Slot: E1+TE1
Faculty: Prof. USHA DEVI G

School of Information Technology & Engineering
Fall-Semester: 2022 – 2023

NIS Project Report
Review - III

Programme: B. Tech (IT) **Course Code: ITE4001**
Course Title : Network and Information Security
Date of Submission: 15.11.2022

Securing Data Using Digital Envelope

Team Members:

1. Moulik Singh Arora - 20BIT0415
2. Preksha J Dadhania - 20BIT0158
3. Rishika Aggrawal – 20BIT0147
4. Shreya Basu – 20BIT0144

Abstract :

We are on the cusp of handling enormous amounts of data in the context of today's technological world. We must also protect those in order to do that. Thus, the idea of a digital envelope exists. A digital envelope is created by electronically encrypting any message and placing it in a sealed envelope to guarantee privacy and security by preventing tampering by unauthorized parties. Here, the RSA (Rivest- Shamir-Adleman) notion is used in the philosophy of Digital Envelopes, where we employ both a secret key and a public key cryptography technique. In this, the public key is used to convey the secret key to the other party, and the secret key is used to encrypt and decode the communication. By utilizing the wrong reader and the wrong decryption key, Digital Envelopes render the entire communication delivered from a sender to a recipient to be twaddle. A well-thought-out mail client, on the other hand, can keep track of which recipients to use digital envelopes with, the kind of digital envelope to use, and the specific key to use for each recipient throughout the process. In this project, we have proposed and briefly addressed the idea of implementing the digital envelope.

Introduction on Digital Envelope:

A digital envelope is a secure electronic data container used to encrypt and authenticate data to protect messages. Using a digital envelope, users can encrypt data quickly using secret keys and conveniently and securely with public keys.

Secret (symmetric) key and public key encryption are used as the two layers of encryption in a digital envelope. The encoding and decoding of messages uses secret key encryption. A secret key is transmitted over a network using public key encryption. Plain text communication is not necessary for this strategy.

It is possible to make a digital envelope using either of the following techniques:

- For message encryption, use algorithms using secret keys like Rijndael or Two fish.
- RSA's public key encryption algorithm is used to encrypt a recipient's secret key using the recipient's public key.

Pretty Good Privacy (PGP), a well-known data cryptography programme that also offers cryptographic privacy and data transmission authentication, is an example of a digital envelope.

Existing Works :

