tools based on IDS systems.

7 FAQ

1. **What are various types of IDS system?**

   There *are two main types of IDS systems*:

   (a) *Network-based IDS (NIDS)*: NIDS systems monitor network traffic in real-time to detect and alert on potential attacks. They analyze packets passing through the network to identify suspicious behavior, such as unusual traffic patterns, unauthorized access attempts, and malware infections.

   (b) *Host-based IDS (HIDS)*: HIDS systems monitor the activity on individual hosts to detect and alert on potential attacks. They analyze system logs, file integrity, and user activity to identify suspicious behavior, such as unauthorized access, malware infections, and changes to system configurations.

2. **What are the popular tools based on IDS systems?**

   There *are several popular tools based on IDS systems, including*:

   (a) *Snort*: an open-source NIDS that can detect a wide range of attacks and anomalies in network traffic.

   (b) *Suricata*: an open-source NIDS that can detect and alert on various network-based attacks, including DDoS, malware infections, and suspicious traffic patterns.

   (c) *OSSEC*: an open-source HIDS that can detect and alert on various host-based attacks, including rootkits, file changes, and unauthorized access attempts.

   (d) *Bro*: an open-source NIDS that can detect and alert on various network-based attacks, including malware infections, network scans, and suspicious traffic patterns.

   (e) *Security Onion*: a Linux distribution that includes several IDS tools, including Snort, Suricata, and Bro, and provides a centralized platform for monitoring and analyzing network and host activity.

3. **What are the detection methods of IDS?**
   IDS systems use several methods for detecting potential attacks, including:

   (a) *Signature-based detection*: IDS systems can detect known attacks by comparing network or host activity to a database of known attack signatures. If a match is found, the IDS can generate an alert.

   (b) *Anomaly-based detection*: IDS systems can detect new or unknown attacks by identifying patterns or behavior that deviate from normal or expected activity. This method requires a baseline of normal activity to be established, which the IDS can then use to identify anomalies.

   (c) *Heuristic-based detection*: IDS systems can detect potential attacks by using algorithms and rules to identify behavior that is indicative of an attack. This method can be more flexible than signature-based detection, but can also result in more false positives.

   (d) *Hybrid detection*: IDS systems can use a combination of signature-based, anomaly-based, and heuristic-based detection methods to increase the accuracy of attack detection and reduce false positives.

MIT World Peace University

Information and Cybersecurity
Second Year B. Tech, Semester 1

Learning to Work with NESSUS tool for Vulnerability Assessment

Lab Assignment 11

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

May 1, 2023

Contents

1 Aim

Configuration and demonstration of NESSUS tool for vulnerability assessment

2 Objectives

To learn authentication technique for access control

3 Theory

3.1 Vulnerability Assessment

Vulnerability assessment is the process of identifying and evaluating potential security vulnerabilities in a computer system or network. This assessment helps organizations to identify and prioritize potential risks and develop strategies to mitigate them. One of the popular vulnerability assessment tools is Nessus.

Nessus is a widely used vulnerability assessment tool that is designed to scan and detect vulnerabilities in networks, systems, and applications. Here are some of the features of Nessus:

1. Multiple scanning options: Nessus offers various scanning options, including network, web application, and database scanning. It supports multiple protocols, such as TCP/IP, SNMP, SSH, and HTTP/HTTPS.
2. Comprehensive vulnerability database: Nessus maintains a comprehensive database of known vulnerabilities and exploits, which it uses to identify potential vulnerabilities in the target system.
3. Customizable policies: Nessus allows users to customize scan policies based on their specific needs. Users can create policies to scan specific IP addresses, ports, or protocols.
4. Real-time alerts: Nessus provides real-time alerts on critical vulnerabilities, enabling organizations to quickly remediate potential threats.
5. Compliance reporting: Nessus generates compliance reports based on industry standards such as PCI DSS, HIPAA, and CIS.
6. Integration with other tools: Nessus can integrate with other security tools such as SIEMs and incident response platforms.

