Name : Munilakshmi. GJ

RegNo: 20BCI0190

Course: Cyber Security and Ethical Hacking

Qn.

Assignment-3: Cryptography Analysis and Implementation

Objective: The objective of this assignment is to analyze cryptographic algorithms and implement them in a practical scenario.

Instructions:
Research: Begin by conducting research on different cryptographic algorithms such as symmetric key algorithms (e.g., AES, DES), asymmetric key algorithms (e.g., RSA, Elliptic Curve Cryptography), and hash functions (e.g., MD5, SHA-256). Understand their properties, strengths, weaknesses, and common use cases.

Analysis: Choose three cryptographic algorithms (one symmetric, one asymmetric, and one hash function) and write a detailed analysis of each. Include the following points in your analysis:

Briefly explain how the algorithm works.

Discuss the key strengths and advantages of the algorithm.

Identify any known vulnerabilities or weaknesses.

Provide real-world examples of where the algorithm is commonly used.

Implementation:
Select one of the cryptographic algorithms you analyzed and implement it in a practical scenario. You can choose any suitable programming language for the implementation.

Clearly define the scenario or problem you aim to solve using cryptography.

Provide step-by-step instructions on how you implemented the chosen algorithm.

Include code snippets and explanations to demonstrate the implementation.
Test the implementation and discuss the results.

Security Analysis:
Perform a security analysis of your implementation, considering potential attack vectors and countermeasures.

Identify potential threats or vulnerabilities that could be exploited.
Propose countermeasures or best practices to enhance the security of your implementation.
Discuss any limitations or trade-offs you encountered during the implementation process.

Conclusion: Summarize your findings and provide insights into the importance of cryptography in cybersecurity and ethical hacking.

Submission Guidelines:
Prepare a well-structured report that includes the analysis, implementation steps, code snippets, and security analysis.

Use clear and concise language, providing explanations where necessary.

Include any references or sources used for research and analysis.

Compile all the required files (report, code snippets, etc.) into a single zip file for submission.

Analysis:

Symmetric Algorithm: DES

The DES (Data Encryption Standard) algorithm is a symmetric-key block cipher created in the early 1970s by an IBM team and adopted by the National Institute of Standards and Technology (NIST). The algorithm takes the plain text in 64-bit blocks and converts them into ciphertext using 48-bit keys.

Since it's a symmetric-key algorithm, it employs the same key in both encrypting and decrypting the data. If it were an asymmetrical algorithm, it would use different keys for encryption and decryption.

DES WORKING:

To put it in simple terms, DES takes 64-bit plain text and turns it into a 64-bit ciphertext. And since we're talking about asymmetric algorithms, the same key is used when it's time to decrypt the text.

The algorithm process breaks down into the following steps:

1. The process begins with the 64-bit plain text block getting handed over to an initial permutation (IP) function.
2. The initial permutation (IP) is then performed on the plain text.
3. Next, the initial permutation (IP) creates two halves of the permuted block, referred to as Left Plain Text (LPT) and Right Plain Text (RPT).
4. Each LPT and RPT goes through 16 rounds of the encryption process.
5. Finally, the LPT and RPT are rejoined, and a Final Permutation (FP) is performed on the newly combined block.
6. The result of this process produces the desired 64-bit ciphertext.

The encryption process step (step 4, above) is further broken down into five stages:

1. Key transformation
2. Expansion permutation
3. S-Box permutation
4. P-Box permutation
5. XOR and swap

For decryption, we use the same algorithm, and we reverse the order of the 16 round keys.

Next, to better understand what is DES, let us learn the various modes of operation for DES.

Advantages of the DES algorithm:

1. It is set as a standard by the US government.
2. When compared to the software, it works faster on hardware.
3. Triple DES, used a 168-bit key which is very hard to crack.

Disadvantages of the DES algorithm:

1. Weakly secured algorithm.
2. There is a threat from Brute force attacks.
3. A DES cracker machine known as Deep Crack is available in the market.

Real-World example:

The DES algorithm is used whenever a not-very-strong encryption is needed. It can be used in random number generators or even as a permutation generator. One of the most important practical applications of the DES algorithm is to create triple DES legacy systems with three keys.

