破密學 Homework 4

B01901169 王愷

Github: https://github.com/b01901169/crypto/tree/master/hw4

Problem 1

1. [Algebraic Attack]
   Solve a system of 5 quadratic equations in 3 variables over GF(7). Compared to the example in the slides, there is one more equation, so the system should be solved by raising to degree 3. You may follow the steps below.
   1) Assign a solution randomly (say, $x_1 = 5$, $x_2 = 2$, $x_3 = 4$).
   2) Generate all coefficients a_{ijk}'s, b_{ik}'s, and c_{ik}'s randomly.
   3) Evaluate $d_k = \Sigma\, a_{ijk} x_i x_j + \Sigma\, b_{ik} x_i^2 + \Sigma\, c_{ik} x_i$ for each k to obtain 5 equations.
   4) Solve the system of equations by XL algorithm.
   5) Compare the solution with the values from 1).

請參考附檔 algebra.py。

Usage: python algebra.py

下表為各個 function 的用途與功能

Function	Output variable	功能
`generateSolution`	Solutions x_1, x_2, x_3	隨機生成三個變數的解。
`assignCoefficient`	polynomial	對每個項隨機給定係數，生成一個 random equation 來當作已知條件，並根據上面隨機生成的解來計算常數項。
`raisePolynomials`	Polynomials	生成 3 次以內的所有首一單項式(ex. x^2y)，並將最後一個變數(z)的次方都放在最後面。
`getCoefficientMatrix`	Coefficient matrix	輸入原本的 random equations 乘上三次以內的首一單項式後得到的 XL equations，照順序輸出這些 equations 的係數為一個矩陣。
`egcd, modinv`	在 finite field 中做乘法反元素運算	輸入 x, modular，計算 $x^{-1}\ (mod\ modular)$
`gaussElimination`	輸入矩陣	對矩陣做 gauss elimination (under modular)，輸出最後算出的上三角矩陣與最後停下的列數。

```
ubuntu@ip-172-31-5-52:~/crypto/hw4$ python algebra.py
random solutions: [0, 1, 2]
equation 0  : 3*x**2 + 5*x*y + 3*x*z + 3*x + 4*y**2 + 6*y*z + 5*y + z**2 + 3
equation 1  : 5*x**2 + 6*x*y + 4*x*z + 6*x + 5*y**2 + 2*y*z + y + z**2 + 4*z + 6
equation 2  : 6*x**2 + x*z + 2*y*z + 3*y + z**2 + 4*z + 2
equation 3  : 6*x*y + 2*x*z + 2*x + 3*y**2 + 4*y*z + 3*y + 2*z + 3
equation 4  : 2*x**2 + 3*x + y**2 + y*z + y + 5*z**2 + 2*z
```

一開始先隨機生成三個變數的解與五條已知和為 0 的方程式。之後再乘上 3 次以內所有的首一單項式，共得到$5 \times 20 = 100$條方程式，之後把所有 2+3 次以內的首一單項式列出並依照順序將上述方程式的係數取出來，輸出成矩陣。

```
gauss elimination ...
solve z by solving: 6*z + 2
solve y by solving: 3*y + 3*z**5 + 3*z**4 + 2*z**2 + 6*z + 1
solve x by solving: 4*x + 3*y + 5*z**4 + 6*z**3 + 6*z**2 + 6
solved (x,y,z): (0, 1, 2)
```

之後對此矩陣做高斯消去法，刪除到最後會得到 z 的一元多次方程式，解出 z 之後，觀察高斯消去法得到的最後上三角矩陣的前面幾列，會發現可以得到只有 y 和 z 的二元多次方程式，代換掉已知解的 z，得到只有 y 的一元多次方程式，解出 y 的解。之後再繼續往前找，可以得到 x,y,z 的三元多次方程式，代換掉 y,z 則可以得到 x 的一元多次方程式，同樣可解出 x。

