

DATA ENCRYPTION OF FINANCIAL DETAILS IN ONLINE TRANSACTIONS

Team Members:

20BCI0153(Niravra Baksi)

20BCT0312(Reet Bhatia)

20BCE0832 (Jatin Dhall)

20BCE2156(Adnan Alam)

20BCI0207(Lokhandwala Mohammed Farhan)

REPORT FOR REVIEW 3

Course Code: CSE4003 –Cyber Security

Slot: E1 + TE1/ E2 + TE2

Professor: Dr. Ilanthenral Kandasamy

Abstract:

The world has seen a huge growth in online transactions (In comparison with 2019, online transactions grew by 80 per cent in 2020) these statistics are only continuing to increase. This, in turn, has increased the use of both credit and debit cards which are becoming the most popular mode of payment for purchases. Naturally, the cases of fraud in such transactions are also increasing. Security of online transactions is a challenging task that includes many areas that need to be taken care of such as securing communication channel and strong data encryption techniques. Hence, we will be using the concepts of Cryptography to make such transactions and the data transferred involved with this, as secure as possible. In this project, we will be proposing a modified way of encrypting the data by randomising the encryption algorithms. This encryption

system will be demonstrated using 5 algorithms mainly, RSA, Cast-128, BlowFish, AES and Triple DES. For each transaction that takes place, one of these algorithms is going to be randomly selected for encryption, and the unique key associated with the algorithm, as well as the key generated for encryption will be sent along with the important data that is needed to be transferred. This will allow the decryption of data with ease, whilst making it extremely difficult for the attacker to figure out which algorithm is being used to encrypt the data therefore, decreasing the chances of fraud.

Keywords: Online Transactions, Randomized, Multiple Algorithms, Attacks

Introduction:

With the Contactless process of online transactions, there comes a huge list of security threats which need to be taken care. If the Personal information of a user is compromised it could be used for spamming, identity theft and unsolicited marketing. So, it is very important that the information is kept secure.

Credit cards hold a lot of essential information such as your name, credit card number, CVV etc. which in turn can provide access to all your bank account information and personal details to the attacker and you may get into a lot of trouble.



So, it is very important that all the information is passed safely.

Our Basic aim is to provide a fully secure system to transfer the information such that the information can't be accessed by any third-party user.

The Algorithms that we are going to study are as follows:

- RSA
- CAST-128
- BLOWFISH
- AES

- TDES

Literature Review Summary Table:

Authors and Year (Reference)	Title (Study)	Concept / Theoretical model/ Framework	Methodology used/ Implementation	Relevant Finding	Limitations/ Future Research/ Gaps identified
TONY CUADRA, IIYAS ELKIN [1]	Secure RSA Credit Card Transaction System.	RSA ALGORITHM	two Atmel AT90S4414 microcontroller which takes the account number and the transaction amount from the store clerk through the keypad, displaying them on the LCD as they are being typed in. It then requests a key pair from the server using software UART.	HOW RSA ALGORITHM IS USED FOR ENCRYPTI ON OF CREDIT CARD DETAILS	DUE TO THE PRESENCE OF HIGH COMPUTATIO NAL POWER TODAY, HACKERS COULD EASILY BREAK THE ENCRYPTION IF THE KNOW THE TYPE OF ALGORITHM IS USED

S. AISHK.DEVI KA, RANI DHIVYAWA RYA, [2]	Online Payment Fraud Prevention Using Cryptographic Algorithm TDES	TRIPLE DES ALGORITHM	whenever a new user registers, the credit card and debit card details are cross checked and then only the user id is generated. This allows only correct users to login each time. At the same time credit card and debit card details are verified each time whenever a customer buys product	HOW TRIPLE DES ALGORITHM IS IMPLEMENTED FOR ENCRYPTION	THOUGH TRIPLE DES ALGORITHM IS VERY SECURE YET THE ENCRYPTION PROCESS IS VERY LENGTHY AND TIME TAKING AS WE APPLY THE ENCRYPTION ON THE SAME DATA THRICE
GABRIEL IWAUSKON [3]	Encryption and Tokenization based credit card security system	RSA algorithm and Tokenization	The implementation of the RSA encryption and tokenization-based platform for securing credit card information was carried out on Microsoft Windows 7 Operating System environment on Pentium IV with 2.0 GHZ Duo Core Processor and 2 GB of RAM. APACHE server and HTML with CSS, JavaScript served as the frontend while MySQL database from WAMP server and PHP were the backend on Mozilla Firefox browser	HOW TOKENIZATION AND RSA ALGORITHM IS USED FOR ENCRYPTION	Using the tokenized data requires it be detokenized and retrieved from a remote service. This introduces a small increase in transaction time and while using it the user may sometime get locked in

HIMANSHU SHARMA [4]	ROLE OF MULTIPLE ENCRYPTION IN SECURE ELECTRONIC TRANSMISSION	RSA AND DOUBLE DES ALGORITHMS	1.It takes original confidential information as plaintext. 2.Employ Simple Hash Algorithm, result comes out as Message Digest. 3.Encrypt Message Digest multiple times with different encryption keys to generate more advance and complex Digital Signature. 4.Transmit the data with newly generated Digital Signature to the receiver side.	HOW MORE THAN ONE ALGORITHMS CAN BE USED TO ENCRYPT THE DATA	DUE TO THE PRESENCE OF MULTIPLE LAYERS OF ENCRYPTION, THE ENCRYPTION PROCESS IS VERY LENGTHY
------------------------	---	-------------------------------	---	--	--

Objective of the project:

We are aware that there are various strict securities present for the smooth and secure transactions, but if the algorithm that is going to be used for the encryption is known by the third party, then he/she might be able to easily find different ways to crack the system easily.

The objective of our project is that we are providing a fully secure system to transfer the information (CVV+Card Number), from receiver end to the sender end and vice versa, in such a way that the information can't be accessed by any third-party user and used for fraud.

The unique feature is that we are using randomization to select which algorithm to use for encryption, the system will randomly select which algorithm to use, by doing so we will make sure that no one is able to know which algorithm is being used for the current encryption.

Also, the system is going to randomly generate a key in a very particular format. A key in cryptography is basically a string of numbers or letters that are stored in a file which when processed through an algorithm, can encode or decode cryptographic data. The key generation will be done from the receiver side. As RSA is asymmetric and CAST-128, BLOWFISH, AES and TDES are symmetric, we have done key generation separately for symmetric and asymmetric algorithms,

- ASYMMETRIC- After the calculation of n,e in RSA, the key will be printed in the format [algonumber + nsize + n + esize + e]
- SYMMETRIC- The format for Symmetric key is : [algonumber + key]

Hence in this way while we are encrypting the information, we are also keeping the algorithm used, unknown.