Asymmetric Algorithm: RSA

The RSA algorithm is a public-key signature algorithm developed by Ron Rivest, Adi Shamir, and Leonard Adleman. Their paper was first published in 1977, and the algorithm uses logarithmic functions to keep the working complex enough to withstand brute force and streamlined enough to be fast post-deployment. The image below shows it verifies the digital signatures using RSA methodology.

RSA WORKING:

Keeping the image above in mind, go ahead and see how the entire process works, starting from creating the key pair, to encrypting and decrypting the information.

Key Generation

You need to generate public and private keys before running the functions to generate your ciphertext and plaintext. They use certain variables and parameters, all of which are explained below:

- Choose two large prime numbers (p and q)
- Calculate n = p*q and z = (p-1)(q-1)
- Choose a number e where 1 < e < z
- Calculate d = e-1mod(p-1)(q-1)
- You can bundle private key pair as (n,d)
- You can bundle public key pair as (n,e)

Encryption/Decryption Function

Once you generate the keys, you pass the parameters to the functions that calculate your ciphertext and plaintext using the respective key.

- If the plaintext is m, ciphertext = me mod n.
- If the ciphertext is c, plaintext = cd mod n

To understand the above steps better, you can take an example where p = 17 and q=13. Value of e can be 5 as it satisfies the condition $1 < e < (p-1)(q-1)$.

$N = p * q = 221$

$D = e\text{-}1 \bmod (p-1)(q-1) = 29$

Public Key pair = (221,5)

Private Key pair = (221,29)

If the plaintext(m) value is 10, you can encrypt it using the formula me mod n = 82.

To decrypt this ciphertext(c) back to original data, you must use the formula cd mod n = 29.

Advantages of RSA:

- No Key Sharing: RSA encryption depends on using the receiver's public key, so you don't have to share any secret key to receive messages from others.
- Proof of Authenticity: Since the key pairs are related to each other, a receiver can't intercept the message since they won't have the correct private key to decrypt the information.
- Faster Encryption: The encryption process is faster than that of the DSA algorithm.
- Data Can't Be Modified: Data will be tamper-proof in transit since meddling with the data will alter the usage of the keys. And the private

key won't be able to decrypt the information, hence alerting the receiver of manipulation.

This sums up this lesson on the RSA Algorithm.

Disadvantages of RSA:

- Because RSA only employs asymmetric encryption and complete encryption requires both symmetric and asymmetric encryption, it might occasionally fail.
- Sometimes, it's necessary for a third party to confirm the dependability of public keys.
- Since so many people are engaged, the data transfer rate is slow.
- RSA cannot be used for public data encryption, such as electoral voting.
- Decryption requires intensive processing on the receiver's end.

Real World examples:

RSA algorithm is asymmetric cryptography algorithm as it operate on two different keys such as public key and private key. The public key is likely to everyone, and private key remains private. The public key includes two numbers, one of which is a multiplication of two large prime numbers.

The RSA algorithm is based on the complexity included in the factorization of large numbers. The RSA algorithm depends on the fact that there is no effective method to factor very large numbers. Therefore, it can deducing an RSA key would take a large amount of time and processing power.

In RSA encryption, a message is encrypted with a code is known as a public key, which does not required to be hidden. It is based on the mathematical features of the RSA algorithm, because a message has been encrypted with the public key, it can only be decrypted by another key, which is known as the private key. Therefore, a set of key, which are public and private keys, is needed to read such messages.

Hash Function: SHA512

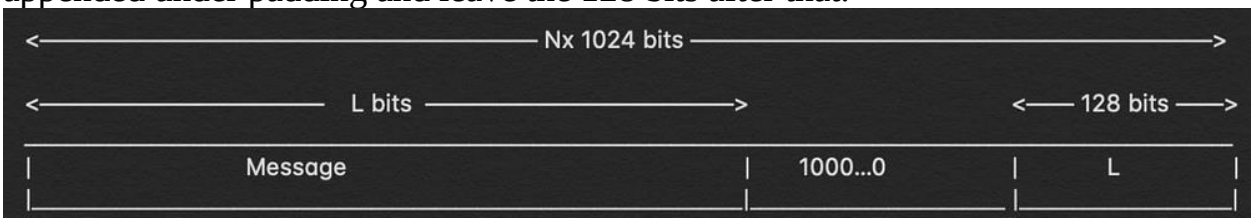
SHA-512 is a hashing algorithm that performs a hashing function on some data given to it. Hashing algorithms are used in many things such as internet security, digital certificates and even blockchains. Since hashing algorithms play such a vital role in digital security and cryptography, this is an easy-to-understand walkthrough, with some basic and simple maths along with some diagrams, for a hashing algorithm called SHA-512. It's part of a group of hashing algorithms called SHA-2 which includes SHA-256 as well which is used in the bitcoin blockchain for hashing.