因此就可以解出原本隨機給定的三個變數的解。

Problem 2

2. [Linear Attack]
   Use the toy cipher with the given 4x4 S-box and the linear approximation in the slides to demonstrate linear attack. Follow the steps below.
   1) Generate a key randomly
   2) Generate enough plaintexts randomly and encrypt them
   3) Use the above plaintext-ciphertext pairs to recover some bits of the key

請參考附檔 linear.py。

Usage: python linear.py

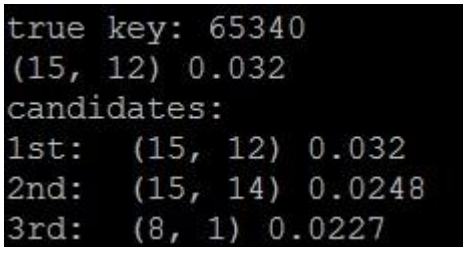
下表為各個 function 的用途與功能

Function	Output variable	功能
`int2bin, bin2int`	Int or binary array	做整數與二進制之間的轉換。
`passSBox, passPBox`	輸出通過給定 sBox, pBox 的結果	將 input 的 message 通過 sBox, pBox，盡量攪亂做成隨機分布。
`encrypt`	將輸入做給定層數與 sBox, pBox 的加密	將整個 toy cipher 的加密系統包在一起，只需要給 input message, sBox, pBox, round number，就會輸出加密後的密文。
`Decrypt`	將輸入做給定層數的解密。	類似 encrypt，不過做的是解密所以要給 sBox, pBox 的乘法反元素，輸入之後就可以得到解密後的明文。
`randomKey` `randomSameKey`	隨機生成數層的 keys	randomKey 是會生成每層不同的密文，而 randomSameKey 則是每層都相同，不過對這於 linear attack 來說沒有差別。

```
ubuntu@ip-172-31-5-52:~/crypto/hw4$ python linear.py
linear attack
keys: [2202, 45846, 39094, 51664, 12089, 65340]
generate 10000 plaintext and ciphertext ...
computing the bias of each guessing key ...
this may take several minutes ...
```

一開始先生成五層共六個不一樣的 keys，之後使用這些 key 跟已知的 sBox, pBox 去做加密，隨機選取 plaintext 加密後得到共 10000 筆明密文對。之後枚舉最後一層的 key 有牽涉到的 8 個 bits，總共試 16*16 次，每次計算密文經由這個最後一層的 key 解密回上一層的結果，並與他的明文中特定幾個 bits 做 xor，觀察這些 bits xor 的 bias，並記錄下來。

(計算每個不同猜測的 bias)



最後則是將這些不同的 bias 排序，取出最大最符合我們預測值$\frac{1}{32} \approx 0.03125$ 的猜測值，因此可以知道最後一層 key 的第 2 個 byte 與第 4 個 byte 分別是 15 和 12 的機會相當高。與實際上生成出的 key 吻合。

Problem 3

3. [Differential Attack]
   Use the toy cipher with the given 4x4 S-box and the differential trail in the slides to demonstrate differential attack. Follow the steps below.
   1) Generate a random key
   2) Generate enough plaintext pairs with the specific differential, and encrypt them
   3) Use the above plaintext-ciphertext pairs to recover some bits of the key

請參考附檔 differential.py。

Usage: python differential.py

各個 function 的用途與功能與 linear attack 相同，因為重複則不列舉於下。

```
ubuntu@ip-172-31-5-52:~/crypto/hw4$ python differential.py
linear attack
keys: [3438, 23878, 14181, 10730, 2257, 61836]
differential: 2816
target: 1542
generate 500 plaintext and ciphertext ...
the last round key: [1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0]
```

