

## CS5231 Assignment 1

Edison Tsui Ka Wing  
HT086372B

### Timing Attack

Timing Attack is a type of Side-Channel Attack in which attackers observe the time used by the cryptosystem from a set of cipher/plaintext.

In this assignment, a set of cipher, algorithm and hardware platform are known and we observe the timing needed to decrypt the cipher in order to guess the private key. The attack works as exponentiation is not a fix time operation and the operation time is dependent on the private key. From below, we can see if bit  $i$  in secret key is 1, it will take more time to compute.

Algorithm:

```
temp ← 1;
for i ← 0 to (k -1)
    temp ← (temp * temp ) mod n;
    if (bi == 1 ) then temp = (temp * c) mod n;
end-for-loop
output temp
```

However, as time recorded will be subjected to noise like, other background process may eat up CPU cycle, measurement accuracy, for each key, we try to iterate for large number of time and close all applications in order to average out and reduce the noise mentioned.

### Modification

Original Blackbox.c provided the algorithm and put generated data to standard out. We can use pipe to save all the output data. In addition, as the attack compose of correlation calculation and hypothesis. Blackbox.c is modified as follows

- i) Generate key time mode  
blackbox will take a private key (in integer, will be translated to key bits automatically)  
key time and first key will be output to stdout
- ii) Crack key mode  
blackbox will take a number of bits try ( $k$ , from last key bit to last  $k$  bit), threshold value and input time file. It will hypothesis key and try to guess the private key by finding correlation between guess key and real key.

Algorithm

Set decryptKey to all 0

For  $i = 0$  to  $k-1$

Change bit  $i = 1$

For All cipher

Run Decryption and observe total time taken

End

Calculate correlation between the timing recorded and the timing generated by the

```

new decryptKey.
If correlation > threshold
    Accept key
Else
    Set decryptKey i bit to 0
End
Output decryptKey

```

I have an initial thought that we would only accept hypothesis that correlation is larger than last accepted. It is because for last accepted test, e.g. 0x000...01000, the next test is 0x000....11000, if next test is closer to the real key, correlation should be higher. However, due to the existence of noise, above assumption may be wrong and it is very hard to determine a noise level value, like

```

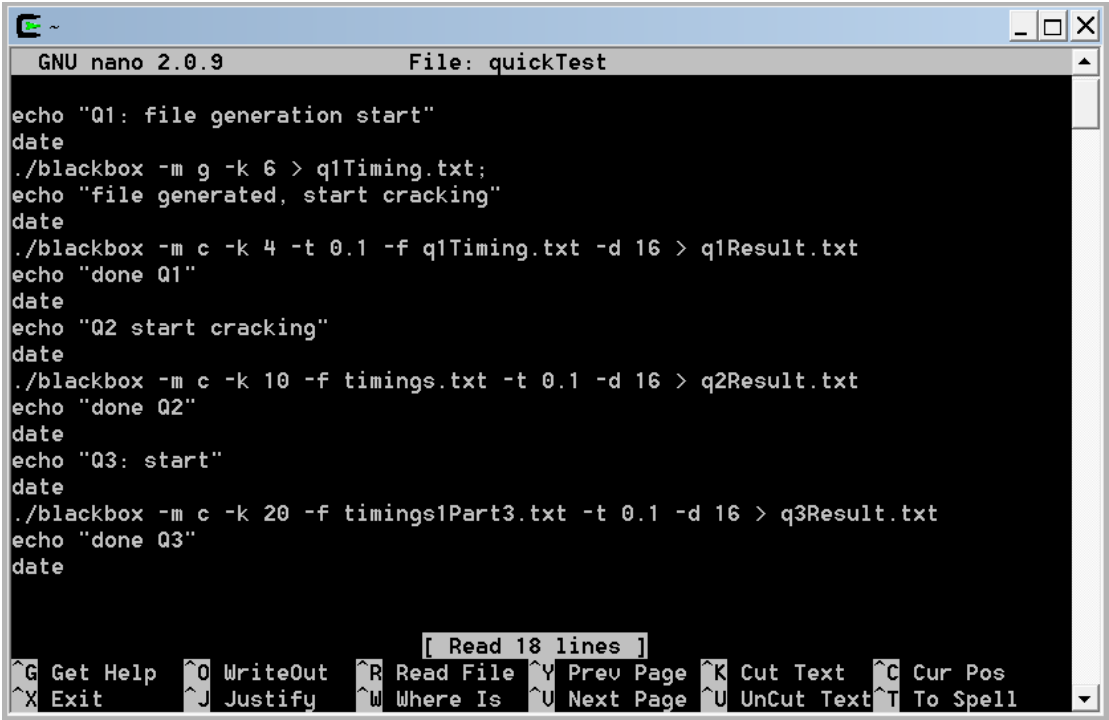
If (correlationthis > correlationlastaccepted - noise)
    {accept}

```

Worst case is we may observe a decreasing correlation as key is guessed, so a fixed threshold is used .

