INFORMATION AND SYSTEM SECURITY

SWE3002

J-COMPONENT FINAL REPORT

TITLE:

SIMPLE MATRIX ENCRYPTION TECHNIQUE **(SMET)**

Submitted to:

Prof. JEYANTHI N

Done by:

V. RAHUL – 18MIS0399

S.S. SOLOMON REX – 18MIS0411

SIMPLE MATRIX ENCRYPTION TECHNIQUE

Rahul V [1] , Solomon Rex S S [2]

School of Information Technology and Engineering, Vellore Institute of Technology,
Vellore, Tamil Nadu, India

ABSTRACT

Use of internet for communication and information sharing has risen dramatically over the past decade. This also means that tons of data are stored in a system to keep it safe but with the growth of technology people have also found ways to break in through the system security to access unauthorized data. In this paper we will be trying to implement an algorithm to encrypt text which is one of the basic form of data that's shared over the internet.

I. INTRODUCTION

Various text encryption algorithms are available in cryptography. We have come up with our own algorithm to encrypt and decrypt text and try to implement the same. We would be testing our algorithm on how efficient it is and provide the security analysis.

II. LITERATURE SURVEY:

1. **Research on Base64 Encoding Algorithm and PHP Implementation**

   In this paper, the principle of Encryption and Decryption of Base64 algorithm is expatiated and explained in detail. It provides examples of applications which help in quick grasp of Base64 algorithm encryption and decryption work and storage principle.

2. **Base64 Character Encoding and Decoding Modelling**

   This paper infers that the method used to secure data is to use a cryptographic system by changing plaintext into ciphertext. Base64 algorithm is one of the encryption processes that is ideal for use in data transmission. Ciphertext obtained is the arrangement of the characters that have been tabulated. These tables have been designed to facilitate the delivery of data during transmission. By applying this algorithm, errors would be avoided, and security would also be ensured.

III. PROPOSED METHODOLOGY

SMET ALGORITHM

- Initially, each character from the text is converted to binary form and separated as a group of 8 bits.
- Three 8-bits segment is grouped and four 6-bits segments are formed.
- The corresponding 6-bits segments are then converted into Base64 character values.

Base64 Table

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	**000000**	**A**	22	**010110**	**W**	44	**101100**	**s**
1	**000001**	**B**	23	**010111**	**X**	45	**101101**	**t**
2	**000010**	**C**	24	**011000**	**Y**	46	**101110**	**u**
3	**000011**	**D**	25	**011001**	**Z**	47	**101111**	**v**
4	**000100**	**E**	26	**011010**	**a**	48	**110000**	**w**
5	**000101**	**F**	27	**011011**	**b**	49	**110001**	**x**
6	**000110**	**G**	28	**011100**	**c**	50	**110010**	**y**
7	**000111**	**H**	29	**011101**	**d**	51	**110011**	**z**
8	**001000**	**I**	30	**011110**	**e**	52	**110100**	**0**
9	**001001**	**J**	31	**011111**	**f**	53	**110101**	**1**
10	**001010**	**K**	32	**100000**	**g**	54	**110110**	**2**
11	**001011**	**L**	33	**100001**	**h**	55	**110111**	**3**
12	**001100**	**M**	34	**100010**	**i**	56	**111000**	**4**
13	**001101**	**N**	35	**100011**	**j**	57	**111001**	**5**
14	**001110**	**O**	36	**100100**	**k**	58	**111010**	**6**
15	**001111**	**P**	37	**100101**	**l**	59	**111011**	**7**
16	**010000**	**Q**	38	**100110**	**m**	60	**111100**	**8**
17	**010001**	**R**	39	**100111**	**n**	61	**111101**	**9**
18	**010010**	**S**	40	**101000**	**o**	62	**111110**	**+**
19	**010011**	**T**	41	**101001**	**p**	63	**111111**	**/**
20	**010100**	**U**	42	**101010**	**q**	64	padding	=
21	**010101**	**V**	43	**101011**	**r**			

- The Base64 code is generated using the table.
- Now, this code is further used in the algorithm.

Encryption

- Find s= [(length of code) mod3] +2
- Split the message into multiple segments of size s.
- Form a s x s matrix with the number equivalent of the code using the MET table given below.

