

암호분석 HW2 1번-20192233 박진철

(a)

```
40 key_bit = 24 # 키공간 24비트 key = [0,*,*,*]
41 #-----
42 # 암호문(32비트)을 다음 단계 암호키(24비트)로 만드는 함수
43 # R: 32비트 -> 24비트
44 def R(ct):
45     #next_key = ct
46     next_key = copy.deepcopy(ct)
47     next_key[0] = 0
48     return next_key
49 #-----
```

키 공간을 24비트로 만들고, 암호문을 24비트의 암호키로 변환하는 함수 R 생성

```
51 #-----
52 # Encryption key chain 만들기
53 # SP = (24비트 랜덤키)
54 # P0 = (선택평문, 고정값)
55 # t = 체인의 길이
56 def chain_EP(SP, P0, t):
57     Xj = SP
58     for j in range(0,t):
59         ct = TC20.TC20_Enc(P0, Xj)
60         Xj = R(ct) # next Xj (출력 암호문 32비트를 암호키 24비트로)
61     return Xj
62 #-----
```

시작점 SP에 대하여 길이 t인 체인을 만드는 함수 chain_EP 생성

```
64 #-----
65 # TMT0 테이블 한개 만들기 (번호=ell)
66 # 입력:
67 #     P0: 선택(고정)평문
68 #     m: #SP (행의 개수)      m=2^8: SP1 ~ SP2^8
69 #     t: 체인의 길이(열)      j=0, ..., j=t
70 #     ell: 테이블 번호        ell = 0 ~ 255
71 # 출력:
72 #     사전 : { (Key=EP, Value=SP) }
73 #     저장위치: ./tmt0_table/TMT0-ell.dic
74 def make_one_tmt0_table(P0, m, t, ell):
75     tmt0_dic = {} # (SP,EP), 정렬기준 EP (EP를 검색하기 위해)
76     for i in range(0,m):
77         # 랜덤한 시작점
78         SP = [0, random.randint(0,255), random.randint(0,255), random.randint(0,255) ]
79         EP = chain_EP(SP, P0, t)
80
81         # 정수로 만들어 (SP, EP)를 사전에 넣는다 { (Key=EP, Value=SP) }
82         SP_int = list2int(SP)
83         EP_int = list2int(EP)
84         tmt0_dic[EP_int] = SP_int
85     # 만든 테이블 사전을 파일로 저장한다
86     # 파일명: TMT0-0, TMT0-1, ..., TMT0-255
87     file_name = 'tmt024_table/TMT0-' + str(ell) + '.dic'
88     save_var_to_file(tmt0_dic, file_name)
89 #-----
```

랜덤한 시작점 SP를 Chain_EP에 넣어 EP를 생성하고, SP와 EP를 사전 형태로 파일로 저장하는 함수 make_one_tmt0_table 생성

```

91 #-----
92 # TMTO 테이블 전체 만들기
93 # 입력:
94 #   P0: 고정평문
95 #   m: 행(row)의 개수 (체인의 개수)
96 #   t: 열(col)의 개수 (체인의 길이)
97 #   num_of_tables: TMTO 테이블 개수 (=256)
98 def make_all_tmto_tables(P0, m, t, num_of_tables):
99     print('making TMTO tables', end='')
100     for ell in range(0, num_of_tables):
101         make_one_tmto_table(P0, m, t, ell)
102         print('.', end='')
103     print('\n All TMTO tables are created.')
104 #-----

```

테이블을 원하는 개수만큼 만들어주는 함수 make_all_tmto_tables 생성

```

106 # 선택평문 (TMTO 테이블 전체에서 고정된 값으로 사용)
107 PT = [1,2,3,4]
108 # 공격 파라미터 설정
109 m = 256 # m: 한 테이블에 들어가는 체인의 개수
110 t = 256 # t: 체인의 길이
111 num_of_tables = 256 # 테이블 개수
112
113 make_all_tmto_tables(PT, m, t, num_of_tables)

```

PT를 [1,1,1,1]로 설정할 경우, 너무 느려지는 현상이 있어, PT를 [1,2,3,4]로 설정











m,t,l=2^8=256으로 설정하여 tmto테이블을 생성

```

PS C:\Users\Jin_Cheol\Desktop\Hw2_20192233막진철> & 'C:\Users\Jin_Cheol\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\Jin_Cheol\.vscode\extensions\ms-py
thon.python-2023.8.0\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '14164' '--' 'C:\Users\Jin_Cheol\Desktop\Hw2_20192233막진철\TMTO24.py'
making TMTO tables.....

