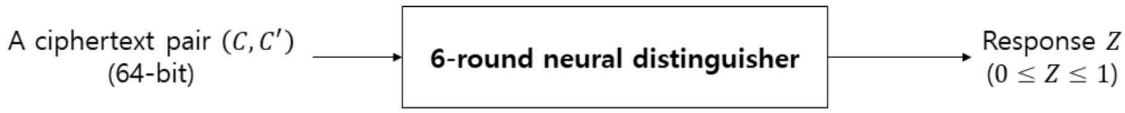
2023 암호분석경진대회

2번 문제

1. Description

주어진 딥러닝 모델(Neural distinguisher)은 6-라운드로 축소된 SPECK-32/64로 암호화된 암호문을 구별하도록 학습된 모델이다. 제시된 6-round neural distinguisher는 다음 그림과 같이 동작하며 약 75%의 구별 정확도를 보인다.



우리는 위의 neural distinguisher를 기반으로 7-라운드 SPECK-32/64에 대한 키 복구 공격을 수행하려고 한다. 다음 물음에 답하고 키 복구 공격을 수행하시오.

2. Solution

1) Neural distinguisher의 학습 데이터셋은 두 가지의 라벨(0과 1)로 구성되었다. 라벨이 1일 경우에는 암호문 쌍 (C, C')에 대응하는 평문 쌍 (P, P')이 P⊕P' = 0x0040/0000을 만족하며, P⊕P' ≠ 0x0040/0000인 암호문 쌍들은 라벨이 0으로 구성된다. (암호화 키는 각 데이터 샘플에 대하여 무작위로 생성된 키가 사용되었다.) 이 데이터셋을 학습한 neural distinguisher의 출력 가 의미하는 바를 설명하시오. (δ = 0x0040/0000)

문제에 제시된 neural distinguisher는 암호문 쌍을 입력값으로 받아 0과 1사이의 값을 출력한다. 이 Neural distinguisher의 학습 데이터셋은 두 가지의 라벨(0과 1)로 구성되었다. 라벨이 1일 경우에는 암호문 쌍 (C, C')에 대응하는 평문 쌍 (P, P')이 P⊕P' = 0x0040/0000을 만족하며, P⊕P' ≠ 0x0040/0000인 암호문 쌍들은 라벨이 0으로 구성된다. 다시 말해 이 neural distinguisher의 학습 데이터는 입력(평문) 차분이 δ일 때 출력 데이터에 해당하는 Label이 1로, 그렇지 않을 경우에는 Label이 0으로 정의되어 있다. 따라서 주어진 AI model은 입력(평문) 차분이 δ와 같을 경우 1을, 그렇지 않을 경우 0을 출력하도록 학습된 것을 알 수 있다.

AI model 의 출력값이 $0 \leq Z \leq 1$ 인 것과 각 라벨이 0 또는 1인점을 토대로 Z의 값이 1에 가까울수록 입력(평문) 차분이 δ일 가능성이 높아진다고 추측할 수 있다. 이에 아래와 같은 코드를 작성하여 Z의 의미를 검증하였다.

```
# Model import
from keras.models import model_from_json
arch =open('arch_neural_distinguisher.json')
json_arch =arch.read()
nr6_speck_distinguisher =model_from_json(json_arch)
nr6_speck_distinguisher.load_weights('weights_nr6_speck.h5')
nr6_speck_distinguisher.compile(optimizer='adam',loss='mse',metrics=['acc'])

# Model test
import speck as sp
size =10 **6
# x_test, y_test = sp.real_differences_data(size, 6)
x_test,y_test =sp.make_train_data(size,6)
res =nr6_speck_distinguisher.predict(x_test)
correct =0
for i,j in zip(res,y_test):
```

```
    if round(float(i[0]))==y:
        correct +=1
results = nr6_speck_distinguisher.evaluate(x_test,y_test,batch_size=10000)
print('test loss, test_acc: ',results)
print('Calculated acc: ',correct/size)
if round(results[1],6)==correct/size:
    print("The result of distinguisher is the prob of Y_label = 1")
```

코드 1. Z 추측 검증 코드

주어진 nr6_speck_dictinguisher의 내장 함수인 predict 함수를 사용하여 입력데이터에 대한 Z 값을 구했고, Z 값을 반올림하여 Label과 비교함으로써 nr6_speck_dictinguisher의 정확도를 측정하였다. 다시 말해 Z값이 0 이상 0.5 이하인 경우는 Label이 0일 가능성이 높으므로 주어진 입력의 Label이 0 이라고 예측하고, 0.5 초과 1 이하인 경우는 Label이 1일 가능성이 높으므로 주어진 입력의 Label이 1 이라고 예측한 것이다.