Innovative component of the project:

Our project differs from other projects because we are using randomization to select one algorithm number out of the five. For each of the transactions that will take place, one of the five algorithms that we are using will be selected making it difficult for a third party to figure out which algorithm will be used hence decreasing the chances of fraud. So basically we are generating a random number by using the rand() function and its modulus 5 value is taken for the generation of the algorithm number that has to be used for encryption.

Work done and implementation:

a. Methodology adapted-

1. Credit card details are taken as input from the user. (CVV + CARD NUMBER)
2. One out of the five encryption algorithms is selected at random.
3. The communication starts from the receiver end. The receiver end performs randomization + key generation.
4. This is sent to the Sender end. The sender end encrypts the text using this information.
5. The Sender End sends the encrypted text to the receiver end.
6. Finally, the Receiver end decrypts it to obtain the plaintext.

b. Dataset used-

We are not using any specific dataset, we are just going to input a random card number and cvv number to show the implementation of the project.

c. Tools used-

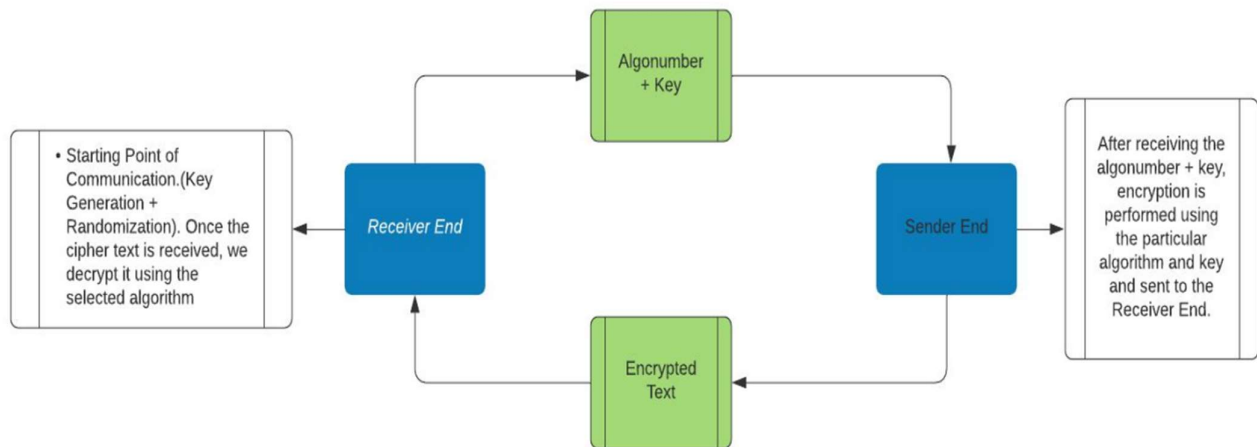
Libraries and Packages Used

1. Pycryptodome Python package
2. Blowfish Python package
Both Installed via pip
3. OS Library
4. base64 Library
5. csv Library
6. binascii Library
7. sys Library

Code written in :

Python(Vs Code)

FLOWCHART-



ENCRYPTION ALGORITHMS USED:

1. RSA:

- (i) Generation of public key:
Select 2 **prime** numbers **P** and **Q**.
First half of key= P*Q
Select a small exponent **e** which is an **integer** and its value lies between **1** and $\phi(n)$. $(\phi(n) = ((P-1)*(Q-1)))$.

Second half of key= e

- (ii) Generation of private key: Private Key= $d = (k * \phi(n) + 1) / e$
where k is some integer.

2. BLOWFISH:

- (i) Generating subkeys: Both encryption and decryption processes require **18 subkeys** which are stored in an array called P. P is initialised with digits of pi.

**32-bit hexadecimal representation of
initial values of sub-keys**

P[0] : 243f6a88	P[9] : 38d01377
P[1] : 85a308d3	P[10] : be5466cf
P[2] : 13198a2e	P[11] : 34e90c6c
P[3] : 03707344	P[12] : c0ac29b7
P[4] : a4093822	P[13] : c97c50dd
P[5] : 299f31d0	P[14] : 3f84d5b5
P[6] : 082efa98	P[15] : b5470917
P[7] : ec4e6c89	P[16] : 9216d5d9
P[8] : 452821e6	P[17] : 8979fb1b

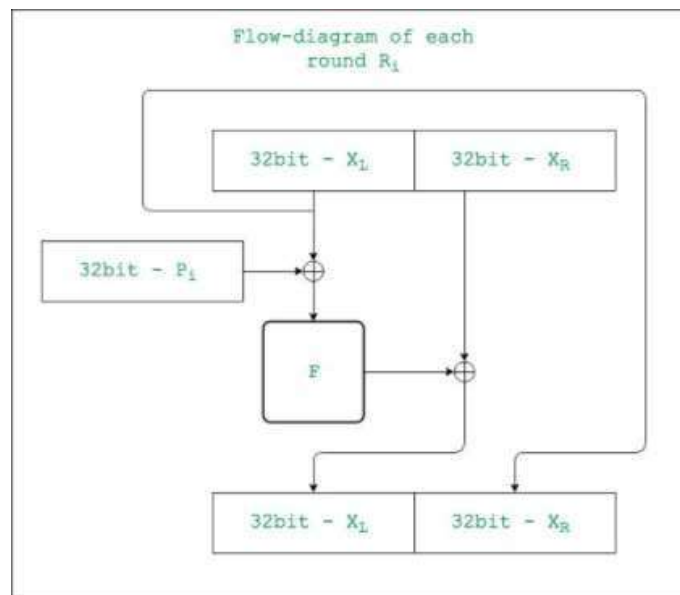
Each subkey changes with respect to the input key and this change follows the formula:

$P[i] = P[i] \text{ xor } (i+1)\text{th 32-bits of input key.}$

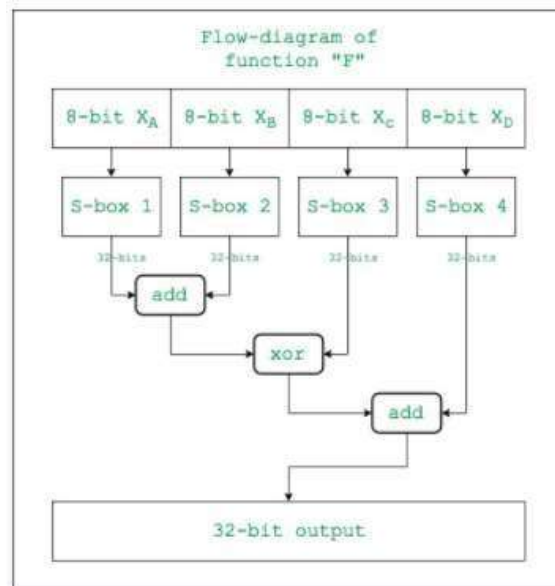
Where $i=\{0,1,2,3,\dots,17\}$

The resultant array of subkeys is used for encryption and decryption.