MET:

Uppercase/Lowercase	**Number equivalent**
A	2
B	1
C	10
D	16
E	3
F	4
G	12
H	17
I	5
J	6
K	13
L	18
M	21
N	24
O	7
P	8
Q	14
R	19
S	22
T	25
U	11
V	9
W	15
X	20
Y	23
Z	26

- Find D = (determinant of matrix)Mod26
- If the character is uppercase, add the corresponding D value(cyclic order) to the character in the matrix.
- If the character is lowercase, subtract the corresponding D value(cyclic order) from the character in the matrix.
- If it is a numerical character:

  Odd number: Add 4

  Even number: Subtract 4
- Finally, the encrypted message is generated.
- A random key is generated.
- The key value is added/subtracted to the D values based on the length of the D value array.
- The D values are appended to the cipher text and sent to the receiver.

Decryption

- Find s.
- Split the cipher text by ‘s’ no. of characters.
- Use the corresponding D value for the ‘s’ group of characters to decrypt the cipher text.
- After decryption decode the code using Base64.
- The message is successfully decrypted.

SAMPLE ENCRYPTION

String: “ ISS project review :- 2 ”

STEP 1

Base64 code: SVNTIHByb2plY3QgcmV2aWV3IDotIDI

Length of string = 31
S= (31 mod 3) + 2 => 1+2=3
S=3

STEP2

SVN TIH Byb 2pl Y3Q gcm V2a WV3 IDo tID I

STEP 3

S V N   22 9 24
T I H   →   25 5 17   det =>   5116 mod 26 = 20
B y b   1 23 1

D=20
SVN TIH Byb -> MPH NCB Veh
----

2 p l   2 8 18
Y 3 Q   →   23 3 14   det =>   818 mod 26= 12
g c m   12 10 21

D=12
2pl Y3Q gcm -> 6dz K0C uqa

V 2 a   9   2   2
W V 3 → 15 9   3   det => 345 mod 26 = 7
I D o   5 16 7

D=7
V2a WV3 IDo -> C6t DC0 PKh

---

t I D   25   5   16
I 1 1 → 5   1   1   det => 44 mod 26 = 18
1 1 1   1   1   1

D=18
tID I -> bAS A

Therefore, the base64 code:

SVNTIHByb2plY3QgcmV2aWV3IDotIDI

becomes => MPHNCBVeh8dzK7CuqaC8tDC7PKhbASA

<u>ENCRYTED MESSAGE</u> →

[MPHNCBVeh6dzK0CuqaC6tDC0PKhbASA, {20,12,7,18}]

IV. IMPLEMENTATION

CODE:

FINALPROJECT.py - C:\Users\rahul\AppData\Local\Programs\Python\Python39\FINALPROJECT.py (3.9.0)

