2023 NTU Computer Security HW1 Writeup

StudentID : R12922054

[Lab] COR

· FLAG{Corre1ati0n_Attack!_!}

解題流程和思路

本題是實作Correlation Attack:

1. 題目給了3個LFSR來決定最後的output，且3個LFSR的tag在0, 1, 2, 5。
2. 第二個和第三個LFSR的輸出和最後的output有75%相似
3. 因此透過brute force的方式，找出和output有75%相似的LFSR2,LFSR3，的候選key組。
4. 處理完LFSR2,LFSR3後，再去brute force LFSR1。
5. 最後再將LFSR1, LFSR2,LFSR3產生的output和題目給的stream做xor，再取index 200之後的bit 取出就是flag了。

```
lambo@Ubuntu-Lambo:~/Desktop/Hw1/COR$ python Q1_solution.py
```

Figure 1: COR_Cmd.

```
b'FLAG{Corre1ati0n_Attack!_!}'
```

Figure 2: COR_Flag.

[Lab] POA

· FLAG{pAdd1NG_0rAcL3_A77aCK}

解題流程和思路

本題是實作Padding Oracle Attack:

1. 有題目得知，server會根據將plaintext解析出來後判斷他的padding是不是"message + /x80 + /x00*(15 - len(message))"
2. 因此我們可以透過將送至server端的iv中對其每個bit $\oplus$一個常數$c(0 \sim 255)$，讓解析出來的pliantext變成都是padding的樣子，例如: /x80/x00/x00/x00/x00...這樣我們就可以透過以下的式子得出原先的plaintext是多少

$$\text{ct} \oplus \text{iv} = \text{pt}$$

$$\text{ct} \oplus \text{iv} \oplus c = \text{pt} \oplus c = /\text{x80}(\text{或者}/\text{x00})$$

$$\text{pt} \oplus c = /\text{x80}(\text{或者}/\text{x00})$$

$$\text{pt} = /\text{x80}(\text{或者}/\text{x00}) \oplus c$$

```
lambo@Ubuntu-Lambo:~/Desktop/Hw1/POA$ python Q2_solution.py
```

Figure 3: POA_Cmd.

```
b'FLAG{pAdd1NG_0rAcL3_A77aCK}\x80\x00\x00\x00\x00'
```

Figure 4: POA_Flag.

[Lab] LSB

· FLAG{Viycx_qsklsjgmeld_fgd_spkgjo}

解題流程和思路

本題是實作Least Significant Bit Attack, 由題目我們可獲得以下資訊:

1. Server會將本地端傳至此地的RSA密文做解密後, 回傳給本地端明文mod 3的結果。

本題作法:

1. 利用RSA的乘法同態。
2. 可以將密文看作:

$$a_n * 3^n + a_{n-1} * 3^{n-1} + \cdots + a_1 * 3^1 + a_0 * 3^0$$

3. 每做一次Oracle後就將明文成1/3(不直接除3是因為除法沒有RSA同態), 但除了a_0, 其餘Oracle的結果含有小數, 因此要將回傳回來的結果減去小數。如此就可以得到$a_0 \sim a_n$的值。
4. 之後再將$a_0 \sim a_n$乘上對應的3的冪次方, 就可以獲得明文。

```
lambo@Ubuntu-Lambo:~/Desktop/Hw1/LSB$ python Q3_solution.py
```

Figure 5: LSB_Cmd.

```
b'FLAG{Viycx_qsklsjgmeld_fgd_spkgjo}'
```

Figure 6: LSB_Flag.

[HW1] LFSR

· FLAG{f5r_15_50_eZZzZzZZZzzZzzz}

解題流程和思路

從題目中可以得到以下資訊:

1. 題目中有random 8 bytes的key，由此我們可以知道要生成64*642的companion_matrix。
2. 由於taps = [0, 2, 17, 19, 23, 37, 41, 53]所以可以知道在最後一Row的地方index為[0, 2, 17, 19, 23, 37, 41, 53]的Column為1。
3. 由題目的內容可知我們會對總長度為len(flag)+70的資料做LFSR，且每經過70輪的shift之後在第71輪的時後拿輸出的state。
4. 題目中只對前面len(flag)所拿到的輸出狀態做對index相對應的flag做XOR，因此我們可以知道最後70的輸出state是乾淨的，可以拿來使用。