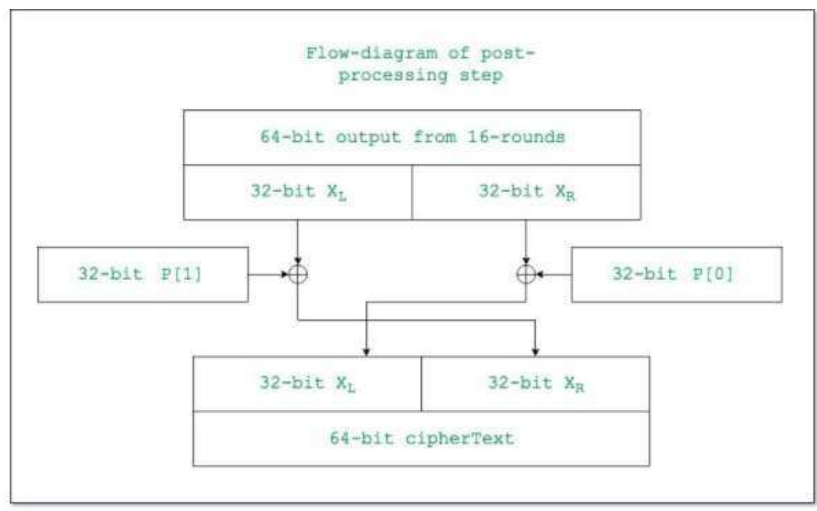
- (ii) 4 S-boxes each with 256 32-bit entries are initialized with the P array.
- (iii) Encryption Occurs in 2 steps.
First we have rounds where each round takes plaintext from previous round and corresponding subkey as input.



Where **function F** is as follows:



Then we go to the post processing step which is as follows:



- (iv) Decryption follows a similar method, the only major difference is that the P array is used in reverse order. [5]

3. CAST-128

It is a 12 or 16-round Feistel network with a 64-bit block size and a key size of between 40 to 128 bits (but only in 8-bit increments).

The full 16 rounds are used when the key size is longer than 80 bits. Components include large 8×32 -bit S-boxes based on bent functions, key-dependent rotations, modular addition and subtraction, and XOR operations. There are three alternating types of round function, but they are similar in structure and differ only in the choice of the exact operation (addition, subtraction or XOR) at various points. [6]

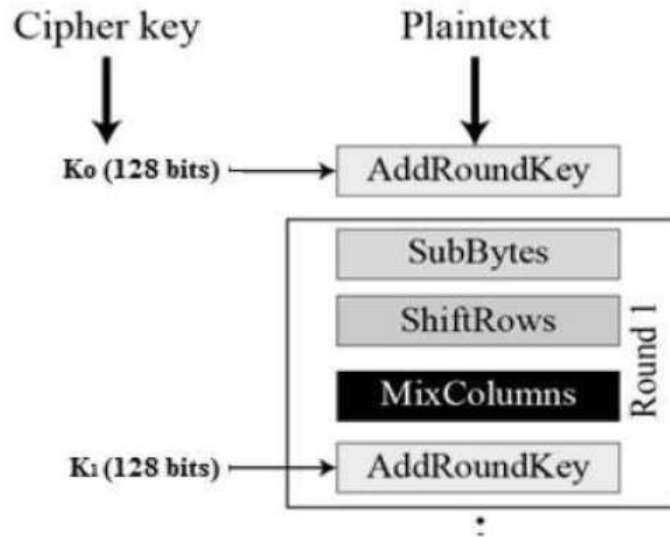
4. AES

Encryption involves 4 steps which are carried out in each round.

- (i) **Byte Substitution:** 16 input bytes are substituted using an SBOX resulting in a **matrix of 4 rows and columns**.
- (ii) **Shiftrows:** All the **rows** of the matrix are **shifted to the left** and the entries that fall off are added to the rightmost side. Each **row is shifted by n-1 bytes** where n is the number of the row, **first row is not shifted** as n-1 becomes 0.
- (iii) **MixColumns:** Each column of the matrix is transformed using a special function which takes the 4 bytes of a column as input and **outputs 4 new bytes**.

- (iv) **Addroundkey:** The 16 bytes are considered as **128 bits** and **XORed with the 128 bit round key**, if in last round the output is considered the Cipher Text.

The Decryption process is similar , the only difference is that the steps are done in reverse order. [7]

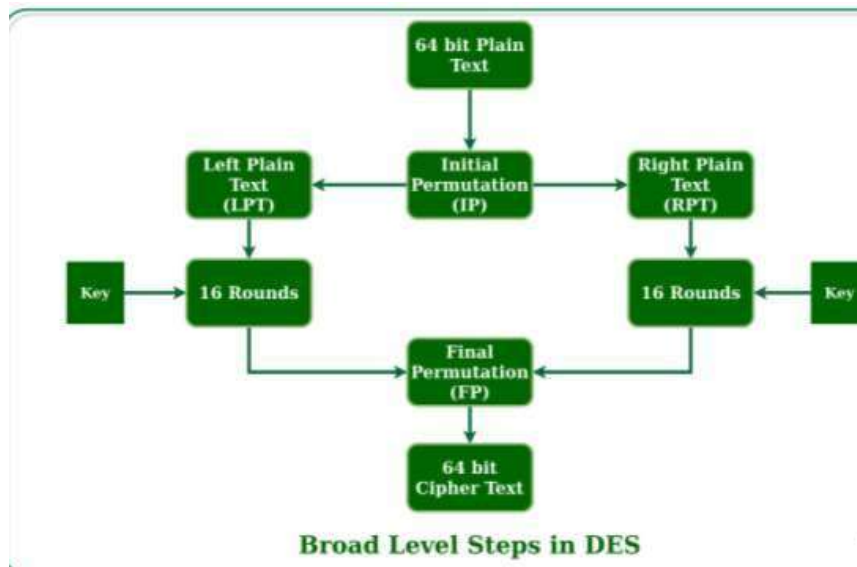


5. TDES:

TDES uses **3 instances of DES** on the same plaintext. **Each instance** use a **different key selection method**. All are different in first instance, 2 keys are same in the second instance and all are same in the third instance.