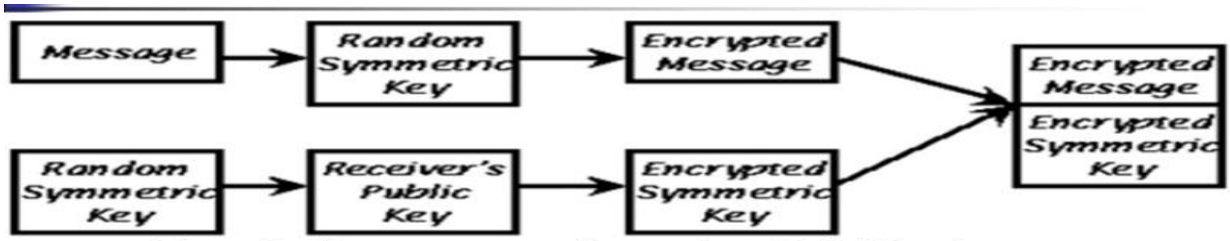


Figure 3: The process used to create a Digital Envelope

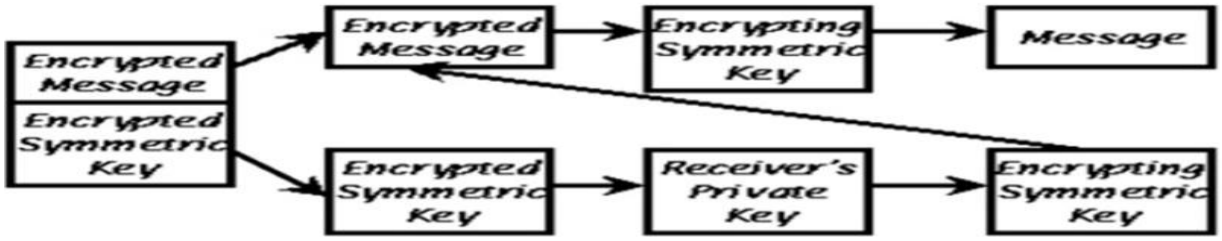
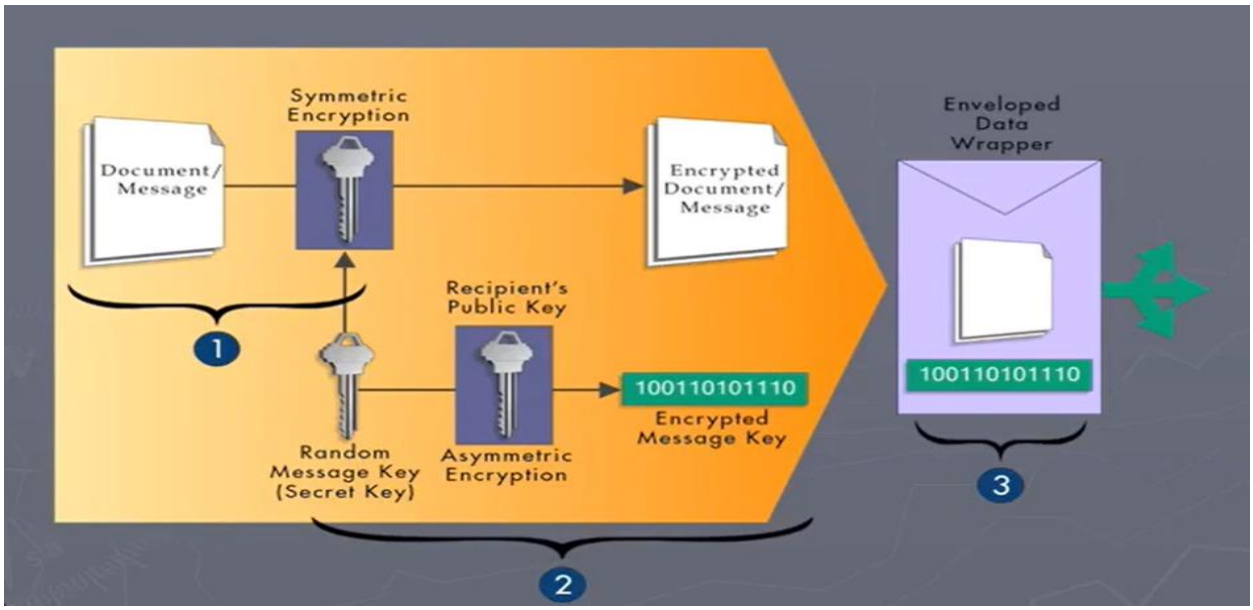


Figure 4: The process used to verify a Digital Envelope

Encryption using Symmetric Key :

A symmetric algorithm session key must first be randomly chosen by the sender of a message in a digital envelope, and it must then be encrypted with the recipient's public key and an asymmetric algorithm. The sender uses the original (unencrypted) symmetric session key to encrypt the message body (also known as the plaintext), and after doing so, transmits the receiver both the encrypted session key and the encrypted message body (also known as the ciphertext).

To get the authentic message body, the recipients of that message must first decode the session key using their own private key, and then decrypt the remainder of the message using the decrypted session key (the plaintext). The session key and the original message can only be recovered by the owner of the recipient's private key, which is ideally just the recipient.



Pictorial representation of Digital Envelope

RSA Algorithm:

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and the Private key is kept private.

Generation of Key Pairs in RSA:

A public key and a private key must be generated by any individual or party who wants to take part in encrypted communication. The steps taken to generate keys are explained below:

1) Create the RSA modulus, first (n)

- Pick the two big primes p and q.
- Determine n=p*q. Let n be a big integer, typically at least 512 bits, for effective uncrackable encryption.

2) Locate the Derived Number (e)

- The value of e ought to be higher than 1 and lower than (p-1)(q-1).
- Except for 1, there must be no common factor between e and (p-1)(q-1). In other words, e and (p - 1)(q - 1) are two numbers that are coprime.

3) Create the public key.

- The RSA public key is made public and is made up of the two numbers (n, e).
- Though n is a part of the public key, it's interesting to note that the difficulty of factorizing a huge prime number assures that an attacker cannot locate the two primes (p & q) necessary to acquire n in a finite amount of time. This is RSA's main strength.

4) Produce a private key.

- Secure Key P, Q, and e are used to determine d. There is a distinct number d for the given n and e.
- The inverse of e modulo (p - 1) is number d. (q – 1). This indicates that d is the number less than (p - 1)(q - 1) such that it equals 1 modulo (p - 1)(q - 1) when multiplied by e.

The mathematical formula for this relationship is : ed = 1 mod (p - 1)(q − 1).

RSA Encryption:

- Consider the case when the sender wants to text someone whose public key is (n, e).
- Then, the sender displays the plaintext as a series of numbers less than n.
- The first plaintext P, which is a number modulo n, must be encrypted. The mathematical step for the encryption process is : C = Pe mod n.