SHA-512 WORKING:

1. Append : Padding bits

The first step is to carry out the padding function in which we append a certain number of bits to the plaintext message to increase its length, which should be exactly 128 bits less than an exact multiple of 1024.

When we are appending these bits at the end of the message we start with a '1' and then keep on adding '0' till the moment we reach the last bit that needs to be appended under padding and leave the 128 bits after that.



2. Append : Length bits

Now, we add the remaining 128 bits left to this entire block to make it an exact multiple of 1024, so that the whole thing can be broken down in ’n’ number of 1024 blocks of message that we will apply our operation on. The way to calculate the rest of the 128 bits is by calculating the modulo with 2^{64}.

Once done, we append it to the padded bit and the original message to make the entire length of the block to be “n x 1024” in length.

3. Initialize the buffers

Now, that we have “n x 1024” length bit message that we need to hash, let us focus on the parts of the hashing function itself. To carry on the hash and the computations required, we need to have some default values initialized.

```
a = 0x6a09e667f3bcc908
b = 0xbb67ae8584caa73b
c = 0x3c6ef372fe94f82b
d = 0xa54ff53a5f1d36f1
e = 0x510e527fade682d1
f = 0x9b05688c2b3e6c1f
g = 0x1f83d9abfb41bd6b
h = 0x5be0cd19137e2179
```

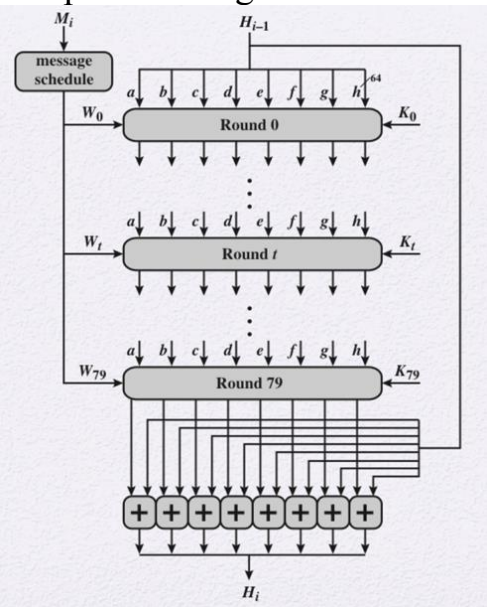
These are the values of the buffer that we will need, there are other default values that we need to initialize as well. These are the value for the ‘k’ variable which we will be using.

```
k[0..79] := [ 0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f, 0xe9b5dba58189dbbc, 0x3956c25bf348b538,
              0x59f111f1b605d019, 0x923f82a4af194f9b, 0xab1c5ed5da6d8118, 0xd807aa98a3030242, 0x12835b0145706fbe,
              0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2, 0x72be5d74f27b896f, 0x80deb1fe3b1696b1, 0x9bdc06a725c71235,
              0xc19bf174cf692694, 0xe49b69c19ef14ad2, 0xefbe4786384f25e3, 0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65,
              0x2de92c6f592b0275, 0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4, 0x76f988da831153b5, 0x983e5152ee66dfab,
              0xa831c66d2db43210, 0xb00327c898fb213f, 0xbf597fc7beef0ee4, 0xc6e00bf33da88fc2, 0xd5a79147930aa725,
              0x06ca6351e003826f, 0x142929670a0e6e70, 0x27b70a8546d22ffc, 0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed,
              0x53380d139d95b3df, 0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edaee6, 0x92722c851482353b,
              0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791, 0xc76c51a30654be30, 0xd192e819d6ef5218,
              0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8, 0x19a4c116b8d2d0c8, 0x1e376c085141ab53,
              0x2748774cdf8eeb99, 0x34b0bcb5e19b48a8, 0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373,
              0x682e6ff3d6b2b8a3, 0x748f82ee5defb2fc, 0x78a5636f43172f60, 0x84c87814a1f0ab72, 0x8cc702081a6439ec,
              0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915, 0xc67178f2e372532b, 0xca273eceea26619c,
              0xd186b8c721c0c207, 0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178, 0x06f067aa72176fba, 0x0a637dc5a2c898a6,
              0x113f9804bef90dae, 0x1b710b35131c471b, 0x28db77f523047d84, 0x32caab7b40c72493, 0x3c9ebe0a15c9bebc,
              0x431d67c49c100d4c, 0x4cc5d4becb3e42b6, 0x597f299cfc657e2a, 0x5fcb6fab3ad6faec, 0x6c44198c4a475817]
```