DES is carried out as follows:

- (i) The 64 bit plaintext is handed to an **initial permutation(IP)** function which performs initial permutation on it.
- (ii) The IP function produces 2 halves of the permuted block as **Left Plain Text (LPT)** and **Right Plain Text(RPT)**.
- (iii) LPT and RPT go through **16 rounds of encryption** process
- (iv) Finally LPT and RPT are **combined** and a **final permutation** is performed to give a 64 bit cyphertext.



Screenshot and demo along with visualization:

RECEIVER SIDE CODE-

```
#Receiver Side Code
import random
import sys
sys.path.append("c:\\users\\jatin dhall\\anaconda3\\lib\\site-packages")
from Crypto.Cipher import AES
from Crypto.Cipher import DES3
from Crypto.Cipher import CAST
import blowfish
from os import urandom
from Crypto.Util.Padding import unpad
from base64 import b64decode
from base64 import b64encode
from Crypto import Random
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad
from binascii import b2a_hex
import csv

#Assigning Algorithms Numbers :-
#0 -> RSA
#1 -> AES
#2 -> TDES
#3 -> BLOWFISH
#4 -> CAST
```

```

def RSAdecryption(n,d,a):
    M=[]
    for i in a:
        M.append(compute(int(i),d,n))
    print("Decrypted text is :-")
    plaintext = ""
    for i in M:
        plaintext += str(i)
    print("CVV : ",plaintext[0:3])
    print("CARD NO. : ",plaintext[3:])

```

```

def compute(a,m,n):
    y=1
    while(m>0):
        r=m%2
        if(r==1):
            y=(y*a)%n
        a = a*a % n
        m = m // 2
    return y

```

```

def CASTdecryption(key,ciphertext,eiv):
    ciphertext = b64decode(ciphertext)
    eiv = b64decode(eiv)

    print("Decoded ciphertext : ",ciphertext,len(ciphertext))
    print("Decoded iv : ",eiv,len(eiv))
    cipher = CAST.new(key, CAST.MODE_OPENPGP, eiv)
    decryptedText = cipher.decrypt(ciphertext)

    print("Decrypted data is :- ")
    print("CVV : ",decryptedText[0:3])
    print("CARD NO. : ",decryptedText[3:])

```

```

def AESdecryption(key,ciphertext,iv):
    ciphertext = b64decode(ciphertext)
    iv = b64decode(iv)

    print("Decoded ciphertext : ",ciphertext,len(ciphertext))
    print("Decoded iv : ",iv,len(iv))
    # To decrypt, use key and iv to generate a new AES object
    mydecrypt = AES.new(key, AES.MODE_CBC, iv)

    # Use the newly generated AES object to decrypt the encrypted ciphertext
    decrypttext = unpad(mydecrypt.decrypt(ciphertext), AES.block_size)
    decrypttext = decrypttext.decode('utf-8')
    #decrypttext = mydecrypt.decrypt(ciphertext)
    print("The decrypted data is:-")
    print("CVV : ",decrypttext[0:3])
    print("CARD NO. : ",decrypttext[3:])

def blowfishdecrypt(key,ciphertext,iv):
    ciphertext = b64decode(ciphertext)
    iv = b64decode(iv)

    print("Decoded ciphertext : ",ciphertext,len(ciphertext))
    print("Decoded iv : ",iv,len(iv))
    cipher = blowfish.Cipher(key)
    data_decrypted = b"".join(cipher.decrypt_cbc(ciphertext, iv))
    decrypttext=data_decrypted.decode('utf-8')
    print("The decrypted data is:-")
    print("CVV : ",decrypttext[0:3])
    print("CARD NO. : ",decrypttext[3:])

```

```

def tripledesdecrypt(ciphertext,bkey):
    key = pad(bkey, 24)
    tdes_key = DES3.adjust_key_parity(key)
    cipher = DES3.new(tdes_key, DES3.MODE_EAX, nonce=b'\0')
    ciphertext = b64decode(ciphertext)
    plaintext=cipher.decrypt(ciphertext)
    decrypttext = plaintext.decode('utf-8')
    print("The decrypted data is:-")
    print("CVV : ",decrypttext[0:3])
    print("CARD NO. : ",decrypttext[3:])

def assymetricfinalkey(res):
    n = str(res[0])
    e = str(res[1])
    nsize = len(n)
    esize = len(e)
    print("ASSYMETRIC : n e nsize esize",n,e,nsize,esize)
    key = str(nsize) + n + str(esize) + e
    print("n = ",n)
    print("e = ",e)
    print("Key Format : algonumber + nsize + n + esize + e")
    return key

```

```

def randomize():
    rand=random.randint(40,4000)
    return (rand%5)

def check_prime(n):
    for i in range(2,n):
        if(n%i==0):
            return 1
    return 0

def gcd(a,b):
    while(1):
        temp=a%b
        if temp==0:
            return b
        a=b
        b=temp

```

```

def asymmetric():
    while(5):
        p=random.randint(1,1000)
        while(check_prime(p)==1):
            p=p+1
        break
    while(5):
        q=random.randint(1,1000)
        while(check_prime(q)==1):
            q=q+1
        break
    n=p*q
    phi_n=(p-1)*(q-1)
    while(5):
        e=random.randint(2,n-1)
        if(gcd(e,phi_n)==1):
            break
    k=0
    while((1+(k*phi_n))%e!=0):
        k+=1
    d=(1+(k*phi_n))//e

    res = [n,e,d]
    print("p = ",p)
    print("q = ",q)
    return res
    # print("n is",n)
    # print("e is",e)
    # print("d is",d)

```



```

algonumber = randomize()
if algonumber == 0: #Meaning the algorithm is RSA(Asymmetric)
    key = str(algonumber)
    res = asymmetric()
    key += asymmetricfinalkey(res)
    d = res[2]
    n = res[0]
    keys = [key]
    print("Asymmetric Key : ",key)
    print("Algo Number : ",algonumber)

    with open('key.csv', 'w', encoding='UTF8') as f:
        writer = csv.writer(f)

        # write the data(cipher,iv)
        writer.writerow(keys)

    f.close()

print("DECRYPTION OF RSA")
input("Please Enter when ciphertext generated : ")
with open("C:\\Users\\Jatin Dhall\\Desktop\\Desktop File\\VIT\\VIT\\SEM 3\\Cyber Security\\PROJECT\\PROJECT\\Sender\\ciphertext.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        if(len(row) == 0):
            break
        ciphertext = row
    csv_file.close()
RSAdecryption(int(n),int(d),ciphertext)