解題方向:

1. 要解出這題的flag，必須先找出本題的Initial State，找出之後就可以根據本題LFSR的規則解出flag。
2. 我們有70個純粹輸出LFSR之後的結果，我們可以從中選取64個，這樣就可以列出64條方程式，64個變數有64條方程式，那就可以解出這64個變數，而這個變數就是Initial State。這題我是直接抓flag之後的64個state，如下方的矩陣。

$$\begin{pmatrix} M^{70+71*256} \\ M^{70+71*257} \\ \vdots \\ M^{70+71*319} \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{63} \end{pmatrix} = \begin{pmatrix} O_{256} \\ O_{257} \\ \vdots \\ O_{319} \end{pmatrix}$$

3. 將矩陣做inverse解出Initial State

$$\begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{63} \end{pmatrix} = \begin{pmatrix} M^{70+71*256} \\ M^{70+71*257} \\ \vdots \\ M^{70+71*319} \end{pmatrix}^{-1} \cdot \begin{pmatrix} O_{256} \\ O_{257} \\ \vdots \\ O_{319} \end{pmatrix}$$

4. 最後將 O_0 到 O_{255}解出後，XOR相對應的Output,即可得出flag

$$\begin{pmatrix} M^{70} \\ M^{70+71*1} \\ \vdots \\ M^{70+71*255} \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{63} \end{pmatrix} = \begin{pmatrix} O_0 \\ O_1 \\ \vdots \\ O_{255} \end{pmatrix}$$

$$\text{flag[i]} = O_i \oplus \text{output[i]}$$

```
lambo@Ubuntu-Lambo:~/Desktop/Hw1/LFSR$ sage LFSR_Solve.sage
```

Figure 7: LFSR_Cmd.

```
b'FLAG{Lf5r_15_50_eZZzZzZZZzzZzzz}'
```

Figure 8: LFSR_Flag.

[HW1] Oracle

· FLAG{Rea11yu5efu110rac1eisntit?}

解題流程和思路

本題的作法和想法：
因為本題的plaintext太過龐大因此得思考先獲得decryted IV和Key，然後在本地端解密，因此在本地端，隨機生成一個key，控制"解密的結果"並把他當作AES_ECB的明文（之後用Dct形容)，並使用AES_ECB將Dct加密生成ct，然後將以下內容傳至Alice:

1. 將隨機生成的key用RSA加密 (作為Alice接收的key)
2. 題目給的實際的RSA key 或iv (作為Alice接收的iv)
3. AES_ECB加密生成ct (作為Alice接收的ct)

因為我們控制AES_ECB解密的結果(Dct)(也就是AES_CBC 經過AES_Block解密的結果，其值等於iv xor pt, Dct = iv xor pt)，又我們可以透過Oracle得知控制Dct後所得出的pt 這樣就可以獲得我們要的內容(題目給的RSA_key 和RSA_iv解密之後的結果, Figure 9 & Figure 10)

```
bytearray(b'K\xa3\xcb\x1c\x13FQ\xc3\xbb\\\xd6\xe3\x81\xc2\x90\x9b')
```

Figure 9: Oracle_decryIV.

```
bytearray(b"I\'m_4_5tr0n9_k3y")
```

Figure 10: Oracle_decryKey.

```
lambo@Ubuntu-Lambo:~/Desktop/Hw1/Oracle$ python Oracle_solve.py
```

Figure 11: Oracle_Cmd.

最後我們在使用解密後的key, iv和原先知道的ct做解密, 解密後得到的binary文字轉成png輸出, 如下圖, Figure 12:

```
FLAG{Rea11yu5efu110rac1eisntit?}
```

Figure 12: Oracle_Flag.