The reason for initiating these values will be clear to you in the very next step that we will explore.

4. Compression function

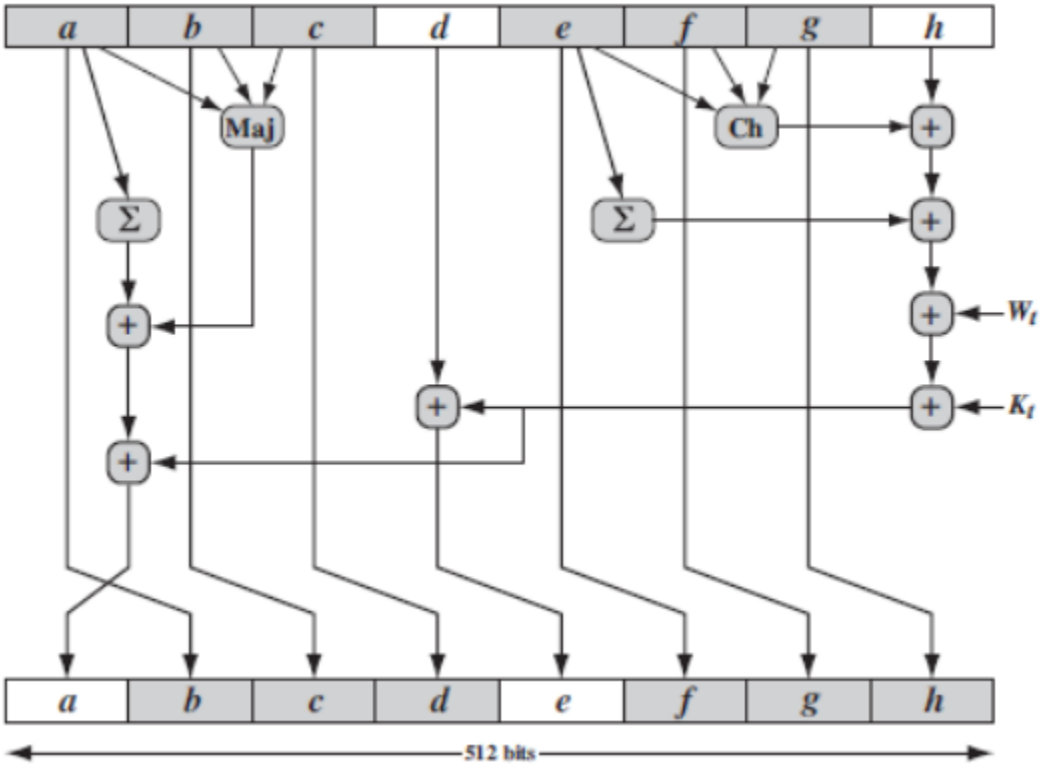
Now, we need to have a wider look at the hash function so that we can understand what is happening. First of all, we take 1024 bits of messages and divide the complete message in ‘n’ bits.



Having a look at the above image you get the idea that we will be working on 80 rounds and the input, W will be derived from the 1024 bits, which is further divided into 16 parts. The value of W from 0 to 15 is the message in plaintext but the values of W from 16 to 79 is calculated from the previous 16 blocks and we have the formula mentioned below.