RSA Decryption:

- For RSA, the decryption procedure is also fairly simple. Assume that cipher text C has been delivered to the recipient of public-key pair (n, e).
- Receiver increases C by his private key's d power. The plaintext P will be the outcome modulo n.
- Plaintext = Cd mod n

Literature Surveys:

1. A Novel Approach for Security Using RSA Parallelization Process

Key exchange for cryptographic system is always a challenge. A private key algorithm is used to encipher and decipher the message, but after the symmetric key encrypts the message the digital envelope concept is used where the public key method or symmetric key algorithms are used to send the private key to the other party. A significant amount of research has focused on methods to improve the efficiency of cryptographic algorithms. Public key based cryptographic algorithms are usually considered as slower than their corresponding symmetric key based algorithms due to their root in modular arithmetic. So for encrypting the large message, symmetric key algorithms are used whereas for the relatively smaller key, asymmetric key algorithm is used. Here we are implementing this digital envelope system using asymmetric key algorithm. The goal is to improve the system by implementing the algorithm with parallelization using OpenMP on the GCC infrastructure.

2. Authenticity and integrity of digital mammography images

Data security becomes more and more important in telemammography which uses a public high-speed wide area network connecting the examination site with the mammography expert center. Generally, security is characterized in terms of privacy, authenticity and integrity of digital data. Privacy is a network access issue and is not considered in this paper. The authors present a method, authenticity and integrity of digital mammography, here which can meet the requirements of authenticity and integrity for mammography image (IM) transmission. The authenticity and integrity for mammography (AIDM) consists of the following four modules. (1) Image preprocessing: To segment breast pixels from background and extract patient information from digital imaging and communication in medicine (DICOM) image header. (2) Image hashing: To compute an image hash value of the mammogram using the MD5 hash algorithm. (3) Data encryption: To produce a digital envelope containing the encrypted image hash value (digital signature) and corresponding patient information. (4) Data embedding: To embed the digital envelope into the image. This is done by replacing the least significant bit of

a random pixel of the mammogram by one bit of the digital envelope bit stream and repeating for all bits in the bit stream. Experiments with digital IMs demonstrate the following. (1) In the expert center, only the user who knows the private key ran open the digital envelope and read the patient information data and the digital signature of the mammogram transmitted from the examination site. (2) Data integrity can he verified by matching the image hash value decrypted from the digital signature with that computed from the transmitted image. (3) No visual quality degradation is detected in the embedded image compared with the original. The authors' preliminary results demonstrate that AIDM is an effective method for image authenticity and integrity in telemammography application.

3. Modern Cryptographic Schemes: Applications and Comparative Study

Cryptography and encryption have been used for secure communication. In the modern world, cryptography is a very important tool for protecting information in computer systems. With the invention of the World Wide Web or Internet, computer systems are highly interconnected and accessible from any part of the world. As more systems get interconnected, more threat actors try to gain access to critical information stored on the network. It is the responsibility of data owners or organizations to keep this data securely and encryption is the main tool used to secure information. In this paper, we will focus on different techniques and the modern application of cryptography. We will study different cryptographic schemes: symmetric, asymmetric (sometimes referred to as public key), and hybrid systems. The paper will present a comparative study for these schemes and their applications in network protocols. Moreover, we will highlight the concept of Quantum Cryptography, which takes advantage of quantum physics at the physical layer.

4. Scalable blockchain model using off-chain IPFS storage for healthcare data security and privacy

Traditional healthcare systems in the present scenario follow centralized client-server architecture to store and process patient-health related information. Data stored in each of the healthcare institution remain in silos which cannot be easily shared with other institutions due to technical and infrastructural constraints. Hospitals do not have an effective and secure data sharing mechanism leading to monetary and resource loss in the case of a person visiting different hospitals. Blockchain, a disruptive technology with secure and reliable decentralized framework, and can be used to circumvent problems in traditional healthcare architecture for secure storage, sharing and retrieval of Electronic Health Records (EHR). A blockchain-based framework integrated with InterPlanetary File System (IPFS) for EHR in healthcare management has been proposed in this paper. This proposed framework will enable healthcare institutions to maintain fail-safe and tamper-proof healthcare ledgers in a decentralized manner. Hospitals and doctors act as lightweight nodes, whereas patient nodes can be full or lightweight nodes. The model proposes two-factor authentication and multi-factor

authentication for preventing fake node attacks. Patient-centric access model allows the patients to act as digital stewards for their health data, allowing access to doctors and hospitals on demand and revoking it after stipulated time. Symmetric key encryption (AES-128) is used for encrypting data before storing into IPFS. Asymmetric encryption (RSA-4096) is used for generating digital envelopes to pass on symmetric key to authorized entities. Digital signatures (RSA-1024) make sure that the transactions are valid and from authorized nodes. Hashing of the encrypted data is done using SHA-256 algorithm. Multiple layers of security implemented in this model makes sure that adversaries cannot obtain data stored in IPFS; even if they retrieve the data, it will not be meaningful since it is encrypted. The proposed framework for off-chain storage of health data using IPFS saves blockchain structure from scalability issues. Further the proposal for blockchain integration with IPFS helps preserve privacy in the healthcare system, making it highly secure, scalable, and robust.