```

```

else:
    bkey = get_random_bytes(16)
    key = str(algonumber)
    bkey1 = b64encode(bkey).decode('utf-8')
    key += bkey1

    keys = [key]
    with open('key.csv', 'w', encoding='UTF8') as f:
        writer = csv.writer(f)

        # write the data(cipher,iv)
        writer.writerow(keys)

    f.close()
    print("Symmetric Key : ",key)
    print("AlgoNumber : ",algonumber)

    input("Please Enter when ciphertext generated : ")
    with open("C:\\Users\\Jatin Dhall\\Desktop\\Desktop File\\VIT\\VIT\\SEM 3\\Cyber Security\\PROJECT\\PROJECT\\Sender\\ciphertext.csv") as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            if(len(row) == 0):
                break
            ciphertext = row[0]
            if algonumber!=2:
                iv = row[1]
        csv_file.close()

    if algonumber == 1: #AES
        AESdecryption(bkey,ciphertext,iv)
    elif algonumber == 2: #TDES
        triplledesdecrypt(ciphertext,bkey)
    elif algonumber == 3: #Blowfish
        blowfishdecrypt(bkey,ciphertext,iv)
    elif algonumber == 4: #CAST
        CASTdecryption(bkey,ciphertext,iv)

```

SENDER SIDE CODE-


```

#Sender Side Code
import sys
sys.path.append("c:\\users\\jatin dhall\\anaconda3\\lib\\site-packages")
from Crypto.Cipher import AES
from Crypto.Cipher import DES3
from Crypto.Cipher import CAST
import blowfish
from os import urandom
from base64 import b64encode
from base64 import b64decode
from Crypto import Random
from Crypto.Util.Padding import pad
from binascii import b2a_hex
import csv

#Assigning Algorithms Numbers :-
#0 -> RSA
#1 -> AES
#2 -> TDES
#3 -> BLOWFISH
#4 -> CAST

```

```

def BlowfishEncrypt(key):
    cipher = blowfish.Cipher(key)
    iv = urandom(8) # initialization vector
    cvv = input("Enter the CVV : ")
    cardno = input("Enter the cardnumber : ")
    data = cvv + cardno
    while((len(data)%8)!=0):
        data=data+" "
    res = data.encode('utf-8')
    print("Data to encrypt",data)
    ciphertext = b"".join(cipher.encrypt_cbc(res, iv))
    print("encrypted data",ciphertext)
    cipher = [b64encode(ciphertext).decode('utf-8'),b64encode(iv).decode('utf-8')]

    with open('ciphertext.csv', 'w', encoding='UTF8') as f:
        writer = csv.writer(f)

        # write the data(cipher,iv)
        writer.writerow(cipher)

    f.close()

def CASTEncrypt(key):
    cipher = CAST.new(key, CAST.MODE_OPENPGP)
    cvv = input("Enter the CVV : ")
    cardno = input("Enter the cardnumber : ")
    plaintext = cvv + cardno
    plaintext = plaintext.encode()
    msg = cipher.encrypt(plaintext)
    eiv = msg[:CAST.block_size+2]
    ciphertext = msg[CAST.block_size+2:]
    print("The encrypted data is:", b64encode(ciphertext).decode('utf-8'))
    print("The iv is:", b64encode(eiv).decode('utf-8'))
    cipher = [b64encode(ciphertext).decode('utf-8'),b64encode(eiv).decode('utf-8')]

    with open('ciphertext.csv', 'w', encoding='UTF8') as f:
        writer = csv.writer(f)

        # write the data(cipher,iv)
        writer.writerow(cipher)

    f.close()

```

```

def AESEncrypt(key):
    cvv = input("Enter the CVV : ")
    cardno = input("Enter the cardnumber : ")
    plain_text = cvv + cardno
    plain_text = plain_text.encode()
    # Generate a non-repeatable key vector with a length
    # equal to the size of the AES block
    iv = Random.new().read(AES.block_size)
    # Use key and iv to initialize AES object, use MODE_CBC mode
    mycipher = AES.new(key, AES.MODE_CBC, iv)

    ciphertext = mycipher.encrypt(pad(plain_text,AES.block_size))
    print("The encrypted data is:", b64encode(ciphertext).decode('utf-8'))
    print("The iv is:", b64encode(iv).decode('utf-8'))
    cipher = [b64encode(ciphertext).decode('utf-8'),b64encode(iv).decode('utf-8')]

    with open('ciphertext.csv', 'w', encoding='UTF8') as f:
        writer = csv.writer(f)

        # write the data(cipher,iv)
        writer.writerow(cipher)

    f.close()

def tripledesencrypt(bkey):
    cvv = input("Enter the CVV : ")
    cardno = input("Enter the cardnumber : ")
    msg = cvv + cardno
    key = pad(bkey,24)
    tdes_key = DES3.adjust_key_parity(key)
    cipher = DES3.new(tdes_key,DES3.MODE_EAX, nonce=b'0')
    ciphertext = cipher.encrypt(msg.encode('utf-8'))
    print("Encrypted text :-", ciphertext)
    cipher = [b64encode(ciphertext).decode('utf-8')]
    with open('ciphertext.csv', 'w', encoding='UTF8') as f:
        writer = csv.writer(f)
        writer.writerow(cipher)
    f.close()

```

```

def compute(a,m,n):
    y=1
    while(m>0):
        r=m%2
        if(r==1):
            y=(y*a)%n
            a = a*a % n
            m = m // 2
    return y

def RSAEncryption(n,e):
    cvv = input("Enter the CVV : ")
    cardno = input("Enter the cardnumber : ")
    M = int((cvv + cardno))
    d1=M
    if M < n:
        C=compute(M,e,n)
        print("Encrypted text is:",C)
    else:
        f=0
        a=[]
        C=[]
        while d1>0:
            mod=d1%100
            a.insert(f,mod)
            f=f+1
            d1= d1//100
        a.reverse()
        for i in a:
            C.append(compute(i,e,n))
        print("Encrypted text is")
        cipher = []
        for i in C:
            print(i)
            cipher.append(i)
        with open('ciphertext.csv', 'w', encoding='UTF8') as f:
            writer = csv.writer(f)

            # write the data(cipher)
            writer.writerow(cipher)

        f.close()

