Cryptanalysis

- 2024-Spring -

Ji, Yong-Hyeon

A document presented for
the Cryptanalysis

Department of Information Security, Cryptology, and Mathematics
College of Science and Technology
Kookmin University

April 22, 2024

Contents

Chapter 1

Midterm Examination

1.1 2023-Spring-Midterm

1. [Substitution Cipher]

```python
Alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
#-- Subst-Encryption
def subst_encrypt(key, msg):
        result = ''
        InSet = Alphabet     # InSet = 'AB...Z'
        OutSet = key         # OutSet = '...' (Key)
        for ch in msg:
                if ch.upper() in InSet:
                        idx = InSet.find(ch.upper())
                        if ch.isupper():
                                result += OutSet[idx].upper()
                        else:
                                result += OutSet[idx].lower()
                else:
                        result += ch
        return result
#-- Subst-Decryption
def subst_decrypt(key, msg):
    result = ''
        InSet = Alphabet
        OutSet = key
        for ch in msg:
                if ch.upper() in InSet:
                        idx = InSet.find(ch.upper())
                        if ch.isupper():
                                result += OutSet[idx].upper()
                        else:
                                result += OutSet[idx].lower()
                else:
                        result += ch
        return result
```

(a)

```python
# A -> A, B -> B, C -> F, ..., Z -> O
key1 = 'ABFWQICPLZYUDNHMGKEXVTSJRO'
pt1 = 'banana'
ct1 = subst_encrypt(key1, pt1)
print('Eng =', Alphabet)
print('key1 =', key1)
print('pt1 =', pt1)
print('ct1 =', ct1)
```

```
Eng = ABCDEFGHIJKLMNOPQRSTUVWXYZ
key1 = ABFWQICPLZYUDNHMGKEXVTSJRO
pt1 = banana
ct1 = banana
```

(b)

```python
def valid_key1(key):
        list_key = list(key)
        list_key.sort() # ASCII; A = 65, a = 97
        key_alphabet = ''.join(list_key).upper()
        return key_alphabet == Alphabet

def valid_key2(key):
        for ch in key:
                if not ch.isalpha():
                        return False
                if len(set(key)) != 26:
                        return False
        return True
```

```python
key_b1 = 'AAADEFGHIJKLMNOPQRSTUVWXYZ'
key_b2 = 'ABCDEFGHIJKLMNOPQRSTUVWxyz'
key_b3 = 'abcDEFGHIJKLMNOPQRSTUVWXYZ'

print(f"b1 | {valid_key1(key_b1)} : {valid_key2(key_b1)}")
print(f"b2 | {valid_key1(key_b2)} : {valid_key2(key_b2)}")
print(f"b3 | {valid_key1(key_b3)} : {valid_key2(key_b3)}")
```

```
b1 | False : False
b2 | True  : True
b3 | False : True
```

(c)

```
def gen_random_key(seed):
        random.seed(seed)
        alpha_list = list(Alphabet)
        random.shuffle(alpha_list)
        shuffled_key = ''.join(alpha_list)
        return shuffled_key
```

```
def gen_random_key(seed):
        random.seed(seed)
        alpha_list = list(Alphabet)
        valid_key = False
        while not valid_key:
                random.shuffle(alpha_list)
                shuffled_key = ''.join(alpha_list)
                valid_key = True
                for i in range(len(Alphabet)):
                        # if A -> A or ... or Z -> Z
                        if Alphabet[i] == shuffled_key[i]:
                                valid_key = False
                                break # for-break
                return shuffled_key

for _ in range(10):
        alpha_list = list(Alphabet)
        for _ in range(255):
                random.shuffle(alpha_list)
        seed = ''.join(alpha_list)
        print(gen_random_key2(seed))
```

```
QODUNVELJPFZGTYCXSAKRIBHMW
NRHSXVFWUCYOJZBTLMPEKDAIQG
CYKXDVPEBMWAZRJHOTISFUNQLG
TDMZLNUSQCVJOGIEAWRFKXYHPB
LYDJTIPFNRWQKEOMBVHXGUSZAC
HSOUNZPTIJRGQWAXLVYFDEMBKC
EDBICNMHUYQOZXSJRLAWFKPTGV
KUDCFEHXYATINRPVQMGWZSLBJO
CKSHBAPROUVINXTLWFEJGQMYZD
KARZCIFPDUMJTYGHWSQEOVLBXN
```