문제에 있는 코드의 evaluate 함수는 자체적으로 전체 입력데이터와 라벨를 이용해 모델의 정확도를 계산한다. 우리가 직접 측정한 정확도 correct / size (correct는 Z의 반올림 값이 Y Label과 일치하는 경우의 수, size는 전체 경우의 수)가 evaluate 함수에서 자체적으로 계산한 accuracy 값과 일치하는 것을 확인할 수 있었다.(전체 경우의 수가 10^6이기에 두 값을 비교할 때 소수점 6번째 자리로 반올림 해주었다.). 따라서 추측한 Z 의 의미가 해당 모델의 개발자들이 의도한 Z의 의미와 일치하다는 것을 확신할 수 있었다.

```
~/De/C/2/2/2/nr6_speck_neural_distinguisher ❯ python3 1.py                    04:
31250/31250 [==============================] - 16s 523us/step
100/100 [==============================] - 2s 22ms/step - loss: 0.1472 - acc: 0.7861
test loss, test_acc:  [0.14722926914691925, 0.7861019968986511]
Calculated acc:  0.786102
The result of distinguisher is the prob of Y_label = 1
~/De/C/2/2/2/nr6_speck_neural_distinguisher ❯                                 28s 04:
```

그림 1. train data를 이용한 검증 결과

```
~/De/C/2/2/2/nr6_speck_neural_distinguisher ❯ python3 1.py                    8s 04:
31250/31250 [==============================] - 16s 511us/step
100/100 [==============================] - 3s 24ms/step - loss: 0.2946 - acc: 0.6018
test loss, test_acc:  [0.2946391999721527, 0.6017670035362244]
Calculated acc:  0.601767
The result of distinguisher is the prob of Y_label = 1
~/De/C/2/2/2/nr6_speck_neural_distinguisher ❯                                 28s 04:
```

그림 2. real data를 이용한 검증 결과

2) Neural distinguisher의 출력 Z를 활용하여, 키 복구 공격 시에 추측된 키가 옳은 키인지를 평가할 수 있는 수식을 구성하고 그 원리를 설명하시오. (δ = 0x0040/0000)

위에서 본 바와 같이 주어진 neural distinguisher는 암호문 쌍을 입력값으로 받아 해당 암호문 쌍이 특정한 입력(평문) 차분을 가질 가능성을 출력한다. 주어진 distinguisher의 경우는 6-round SPECK-32/64 에 대한 입력(평문) 차분이 δ 인지의 여부를 판단하는 것이기 때문에 Neural distinguisher의 출력 Z를 활용하여 7-라운드 SPECK-32/64에 대한 키 복구 공격을 수행한다는 것은 7-round-speck cipher에서 final round key를 복구한다는 뜻으로 해석할 수 있다.

키를 복구 공격은 대체로 특정 키가 옳은 키일 점수을 계산하는 수식을 만들어, 모든 키에 대한 점수를 구한 후에, 이 점수가 가장 높은 키를 구하는 방식으로 이루어진다. 문제에서 명시된 7-round speck cipher의 경우에, 1개의

라운드에 쓰이는 모든 key의 경우의 수는 2^{16} 이기 때문에, 충분히 전수조사를 통해서 모든 key에 대한 점수를 구할 수 있다고 판단했다.

추가적으로, 키 복구 공격 시 제 3자는 암호 알고리즘 내부의 값들을 알아낼 수 없다. 따라서 입력값과 출력값만을 이용한 분석을 통해 어떤 키가 옳은 키인지를 판별할 수 있어야 한다. 즉, 어떤 키가 올바른 키가 될 확률을 구하기 위한 수식에는 입출력 데이터만이 포함되어야 한다.

final round key 가 주어져 있다면, 우리는 주어진 출력 데이터를 이용해서 한 라운드를 복호화할 수 있을 것이다. 그렇게 되면 원래 입력 데이터로부터 6 round 암호화된 데이터들을 얻어낼 수 있을 것이고, 이때 6-round nerual distinguisher를 사용하게 되면, 입력(평문) 차분이 주어진 δ 와 같은지 구분할 수 있다.