$\mathbf{W(t) = \sigma^1(W^{t-2}) + W^{t-7} + \sigma^0(W^{t-15}) + W^{t-16}}$
where,
$\mathbf{\sigma^0(x)}$ = $ROTR^1(x) \phi ROTR^8(x) \phi SHR^7(x)$
$\mathbf{\sigma^1(x)}$ = $ROTR^{19}(x) \phi ROTR^{61}(x) \phi SHR^6(x)$
$\mathbf{ROTR^n(x)}$ = Circular right rotation of 'x' by 'n' bits
$\mathbf{SHR^n(x)}$ = Circular right shift of 'x' by 'n' bits
ϕ = addition modulo 2^{64}

Now that we know the methodology to obtain the values of W for 0 to 79 and already have the values for K as well for all the rounds from 0 to 79 we can proceed ahead and see where and how we do put in these values for the computation of the hash.



Depiction of a "round.In the image above we can see exactly what happens in each round and now that we have the values and formulas for each of the functions carried out we can perform the entire hashing process.

Ch(E, F, G) = (E AND F) XOR ((NOT E) AND G)
Ma(A, B, C) = (A AND B) XOR (A AND C) XOR (B AND C)
∑(A) = (A >>> 2) XOR (A >>> 13) XOR (A >>> 22)

$\sum(E) = (E >>> 6) \text{ XOR } (E >>> 11) \text{ XOR } (E >>> 25)$
$+ \quad =$ addition modulo 2^{64}

These are the functions that are performed in each of the 80 rounds that are performed over and over for 'n' number of times.

Advantages of SHA-512:

The reason why SHA-512 is faster than SHA-256 on 64-bit machines is that has 37.5% less rounds per byte (80 rounds operating on 128 byte blocks) compared to SHA- 256 (64 rounds operating on 64 byte blocks), where the operations use 64-bit integer arithmetic.

Disadvantages of SHA-512:

- Hash is inefficient when there are many collisions.
- Hash collisions are practically not be avoided for large set of possible keys.
- Hash does not allow null values.
- Hash tables have a limited capacity and will eventually fill up.
- Hash tables can be complex to implement.

- Hash tables do not maintain the order of elements, which makes it difficult to retrieve elements in a specific order.

Real-World Example of SHA-512:

- Hash is used for cache mapping for fast access of the data.
- Hash can be used for password verification.
- Hash is used in cryptography as a message digest.

Aim and Abstract:

In modern days with lots of online transactions Credit cards frauds increasing. To prevent unauthorized access of credit cards we will be encrypting the credit card using RSA Algorithm and Steganography technique. This makes it difficult for any unauthorized person to know whether the object is a credit card or not. As we can see these days all the users are going to prefer the online transaction for online shopping, money transfer, and for many purposes which require money transaction. These days it is very often that users lose their wallet including ATM cards, which increases the chances of monetary loss. So, to avoid all these things we can do encryption of ATM cards.

Techniques Used & Experimental Setup:

Techniques and Modules:

Python: Entire encryption code is written using python language along with modules of python, numpy and matplotlib. Encryption is done by public key, which is generated by the python code, It also calculates the private key and shows it to the user. Encrypted card is stored in database of merchant, and it is decrypted with the help of private key while performing the transaction.

Steps Involved:

Step 1: Generate two large prime numbers p and q

Step 2: Calculate N= p*q and Euler Totient function = (p-1)*(q-1)

Step 3: Check the number is prime or not using Miller Rabin method,

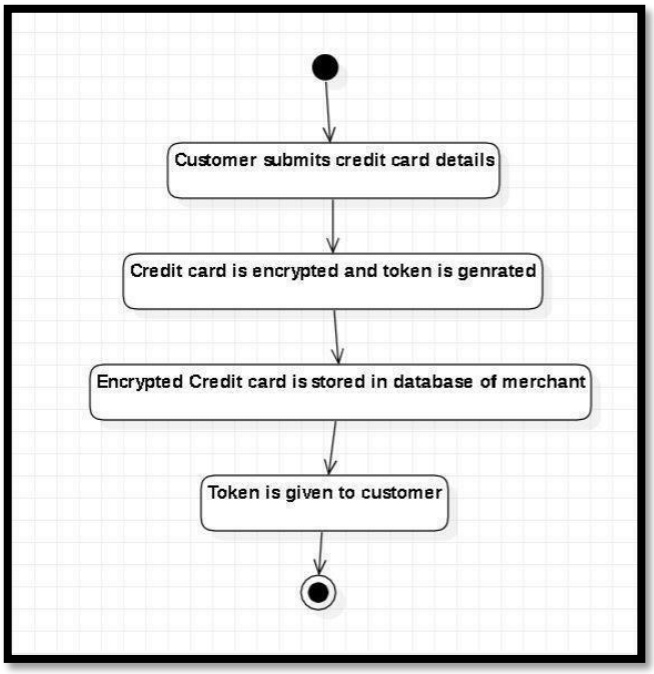
Step 4: Find E such that GCD(E,eulerTotient)=1(i.e., e should be coprime) such that it satisfies this condition