3.2 How Nessus Works

1. Discovery: Nessus starts by discovering devices on the network, including servers, workstations, routers, and switches.
2. Scan configuration: The user configures the scan policy based on their needs, such as IP range, scanning ports, protocols, and vulnerabilities to scan for.
3. Scan execution: Nessus then scans the target system for vulnerabilities based on the configured policy. The scanning process may take some time, depending on the size of the network and the complexity of the scan policy.

4. Vulnerability assessment: Nessus then analyzes the scan results and reports potential vulnerabilities. It provides detailed information on the type of vulnerability, the level of severity, and possible remediation actions.

5. Reporting: Nessus generates a report that summarizes the vulnerabilities found during the scan. The report includes detailed information on each vulnerability, including its severity level and recommended remediation actions.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters : Python 3.10.1

5 Input and Output

6 Conclusion

Thus, we have learnt about the authentication technique for access control, and implemented it using NESSUS tool for vulnerability assessment.Thus, we have seen how to implement digital signatures using DSA algorithm.

7 FAQ

1. **What vulnerabilities can Nessus detect?**

   Nessus is a vulnerability scanner that can detect a wide range of vulnerabilities in an IT environment, including:

   (a) Operating system vulnerabilities: Nessus can identify vulnerabilities in the operating systems of network devices, servers, and workstations, including Windows, Linux, Unix, and macOS.

   (b) Application vulnerabilities: Nessus can detect vulnerabilities in popular applications such as web browsers, email clients, and office productivity software.

   (c) Network vulnerabilities: Nessus can identify network vulnerabilities, including weak passwords, open ports, and misconfigured network devices.

   (d) Malware and botnets: Nessus can detect signs of malware and botnets on network devices and workstations.

   (e) Web application vulnerabilities: Nessus can identify vulnerabilities in web applications, including SQL injection, cross-site scripting, and directory traversal.

2. **What are the limitations of Nessus essentials?**

   Nessus Essentials is a free version of the Nessus vulnerability scanner that is limited in its capabilities compared to the paid version. Some of the limitations of Nessus Essentials include:

   - Limited scanning: Nessus Essentials is limited to scanning up to 16 IP addresses or hosts at a time, while the paid version can scan thousands of hosts.
   - No scheduling: Nessus Essentials does not allow users to schedule scans or automated reporting, which can be inconvenient for organizations with large IT environments.
   - Limited vulnerability coverage: Nessus Essentials has a smaller set of plugins for detecting vulnerabilities compared to the paid version, which can limit its effectiveness in identifying security issues.
   - No support: Nessus Essentials does not come with technical support from Tenable, the company behind Nessus.

3. **How can you identify a false positive vulnerability in Nessus?**

   A false positive vulnerability in Nessus occurs when the scanner reports a vulnerability that does not actually exist. To identify false positive vulnerabilities in Nessus, follow these steps:

   - Verify the vulnerability: Check the affected device or application to verify if the reported vulnerability actually exists.
   - Check the plugin output: Review the Nessus plugin output to identify any discrepancies or errors that may have led to the false positive.
   - Confirm with multiple sources: Check other vulnerability scanners, security advisories, and technical forums to confirm if the vulnerability is a known issue.

- Perform additional testing: Conduct additional testing or penetration testing to confirm if the vulnerability can be exploited.

4. **How many hosts can Nessus scan? What port does Nessus use?**

   The number of hosts that Nessus can scan depends on the version of the software and the license purchased. The free version of Nessus, Nessus Essentials, can scan up to 16 IP addresses or hosts at a time, while the paid versions can scan thousands of hosts.

   Nessus uses various ports to communicate with the devices being scanned. The default port used by Nessus is 8834 for web-based communication, but Nessus can also use other ports depending on the type of scan being performed. For example, Nessus may use port 22 for SSH communication or port 445 for SMB communication.