입력(평문) 차분이 δ 라고 생각해보자. key schedule을 통해 만들어진 임의의 key 로 이 평문이 암호화되었을 때, 어떤 final round key로 이를 한 라운드 복호화하고, 이를 nerual distinguisher의 입력값으로 넣어주게 되면, 평문의 차분이 δ인 가능성 Z가 결과값으로 나올 것이다. 이를 final round key의 개별 점수라고 정의하자. 이 결과값은 약 75%의 정확도를 가지고 있기에, 이를 여러 번 반복하게 되면 충분히 의미있는 값을 계산해낼 수 있겠다.

따라서 문제의 참고 논문에 제시된 다음의 공식을 사용하여 각각의 개별 점수를 키에 대한 점수로 결합했다.

$$v_k = \sum_{i=1}^{n} \log_2(Z_{i,k}/(1-Z_{i,k}))$$

이를 키가 옳은 키인지를 평가할 수 있는 점수로 정의한다. 해당 점수가 높을수록 옳은 키일 확률이 높다.

이를 코드 2와 같이 구현했다. 입력(평문) 차분이 δ 인 pair_num개의 입력값을 이용해서 key 분석을 진행했고, 각 입력에 대한 Z값, 즉 key의 개별 점수를 구하는 과정은 calc_prob 함수로 구현하였다. score_for_key 함수에서는 개별 점수를 이용해 키에 대한 점수를 계산해주었고, 이렇게 구한 값은 score 변수에 저장한다.

```
# Model import
from keras.models import model_from_json

arch =open('arch_neural_distinguisher.json')
json_arch =arch.read()
nr6_speck_distinguisher =model_from_json(json_arch)
nr6_speck_distinguisher.load_weights('weights_nr6_speck.h5')
nr6_speck_distinguisher.compile(optimizer='adam',loss='mse',metrics=['acc'])

# Model test
import speck as sp
import numpy as np
import os
import math
from Crypto.Util.number import *

## logic should be INPUT : (C, C') + KEY --> OUTPUT : (Correct Key!!, Wrong Key!!)
## A model that predicts whether a ciphertext is encrypted with a given key

delta =[0x0040,0000]
```

```
21
22 def calc_prob(p1,p2,c1,c2,final_key):
23     assert p1[0] ^ p2[0] == delta[0] and p1[1] ^ p2[1] == delta[1]
24
25     rev_c1,rev_c2 = sp.dec_one_round(c1,final_key),sp.dec_one_round(c2,final_key)
26     tmp = [None for _ in range(4)]
27
28     tmp[0] = np.array([rev_c1[0]],dtype=np.uint16)[np.newaxis,:]
29     tmp[1] = np.array([rev_c1[1]],dtype=np.uint16)[np.newaxis,:]
30     tmp[2] = np.array([rev_c2[0]],dtype=np.uint16)[np.newaxis,:]
31     tmp[3] = np.array([rev_c2[1]],dtype=np.uint16)[np.newaxis,:]
32
33     X = sp.convert_to_binary(tmp)
34     res = nr6_speck_distinguisher.predict(X)
35
36     return res[0][0]
37
38 pair_num = 10
39
40 key = [bytes_to_long(os.urandom(2)) for _ in range(4)]
41 key = sp.expand_key(key,7)
42
43 fk = key[-1]
44
45 def score_for_key(key,final_key):
46     score = 0
47     for n in range(pair_num):
48         pt1 = [bytes_to_long(os.urandom(2)) for _ in range(2)]
49         pt2 = [pt1[0] ^ delta[0],pt1[1] ^ delta[1]]
50
51         ct1 = sp.encrypt(pt1,key)
52         ct2 = sp.encrypt(pt2,key)
53
54         Z = calc_prob(pt1,pt2,ct1,ct2,final_key)
55         score += math.log2(Z/(1-Z))
56
57     return score
58
59 print(score_for_key(key,fk))
60 print(score_for_key(key,bytes_to_long(os.urandom(2))))
```

코드 2. 임의의 키에 대한 점수 코드

그림 3과 같이 암호화 key 에 대응되는 final key와 랜덤값의 score을 구해봄으로써 주어진 수식이 올바르다고 확신할 수 있었다. 옳은 키 값의 score는 양수이고 틀린 키 값의 score는 음수로 score에 확연한 차이가 있기 때문이다.