2. [Index of Coincidence]

Index of Coincidence

For a given ciphertext, the **index of coincidence** I is defined to be the probability that two randomly selected letters in the ciphertext represent the same plaintext symbol. For a given ciphertext, let $n_0, n_1, \ldots, n_{25}$ be the respective letter counts of `A`, `B`, `C`, ..., `Z` in the ciphertext, and set $n = n_0 + n_1 + \cdots + n_{25}$. Then, the index of coincidence can be computed as

$$I = \frac{\binom{n_1}{2} + \binom{n_2}{2} + \cdots \binom{n_{25}}{2}}{\binom{n_1+\cdots+n_{25}}{2}} = \frac{\sum_{i=0}^{25}\binom{n_i}{2}}{\binom{n}{2}} = \frac{1}{n(n-1)}\sum_{i=0}^{25}[n_i(n_i-1)].$$

To see why the index of coincidence gives us useful information, first note that the empirical probability of randomly selecting two identical letters from a large English plaintext is

$$\sum_{i=0}^{25} p_i^2 \approx 0.065.$$

(a) Attack with Index of Coincidence

Caesar Cipher	`IC(PT)=IC(CT)`
Substitution Cipher	`IC(PT)=IC(CT)`
Vigenére Cipher	`IC(CT)≈IC(rand)`, $IC(CT)<IC(PT)$

(b) 알파벳 26 글자 중 'A'의 빈도가 30%이고, 'B'-'K'까지 10글자가 같은 비율로 사용되며, 나머지 알파벳은 쓰지 않는 언어가 있다고 가정하자. 이 언어로 된 문서의 'Index of Coincidence'는 얼마인가?

Sol.

- Document with N characters.
- Frequency of 'B' - 'K' : 70% (for each 7%)

Then

$$IC = \frac{1}{N(N-1)}\left[0.3N(0.3N-1) + 0.07N(0.07-1)\times 10\right]$$

and so

$$\lim_{N\to\infty} IC = \frac{(0.3)^2\cdot N^2 + \cdots + (0.07)^2\cdot 10\cdot N^2 + \cdots\times}{N^2+\cdots} = 0.09 + 0.049 = 0.139.$$

□

(c) 다음은 영문을 Vigenére 암호로 암호화한 문서에 대하여 키 길이 key len 을 1부터 증가시켜가며 암호문을 키 길이 간격으로 추출한 sub msg에 대하여 'IC(Index of Coincidence)'를 계산한 결과이다. 아래 결과로부터 사용된 암호키 는 몇 글자로 추정되는가?

```
key_len =  1 :IC(sub_msg) = 0.0435
key_len =  2 :IC(sub_msg) = 0.0493
key_len =  3 :IC(sub_msg) = 0.0428
key_len =  4 :IC(sub_msg) = 0.0598
key_len =  5 :IC(sub_msg) = 0.0424
key_len =  6 :IC(sub_msg) = 0.0477
key_len =  7 :IC(sub_msg) = 0.0444
key_len =  8 :IC(sub_msg) = 0.0597
key_len =  9 :IC(sub_msg) = 0.0418
key_len = 10 :IC(sub_msg) = 0.0492
key_len = 11 :IC(sub_msg) = 0.0445
key_len = 12 :IC(sub_msg) = 0.0578
key_len = 13 :IC(sub_msg) = 0.0469
key_len = 14 :IC(sub_msg) = 0.0505
key_len = 15 :IC(sub_msg) = 0.0416
key_len = 16 :IC(sub_msg) = 0.0638
key_len = 17 :IC(sub_msg) = 0.0469
```

Sol. Length of Key = 4 (Size of Interval) □

3.