Proposed Methodology:

- ***Making the contents of the letter that will be sent by shared secret key more complicated***

  In this instance of a digital envelope, the sender transmits an envelope containing cipher text and cipher key information. By using the public key generated by the RSA algorithm to encrypt the plain message, cipher text is produced. The shared secret key N and the private key, which is produced via the RSA technique, are combined to form the cipher key. The sender must input shared secret key number N. On the other side, the recipient receives the digital envelope from the sender. The recipient's envelope contains both cipher text and cipher key, and the former can be used to decrypt the latter into the message using the latter's private key. Upon entering the proper shared secret number N, the receiver can retrieve the private key using a cipher key.

- ***The receiver is receiving the digital envelope containing the encrypted text and key combination.***

  When the shared number N is entered correctly, the receiver can extract the private key from the cipher key and obtain the private key.

Sender’s Side:

To get the cipher text, the message is encrypted using the public key.

- N will be given a random shared secret number (let's assume N=7). To make it more difficult, the private key and the number N are combined.
- Let's say that N = 7 and the private key is 10.
- Consequently, the cipher key (C.K = 17) was obtained.
- Finally, the recipient receives the digital envelope that was created using the cipher text and cipher key.

Receiver's Side:

- The receiver is receiving the digital envelope that contains the cipher text and key combination.
- The receiver extracts the private key from the encryption key by entering the shared number N, which, if entered correctly, yields the private key.
- (We know that the cipher text is 17 and N is 7, therefore we can calculate the private key as being 10).
- If N is entered incorrectly, the obtained private key and the message that was decrypted using the private key will also be incorrect. As a result, the attacker will get the erroneous message.
- The encryption text is decrypted to reveal the plain message using the private key.