[HW1] Oracle_Revenge

- FLAG{Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_oracle_of_oracle_oO_oO_o_oO_oO_o_oO_oO_o_oO_oO_o_oO_oO_o_oO_oO_o_oO_oO}

解題流程和思路

本題的解題技巧是使用前一題Oracle的解法 + Oracle LSB的概念來實現:

1. 將encrypted_flag作為給Alice的RSA_IV送至Server。
2. 接著透過Oracle的方式，得知decrypted iv是多少(也就是decrypted flag的倒數16bytes)
3. 接著往右shift 16位(也就是做將encrypted flag 乘上RSA加密過後的 $\frac{1}{2^{128}}$)，繼續丟給Alice，回傳回來的答案減去前一回合的答案乘上$\frac{1}{2^{128}}$，就是倒數32-17位的值，如此繼續到求出完整的decrypted flag。
4. 其實本題的作法就跟LSB一樣只是一個是乘上$\frac{1}{3}$，$\frac{1}{2^{128}}$，並且都要去扣除前項答案變成的餘數。

```
lambo@Ubuntu-Lambo:~/Desktop/HW1/Oracle_Revenge$ python Oracle_Revenge_solve.py
```

Figure 13: Oracle_Revenge_Cmd.

```
b'FLAG{Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_Oo_oracle_of_oracle_oO_oO_o_
oO_oO_o_oO_oO_o_oO_oO_o_oO_oO_o_oO_oO}'
```

Figure 14: Oracle_Revenge_Flag.

[Lab] dlog

- FLAG{YouAreARealRealRealRealDiscreteLogMaster}

解題流程和思路

本題的解法:(因為基本上都是按照助教上課的內容實作,所以就稍微說明)

1. 要能順利解出dicrete log的問題, 基本上就是產生很smooth的order因此我們可以在本地端產生一個order很smooth的質數給server, 就可以很輕鬆解出我們要的答案

```
lambo@Ubuntu-Lambo:~/Desktop/HW1/dlog$ python dlog_solve.py
```

Figure 15: dlog_Cmd.

```
b'FLAG{YouAreARealRealRealRealDiscreteLogMaster}'
```

Figure 16: dlog_Flag.

[Lab] signature

- FLAG{EphemeralKeyShouldBeRandom}

解題流程和思路

本題的解法:(因為基本上都是按照助教上課的內容實作,所以就稍微說明)

1. 基本上解法就跟下圖Figure 17 一樣, 不過k_1, k_2的值並不相等,不過從題目中可以得知,$k_2 = 1337 * k_1$, 因此可以將公式化簡後, 利用下圖的方法求出答案即可。

- $k_1 = s_1^{-1}H_1 + d(s_1^{-1}r_1) \bmod q$
- $k_2 = s_2^{-1}H_2 + d(s_2^{-1}r_2) \bmod q$
- $d = (s_1^{-1}H_1 - s_2^{-1}H_2) \,/\, (s_2^{-1}r_2 - s_1^{-1}r_1)$

Figure 17: signature_idea.

```
lambo@Ubuntu-Lambo:~/Desktop/HW1/signature$ python signature_solve.py
```

Figure 18: signature_Cmd.

```
b'FLAG{EphemeralKeyShouldBeRandom}'
```

Figure 19: signature_Flag.

[Lab] coppersmith

- FLAG{RandomPaddingIsImportant}

解題流程和思路

本題的解法:(因為基本上都是按照助教上課的內容實作,所以就稍微說明)

1. 本題就是上課講義內容, 透過Lattice matrix做LLL將RSA的密文解析出來, 至於Lattice matrix的樣子就如下圖 Figure 20 。

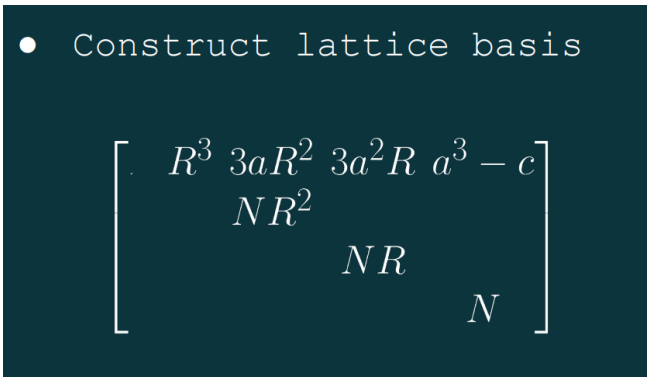

Figure 20: coppersmith_idea.

```
lambo@Ubuntu-Lambo:~/Desktop/HW1/coppersmith$ sage coppersmith_solve.sage
```

Figure 21: coppersmith_Cmd.

```
b'FLAG{RandomPaddingIsImportant}'
```

Figure 22: coppersmith_Flag.