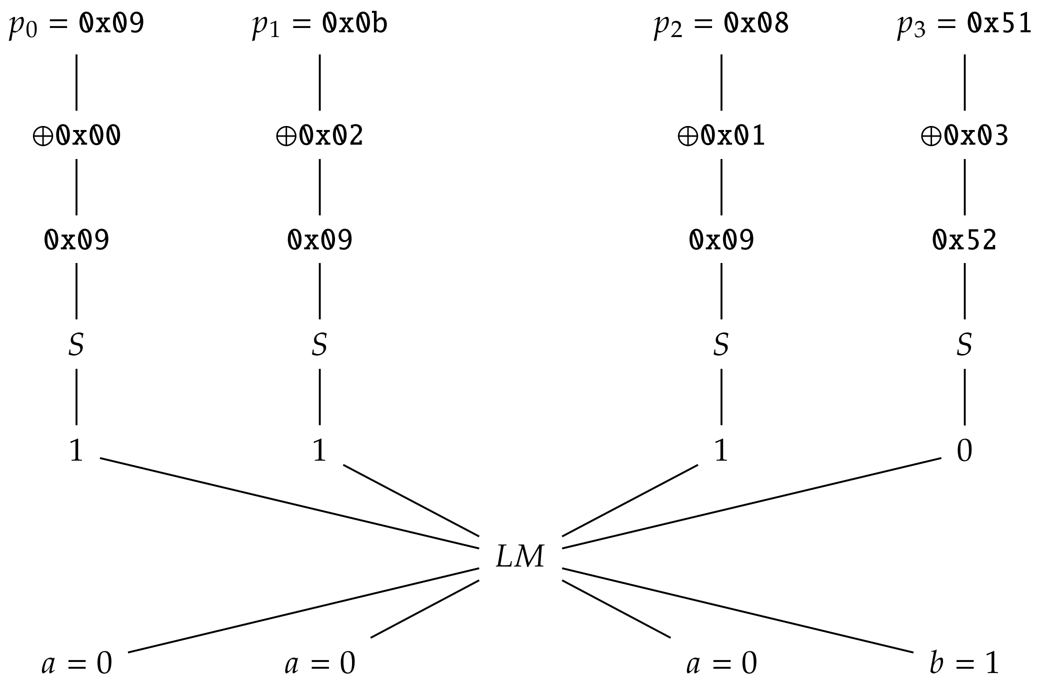
```
#==========================================
# TC1 - Toy Cipher (encryption/decryption)
# - n, k: 32-bit
# - Identical Round Function
# - Key Schedule (X)
#==========================================

NUM_ROUND = 10

#---------------------------------------
#  Encryption
#---------------------------------------

#--- S-Box (AES)
Sbox = [ ... ]
#--- Inverse S-Box (AES)
ISbox = [ ... ]

#-- AR: Add Roundkey
def AR(in_state, rkey):
        out_state = [0] * len(in_state)
        for i in range(len(in_state)):
                out_state[i] = in_state[i] ^ rkey[i]
        return out_state

#-- SB: Sbox layer
def SB(in_state):
        out_state = [0] * len(in_state)
        for i in range(len(in_state)):
                out_state[i] = Sbox[in_state[i]]
        return out_state

#-- LM: Linear Map
# |Y0|     |0 1 1 1||x0|    |x1^x2^x3|
# |  |     |       ||  |    |        |
# |Y1|     |1 0 1 1||x1|    |x0^x2^x3|
# |  |  =  |       ||  | =  |        |
# |Y2|     |1 1 0 1||x2|    |x0^x1^x3|
# |  |     |       ||  |    |        |
# |Y3|     |1 1 1 0||x3|    |x0^x1^x2|
def LM(in_state):
        out_state = [0] * len(in_state)
        All_Xor = in_state[0] ^ in_state[1] ^ in_state[2] ^ in_state[3]
        for i in range(len(in_state)):
                out_state[i] = All_Xor ^ in_state[i]
        return out_state
```

```
#-- Enc_Round
def Enc_Round(in_state, rkey):
        out_state = [0] * len(in_state)
        out_state = AR(in_state, rkey)
        in_state = SB(out_state)
        out_state = LM(in_state)
        return out_state

#- TC1 Encryption
def TC1_Enc(PT, key):
        NROUND = NUM_ROUND # Number of Round = 10
        CT = PT #CT = [0] * len(PT)
        for i in range(NROUND):
                CT = Enc_Round(CT, key)
        return CT

#---------------------------------------
#  Decryption
#---------------------------------------

#-- SB: Sbox layer
def ISB(in_state):
out_state = [0, 0, 0, 0]
        for i in range(len(in_state)):
                out_state[i] = ISbox[in_state[i]]
        return out_state

#-- Decrypt Round
def Dec_Round(in_state, rkey):
        out_state1 = [0, 0, 0, 0]
        out_state2 = [0, 0, 0, 0]
        out_state3 = [0, 0, 0, 0]
        out_state1 = LM(in_state)
        out_state2 = ISB(out_state1)
        out_state3 = AR(out_state2, rkey)
        return out_state3

#- TC1 Decryption
def TC1_Dec(input_state, key):
        state = input_state
        numRound = NUM_ROUND
        for i in range(0, numRound):
                state = Dec_Round(state, key)
        return state
```