```

```

with open("C:\\Users\\Jatin Dhall\\Desktop\\Desktop File\\VIT\\VIT\\SEM 3\\Cyber Security\\PROJECT\\PROJECT\\Receiver\\key.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        if(len(row) == 0):
            break
        keyreceived = row[0]
    csv_file.close()

#keyreceived = input("Enter the key recieved by the reciever side : ")
input("Please press enter once the key is generated from the receiver side : ")
print(keyreceived)
print("Algo Number : ",keyreceived[0])

algonumber = keyreceived[0]
key = keyreceived[1:]
print(algonumber)
print(key)

if(algonumber == '0'):
    nsize = int(key[0])
    n = key[1:nsize+1]
    esize = int(key[nsize+1])
    e = key[nsize+2:]
    print("n = ", n)
    print("e = ", e)
    RSAEncryption(int(n),int(e))

elif(algonumber == '1'):
    key = b64decode(key)
    print("KEY :",key)
    AESEncrypt(key)

elif(algonumber == '2'):
    key = b64decode(key)
    print("KEY :",key)
    tripldesencrypt(key)

elif(algonumber == '3'):
    key = b64decode(key)
    print("KEY :",key)
    BlowfishEncrypt(key)

```

```

elif(algonumber == '4'):
    key = b64decode(key)
    print("KEY :",key)
    CASTEncrypt(key)

```

BRIEF EXPLANATION OF ALGORITHM CODES:

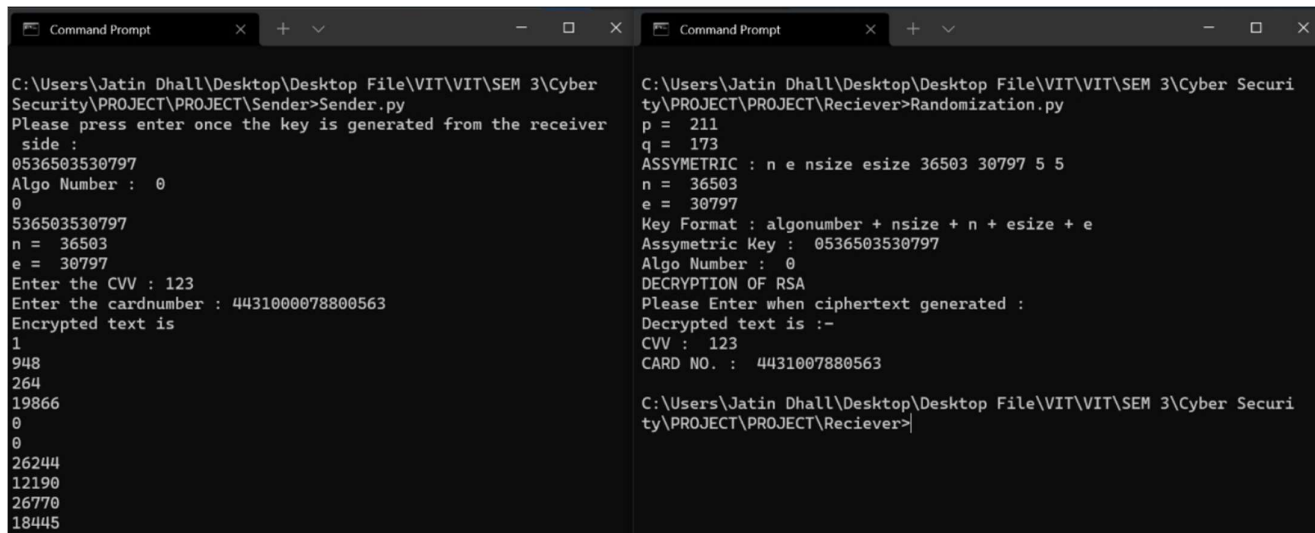
1. **RSA:** To implement RSA we first generate the prime numbers p and q randomly using the randint() and check whether they are prime using checkprime() function if the generated value of p and q are not prime their values are incremented by one and this continues until the value of generated p and q becomes prime. We find the value of n and phi_n from p and q. we randomly generate the value of e such that e is relatively prime with phi_n by checking it with the gcd() function which is declared previously. Then we calculate the value of d from e, then we combine the private keys e and n in an array and also append the algorithm number for the rsa algorithm and read it into a csv file and send to the sender end from the receiver end. For encryption we generate the cipher text using compute(M,e,n) where M is the plain text, compute(a,b,m) is a function used for calculating $a^b \text{ mod } m$ quickly using exponentiation. If the value of plain text is greater than n, we extract 2 numbers from the plain text block-wise and then encrypt them serially and finally append them into a list. The cipher text generated is read into the csv file and send to the receiver. The receiver receives the file from the sender decodes it into usable format and then the cipher text is divided into

blocks and decrypted block by block using the formula $\text{compute}(\text{int}(i), d, n)$ where i is each block. Then finally they are appended into a string and displayed

2. **AES:** To implement AES we first import AES and Random from Pycryptodome library. The function `symmetric()` generates a randomized 16 byte(128 bit) key and we also generate a 16 byte long initialization vector via randomization. The `new()` function is used to create the cipher object and mode is set to CBC(Cypher Block Chaining). `Encrypt()` function is used to carry out the encryption, we also use a padding function to ensure that all data is consistent with the block size, which is essential for the algorithm to work. `b64encode()` is used to to encode binary data that needs to be stored and transferred over media that are designed to deal with ASCII. For decryption we get the key, iv and ciphertext from the file and then we decrypt using `decrypt()` function. Padding is removed using `unpad()` and `decode()` function is used to bring everything back to the normal format.
3. **TDES:** To implement Triple DES ,at first we import DES3 from Pycryptodome library for the execution.To generate the keys ,we recieve the 16 bits key from the `symmetric()` function and pad it to make 24 bits then using `DES3.adjust_key_parity(key)` we adjust the key parity to be used for tdes. We create the cipher object using the `new()` function and declare the mode as EAX object and declare nonce =b'0'.The ciphertext is generated using the key and the tdes cipher EAX object and plain text.The cipher text generated is using `cipher.encrypt()` and encoded using the 'utf-8'and read into a csv file.To decrypt the function `tripledesdecrypt(ciphertext,bkey)` accepts the cipher text from the sender file and it is decoded back to a string using `b64decode()` and then the EAX cipher object is generated,and the final decrypted text is using `cipher.decrypt()`
4. **BLOWFISH:** We are using library blowfish to implement blowfish. We first import `urandom` which helps in generating a random number. The key is generated using `blowfish.Cipher()` function. Blowfish is implemented using CBC(Cipher Block Chaining) mode. 8 bit initialisation vector is randomly generated using `urandom` function. Padding is done to ensure that all data is consistent with the block size, which is essential for the algorithm to work. Then the data is converted to bytes using `encode()` function. The encryption is done using key and function `cipher.encrypt_cbc()` function. The encrypted text is decrypted using key and `cipher.decrypt_cbc()` , after decryption the data is converted to string using `.decode()` and the decrypted data is displayed.
5. **CAST128:** To implement CAST128 we first import CAST and Random from Pycryptodome library. The function `symmetric()` generates a randomized 16 byte(128 bit) key and we also generate a 16 byte long initialization vector via randomization. The `new()` function is used to create the cipher object and mode is set to OPENPGP. `Encrypt()` function is used to carry out the encryption, we also

use a padding function to ensure that all data is consistent with the block size, which is essential for the algorithm to work. `b64encode()` is used to to encode binary data that needs to be stored and transferred over media that are designed to deal with ASCII. For decryption we get the key, iv and ciphertext from the file and then we decrypt using `decrypt()` function. Padding is removed using `unpad()` and `decode()` function is used to bring everything back to the normal format.