## Results

Bash script is developed to perform test.



```

GNU nano 2.0.9      File: quickTest

echo "Q1: file generation start"
date
./blackbox -m g -k 6 > q1Timing.txt;
echo "file generated, start cracking"
date
./blackbox -m c -k 4 -t 0.1 -f q1Timing.txt -d 16 > q1Result.txt
echo "done Q1"
date
echo "Q2 start cracking"
date
./blackbox -m c -k 10 -f timings.txt -t 0.1 -d 16 > q2Result.txt
echo "done Q2"
date
echo "Q3: start"
date
./blackbox -m c -k 20 -f timings1Part3.txt -t 0.1 -d 16 > q3Result.txt
echo "done Q3"
date

[ Read 18 lines ]
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text    ^C Cur Pos
^X Exit      ^J Justify    ^W Where Is   ^U Next Page  ^U UnCut Text ^T To Spell

```

```
Edison@Edison-notebook ~
$ nano quickTest

Edison@Edison-notebook ~
$ nano quickTest

Edison@Edison-notebook ~
$ ./quickTest
Q1: file generation start
Sun Mar  1 23:16:14 CST 2009
file generated, start cracking
Sun Mar  1 23:37:29 CST 2009
done Q1
Mon Mar  2 01:03:05 CST 2009
Q2 start cracking
Mon Mar  2 01:03:05 CST 2009
done Q2
Mon Mar  2 10:34:19 CST 2009
Q3: start
Mon Mar  2 10:34:19 CST 2009

Edison@Edison-notebook ~
$
```

- 1) We selected a simple key: 00....0100 (0x04) as the key and generated time file part1\_k4.txt. And the key is found by setting threshold = 0.2. Below diagram shows blackbox iterates for 4 bits.

```
Edison@Edison-notebook ~
$ ./genFile

Edison@Edison-notebook ~
$ ./blackbox -m c -k 4 -t 0.2 -f key2_test.txt -d 16
correlation 0, keyVal:1 :0.0578274
cov: 28.338326,
  T: sd: 27.917071, mean: 434.749875,
  M: sd: 17.553790, mean: 96.901951
correlation 1, keyVal:2 :0.124432
cov: 69.445691,
  T: sd: 27.917071, mean: 434.749875,
  M: sd: 19.991351, mean: 270.960980
correlation 2, keyVal:4 :0.256371
cov: 145.794078,
  T: sd: 27.917071, mean: 434.749875,
  M: sd: 20.370451, mean: 433.275138
correlation 3, keyVal:12 :0.165594
cov: 113.669709,
  T: sd: 27.917071, mean: 434.749875,
  M: sd: 24.588339, mean: 601.318159
Key: 4

Edison@Edison-notebook ~
$
```

- 2) 9 hours is used to crack the key, by setting correlation to 0.1 to avoid noise. The key is guessed to be 0x00....0FF, i.e. last 8 bits are 1. Below is the data generated (q2Result.txt).

```
Iteration: 0, keyVal:1, correlation:0.197689
cov: 58.712859,
```

```

    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 15.033179, mean: 96.433717
key accepted, ref: 0.1
Iteration: 1, keyVal:3, correlation:0.228678
cov: 60.631727,
    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 13.420693, mean: 463.351676
key accepted, ref: 0.197689
Iteration: 2, keyVal:7, correlation:0.19521
cov: 101.932674,
    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 26.430806, mean: 831.386193
key accepted, ref: 0.228678
Iteration: 3, keyVal:15, correlation:0.176251
cov: 84.147867,
    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 24.166420, mean: 1182.150575
key accepted, ref: 0.19521
Iteration: 4, keyVal:31, correlation:0.157977
cov: 75.312571,
    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 24.130937, mean: 1543.013007
key accepted, ref: 0.176251
Iteration: 5, keyVal:63, correlation:0.127734
cov: 73.524750,
    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 29.135858, mean: 1906.559780
key accepted, ref: 0.157977
Iteration: 6, keyVal:127, correlation:0.135438
cov: 92.553930,
    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 34.590176, mean: 2270.801901
key accepted, ref: 0.127734
Iteration: 7, keyVal:255, correlation:0.115746
cov: 82.876573,
    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 36.243182, mean: 2633.513757
key accepted, ref: 0.135438
Iteration: 8, keyVal:511, correlation:0.0941415
cov: 74.607930,
    T: sd: 19.756063, mean: 2374.749375,
    M: sd: 40.114707, mean: 2994.314657
key rejected, ref: 0.115746
Iteration: 9, keyVal:767, correlation:0.0400662

```

```
cov: 34.220266,  
T: sd: 19.756063, mean: 2374.749375,  
M: sd: 43.231935, mean: 3175.872436  
key rejected, ref: 0.115746  
Key: 255
```

- 3) I believe the key can be guessed as in optimal case, we only need to scan all the 20 bits once, i.e. 20 guess. From the timing file, it is observed that average decryption time is around 5000ms. Even the first guess will be faster and last few guess will be longer, still we can do a rough estimation. So, for 2000 test, we will need around  $20 * 2000 * 5 = 200000s = 55.6$  hours = 2.3 days.

However, in order to maintain low noise level, PC should be dedicated for the test only. So I haven't tried this test.

- 4) Please refer to modification section for modification done.

Further Improvement:

i) In the assignment, as ciphers are fixed, we can save time generated by guess key to file so as to reduce recalculation.

ii) As noise cannot be 100% mimic to the time the real decryption is done, this gives challenge to set correlation threshold. And honestly, correlation threshold is very hard to be accurate. An improvement can be like doing cheap, low resolution for larger set of guess key and refine those with better correlation to determine which guess key better fit. This can hopefully lower the consequent of guessing a wrong bit.

Details work can be done like below

For threshold = 0 to 0.5 (increment by 0.05)

Run test, for number of cipher = 100, number of iteration = 5000

Save guess key

End

For each guess key

Run test with original number of cipher and iteration

Save correlation

End

Get guess key with highest correlation

## Appendix

- 1) blackbox.c
- 2) q2Result.txt
- 3) quickTest - bash script to get result for question 1-3