(a) 암호키 32비트가 `[0x00, 0x02, 0x01, 0x03]`으로 설정되었다고 하자. 1라운드 출력의 처음 3바이트가 동일한 입력의 예를 하나만 만들면?

$$[p_0, p_1, p_2, p_3] \implies [a, a, a, b]$$

Sol. Consider $[a, a, a, b] = [0, 0, 0, 1]$. Then $[p_0, p_1, p_2, p_3] =$ `[0x09,0x0b,0x08,0x51]`



□

(b) Note that

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = M \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

Let $M = I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$. Then $c_0 = f_0(p_0, k_0)$ and so we can find k_0 with $2^8 = 256$ complexity.

4. [Double Encryption]

- Block Cipher `E1`: in/out 64-bit, key 64-bit
- Block Cipher `E2`: in/out 64-bit, key 52-bit
- Formula: $CT = E2(E1(PT, K1), K2)$

$$\begin{tikzcd}
PT \arrow[d, "64"'] & \\
E1 \arrow[d, "64"'] & K_1 \arrow[l, "64"'] \\
E2 \arrow[d, "64"'] & K_2 \arrow[l, "56"'] \\
CT &
\end{tikzcd}$$

(a) (평문, 암호문) 쌍을 여러 개 수집한 후 공격을 시작한다고 가정하자. 수집한 평문을 E1으로 암호화하여, 중간값을 저장하고, 수집한 암호문을 E2로 복호화하여 비교하는 방식으로 공격한다면, 이 때, 사용되는 메모리와 계산량은 각각 얼마인가?

Sol. Memory = $2^{k_1} = 2^{64}$, Time = $2^{k_2} = 2^{56}$. □

(b) (a)와 동일한 조건에서 블록암호 E1과 E2의 역할을 반대로 하여 공격한다 면, 즉, $PT = E1^{-1}(E2^{-1}(CT, K2), K1)$를 이용하여 같은 공격을 수행한다면, 이 때, 사용되는 메모리와 계산량은 각각 얼마인가?

Sol. Memory = $2^{k_2} = 2^{56}$, Time = $2^{k_1} = 2^{64}$. □

(c) 높은 확률로 하나의 키 후보만 남도록 하려면 몇개의 (평문, 암호문) 쌍을 수집해야 하는가?

Sol.

- 64-bit 블록암호에서 우연히 중간값이 일치할 확률 = $\frac{1}{2^{64}}$
- 2^{k_1} (테이블 크기, 저장 되어 있는 중간값의 개수)
- 2^{k_2} (비교 횟수)

Thus,

$$\frac{1}{2^{64}} \times 2^{k_1} \times 2^{k_2} = 2^{k_1+k_2-64} = 2^{64+56-64} = 2^{56}.$$

추가로 (PT,CT) 한 쌍을 확인하면 $2^{56} \times \frac{1}{2^{64}} = \frac{1}{8}$. MIMT + 한쌍 or 처음부터해서 두 쌍. □

(d) 선택평문공격(chosen plaintext attack)이 가능하다면, 고정된 평문에 대 한 사전계산(pre-computation)을 이용하여 공격할 수 있다. 이 때 최적의 공격 방법을 구성하고 사전계산으로 얻어지는 장점을 설명하라.