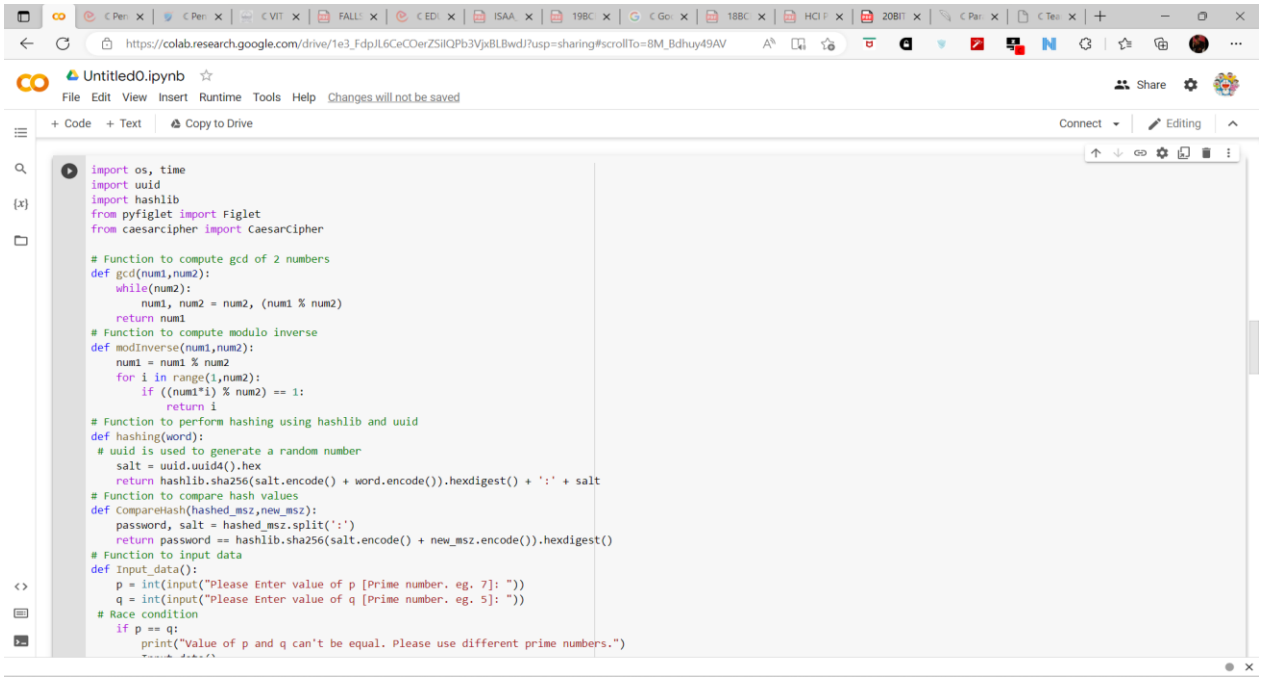
Code:

```
import os, time
import uuid
import hashlib
from pyfiglet import Figlet
from caesarcipher import CaesarCipher

# Function to compute gcd of 2 numbers
def gcd(num1,num2):
   while(num2):
      num1, num2 = num2, (num1 % num2)
   return num1
# Function to compute modulo inverse
def modInverse(num1,num2):
   num1 = num1 % num2
   for i in range(1,num2):
      if ((num1*i) % num2) == 1:
         return i
# Function to perform hashing using hashlib and uuid
def hashing(word):
 # uuid is used to generate a random number
   salt = uuid.uuid4().hex
   return hashlib.sha256(salt.encode() + word.encode()).hexdigest() + ':' + salt
# Function to compare hash values
def CompareHash(hashed_msz,new_msz):
   password, salt = hashed_msz.split(':')
```

```
    return password == hashlib.sha256(salt.encode() + new_msz.encode()).hexdigest()
# Function to input data
def Input_data():
    p = int(input("Please Enter value of p [Prime number. eg. 7]: "))
    q = int(input("Please Enter value of q [Prime number. eg. 5]: "))
 # Race condition
    if p == q:
        print("Value of p and q can't be equal. Please use different prime numbers.")
        Input_data()
        exit()
 # Common key to be shared between respondents for symmetric communication.
    common_key = int(input("Please enter the common key to be shared: "))
    print("\nThanks!")
    return p,q,common_key
# RSA Key generation process
def RSAKeyGeneration(p,q):
    n = p * q
    totient_func = (p-1)*(q-1)
    flag = 1
    while(flag):
        e = int(input("Please select value of e: "))
        if gcd(totient_func,e) != 1:
            print("\nGCD of totient_func and e should be 1 (Relatively prime).")
            print("Please try again!")
            continue
        flag = 0
    if e>1 and e<totient_func:
        d = modInverse(e,totient_func);
        print("\nValue of computed d is: %s" %(d))

    print("\nPublic Key here is - PU(%s,%s)" %(e,n))
    print("Private Key here is - PR(%s,%s)" %(d,n))
    return n,e,d
# RSA Encryption process
def RSAEncryption(e,n,common_key):
    Cipher = (common_key**e) % n
    print("\nCipher text generated is: %s" %(Cipher))
    return Cipher
# Symmetric Encryption using shared common key
def SymmetricEncryption(common_key):
    msz = input("\nPlease enter the message to be shared: ")
    hashed_msz = hashing(msz)
    print("\nHash for message given is: %s" %(hashed_msz))
    cipher = CaesarCipher(msz, offset= common_key)
    encoded_msz = cipher.encoded
    print("Symmetrically encrypted data is: %s" %(encoded_msz))
    return hashed_msz, encoded_msz
# RSA Decryption Process
def RSADecryption(n,d,Cipher,common_key):
```

```
    Decipher = (Cipher**d) % n
    print("\nDeciphered Common key is %s" %(Decipher))
    print("\nWhich match the sent key - %s" %(common_key))
    return Decipher
# Symmetric Decryption using shared common key
def SymmetricDecryption(hashed_msz,encoded_msz,Decipher):
    decipher = CaesarCipher(encoded_msz, offset= Decipher)
    decoded_msz = decipher.decoded
    print("\nDecrypted message is: %s" %(decoded_msz))
    print("\n##### RESULT #####")
    if CompareHash(hashed_msz,decoded_msz):
        print("\n--> The hash is different. The data is incorrect.")
    else:
        print("\n-->The hash is same. The data is correct")
# Input Menu
def Menu():
    print ("Please Select the mode of operation:- \n")
    print ("1) Give Input.")
    print ("2) Initiate Key Generation Process.")
    print ("3) Initiate RSA Encryption For Assymetric Common Key sharing.")
    print ("4) Initiate Symmetric Encryption for conversation after ke sharing.")
    print ("5) Decrypt Asymmetricly shared common key.")
    print ("6) Decrypt message sent through Symmetric Conversation.")
    print ("7) Exit the program :[.\n")
# Actual Program
if __name__ == '__main__':
    die =1
    try:
        while(die):
            os.system("cls")
            Menu()
            choice = int(input("Enter your choice: "))
            print("\n")
            if choice == 1:
                print("##### DATA INPUT #####")
                p,q,common_key = Input_data()
                time.sleep(10)
            elif choice == 2:
                print("##### KEY GENERATION PROCESS #####")
                n,e,d = RSAKeyGeneration(p,q)
                time.sleep(10)
            elif choice == 3:
                print("##### RSA ENCRYPTION PROCESS #####")
                Cipher = RSAEncryption(e,n,common_key)
                time.sleep(10)
            elif choice == 4:
                print("##### SYMMETRIC ENCRYPTION PROCESS #####")
                hashed_msz,encoded_msz = SymmetricEncryption(common_key)
                time.sleep(10)
```

```
        elif choice == 5:
            print("##### RSA DECRYPTION PROCESS #####")
            Decipher = RSADecryption(n,d,Cipher,common_key)
            time.sleep(10)
        elif choice == 6:
            print("##### SYMMETRIC DECRYPTION PROCESS #####")
            SymmetricDecryption(hashed_msz,encoded_msz,Decipher)
            time.sleep(10)
        elif choice == 7:
            die = 0
            quit()
        else:
            print("This choice is not in the menu. Try Again!!!")
            die=1
    except KeyboardInterrupt:
        print("\n\nInterrupt received! Exiting cleanly...\n")
```