[HW1] Invalid_Curve_Attack

- FLAG{YouAreARealECDLPMaster}

解題流程和思路

本題的解題想法:

1. 本題server並沒有判斷client給的點是否在curve上所以，可以透過這點做攻擊。
2. 因為一開始題目給的a太大了，所以我直接忽略可能要透過singular curve的想法。
3. 然後呢，在discord上助教有提到不要想著找一個b可以讓curve變得smooth的。因此呢就有我以下的作法

本題的作法:

1. 隨機產生一個b然後產生elliptic curve(注意:這邊要避免產生的b使得我們的elliptic curve變singular)
2. 得到該elliptic curve的order後，對order進行質因數分解，然後選擇其中的質數，這題裡，我選擇最大的質數，但最大值我有讓它不超過2^{40}，我怕會之後的discrete log會算太久
3. 然後將E產生的generator乘上 “E的order除以該質數"的G(讓其每order/prime次為一次循環)

4. 讓G產生一個點丟置server，讓server回傳運算完的點(K)
5. 讓回傳的點K和G做discrete log就可以得知在mod prime底下的log

$$\text{flag} \equiv \log \bmod \text{prime}$$

6. 如果產生多個log (d_i) 和 prime(p_i)，就可以做CRT了。

$$\text{flag} \equiv d_i \bmod p_i$$

7. 我最後跑了20次不同的curve產生20個不同的log和prime然後做CRT，就得到結果了(至於為什麼是20，我想說先隨便設一個數字但又不要太小，怕CRT解不出來或解太久，然後20剛好就可以解出來了XD)

```
lambo@Ubuntu-Lambo:~/Desktop/Hw1/invalid_curve_attack$ sage --python invalid_curve_at
tack_solve.py
```

Figure 23: Invalid_curve_attack_Cmd.

```
flag =  b'FLAG{YouAreARealECDLPMaster}'
```

Figure 24: Invalid_curve_attack_Flag.

[HW1] Signature_Revenge

· FLAG{LLLisreaLLyusefuL}

解題流程和思路

本題的解題想法:

1. 這題給的橢圓曲線其order, n為2^{256}，所以其實可以知道我們要設定的K值至少要小於2^{128}，然後要求的值要小於K, 也就是$2^{128} > K \geq k_1$、k_2 這裡的k_1、k_2指的是要求的數值不是指 ephemeral key
2. 那這題的解題關鍵在於我們的 ephemeral key 是從兩個數據magic1、magic2而來, 而且兩個的數值都是小於2^{128}, 根據上述其實我們就可以把 ephemeral key的等式

$$k_1 \equiv s_1^-1 h_1 + d s_1^-1 r_1 \bmod n$$

$$k_2 \equiv s_2^-1 h_2 + d s_2^-1 r_2 \bmod n$$

變成

$$\text{magic1} * 2^{128} + \text{magic2} \equiv s_1^-1 h_1 + d s_1^-1 r_1 \bmod n$$

$$\text{magic2} * 2^{128} + \text{magic1} \equiv s_1^-1 h_1 + d s_1^-1 r_1 \bmod n$$

3. 移向將d消去，再生成Lattice Matrix，並用LLL就可以求出我們要的magic1、magic2(LLL之後我還有再做一些linear combination)，最後再將magic1、magic2組成我們要的 ephemeral key ，就可以解出我們的d，也就是flag了。