Sol.

i. 고정 평문에 대하여 2^{k_1} 테이블을 사전계산한다.
ii. 암호문 수집 후 2^{k_2} 정도의 복호화하면서 찾을 수 있다.

□

5. [TMTO Attack]

(1) m: number of random staring points

(2) t: length of chain

(3) ℓ : number of Hellman Table

$$X_{i,j+1} = f(X_{i,j}) = E(PT, X_{i,j}), \quad (i = 1, 2, \ldots, m, j = 0, 1, \ldots, t).$$

Let $m = 2^{24}, t = 2^{20}$ and $\ell = 2^{20}$.

$$SP_1 = X_{1,0} \longrightarrow X_{1,1} \longrightarrow X_{1,2} \longrightarrow \cdots \longrightarrow X_{1,t-1} \longrightarrow X_{1,t} = EP_1$$

$$SP_2 = X_{2,0} \longrightarrow X_{2,1} \longrightarrow X_{2,2} \longrightarrow \cdots \longrightarrow X_{2,t-1} \longrightarrow X_{2,t} = EP_2$$

$$SP_i = X_{i,0} \longrightarrow X_{i,1} \longrightarrow X_{i,2} \longrightarrow \cdots \longrightarrow X_{i,t-1} \longrightarrow X_{i,t} = EP_i$$

$$SP_m = X_{m,0} \longrightarrow X_{m,1} \longrightarrow X_{m,2} \longrightarrow \cdots \longrightarrow X_{m,t-1} \longrightarrow X_{m,t} = EP_m$$

(a) 암호문 CT 에 f 를 a번 적용하여 EPi와 일치함을 확인했다면, 암호키를 어떻게 결정하는가? 이 과정에서 필요한 계산량은?

$$EP_i \underbrace{(f \circ f \circ \cdots \circ f)}_{a}(CT)$$

Sol. $SP_i = X_{i,0}$를 $(t - a - 1)$번 암호화 하면 $X_{t-a-1}(= key)$를 얻는다. □

(b) 테이블 준비에 필요한 사전 계산량(pre-computation)은 얼마인가? 전수 조사보다 많은 사전 계산량을 사용해도 의미있는 공격이 가능함을 설명하라.

Sol. $m \times t \times l = 2^{24} \times 2^{20} \times 2^{20} = 2^{64}$. 실제 암호문 획득 이전에 미리 테이블을 만들어두면 실제 암호문을 획득 했을 때 해당되는 키를 빠르게 얻을 수 있다. □

(c) 사전계산 후 공격에 필요한 메모리와 계산량은 각각 얼마인가?

Sol. 사전 계산량 mtl를 제외한다면 $M = m \times l$ and $T = t \times l$. □

(d) 높은 확률로 하나의 키 후보만 남기도록 하려면, 추가적으로 필요한 (평문, 암호문)의 쌍은 몇 개인가?