Output:

```
Please Select the mode of operation:-

1) Give Input.
2) Initiate Key Generation Process.
3) Initiate RSA Encryption For Assymetric Common Key sharing.
4) Initiate Symmetric Encryption for conversation after ke sharing.
5) Decrypt Asymmetricly shared common key.
6) Decrypt message sent through Symmetric Conversation.
7) Exit the program :[.

Enter your choice: 1


##### DATA INPUT #####
Please Enter value of p [Prime number. eg. 7]: 17
Please Enter value of q [Prime number. eg. 5]: 11
Please enter the common key to be shared: 187

Thanks!

Please Select the mode of operation:-

1) Give Input.
2) Initiate Key Generation Process.
3) Initiate RSA Encryption For Assymetric Common Key sharing.
4) Initiate Symmetric Encryption for conversation after ke sharing.
5) Decrypt Asymmetricly shared common key.
6) Decrypt message sent through Symmetric Conversation.
7) Exit the program :[.

Enter your choice: 2


##### KEY GENERATION PROCESS #####
Please select value of e: 7

Value of computed d is: 23

Public Key here is - PU(7,187)
Private Key here is - PR(23,187)
```

```
Please Select the mode of operation:-

1) Give Input.
2) Initiate Key Generation Process.
3) Initiate RSA Encryption For Assymetric Common Key sharing.
4) Initiate Symmetric Encryption for conversation after ke sharing.
5) Decrypt Asymmetricly shared common key.
6) Decrypt message sent through Symmetric Conversation.
7) Exit the program :[.

Enter your choice: 3


##### RSA ENCRYPTION PROCESS #####

Cipher text generated is: 0
```

```
Please Select the mode of operation:-

1) Give Input.
2) Initiate Key Generation Process.
3) Initiate RSA Encryption For Assymetric Common Key sharing.
4) Initiate Symmetric Encryption for conversation after ke sharing.
5) Decrypt Asymmetricly shared common key.
6) Decrypt message sent through Symmetric Conversation.
7) Exit the program :[.

Enter your choice: 5


##### RSA DECRYPTION PROCESS #####

Deciphered Common key is 0

Which match the sent key - 187
```

```
Please Select the mode of operation:-

1) Give Input.
2) Initiate Key Generation Process.
3) Initiate RSA Encryption For Assymetric Common Key sharing.
4) Initiate Symmetric Encryption for conversation after ke sharing.
5) Decrypt Asymmetricly shared common key.
6) Decrypt message sent through Symmetric Conversation.
7) Exit the program :[.

Enter your choice: 4


##### SYMMETRIC ENCRYPTION PROCESS #####

Please enter the message to be shared: hello

Hash for message given is: 3e4e16f91b5825ab83a0acf58b87a5064369ff33617e9a0e2a659c1a0e6d5b52:5440fe3882724eb4a6840b7cd430b1e0
Symmetrically encrypted data is: mjqqt
```

```
Please Select the mode of operation:-

1) Give Input.
2) Initiate Key Generation Process.
3) Initiate RSA Encryption For Assymetric Common Key sharing.
4) Initiate Symmetric Encryption for conversation after ke sharing.
5) Decrypt Asymmetricly shared common key.
6) Decrypt message sent through Symmetric Conversation.
7) Exit the program :[.

Enter your choice: 6


##### SYMMETRIC DECRYPTION PROCESS #####

Decrypted message is: mjqqt

##### RESULT #####

-->The hash is same. The data is correct
```

```
Please Select the mode of operation:-

1) Give Input.
2) Initiate Key Generation Process.
3) Initiate RSA Encryption For Assymetric Common Key sharing.
4) Initiate Symmetric Encryption for conversation after ke sharing.
5) Decrypt Asymmetricly shared common key.
6) Decrypt message sent through Symmetric Conversation.
7) Exit the program :[.

Enter your choice: 7
```

Security Parameters :

1) Message Length

Our message length will be high as it will directly affect the number of iterations required to crack the code, i.e. higher the message length, higher the number of iterations required.