Step 5: Finding D: It must satisfies this property:- (D*E)Mod(eulerTotien t)=1

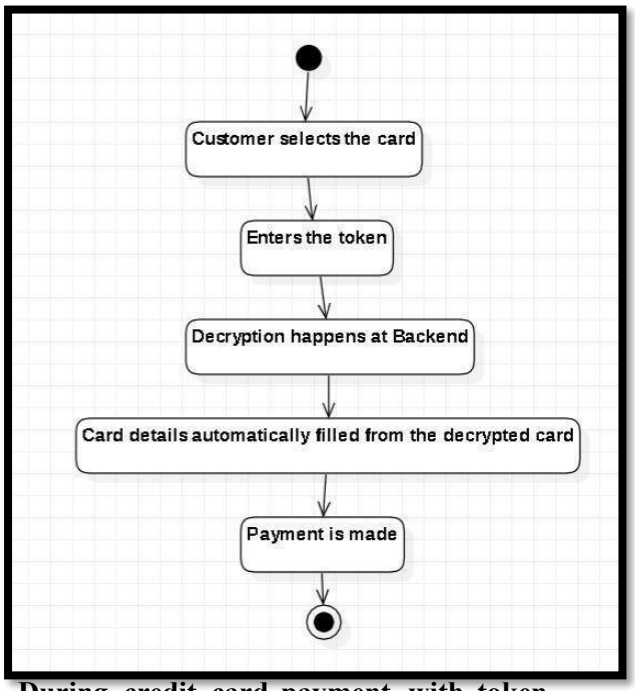
Step 6: Encryption

Step 7: Decryption

Flow of Events



One time token Generation



During credit card payment with token

IMPLEMENTATION:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
%matplotlib inline
my_img = cv2.imread('drive/My Drive/Colab Notebooks/ccard.jpg')
plt.imshow(my_img, cmap="gray")
print()
cv2_imshow(my_img)
print()
from random import randrange, getrandbits
def power(a,d,n):
    ans=1;
    while d!=0:
        if d%2==1:
            ans=((ans%n)*(a%n))%n
        a=((a%n)*(a%n))%n
        d>>=1
    return ans;
def MillerRabin(N,d):
    a = randrange(2, N - 1)
    x=power(a,d,N);
    if x==1 or x==N-1:
```

```
        return True;
    else:
        while(d!=N-1):
            x=((x%N)*(x%N))%N;
            if x==1:
                return False;
            if x==N-1:
                return True;
            d<<=1;
        return False;
def check_prime(N,K):
    if N==3 or N==2:
        return True;
    if N<=1 or N%2==0:
        return False;
    d=N-1
    while d%2!=0:
        d/=2;
    for _ in range(K):
        if not MillerRabin(N,d):
            return False;
    return True;
def generate_prime_candidate(length):
    p = getrandbits(length)
    p |= (1 << length - 1) | 1
```