Sol. 다른 한 쌍 (P', C')를 추가적으로 더 확인해보면 높은 확률로 하나의 키 후보를 남기게 된다. □

Chapter 2

Practice Code

2.1 Toy Cipher TC1

TC1Lib.py

```
#==========================================
# TC1 - Toy Cipher (encryption/decryption)
# - n, k: 32-bit
# - Identical Round Function
# - Key Schedule (X)
#==========================================

NUM_ROUND = 10

#--------------------------------------
#  Encryption
#--------------------------------------

#--- S-Box (AES)
Sbox = [ ... ]
#--- Inverse S-Box (AES)
ISbox = [ ... ]

#-- AR: Add Roundkey
def AR(in_state, rkey):
   out_state = [0] * len(in_state)
   for i in range(len(in_state)):
      out_state[i] = in_state[i] ^ rkey[i]
   return out_state

#-- SB: Sbox layer
def SB(in_state):
   out_state = [0] * len(in_state)
   for i in range(len(in_state)):
      out_state[i] = Sbox[in_state[i]]
   return out_state
```

```
#-- LM: Linear Map
def LM(in_state):
   out_state = [0] * len(in_state)
   All_Xor = in_state[0] ^ in_state[1] ^ in_state[2] ^ in_state[3]
   for i in range(len(in_state)):
      out_state[i] = All_Xor ^ in_state[i]
   return out_state

#-- Enc_Round
def Enc_Round(in_state, rkey):
   out_state = [0] * len(in_state)
   out_state = AR(in_state, rkey)
   in_state = SB(out_state)
   out_state = LM(in_state)
   return out_state

#- TC1 Encryption
def TC1_Enc(PT, key):
   NROUND = NUM_ROUND # Number of Round = 10
   CT = PT #CT = [0] * len(PT)
   for i in range(NROUND):
      CT = Enc_Round(CT, key)

   return CT

#---------------------------------------
#  Decryption
#---------------------------------------

#-- SB: Sbox layer
def ISB(in_state):
   out_state = [0, 0, 0, 0]
   for i in range(len(in_state)):
      out_state[i] = ISbox[in_state[i]]
   return out_state

#-- Decrypt Round
def Dec_Round(in_state, rkey):
   out_state1 = [0, 0, 0, 0]
   out_state2 = [0, 0, 0, 0]
   out_state3 = [0, 0, 0, 0]
   out_state1 = LM(in_state)
   out_state2 = ISB(out_state1)
   out_state3 = AR(out_state2, rkey)
   return out_state3
```

```python
#- TC1 Decryption
def TC1_Dec(input_state, key):
   state = input_state
   numRound = NUM_ROUND
   for i in range(0, numRound):
      state = Dec_Round(state, key)
   return state

#=======================================
def main():
   message = 'ARIA'
   PT = [ ord(ch) for ch in message ]
   print('Message =', message)
   print('PT =', PT)

   key = [0, 1, 2, 3]
   CT = TC1_Enc(PT, key)
   print('CT =', CT)

   hexPT = [hex(item) for item in PT]
   hexCT = [hex(item) for item in CT]

   print('hexPT =', hexPT)
   print('hexCT =', hexCT)

   bytePT = bytes(PT)
   byteCT = bytes(CT)
   print('bytePT =', bytePT)
   print('byteCT =', byteCT)

   input_state = [202, 134, 119, 230]
   output_state = TC1_Dec(input_state, key)
   print('input ciphertext =', input_state)
   print('output plaintext =', output_state)

if __name__ == '__main__':
   main()
```

```
user@host:~$ python3 TC1Lib.py
Message = ARIA
PT = [65, 82, 73, 65]
CT = [202, 134, 119, 230]
hexPT = ['0x41', '0x52', '0x49', '0x41']
hexCT = ['0xca', '0x86', '0x77', '0xe6']
bytePT = b'ARIA'
byteCT = b'\xca\x86w\xe6'
input ciphertext = [202, 134, 119, 230]
output plaintext = [65, 82, 73, 65]
```