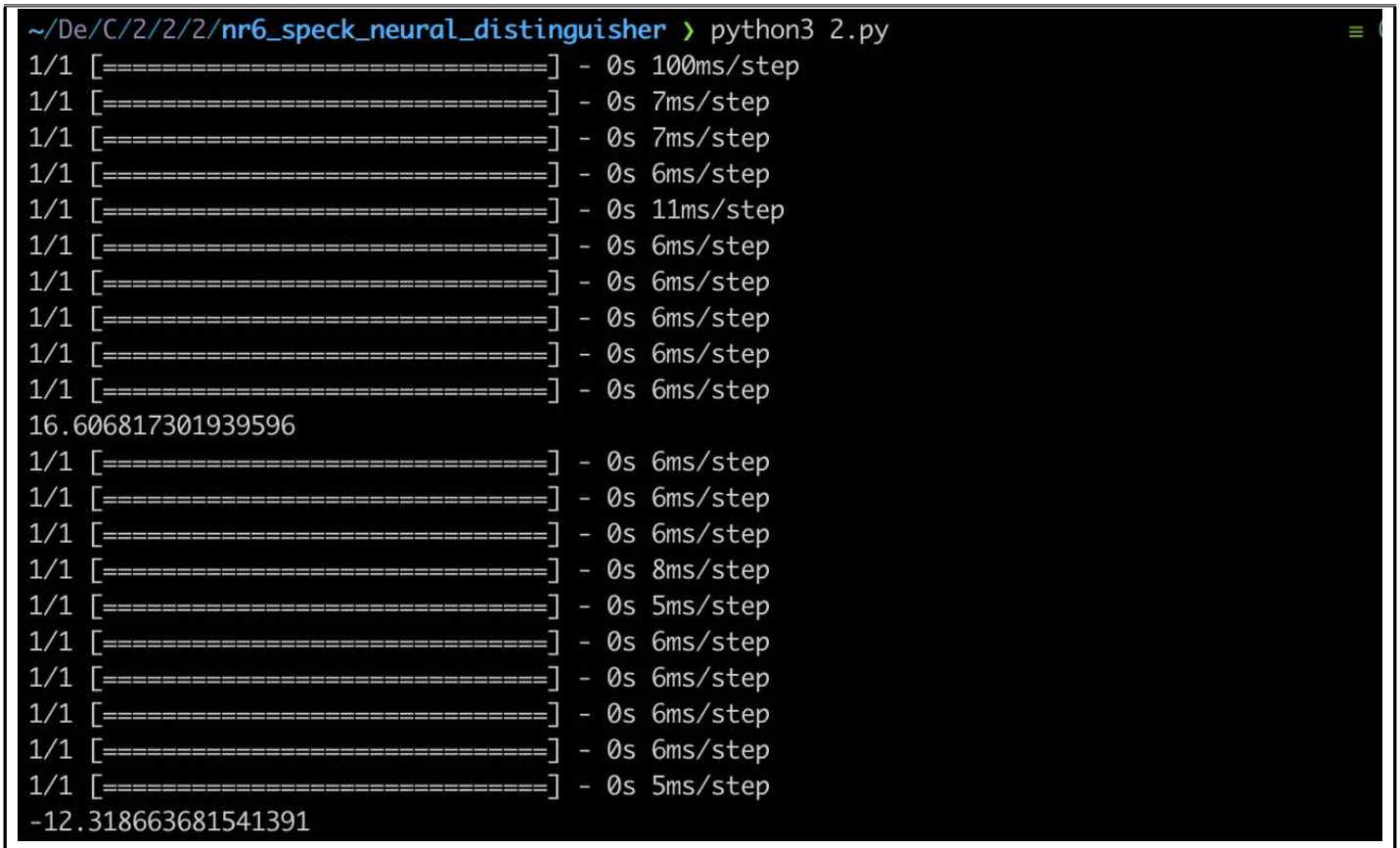


그림 3. 코드 실행 결과: 옳은 키의 점수(위)와 랜덤값의 점수(아래)

3) 2)에서 구성한 수식을 기반으로 7-라운드 SPECK-32/64 서브키를 복구하기 위한 알고리즘을 구성 및 설명하고 구현 결과를 제시하시오. (여기서 키 복구 공격 구현을 위한 평문과 키는 임의로 설정하며, 구현 결과가 키 를 복구하는 것을 확인할 수 있도록 구성하시오.)