File Edit Format Run Options Window Help

```python
import base64
import numpy as np
import random
def split(word):
    return [char for char in word]
def met(x):
    switcher={
        'a':2,
        'b':1,
        'c':10,
        'd':16,
        'e':3,
        'f':4,
        'g':12,
        'h':17,
        'i':5,
        'j':6,
        'k':13,
        'l':18,
        'm':21,
        'n':24,
        'o':7,
        'p':8,
        'q':14,
        'r':19,
        's':22,
        't':25,
        'u':11,
        'v':9,
        'w':15,
        'x':20,
        'y':23,
        'z':26,
        'A':2,
        'B':1,
        'C':10,
        'D':16,
        'E':3,
        'F':4,
        'G':12,
        'H':17,
        'I':5,
        'J':6,
        'K':13,
        'L':18,
        'M':21,
        'N':24,
        'O':7,
        'P':8,
        'Q':14,
        'R':19,
        'S':22,
        'T':25,
        'U':11,
        'V':9,
        'W':15,
        'X':20,
        'Y':23,
        'Z':26,
        '0':0,
        '1':1,
        '2':2,
        '3':3,
        '4':4,
        '5':5,
        '6':6,
        '7':7,
        '8':8,
        '9':9
        }
    return switcher.get(x)
print("\t\t\t\t\t\t\t\t\tSIMPLE MATRIX ENCRYPTION TECHNIQUE")
print("\nENCRYPTION:\n")
plain_txt = input("Enter text to be encrypted:")
plain_txt_bytes = plain_txt.encode("ascii")

base64_bytes = base64.b64encode(plain_txt_bytes)
base64_string = base64_bytes.decode("ascii")

print(f"Encoded string: {base64_string}\n")

l= len(base64_string)
c=base64_string.count("=")
l=l-c
code=base64_string[0:l]
s=(l%3)+2
substrings = [code[i:i+s] for i in range(0, l, s)]
#print(substrings)
nlist=[code[i:i+s] for i in range(0,l,s)]

for x in range(len(substrings)):
    substrings[x]=split(substrings[x])

for x in range(len(nlist)):
    nlist[x]=split(nlist[x])

for i in range(len(substrings)):
    for j in range(len(substrings[i])):
        substrings[i][j]=met(substrings[i][j])
```

```
#print(substrings)

while len(substrings[len(substrings)-1])!=len(substrings[len(substrings)-2]):
    substrings[len(substrings)-1].append(1)
m=len(substrings)%s
t=s-m
if s==2:
    while t!=0:
        substrings.append([1,1])
        t-=1

elif s==3:
    while t!=0:
        substrings.append([1,1,1])
        t-=1

elif s==4:
    while t!=0:
        substrings.append([1,1,1,1])
        t-=1


#print(substrings)

detm=[]
if s==2:
    for i in range(0,len(substrings),s):
        arr2=np.array([substrings[i],substrings[i+1]])
        #print(arr)
        D = np.linalg.det(arr2)
        detm.append(round(D%26))
    print("Determinants:",detm)

elif s==3:
    for i in range(0,len(substrings),s):
        arr3=np.array([substrings[i],substrings[i+1],substrings[i+2]])
        #print(arr)
        D = np.linalg.det(arr3)
        detm.append(round(D%26))
    print("Determinants:",detm)

elif s==4:
    for i in range(0,len(substrings),s):
        arr4=np.array([substrings[i],substrings[i+1],substrings[i+2],substrings[i+3]])
        #print(arr)
        D = np.linalg.det(arr4)
        detm.append(round(D%26))
    print("Determinants:",detm)
r=random.randint(1,99)
print("\nSecret key:",r)
```

```
for i in range(len(detm)):
    if len(detm)%2==0 :
        res = [x + r*i for x in detm]
    else:
        res = [x - r*i for x in detm]
print("\nEncrypted determinants:" + str(res))
#print(nlist)
k=0
cnt=0
f1=[]

for i in range(len(nlist)):
    for j in range(len(nlist[i])):
        if ord(nlist[i][j])>=65 and ord(nlist[i][j])<=90:
            d=ord(nlist[i][j])
            x=d-64
            #print(x)
            A=x+detm[k]
            if A>26:
                A-=26
            A=chr(A+64)
            #nlist[i][j]=A
            f1.append(A)

        elif ord(nlist[i][j])>=97 and ord(nlist[i][j])<=122:
            l=ord(nlist[i][j])
            p=l-96
            #print(p)
            b=p-detm[k]
            if b<=0:
                b+=26
            b=chr(b+96)
            #nlist[i][j]=a
            f1.append(b)

        elif ord(nlist[i][j])>=48 and ord(nlist[i][j])<=57:
            m=ord(nlist[i][j])
            n=m-48
            #print(n)
            #nlist[i][j]=m
            if n%2==0:
                n+=4
                if n>9:
                    n-=10
            else:
                n-=4
                if n<0:
                    n+=10

            f1.append(str(n))
```

```python
        cnt+=1

    if cnt==(s*s):
        k+=1
        cnt=0

#print(f1)
ct=""
ct=ct.join(f1)
#print(ct)

print("\nENCRYPTED CODE:",ct,",",str(res))

#DECRYPTION
for i in range(len(res)):
    if len(res)%2==0 :
        dd = [x - r*i for x in res]
    else:
        dd = [x + r*i for x in res]
print("\n\nDECRYPTION:\n")
print("Decrypted determinants:" + str(dd))

f=(len(ct)%3)+2
dstr = [ct[i:i+s] for i in range(0,len(ct),f)]
#print(dstr)
for x in range(len(dstr)):
    dstr[x]=split(dstr[x])
#print(dstr)



dk=0
dcnt=0
df1=[]

for i in range(len(dstr)):
    for j in range(len(dstr[i])):
        if ord(dstr[i][j])>=65 and ord(dstr[i][j])<=90:
            d=ord(dstr[i][j])
            x=d-64
            #print(x)
            A=x-dd[dk]
            if A<=0:
                A+=26
            A=chr(A+64)
            #nlist[i][j]=A
            df1.append(A)

        elif ord(dstr[i][j])>=97 and ord(dstr[i][j])<=122:
            l=ord(dstr[i][j])
            p=l-96
            #print(p)
            b=p+dd[dk]
            if b>26:
                b-=26
            b=chr(b+96)
            #nlist[i][j]=a
            df1.append(b)

        elif ord(dstr[i][j])>=48 and ord(dstr[i][j])<=57:
            m=ord(dstr[i][j])
            n=m-48
            #print(n)
            #nlist[i][j]=m
            if n%2==0:
                n-=4
                if n<0:
                    n+=10
            else:
                n+=4
                if n>9:
                    n-=10

            df1.append(str(n))

        dcnt+=1

    if dcnt==(s*s):
        dk+=1
        dcnt=0

#print(df1)
dt=""
dt=dt.join(df1)
while c!=0:
    dt=dt + "="
    c-=1
print("\nDECRYPTED CODE:",dt)

dbase64_bytes = dt.encode("ascii")

dstring_bytes = base64.b64decode(dbase64_bytes)
dstring = dstring_bytes.decode("ascii")

print(f"\nDECRYPTED MESSAGE: {dstring}")
```