本題的作法: 如下圖, Figure 25

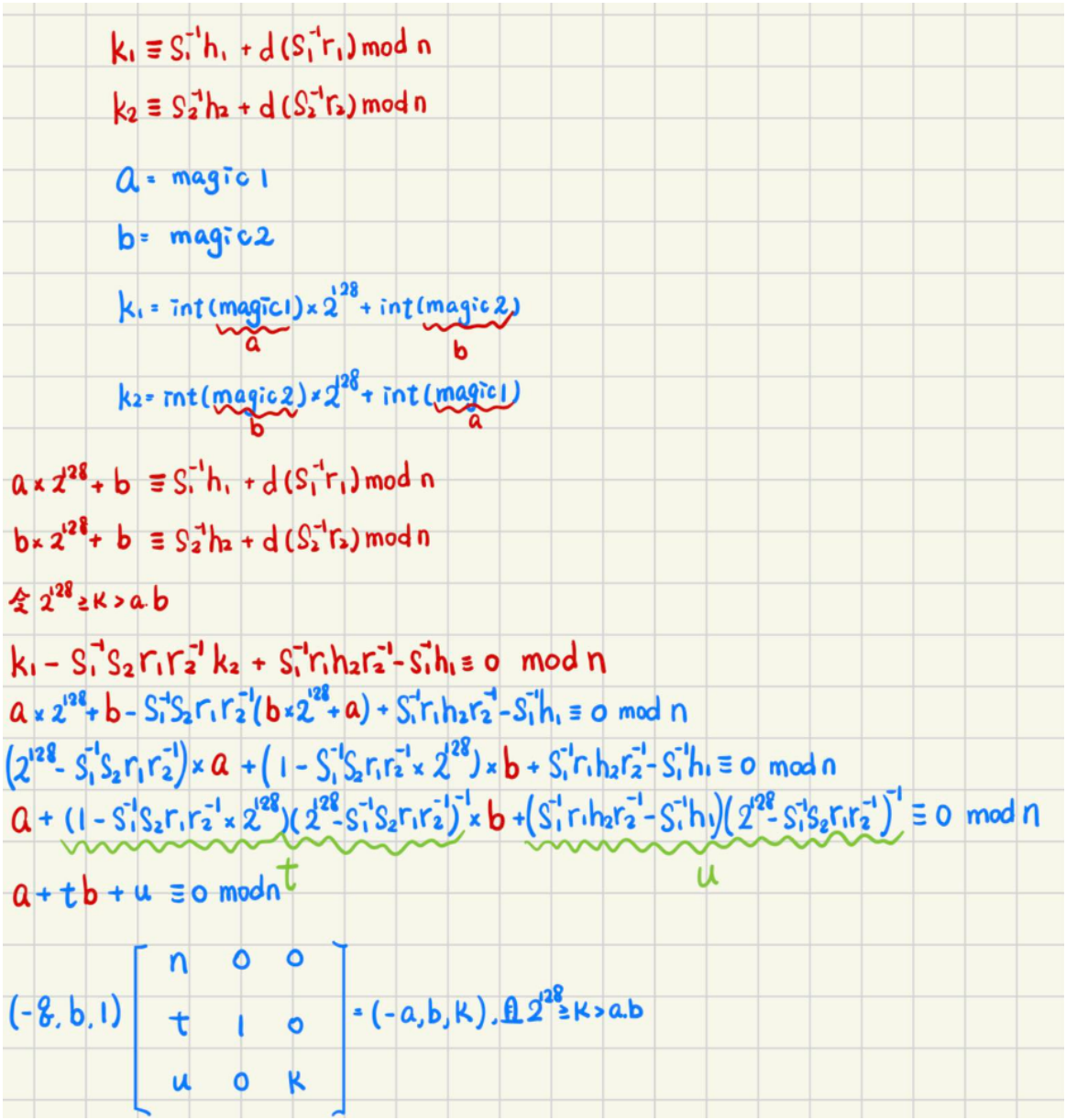

Figure 25: signature_revenge_idea.

```
lambo@Ubuntu-Lambo:~/Desktop/Hw1/signature_revenge$ sage --python signature_revenge_solve.py
```

Figure 26: signature_revenge_Cmd.

```
b'\xad\xc4u\xcf\x11\x1f\xd7R$FLAG{LLLisreaLLyusefuL}'
```

Figure 27: signature_revenge_Flag.

[HW1] Power Analysis

- FLAG{W0ckAwocKaWoCka1}

解題流程和思路

本題的解題作法:

1. 因為我基本上都是按照ppt上面的流程來實作所以就不太敘述我的想法了。(不過sbox以及實作上的過程有和r12922146、chatGPT一起討論)
2. 不過題目給的zip檔中, 我只有用到trace.json其他都沒用到。
3. 迴圈最外層是byte的位置(由最小byte到最大, 0-15)

4. 接著是, data的數量(遍歷每個byte的所有data, 也就是指定一個byte位置讓所有data先針對這個byte位置, 做analysis)
5. 最後做correlation。
6. 重複以上步驟至byte遍歷完畢即可。

```
lambo@Ubuntu-Lambo:~/Desktop/Hw1/Power_Analysis$ python power_analysis_solve.py
```

Figure 28: power_analysis_Cmd.

```
W0ckAwocKaWoCka1
```

Figure 29: power_analysis_Flag.