Results and discussion along with visualization:



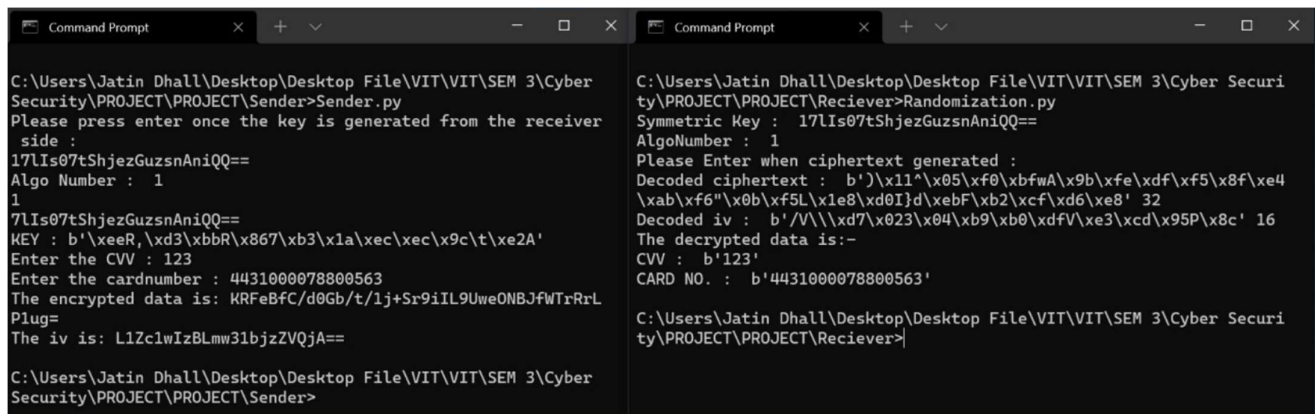
The screenshot shows two Command Prompt windows. The left window displays the output of the `Sender.py` script, which generates an asymmetric key and encrypts a card number. The right window displays the output of the `Reciever>Randomization.py` script, which decrypts the ciphertext and retrieves the original card number.

```
C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Sender>Sender.py
Please press enter once the key is generated from the receiver side :
0536503530797
Algo Number : 0
0
536503530797
n = 36503
e = 30797
Enter the CVV : 123
Enter the cardnumber : 4431000078800563
Encrypted text is
1
948
264
19866
0
0
26244
12190
26770
18445

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>Randomization.py
p = 211
q = 173
ASYMMETRIC : n e nsize esize 36503 30797 5 5
n = 36503
e = 30797
Key Format : algonumber + nsize + n + esize + e
Asymmetric Key : 0536503530797
Algo Number : 0
DECRYPTION OF RSA
Please Enter when ciphertext generated :
Decrypted text is :-
CVV : 123
CARD NO. : 443100078800563

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>
```

RSA OUTPUT



The screenshot shows two Command Prompt windows. The left window displays the output of the `Sender.py` script, which generates a symmetric key and encrypts a card number. The right window displays the output of the `Reciever>Randomization.py` script, which decrypts the ciphertext and retrieves the original card number.

```
C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Sender>Sender.py
Please press enter once the key is generated from the receiver side :
17LI07tShjezGuzsnAniQQ==
Algo Number : 1
1
7LI07tShjezGuzsnAniQQ==
KEY : b'\xeeR,\xd3\xbbR\x867\xb3\x1a\xec\xec\x9c\t\xe2A'
Enter the CVV : 123
Enter the cardnumber : 4431000078800563
The encrypted data is: KRFeBfC/d0Gb/t/lj+Sr9iIL9UweONBJfWTrRrL
Plug=
The iv is: L1Zc1wIzBLmw31bjzZVQjA==

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>Randomization.py
Symmetric Key : 17LI07tShjezGuzsnAniQQ==
AlgoNumber : 1
Please Enter when ciphertext generated :
Decoded ciphertext : b')\x11^\x05\xf0\xbfwA\x9b\xfe\xdf\xf5\x8f\xe4\xab\xf6"\x0b\xf5L\x1e8\xd0I}d\xebF\xb2\xcf\xd6\xe8' 32
Decoded iv : b'/V\\ \xd7\x023\x04\xb9\xb0\xdfV\xe3\xcd\x95P\x8c' 16
The decrypted data is:-
CVV : b'123'
CARD NO. : b'4431000078800563'

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>
```

AES OUTPUT


```
Command Prompt
C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Sender>Sender.py
Please press enter once the key is generated from the receiver side :
2DWAYEGxm7sIN4vHiDfu03w==
Algo Number : 2
2
DWAYEGxm7sIN4vHiDfu03w==
KEY : b'\x18\x12\x05\xe6\xee\xc2\r\xe2\xf1\xe2\r\xfb\xe\xdf'
Enter the CVV : 123
Enter the cardnumber : 4431000078800563
Encrypted text :- b'\xb0\xd4\x97\xd\x19\x1e8\x87\xbc\x1b\x00\x90&\xc9\xce\x1f\xee\x00\xdf'

Command Prompt
C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>Randomization.py
Symmetric Key : 2DWAYEGxm7sIN4vHiDfu03w==
AlgoNumber : 2
Please Enter when ciphertext generated :
The decrypted data is:-
CVV : 123
CARD NO. : 4431000078800563

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>
```