RESULTS:

Python 3.9.0 Shell

File Edit Shell Debug Options Window Help

```
                                    SIMPLE MATRIX ENCRYPTION TECHNIQUE

ENCRYPTION:

Enter text to be encrypted:ISS project Review:-3

Encoded string: SVNTIHByb2plY3QgUmV2aWV3Oi0z

Determinants: [20, 8, 9, 0]

Secret key: 56

Encrypted determinants:[188, 176, 177, 168]

ENCRYPTED CODE: MPHNCBVeh6hdG9YyCeE6rFE9Xz4z , [188, 176, 177, 168]


DECRYPTION:

Decrypted determinants:[20, 8, 9, 0]

DECRYPTED CODE: SVNTIHByb2plY3QgUmV2aWV3Oi0z

DECRYPTED MESSAGE: ISS project Review:-3
>>>
===================================================== RESTART: C:\Users\rahul\AppData\Local\Programs\Python\Python39\FINALPROJECT.py ===============================================
                                    SIMPLE MATRIX ENCRYPTION TECHNIQUE

ENCRYPTION:

Enter text to be encrypted:HII!!! This is the final report for SMET

Encoded string: SElJISEhIFRoaXMgaXMgdGhlIGZpbmFsIHJlcG9ydCBmb3IgU01FVA==

Determinants: [0, 19, 11, 20, 20, 6, 14, 16, 14, 18, 14, 23, 18, 7]

Secret key: 98

Encrypted determinants:[1274, 1293, 1285, 1294, 1294, 1280, 1288, 1290, 1288, 1292, 1288, 1297, 1292, 1281]

ENCRYPTED CODE: SElJBLXoTQCdgRGmgRGmxMbfWUNblwVcWVXxkY5gpQPye9FjM47XCH , [1274, 1293, 1285, 1294, 1294, 1280, 1288, 1290, 1288, 1292, 1288, 1297, 1292, 1281]


DECRYPTION:

Decrypted determinants:[0, 19, 11, 20, 20, 6, 14, 16, 14, 18, 14, 23, 18, 7]

DECRYPTED CODE: SElJISEhIFRoaXMgaXMgdGhlIGZpbmFsIHJlcG9ydCBmb3IgU01FVA==

DECRYPTED MESSAGE: HII!!! This is the final report for SMET
                                    SIMPLE MATRIX ENCRYPTION TECHNIQUE

ENCRYPTION:

Enter text to be encrypted:SimPLe MaTrix Encrypt10n !@#$^$&#*(@345t

Encoded string: U2ltUExlIE1hVHJpeCBFbmNyeXB0MTBuICFAIyReJCYjKihAMzQ1dA==

Determinants: [5, 8, 4, 22, 2, 13, 6, 24, 22, 20, 14, 19, 21, 14]

Secret key: 4

Encrypted determinants:[57, 60, 56, 74, 54, 65, 58, 76, 74, 72, 66, 71, 73, 66]

ENCRYPTED CODE: Z6goCMpdMI7dRDFtcEDHozAlyDH4KRZwEYBWCeLkXQMvDpoTHeL7pO , [57, 60, 56, 74, 54, 65, 58, 76, 74, 72, 66, 71, 73, 66]


DECRYPTION:

Decrypted determinants:[5, 8, 4, 22, 2, 13, 6, 24, 22, 20, 14, 19, 21, 14]

DECRYPTED CODE: U2ltUExlIE1hVHJpeCBFbmNyeXB0MTBuICFAIyReJCYjKihAMzQ1dA==

DECRYPTED MESSAGE: SimPLe MaTrix Encrypt10n !@#$^$&#*(@345t
>>>
                                    SIMPLE MATRIX ENCRYPTION TECHNIQUE

ENCRYPTION:

Enter text to be encrypted:!@#$(#)@3437219234r

Encoded string: IUAjJCgjKUAzNDM3MjE5MjM0cg==

Determinants: [20, 13]

Secret key: 4

Encrypted determinants:[24, 17]

ENCRYPTED CODE: COUpDWmpEOUfHXG9ZwR1ZwZ4pt , [24, 17]


DECRYPTION:

Decrypted determinants:[20, 13]

DECRYPTED CODE: IUAjJCgjKUAzNDM3MjE5MjM0cg==

DECRYPTED MESSAGE: !@#$(#)@3437219234r
>>>
```