2) Common Key Size

Size of Common Key will be kept big so as to make it more difficult for the intruder to decrpypt.

3) Size of salt in hash

Size of salt in hash is kept large as it is used to combine with the message in order to make the encrypted value bigger so that it becomes more difficult to decrypt the message.

4) Encryption and decryption key size in RSA algorithm

The key used in encryption and decryption will be kept big so as to make the decryption of the message more difficult for third party.

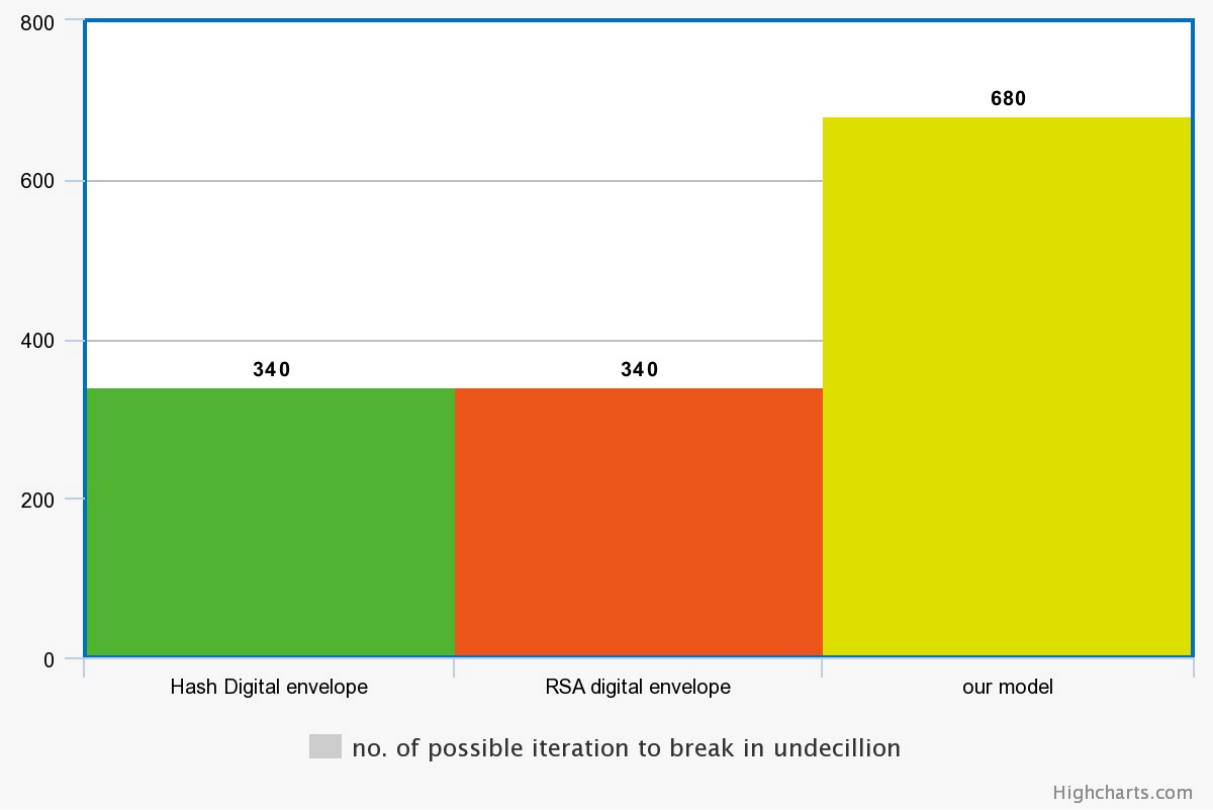
5) Size of random number N

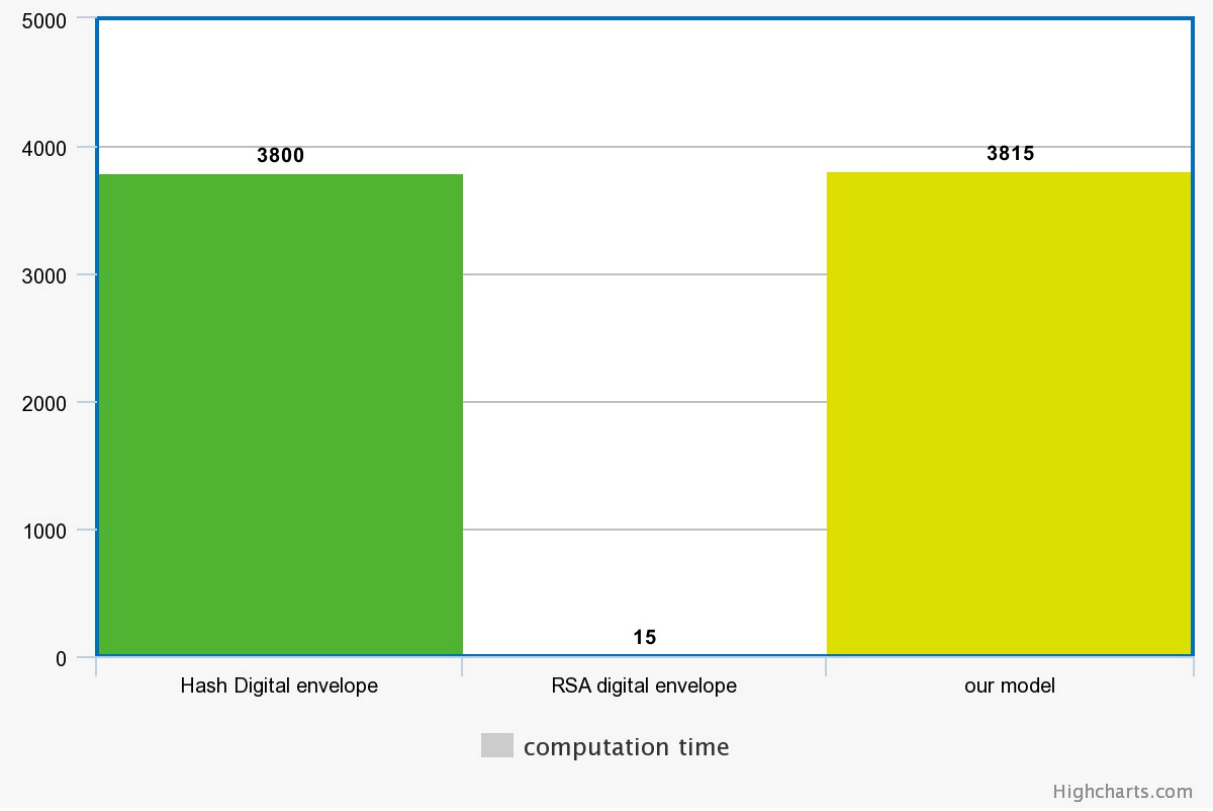
Random number N is used to generate the encryption key size. When the encryption key is bigger, it becomes more difficult to crack the code, hence size of random number N should also be kept high.

Comparison Table :

Algorithm	No. of times to crack the code
RSA digital envelope	2^128
Hash algo digital envelope	2^128
RSA + hash + Caesar cipher digital envelope	2^128+ 2^128

Graphs :


800
600
400
200
0
340
340
680
Hash Digital envelope
RSA digital envelope
our model
no. of possible iteration to break in undecillion
Highcharts.com


5000
4000
3000
2000
1000
0
3800
15
3815
Hash Digital envelope
RSA digital envelope
our model
computation time
Highcharts.com

Conclusion:

The system is resistant to Man-in-the-Middle attacks. Since the encryption must be decrypted, the private key must be obtained. The issue of transferring keys is solved since the sender generates the keys and adds the private key to the encryption.

The envelope stands to be confusing to the attacker's eyes even if the file is obtained by an unauthorized person and if any form of reverse engineering is to be done. A symmetric key used in the classic digital envelope approach makes the communication subject to compromise if a brute force attack is used to decrypt it.

We chose asymmetric encryption over symmetric encryption in our project, which primarily deals with secret key exchange. Although someone can spoof an asymmetric encryption because there are two separate keys at each end, it nevertheless offers higher security than symmetric encryption.

References:

1. A Secure and Fast Approach for Encryption and Decryption of Message Communication, Ekta Agrawal1 , Dr. Parashu Ram Pal, 2017.
2. Jain, Amrita, and Vivek Kapoor. "Secure communication using RSA algorithm for network environment." International Journal of Computer Applications 118.7 (2015).
3. https://www.geeksforgeeks.org/rsa-algorithm-cryptography/
4. https://www.tutorialspoint.com/cryptography/public_key_encryption.htm