```

이름

-  TMTO-0.dic
-  TMTO-1.dic
-  TMTO-2.dic
-  TMTO-3.dic
-  TMTO-4.dic
-  TMTO-5.dic
-  TMTO-6.dic
-  TMTO-7.dic
-  TMTO-8.dic
-  TMTO-9.dic

테이블이 생성되어 tmto24_table폴더에 저장됨

(b)

```
66 #-----
67 # 한개의 테이블에 대한 키 탐색 함수
68 def one_tmto_table_search(ct, P0, m, t, ell):
69     key_candid_list = []
70     file_name = 'tmto24_table/TMTO-' + str(ell) + '.dic'
71     tmto_dic = load_var_from_file(file_name)
72
73     Xj = R(ct)
74     current_j = t
75     for idx in range(0,t):
76         Xj_int = list2int(Xj)
77
78         if Xj_int in tmto_dic: # Xj가 EP에 있는가?
79             SP = int2list(tmto_dic[Xj_int]) # dic = { EP:SP }
80             key_guess = chain_EP(SP, P0, current_j - 1)
81             key_candid_list.append(key_guess)
82
83             new_ct = TC20.TC20_Enc(P0,Xj)
84             Xj = R(new_ct)
85             current_j = current_j - 1
86
87     return key_candid_list
88 #-----
```

한 개의 테이블에서 후보키들을 찾는 함수 one_tmto_table_search 생성

```
90 #-----
91 def attack(PT,num_of_tables):
92     #찾아야 할 키 생성
93     key=[0, random.randint(0,255), random.randint(0,255), random.randint(0,255)]
94     C1=TC20.TC20_Enc(PT,key) #찾아야할 키로 PT를 암호화
95     P2=[5,6,7,8] #key 후보들 중에서 정확한 키를 찾아줄 새로운 평문 생성
96     C2=TC20.TC20_Enc(P2,key)
97     print('key=',key)
98     print('CT1=',C1)
99     print('CT2=',C2)
100
101     key_pool = [] #키 후보들
102     print("TMTO Attack", end='')
103     for ell in range(0, num_of_tables):
104         key_list = one_tmto_table_search(C1, PT, m, t, ell)
105         key_pool += key_list #key_pool에 후보키들이 들어감
106         print('.', end='')
107     print('Attack complete!')
108     print('key_pool =', key_pool[0:4])
109
110     final_key = [] #정확한 키
111     for key in key_pool:
112         ct_result = TC20.TC20_Enc(P2, key)
113         if ct_result == C2: #C2와 후보키로 암호화한 ct_result가 같다면
114             final_key.append(key) #후보키를 찾아야할 키로 결정
115
116     print('Final key =', final_key)
117     if final_key:
118         return True #키를 찾았다면, True를 반환
119 #-----
```

attack 함수에서는 후보키 중에서 정확한 키를 추출함

선택평문, 새로운 평문을 이용하여, tmto공격을 수행

키를 찾았다면, True를 반환

```

121 # 선택평문 (TMT0 테이블 전체에서 고정된 값으로 사용)
122 PT = [1,2,3,4]
123 # 공격 파라미터 설정
124 m = 256 # m: 한 테이블에 들어가는 체인의 개수
125 t = 256 # t: 체인의 길이
126 num_of_tables = 256 # 테이블 개수
127
128 counter=0 #성공 횟수
129 for i in range(100):
130     print("attack ",i+1)
131     chk=attack(PT,num_of_tables) #키를 찾았는지를 확인
132     if chk==True:
133         counter+=1 #키를 찾았다면, 성공 횟수에 추가
134     print('counter=',counter,'\n')
135
136 print("성공 확률=", counter/100)

```

tmt0공격을 하여, 성공횟수를 기록하여 성공 확률을 계산

```

attack 1
key= [0, 26, 68, 198]
CT1= [214, 255, 4, 243]
CT2= [14, 0, 241, 1]
TMT0 Attack.....
.....Attack complete!
key_pool = [[0, 153, 66, 154], [0, 92, 23, 128], [0, 92, 23, 128], [0, 153, 66, 154]]
Final key = [[0, 26, 68, 198], [0, 26, 68, 198]]
counter= 1

```

찾아질 경우, counter에 1씩 더해짐

```

attack 100
key= [0, 36, 107, 253]
CT1= [198, 76, 192, 43]
CT2= [107, 207, 77, 94]
TMT0 Attack.....
.....Attack complete!
key_pool = [[0, 208, 26, 121], [0, 45, 3, 104], [0, 103, 58, 22], [0, 46, 145, 25]]
Final key = []
counter= 10
성공 확률= 0.1