```
    return p
def generatePrimeNumber(length):
    A=4
    while not check_prime(A, 128):
        A = generate_prime_candidate(length)
    return A
length=5
P=generatePrimeNumber(length)
Q=generatePrimeNumber(length)
print("Randomly generated Prime Numbers")
print(P," and ",Q)
N=P*Q
eulerTotient=(P-1)*(Q-1)
def GCD(a,b):
    if a==0:
        return b;
    return GCD(b%a,a)
E=generatePrimeNumber(4)
while GCD(E,eulerTotient)!=1:
    E=generatePrimeNumber(4)
print("Public key: ",N," , ",E)
print("Euler's totient value: ",eulerTotient)
def gcdExtended(E,eulerTotient):
    a1,a2,b1,b2,d1,d2=1,0,0,1,eulerTotient,E
    while d2!=1:
```

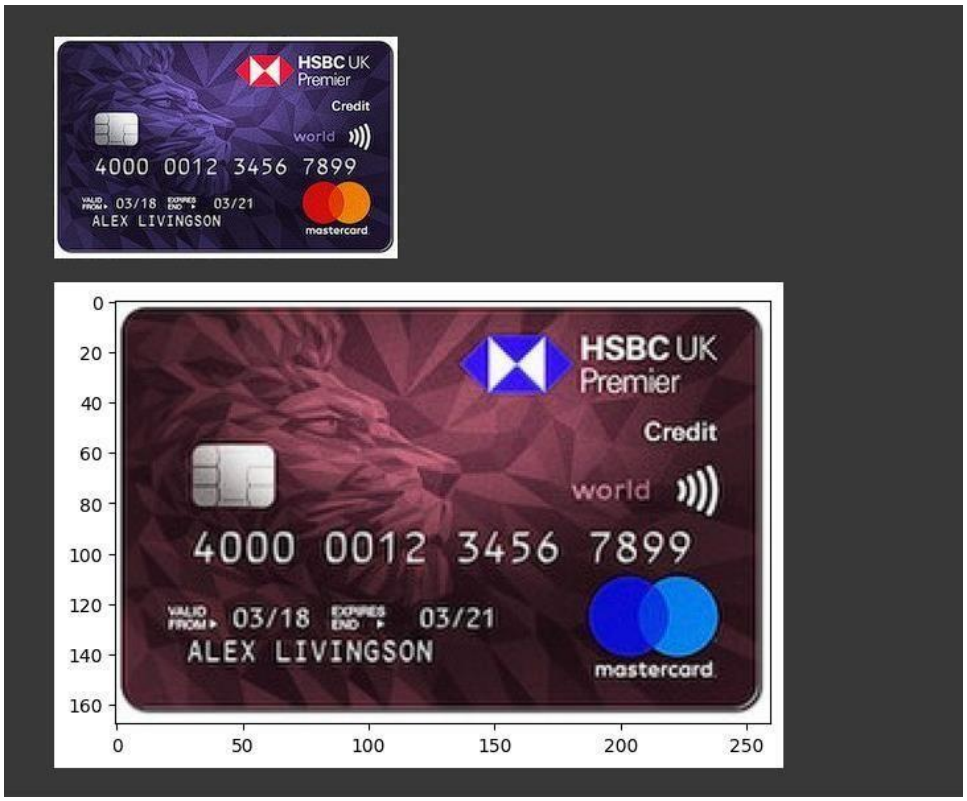
```
        k=(d1//d2)
        temp=a2
        a2=a1-(a2*k)
        a1=temp
        temp=b2
        b2=b1-(b2*k)
        b1=temp
        temp=d2
        d2=d1-(d2*k)
        d1=temp
        D=b2
    if D>eulerTotient:
        D=D%eulerTotient
    elif D<0:
        D=D+eulerTotient
    return D
D=gcdExtended(E,eulerTotient)
print("Private key/token",D)
row,col=my_img.shape[0],my_img.shape[1]
enc = [[0 for x in range(3000)] for y in range(3000)]
for i in range(0,160):
    for j in range(0,255):
        r,g,b=my_img[i,j]
        C1=power(r,E,N)
        C2=power(g,E,N)
```

```
        C3=power(b,E,N)
        enc[i][j]=[C1,C2,C3]
        C1=C1%256
        C2=C2%256
        C3=C3%256
        my_img[i,j]=[C1,C2,C3]
print("Encrypted card")
print()
cv2_imshow(my_img)
for i in range(0,160):
    for j in range(0,255):
        r,g,b=enc[i][j]
        M1=power(r,D,N)
        M2=power(g,D,N)
        M3=power(b,D,N)
        my_img[i,j]=[M1,M2,M3]
print()
print("Decrypted card")
print()
cv2_imshow(my_img)
```

Experimental Outcome:

```
Randomly generated Prime Numbers
19  and  23
Public key:  437  ,  13
Euler's totient value:  396
Private key/token 61
Encrypted card
```

Decrypted card


HSBC UK
Premier
Credit
world
4000 0012 3456 7899
VALID FROM 03/18 EXPIRES END 03/21
ALEX LIVINGSON
mastercard
HSBC UK
Premier
Credit
world
4000 0012 3456 7899
VALID FROM 03/18 EXPIRES END 03/21
ALEX LIVINGSON
mastercard
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250