V. SECURITY ANALYSIS

i. BRUTE FORCE ATTACK

Brute force attack is a means by which the attacker tries to decrypt or crack the code by trial-and-error method. This is heavily countered by SMET because of the multiple layers of encryption and by the different methods used to encrypt different types of data input. Another effort taken to prevent brute force attack is that a random number generated is used as a secret key between sender and receiver and it is shared between them. These steps that are taken will highly reduce the chances of the cipher text being compromised through brute force.

ii. MAN IN THE MIDDLE ATTACK

This is a type of attack where an attacker tries to eavesdrop or impersonate a member on the network. This type of attack is prevented in SMET by the use of secret key to identify and authenticate the members involved. Also, even if the attacker gets hold of the cipher text, he can't decrypt the message without the secret key. Hence SMET also provides protection from man in the middle attack.

iii. MASQUERADE

No attacker can try to impersonate or masquerade as the sender or receiver because both of them share a secret key that is communicated before the transaction. The sender and receiver are uniquely identified by their personal ID so unless the users leak their information SMET is safe from masquerade type of attacks.

iv. MODIFICATION OF MESSAGES

This is a type of attack where the attacker tries to modify the content of the message to produce undesired outcomes. IN SMET since the message is heavily encrypted the attacker does not have any clue as to what to modify in the given cipher text, he can try to alter messages randomly but it will not produce the plain text when the receiver decrypts it so they will know not to trust the message since it could have been compromised.

VI. CONCLUSION AND FUTURE WORKS

In this paper, we proposed and implemented a text encryption algorithm using matrix which converts the given text into a matrix format and generates the cipher text. In future work, we plan to improve the security of our algorithm by providing better encryption for the determinants. Currently the algorithm is limited to text-only but in the future updates we plan to include image and file encryptions as well.

VII. REFERENCES

[1] S. Wen and W. Dang, "Research on Base64 Encoding Algorithm and PHP Implementation," 2018 26th International Conference on Geoinformatics,2018, pp. 1-5, doi: 10.1109/GEOINFORMATICS.2018.8557068.

[2] Sumartono, Isnar & Siahaan, Andysah Putera Utama & Arpan, Arpan. (2016). Base64 Character Encoding and Decoding Modeling. International Journal of Recent Trends in Engineering & Research. 2. 63-68.