一開始先設定好x, x*的 xor 值(differential) 為 $x' = 2816 = 0000101100000000_2$ (如同講義中所寫的)，與理論上會佔 output 較大機率的 target $= 1542 = 0000011000000110_2$，根據講義上所述，在選擇明文時的 xor 值為 differential 時，有$\frac{1}{32}$ 以上的機率會得到target。因此隨機生成500筆明文與明文xor differential，將這些值都做加密，得到密文後，根據猜測的 key 的第 2 個 byte 與第 4 個 byte 的值，計算回前一輪的輸出值，將兩個輸出值做 xor 看結果是不是 target，計算總共有多少機率會變成 target，記錄下來為這個猜測的 bias。

```
(1, 9) prob: 0.012
(1, 10) prob: 0.01
(1, 11) prob: 0.004
(1, 12) prob: 0.028
(1, 13) prob: 0.0
(1, 14) prob: 0.016
(1, 15) prob: 0.01
```

```
true key: 61836
(1, 12) 0.028
candidates:
1st:   (1, 12) 0.028
2nd:   (1, 14) 0.016
3rd:   (4, 12) 0.012
```

```
count = 0
key = (key2 << 8) + key4
for i in range(plaintext_number):
  m1 = random.randint(0,2**bits-1)
  m2 = m1 ^ differential
  #plaintext.append(m)
  #print 'm:', m
  c1 = encrypt(m1,sBox,pBox,round_num,keys)
  c2 = encrypt(m2,sBox,pBox,round_num,keys)
  v1 = c1 ^ key
  v2 = c2 ^ key
  u1 = passSBox(v1,rev_sBox)
  u2 = passSBox(v2,rev_sBox)
  diff = u1 ^ u2
  if diff == target:
    count += 1
prob = float(count) / plaintext_number
bias[(key2,key4)] = prob
print (key2,key4),"prob:",prob
```

根據計算出的各個猜測命中 target 的機率，取出最大的那個，即為對 key 的第 2,4 的 byte 的合理猜測，可以注意到跟原本隨機生成的正確 key 吻合。而且這個攻擊方式所需要的明密文對相對的小得多，正確率也較高，只是需要可以選擇明文。

Bonus Problem

B. Take another 4x4 S-box, find a good differential trail, and redo problem 3.

請參考附檔 general_differential.py。

Usage: python general_differential.py

多數 function 的用途與功能與 differential attack 相同，重複的部分則不列舉於下。

Function	Output variable	功能
`buildTable`	Table of sBox bias	
`passSBoxWithTable`	輸出使用 table 最高機率的輸出與他的機率。	使用建好的 differential table 來取代 sBox，改成輸入 differential 到 sBox 後都取出最高機率的那個輸出來當作通過 sBox 的值。
`encryptWithTable`	輸出使用 table 最高機率的輸出與他的機率。	同上，不過將 pBox 也包含進來，並且照著加密所需要多輪的 sBox, pBox 都做完。
`findPath`	輸出講義上已經幫我們找好的 sBox input differential, output differential(target), expected probability	根據計算出來的 table，取出第一層最高機率的 input differential，並且嘗試放在 input layer 的 4 個不同 bytes 位置，送入 encryptWithTable 去計算該 input differential 最可能產生的 output differential(target)與他的機率 prob。
`seperateKey`	An array containing 4 bytes of key	將 key 切成 4 個 byte，方便檢視用。

這個 bonus problem 跟 problem 3 的唯一差別就是不同的 sBox 會產生不同的最佳 input differential, output differential，因此多做了建 table 的 function，以及用 table 來計算輸入該 input differential 時，會得到什麼樣最大的 output differential 與他對應的 output probability。

根據上面找到的最大機率的 input output differential pair，將他當作如同講義所述的 input, output 來計算 bias，重複 problem 3 所做的事情即可還原最後一層 key 的一部分 bits。

此外注意到有些 sBox 會得到很均勻的分布，這時 input, output pair 的 bias 會相當小，因此很可能無法被攻擊。在程式裡面有對 probability 做判斷，如果 probability 小於 0.01 的話，則視該 sBox 為安全的，也就是很難被 differential attack 攻擊成功，在這種情況下就不繼續執行攻擊，需要重新跑一次程式試試看其他的 sBox。