TDES OUTPUT

```
Command Prompt
C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Sender>Sender.py
Please press enter once the key is generated from the receiver side :
3b+wpXE0qrUImNWARgjPDGw==
Algo Number : 3
3
b+wpXE0qrUImNWARgjPDGw==
KEY : b'o\xec\\M*\xadB&5'\x11\x823\xc3\x1b'
Enter the CVV : 123
Enter the cardnumber : 4431000078800563
Data to encrypt 1234431000078800563
encrypted data b'\xe7\r\xbd\xd6T\xec\xa2\xa2\xab\xed\x3s\xac\xe8\xd2\x18,\xfc:\xe8\xd2\x8a\x14\x7f'

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Sender>

Command Prompt
C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>Randomization.py
Symmetric Key : 3b+wpXE0qrUImNWARgjPDGw==
AlgoNumber : 3
Please Enter when ciphertext generated :
Decoded ciphertext : b'\xe7\r\xbd\xd6T\xec\xa2\xa2\xab\xed\x3s\xac\xe8\xd2\x18,\xfc:\xe8\xd2\x8a\x14\x7f' 24
Decoded iv : b'\x17\x1c\x18:\xf2p>h' 8
The decrypted data is:-
CVV : 123
CARD NO. : 4431000078800563

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>
```

BLOWFISH OUTPUT

```
Command Prompt
C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Sender>Sender.py
Please press enter once the key is generated from the receiver side :
4tdlHmETnvTL7LH01mg00yQ==
Algo Number : 4
4
tdlHmETnvTL7LH01mg00yQ==
KEY : b'\xb5\xd9G\x98D\xe7\xbd2\xfb,s\x5\x9a\r\x0e\xc9'
Enter the CVV : 123
Enter the cardnumber : 4431000078800563
The encrypted data is: jtPGnbohvpfc6jaurHfYC4rVVw==
The iv is: jD1oCXGrUhYMLw==

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Sender>

Command Prompt
C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>Randomization.py
Symmetric Key : 4tdlHmETnvTL7LH01mg00yQ==
AlgoNumber : 4
Please Enter when ciphertext generated :
Decoded ciphertext : b'\x8e\xd3\xc6\x9d\xba!\xbe\x97\xdc\xea6\xae\xacw\xd8\x0b\x8a\xd5W' 19
Decoded iv : b'\x8c=h\tq\xabR\x16\x0c\x97' 10
Decrypted data is :-
CVV : b'123'
CARD NO. : b'4431000078800563'

C:\Users\Jatin Dhall\Desktop\Desktop File\VIT\VIT\SEM 3\Cyber Security\PROJECT\PROJECT\Reciever>
```

CAST-128 OUTPUT

BRIEF COMPARATIVE ANALYSIS OF THE ALGORITHMS USED:

Comparison of Algorithm Characteristics:

FACTORS	CAST 128	TDES	BLOWFISH	AES	RSA
KEY LENGTH	128 bits	168 (56*3) bits	448 bits	128 bits	2048 bits is the ideal length, length depends on size of prime numbers used
CYPHER TYPE	Symmetric Block Cypher	Symmetric Block Cypher	Symmetric Block Cypher	Symmetric Block Cypher	Asymmetric Cypher
DEVELOPED IN	1996	1995 (Derived from DES (1975))	1993	2000	1977
SPEED	Fast	Slow	Fast	Fast	Slow
SECURITY	Relatively less secure	Secure enough	Secure enough	Very Secure	Most Secure
BLOCK SIZE	64 bits	64 bits	64 bits	128 bits	NA
BASIC STRUCTURE	Feistel Network	Feistel Network	Feistel Network	Substitution-permutation Network	Prime Factorization

SECURITY LEVEL COMPARISON:

1. **CAST-128:** by means of a known plain text attack Key of CAST 128 can be known by linear cryptanalysis. It can be broken by 2^{17} chosen plaintexts along with one related-key query in offline work of 2^{48} .
2. **TDES:** Triple DES is vulnerable to meet-in-the middle attack because of which it give total security level of 2^{112} instead of using 168 bit of key. The block collision attack can also be done because of short block size and using same key to encrypt large size of text. It is also vulnerable to sweet32 attack.
3. **BLOWFISH:** Blowfish is a very secure algorithm but Initial 4 rounds of blowfish are observed unprotected from 2nd -order differential attack.
4. **AES:** No any such kind of weakness has been observed in AES. Some initial rounds of AES are observed unprotected i.e. initial rounds can break by square method.
5. **RSA:** The strength and security of the RSA algorithm rely on the difficulty of factoring large numbers. So far there has been no mathematically proven method which can crack an RSA encryption of sufficient key length.

Results:

The result is the creation of a strong and fast system that can encrypt the data in a randomized way, making it difficult for attackers to decrypt the data, whilst keeping it simple for the other end to decrypt the data with ease, not causing any trouble. The system successfully secures essential financial transaction details by relying on the security provided by the encryption algorithms while also adding an additional layer of protection by adding the unpredictability of randomization.

References:

- [1] T.Cuadra, L.Elkin, “Final Project-Secure RSA Credit Card Transaction System,” 2000. [Online]. Available: <https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2000/cuadra/final.html> [Accessed: 21-Sep-2021].
- [2] S.Aishwarya, K.D.R.Dhivya, “Online payment Fraud Prevention Using Cryptographic Algorithm TDES,” *International Journal of Computer Science and Mobile Computing*, Vol. 4, Issue. 4, pg.317 – 323, April 2015. [Online]. Available: <https://ijcsmc.com/docs/papers/April2015/V4I4201581.pdf> [Accessed: 22-Sep-2021].
- [3] G.B.Iwasokun, T.G.Omomule, R.O.Akinyede , “Encryption and Tokenization-Based System for Credit Card Information Security,” *International Journal of Cyber-Security and Digital Forensics*, August 2018. [Online]. Available: https://www.researchgate.net/publication/326753224_Encryption_and_Tokenization-Based_System_for_Credit_Card_Information_Security [Accessed: 22-Sep-2021].
- [4] H.Gupta, V.K.Sharma, “ Role of Multiple Encryption in Secure Electronic Transaction,” *International Journal of Network Security & Its Applications*, Vol.3, No.6, November 2011 . [Online]. Available: https://www.researchgate.net/publication/286732622_Role_of_Multiple_Encryption_in_Secure_Electronic_Transaction [Accessed: 23-Sep-2021].
- [5] Blowfish Algorithm with Examples [Online]. Available: <https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/> [Accessed: 21-Sep-2021].
- [6] Y.M.koukou, S.H.Othman, “Comparative study of AES, Blowfish,CAST-128 and DES algorithm,” *IOSR Journal of Engineering*, Vol.6, Issue 6, June 2016 [Online]. Available: [paper published 2.pdf \(sciencedev.net\)](https://www.researchgate.net/publication/326753224) [Accessed: 30-Sep-2021].
- [7] *Advanced Encryption Standard* [Online]. Available: https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm [Accessed: 22-Sep-2021].