```

공격 결과, 성공 횟수 10회로, 성공 확률이 약 10%라는 것을 알 수 있음

(c)

```

54 #-----
55 # chain에서 암호키들을 리스트에 저장
56 def chain_EP(SP, P0, t):
57     key_list=[]
58     Xj = SP
59     for j in range(0,t):
60         key_list.append(Xj)
61         ct = TC20.TC20_Enc(P0, Xj)
62         Xj = R(ct) # next Xj (출력 암호문 32비트를 암호키 24비트로)
63     return key_list
64 #-----

```

chain 생성 단계에서 암호키들을 저장

```

66 #-----
67 # 한개의 테이블에서 얻어지는 모든 암호키를 리스트로 저장
68 #저장한 암호키를 이용하여 한개의 테이블의 ECR 반환
69 def make_one_tmto_table(P0, m, t, ell):
70     table_list = []
71     table_dic={}
72     for i in range(0,m):
73         # 랜덤한 시작점
74         SP = [0, random.randint(0,255), random.randint(0,255), random.randint(0,255) ]
75         key_list=chain_EP(SP,P0,t)
76         for i in range(0,len(key_list)):
77             table_list.append(key_list[i])
78     for i in range(0,len(table_list)):
79         key_int=list2int(table_list[i])
80         table_dic[key_int]=i
81     H=len(table_dic)
82     ECR=H/(m*t)
83     return ECR
84 #-----

```

한 개의 테이블을 만들 때, 테이블에서 나오는 모든 암호키들을 저장

저장한 암호키들 중 중복되는 값을 table_dic라는 사전을 이용하여 제거

사전의 key부분에 암호키들을 넣으면, 중복된 암호키는 사전의 길이에 영향을 주지 않음

->사전의 길이가 중복되지 않은 암호키의 개수

사전의 길이를 통해 테이블의 ECR 계산

```

86 #-----
87 # TMTO 테이블 전체 만들기
88 # 입력:
89 #   P0: 고정평문
90 #   m: 행(row)의 개수 (체인 의 개수)
91 #   t: 열(col)의 개수 (체인의 길이)
92 #   num_of_tables: TMTO 테이블 개수 (=256)
93 def make_all_tmto_tables(P0, m, t, num_of_tables):
94     print('making TMTO tables', end='')
95     a=[]
96     ECR_list=[]
97     for ell in range(0, num_of_tables):
98         ECR=make_one_tmto_table(P0,m,t,ell)
99         ECR_list.append(ECR)
100         print('.',end='')
101
102     print('\n All TMTO tables are created.')
103     return ECR_list
104 #-----

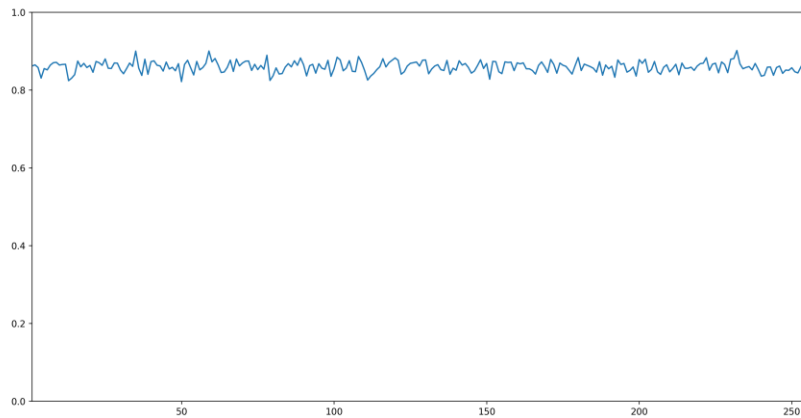
```

전체 테이블들의 ECR을 저장

```
6 from matplotlib import pyplot as plt
```

```
106 # 선택평문 (TMT0 테이블 전체에서 고정된 값으로 사용)
107 PT = [1,2,3,4]
108 # 공격 파라미터 설정
109 m = 256 # m: 한 테이블에 들어가는 체인의 개수
110 t = 256 # t: 체인의 길이
111 num_of_tables = 256 # 테이블 개수
112
113 ECR=make_all_tmto_tables(PT, m, t, num_of_tables)
114 print('ECR=',ECR)
115
116 x=[i for i in range(1,257)]
117 y=ECR
118 plt.ylim(0,1)
119 plt.xlim(1,256)
120 plt.plot(x,y)
121 plt.show()
```

matplotlib를 이용하여 ECR의 그래프 출력



ECR그래프가 위와 같이 생성됨

```
123 ECR_sum=0
124 print(len(ECR))
125 for i in range(0,len(ECR)):
126     ECR_sum+=ECR[i]
127 print("ECR 평균=",ECR_sum/len(ECR))
```

```
ECR 평균 = 0.8597521185874939
```