2.2 TMTO Attack

TC1-TMTO-Table.py

```
#---------------------------------
# TMTO Attack for TC1
# - n: 32-bit, k: 24-bit
# - Parameter: m=t=l=2^8 (mtl = 2^24)
# -- m: number of starting points
# -- l: number of tables
# -- t: length of chain
# - Memory = m*l = 2^16
# - Time   = t*l = 2^16
#---------------------------------

import TC1Lib as TC1
import pickle # store variable
import random # generate random number
import copy   # deep copy

#--- int(4bytes) to list 0x12345678 -> [ 0x12, 0x34, 0x56, 0x78 ]
def int2list(n):
   out_list = []
   out_list.append( (n >> 24) & 0xff )
   out_list.append( (n >> 16) & 0xff )
   out_list.append( (n >>  8) & 0xff )
   out_list.append( (n      ) & 0xff )
   return out_list

#--- list to int [ 0x12, 0x34, 0x56, 0x78 ] -> 0x12345678
def list2int(l):
   n = 0
   num_byte = len(l)
   for i in range(len(l)):
      n += l[i] << 8*(num_byte - i -1)
   return n

#--- Save Variable to File
def save_var_to_file(var, filename):
   f = open(filename, 'w+b')
   pickle.dump(var, f)
   f.close()

#--- Load Variable from File
def load_var_from_file(filename):
   f = open(filename, 'rb')
   var = pickle.load(f)
   f.close()
   return var
```

```
#==============================================================
#  TMTO Attack
#==============================================================

# 32-bit Enc/Dec : PT = [*,*,*,*] --> CT = [*,*,*,*]
# 24-bit Key     : key = [0,*,*,*]
key_bit = 24

#---------------------------------
# TMTO Table (Dictionary): { (SP:EP) }
#   #SP = #EP = 2^8,   #chains: m = 2^8, #tables: l = 2^8
#---------------------------------

#---------------------------------
# P0 : Chosen Plaintext
# X_{j+1} = E(P0, X_{j})                  # if k = n
# X_{j+1} = R( E(P0, X_{j}) )             # R: 32-bit -> 24-bit
# SP = X0 -> X1 -> X2 -> ... -> Xt = EP   # Encryption Key Chain
#---------------------------------

#-- Reduction Function
#-- R: 32-bit [*,*,*,*] -> 24-bit [0,*,*,*]
def R(ct):
   next_key = copy.deepcopy(ct)
   next_key[0] = 0
   return next_key

#-- Create Encryption Key Chain
#-- SP : random key (24-bit)
#-- P0 : chosen plaintext (fixed)
#-- t  : length of chain
def chain_EP(SP, P0, t):
   Xj = SP
   for j in range(0,t):
      ct = TC1.TC1_Enc(P0, Xj)
      Xj = R(ct)   # next Xj (32-bit -> 24-bit)
   return Xj

#--- Debugging Chain
def chain_EP_debug_print(SP, P0, t):
   Xj = SP
   print('SP =', SP)
   for j in range(0,t):
      ct = TC1.TC1_Enc(P0, Xj)
      Xj = R(ct)   # next Xj
      print(' -> ', ct, ' -> ', Xj)
   return Xj
```

```
#--- Debugging Chain
#--- Xj[0,*,*,*] --> ct[*,*,*,*] --> R(ct)[0,*,*,*]
def chain_EP_debug_file(SP, P0, t, chain_num, table_num):
   file_name = 'debug/TMTO-chain-' + str(table_num) + '-' + str(chain_num) +
    ↪  '.txt'
   f = open(file_name, 'w+')
   Xj = SP
   f.write('SP = [0, %d, %d, %d] \n', %(Xj[1], Xj[2], Xj[3]))

   for j in range(0,t):
      ct = TC1.TC1_Enc(P0, Xj)
      Xj = R(ct)
      f.write(' --> [%d, %d, %d, %d] ' %(ct[0], ct[1], ct[2], ct[3]))
      f.write(' --> [%d, %d, %d, %d] \n' %(Xj[0], Xj[1], Xj[2], Xj[3]))
   f.close()

   return Xj

#--------------------------------
# Create one TMTO Table (Number=ell)
# Input:
#       P0: chosen plaintext (fixed)
#        m: number of SPs (row)         m=2^8: SP1 ~ SP2^8
#        t: length of chain (column)    j=0, ... , j=t
#      ell: table number                ell = 0 ~ 255
# Output:
#     dic : { (Key=EP, Value=SP) }
#     path: ./tmto_table/TMTO-ell.dic
#--------------------------------

def make_one_tmto_table(P0, m, t, ell):
   tmto_dic = {}  # (Key, Value) = (EP,SP)
   for i in range(0,m):
      # random starting point
      SP = [0, random.randint(0,255), random.randint(0,255),
       ↪  random.randint(0,255) ]
      EP = chain_EP_debug_file(SP, P0, t, i, ell)
      #EP = chain_EP(SP, P0, t)

      # { (Key=EP, Value=SP) }
      SP_int = list2int(SP)
      EP_int = list2int(EP)
      # EP is Key
      tmto_dic[EP_int] = SP_int

   # files: TMTO-0, TMTO-1, ... , TMTO-255
   file_name = 'tmto_table/TMTO-' + str(ell) + '.dic'
   save_var_to_file(tmto_dic, file_name)
```

```
#---------------------
# Create total TMTO
# Input:
#   P0: Fixed Plaintext
#   m: no. rows (number of chain)
#   t: no. cols (length of chain)
#   num_of_tables: 2^8 (=256)
#---------------------

def make_all_tmto_tables(P0, m, t, num_of_tables):
   print('Making TMTO tables', end='')
   for ell in range(0, num_of_tables):
      make_one_tmto_table(P0, m, t, ell)
      print('.', end='', flush=True)

   print('\n All TMTO tables are created.')

#=====================
# Test Run

#random.seed(1234)  #fixed seed --> identical result
random.seed(2024)  #fixed seed --> identical result

# Fixed Plaintext
P0 = [2,2,5,0]
# Setup Parameter
m = 256             # m: number of chain
t = 256             # t: length of chain
num_of_tables = 256

#=====================
# (Step 1) Create TMTO Table (Pre-computation)
# TMTO-0, TMTO-1, ...
#=====================
make_all_tmto_tables(P0, m, t, num_of_tables)
```

```
user@host:~@ python3 TC1-TMTO-Table.py
TMTO Table Generation...
Greating table 82/256: [########----------------------] 31%
```

```
All TMTO tables are created successfully!
```