3번 문제는 2번에서 구성한 수식을 통해서 전수조사를 진행하면 키를 복구할 수 있다. 가능한 모든 key 값에 대해서 각각 score를 계산한 후 가장 높은 score를 받은 key이 정당한 키가 된다.

```
from keras.models import model_from_json

arch = open('arch_neural_distinguisher.json')
json_arch = arch.read()
nr6_speck_distinguisher = model_from_json(json_arch)
nr6_speck_distinguisher.load_weights('weights_nr6_speck.h5')
nr6_speck_distinguisher.compile(optimizer='adam',loss='mse',metrics=['acc'])

# Model test
import speck as sp
import numpy as np
import os
from Crypto.Util.number import *
from tqdm import *

```

```
delta =[0x0040,0000]

def calc_prob(p1,p2,c1,c2,final_key):
    assert p1[0]^p2[0]==delta[0] and p1[1]^p2[1]==delta[1]

    rev_c1,rev_c2 =sp.dec_one_round(c1,final_key),sp.dec_one_round(c2,final_key)
    tmp =[None for _in range(4)]

    tmp[0]=np.array([rev_c1[0]],dtype=np.uint16)[np.newaxis,:]
    tmp[1]=np.array([rev_c1[1]],dtype=np.uint16)[np.newaxis,:]
    tmp[2]=np.array([rev_c2[0]],dtype=np.uint16)[np.newaxis,:]
    tmp[3]=np.array([rev_c2[1]],dtype=np.uint16)[np.newaxis,:]

    X =sp.convert_to_binary(tmp)
    res =dr6_speck_distinguisher.predict(X)

    return res[0][0]

pair_num =5

key =[bytes_to_long(os.urandom(2))for _in range(4)]
ks =sp.expand_key(key,7)

def score_for_key(key,final_key):
    score =0
    for n in range(pair_num):
        pt1 =[bytes_to_long(os.urandom(2))for _in range(2)]
        pt2 =[pt1[0]^delta[0],pt1[1]^delta[1]]

        ct1 =sp.encrypt(pt1,key)
        ct2 =sp.encrypt(pt2,key)

        Z =calc_prob(pt1,pt2,ct1,ct2,final_key)
        score +=math.log2(Z/(1-Z))

    return score

scoreboard =list()
for predict_key in trange(pow(2,16)):
        scoreboard.append(score_for_key(key,predict_key))

max_score =max(scoreboard)
calc_key =scoreboard.index(max_score)

print(f"real key is {hex(key[-1])}")
print(f"calculated key is {hex(calc_key)}")
```

코드 3. final round key 에측 코드

코드 2와 같이 final key 로 가능한 key의 개수는 2^{16}이기 때문에 반복문을 이용해서 모든 키의 score를 구해주었고, max(scoreboard)를 찾아서 가장 높은 점수를 받은 key를 출력해주었다.

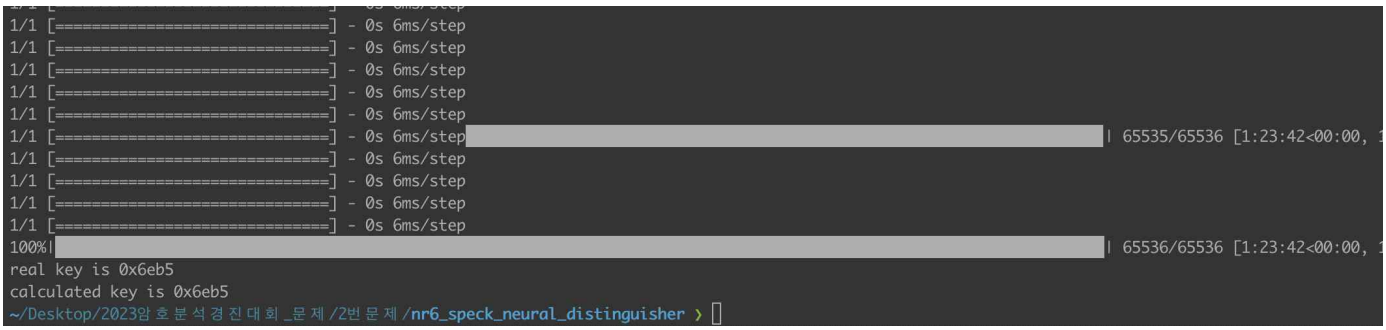


그림 4. 코드 실행 결과

실제로 사용되었던 key와 키복구 알고리즘으로 추측한 key를 출력했더니, 두 키의 값이 동일했다.(그림 4). 따라서 6-round neural distinguisher 모델을 사용하여 7 round speck cipher의 final key를 성공적으로 복구해냈음을 알 수 있다.