ECR평균은 약 0.85라는 것을 알 수 있음

(d)

m, t, l을 동일한 값으로 설정하려 함

$m=2^a$, $t=2^b$, $l=2^c$, $a=b=c$ 이고, $a+b+c \leq 32$ 인 경우는 $a=b=c=10$ 인 경우

따라서 $m, t, l=2^{10}$ 으로 설정하여 계산

```

40 key_bit = 32
41
42 #-----
43 # Encryption key chain 만들기
44 #   SP = (24비트 랜덤키)
45 #   P0 = (선택평문, 고정값)
46 #   t = 체인의 길이
47 def chain_EP(SP, P0, t):
48     Xj = SP
49     for j in range(0,t):
50         ct = TC20.TC20_Enc(P0, Xj)
51         Xj = ct
52     return Xj
53 #-----

```

```

55 #-----
56 # TMTO 테이블 한개 만들기 (번호=ell)
57 # 입력:
58 #   P0: 선택 (고정) 평문
59 #   m: #SP (행의 개수)      m=2^8: SP1 ~ SP2^8
60 #   t: 체인의 길이(열)      j=0, ..., j=t
61 #   ell: 테이블 번호        ell = 0 ~ 255
62 # 출력:
63 #   사전 : { (Key=EP, Value=SP) }
64 #   저장위치: ./tmto_table/TMTO-ell.dic
65 def make_one_tmto_table(P0, m, t, ell):
66     tmto_dic = {} # (SP,EP), 정렬기준 EP (EP를 검색하기 위해)
67     for i in range(0,m):
68         # 랜덤한 시작점
69         SP = [random.randint(0,255), random.randint(0,255), random.randint(0,255), random.randint(0,255) ]
70
71         EP = chain_EP(SP, P0, t)
72
73         # 정수로 만들어 (SP, EP)를 사전에 넣는다 { (Key=EP, Value=SP) }
74         SP_int = list2int(SP)
75         EP_int = list2int(EP)
76         tmto_dic[EP_int] = SP_int
77     # 만든 테이블 사전을 파일로 저장한다
78     # 파일명: TMTO-0, TMTO-1, ..., TMTO-1023
79     file_name = 'tmto32_table/TMTO-' + str(ell) + '.dic'
80     save_var_to_file(tmto_dic, file_name)
81 #-----

```

```

83 #-----
84 # TMTO 테이블 전체 만들기
85 # 입력:
86 #   P0: 고정평문
87 #   m: 행(row)의 개수 (체인의 개수)
88 #   t: 열(col)의 개수 (체인의 길이)
89 #   num_of_tables: TMTO 테이블 개수 (=1024)
90 def make_all_tmto_tables(P0, m, t, num_of_tables):
91     print('making TMTO tables', end='')
92     for ell in range(0, num_of_tables):
93         make_one_tmto_table(P0, m, t, ell)
94         print('.', end='')
95     print('\n All TMTO tables are created.')
96 #-----
97
98 # 선택평문 (TMTO 테이블 전체에서 고정된 값으로 사용)
99 PT = [1,2,3,4]
100 # 공격 파라미터 설정
101 m = 1024          # m: 한 테이블에 들어가는 체인의 개수
102 t = 1024          # t: 체인의 길이
103 num_of_tables = 1024 # 테이블 개수
104
105 make_all_tmto_tables(PT, m, t, num_of_tables)

```

TMTO24.py에서 파라미터와 키 비트만 다르게 하여 실행

이름

TMTO-1023.dic
TMTO-1022.dic
TMTO-1021.dic
TMTO-1020.dic
TMTO-1019.dic
TMTO-1018.dic

1024개의 테이블이 생성

```
making TMTO tables.....  
All TMTO tables are created.  
ECR= [0.9618797302246094, 0.9611616134643555, 0.9617824554443359, 0.  
9599542617797852, 0.9616985321044922, 0.9591426849365234, 0.95936012  
26806641, 0.9676198959350586, 0.9654169082641602, 0.9599189758300781  
]  
10  
ECR 평균 = 0.9617935180664062
```