TC1-PTCT.py

```python
#-------------------------
# TC1 - Create PT and CT
#-------------------------
import TC1Lib as TC1

pt1 = [2, 2, 5, 0]  # P0 used in TMTO
key = [0, 218, 190, 65]  # TMTO-chain-89-237
ct1 = TC1.TC1_Enc(pt1, key)

pt2 = [3, 19, 37, 57]
ct2 = TC1.TC1_Enc(pt2, key)

print('PTCT for TMTO attack')

print('pt1 =', pt1)
print('ct1 =', ct1)

print('pt2 =', pt2)
print('ct2 =', ct2)

print('key =', key)
```

```
user@host:~@ python3 TC1-PTCT.py
PTCT for TMTO attack
pt1 = [2, 2, 5, 0]
ct1 = [135, 9, 44, 221]
pt2 = [3, 19, 37, 57]
ct2 = [150, 236, 50, 83]
key = [0, 218, 190, 65]
```

TC1-TMTO-Attack.py

```
...

# Chosen Plaintext (Fixed on TMTO Table)
P0 = [2,2,5,0]
# Parameter for Attack
m = 256             # m: Number of Chains over One Table
t = 256             # t: Length of Chain
num_of_tables = 256 # Number of Tables

#======================
# (Step 2) Attack
#======================

'''
PTCT for TMTO attack
pt1 = [2, 2, 5, 0]
ct1 = [135, 9, 44, 221]
pt2 = [3, 19, 37, 57]
ct2 = [150, 236, 50, 83]
key = [0, 218, 190, 65]
'''

#--------------
# Key Search for one Table
def one_tmto_table_search(ct, P0, m, t, ell):
   key_candid_list = []
   file_name = f'tmto_table/TMTO-{ell}.dic'
   #file_name = 'tmto_table/TMTO-' + str(ell) + '.dic'
   tmto_dic = load_var_from_file(file_name)

   Xj = R(ct)
   current_j = t
   for idx in range(0,t):
      Xj_int = list2int(Xj)

      if Xj_int in tmto_dic: # Is Xj in EP?
         SP = int2list(tmto_dic[Xj_int]) # dic = { EP:SP }
         key_guess = chain_EP(SP, P0, current_j - 1)
         key_candid_list.append(key_guess)

      new_ct = TC1.TC1_Enc(P0,Xj)
      Xj = R(new_ct)
      current_j = current_j - 1

   return key_candid_list
```

```
#================

ct1 = [135, 9, 44, 221] # (random.seed(2024))
key_pool = []
print("TMTO Attack", end='')
for ell in range(0, num_of_tables):
   key_list = one_tmto_table_search(ct1, P0, m, t, ell)
   key_pool += key_list
   print('.', end='')

print('\n Attack complete!\n')
print('key_pool =', key_pool)

# Choose final key through anther PT-CT pair
pt2 = [3, 19, 37, 57]
ct2 = [150, 236, 50, 83] # (random.seed(2024))
final_key = []

for key in key_pool:
   ct_result = TC1.TC1_Enc(pt2, key)
   if ct_result == ct2:
   final_key.append(key)

print('Final key =', final_key)
```