32비트의 ECR을 구해보면 약 0.95가 나옴

$$\begin{aligned} \text{성공률} &\approx 1 - \exp\left(-\frac{\text{len}(\text{ECR})}{N}\right) = 1 - \exp\left(-\frac{2^0 2^0 2^0 \text{ECR}}{2^3}\right) \\ &= 1 - \exp\left(-\frac{\text{ECR}}{2^1}\right) = 1 - \exp\left(-\frac{0.95}{4}\right) = 1 - \exp(-0.23) \\ &= 1 - \frac{1}{e^{0.23}} \approx 1 - \frac{1}{1.25} = 1 - 0.8 = 0.2 \end{aligned}$$

따라서 약 0.2의 공격 성공률로 예상

(e)

```
55 #-----  
56 # 한개의 테이블에 대한 키 탐색 함수  
57 def one_tmto_table_search(ct, P0, m, t, ell):  
58     key_candid_list = []  
59     file_name = 'tmto32_table/TMTO-' + str(ell) + '.dic'  
60     tmto_dic = load_var_from_file(file_name)  
61  
62     Xj = ct  
63     current_j = t  
64     for idx in range(0,t):  
65         Xj_int = list2int(Xj)  
66  
67         if Xj_int in tmto_dic: # Xj가 EP에 있는가?  
68             SP = int2list(tmto_dic[Xj_int]) # dic = { EP:SP }  
69             key_guess = chain_EP(SP, P0, current_j - 1)  
70             key_candid_list.append(key_guess)  
71  
72             new_ct = TC20.TC20_Enc(P0,Xj)  
73             Xj = new_ct  
74             current_j = current_j - 1  
75  
76     return key_candid_list  
77 #-----
```



```

79 #-----
80 def attack(PT,num_of_tables):
81     #찾아야 할 키 생성
82     key=[random.randint(0,255), random.randint(0,255), random.randint(0,255), random.randint(0,255)]
83     C1=TC20.TC20_Enc(PT,key) #찾아야 할 키로 PT를 암호화
84     P2=[5,6,7,8] #key 후보들 중에서 정확한 키를 찾아줄 새로운 평문 생성
85     C2=TC20.TC20_Enc(P2,key)
86     print('key=',key)
87     print('CT1=',C1)
88     print('CT2=',C2)
89
90     key_pool = [] #키 후보들
91     print("TMTO Attack", end='')
92     for ell in range(0, num_of_tables):
93         key_list = one_tmto_table_search(C1, PT, m, t, ell)
94         key_pool += key_list #key_pool에 후보키들이 들어감
95         print('.', end='')
96     print('Attack complete!')
97     print('key_pool =', key_pool[0:4])
98
99     final_key = [] #정확한 키
100    for key in key_pool:
101        ct_result = TC20.TC20_Enc(P2, key)
102        if ct_result == C2: #C2와 후보키로 암호화한 ct_result가 같다면
103            final_key.append(key) #후보키를 찾아야 할 키로 결정
104
105    print('Final key =', final_key)
106    if final_key:
107        return True #키를 찾았다면, True를 반환
108    #-----

```

```

110 # 선택평문 (TMTO 테이블 전체에서 고정된 값으로 사용)
111 PT = [1,2,3,4]
112 # 공격 파라미터 설정
113 m = 1024 # m: 한 테이블에 들어가는 체인의 개수
114 t = 1024 # t: 체인의 길이
115 num_of_tables = 1024 # 테이블 개수
116
117 counter=0 #성공 횟수
118 for i in range(100):
119     print("attack ",i+1)
120     chk=attack(PT,num_of_tables) #키를 찾았는지를 확인
121     if chk==True:
122         counter+=1 #키를 찾았다면, 성공 횟수에 추가
123         print('counter=',counter,'\n')
124
125 print("성공 확률=", counter/100)

```

24비트 TMTO공격에서 파라미터와 키 크기만 변경하여 공격 수행

```

attack 36
key= [119, 130, 227, 94]
CT1= [62, 112, 152, 250]
CT2= [280, 94, 42, 89]
TMTO Attack.....
.....
.....Attack complete!
key_pool = [[119, 130, 227, 94], [231, 118, 220, 159], [249, 150, 108, 92], [248, 10, 168, 50]]
Final key = [[119, 130, 227, 94], [119, 130, 227, 94], [119, 130, 227, 94]]
counter= 1

```

성공할 경우 counter에 1씩 추가됨

```

.....Attack complete!
key_pool = [[141, 89, 19, 62], [3, 90, 210, 142], [131, 40, 129, 86], [36, 185, 149, 125]]
Final key = []
counter= 2
성공 확률= 0.02

```

공격 결과, 성공 확률이 약 2%라는 것을 알 수 있음->추정했던 값보다는 적은 성공률이 나옴