CTF

EggRoll Taiyaki

January 27, 2022

雖然平時有跟 idek 參與很多 CTF，但眞的有把時間砸下去的不多，以下列出有認眞打，並且覺得相對有趣的比賽。

1. HK Cyber Security New Generation CTF Challenge
2. SECCON
3. SCTF

Contents

1 HK Cyber Security New Generation CTF Challenge

1.1 Freedom

題目簡單來說，就是提供一個 AES 的 oracle，可以用 5 種不同的模式

ecb, cbc, cfb, ofb, ctr

去加密 flag 或是任意使用者給的字串，每種模式的 key 相同，並且只能使用一次。

剛看到這題的想法是：要找出兩個模式 A,B，它們加密的方式等同於 XOR 生成的 stream 跟 plaintext，那拿它們分別去加密 flag 跟已知的 data，應該就能將 flag 拿回來。那這不就只能選 CTR 跟 CFB 嗎？可是它們生成的 stream 並不相同，到這裡就卡住了... 出門吃火鍋的時候，才突然想起 OFB 跟 CFB 其實只差了一點，只要 OFB 的 plaintext 都是 null bytes，那麼生成的 stream 就跟 CFB 一模一樣。

1.2 長話短說

我們有個怪東西，它能夠做以下 4 件事情：

- 看似不太 random 地生成一個新的 1024-RSA
- 用 RSA 加密使用者提供的 m，public key 爲 17
- 回傳 RSA 加密過後的 secret，大小是 256-bits
- 回傳 AES 加密過後的 flag (以 secret 當 key)

此外，我們被限制只能做 17 次操作。

看到那麼小的公鑰，就會想要去開根號。不過 secret 沒有那麼小，因此要有 k 個加密過的 secret，其中 k 滿足

$$256 \times 17 < 1024 \times k \to 5 \leq k$$

要注意到，怪東西並沒有直接跟我們說 RSA 的模數，必須要自己找出它。這裏可以簡單算一下，扣除拿 flag 的一次操作，在一個 RSA 上，平均會有

$$\lfloor \frac{17-1}{5} \rfloor = 3$$

個操作可以使用，這還必須包含操作 1 跟 3，所以要使用一次操作 2 來拿到模數。由 RSA 的生成方式，可知

$$2^{1022} < N < 2^{1024}$$

若我們取 $m = \lceil 2^{\frac{1024}{17}} \rceil = 1357157737484444931$，則得到的加密結果 c 滿足 $m^e - c$ 爲 N 的小倍數。把 $m^e - c$ 中小的因數都除掉，得到的即是 N。

1.3 Braceless

這題跟「長話短說」類似，同樣有個怪東西，提供了 4 種操作：

- (pkey) 看似不太 random 地生成一個新的 1024-RSA
- (send m) 用 RSA 加密使用者提供的 m，public key 爲 65537
- (backup) 回傳 RSA 加密過後的 secret，大小是 256-bits
- 回傳 AES 加密過後的 flag (以 secret 當 key)

比較不同的是，題目提供的是某個人操作過後的 log，它先執行操作 4，接著反覆進行以下操作 16384 次：

1. pkey
2. send 2
3. send 3
4. backup

首先呢，來看看用同樣的做法有沒有機會。用 CRT 會得到

$$secret^{65537} \pmod{\prod N_i}$$

原本 secret 的 65537 次大約是 256×65537 bits，而模數大約是 $1024 \times \frac{65536}{4} = 256 \times 65536$ bits。差了 256bits，也每辦法用暴力解決，所以當時就想說沒救了。只能期望有兩個模數有公因數，那根據質數定理，1024 bits 的質數大概有

$$\frac{2^{1024}}{1024} = 2^{1014}$$

另一方面，由生日問題，需要有 $\sqrt{n} = 2^{507}$ 個才有高機率重複。但感覺也沒有其他洞可以鑽，只能預期那個不夠 random 的方式會產出一樣的質數。如果直接兩兩配對取 gcd 的話會太久，於是我們需要用 gcdtree 去計算 (從 2021 RCTF 那裡學到的)。

1.4 FreeRider

首先它會生成 16 個 random bytes，然後根據它們弄出 16 個 AES -CTR 的 cipher，並且給了明文前 26 個跟最後一個 byte。

注意到，CTR 本質就是 stream cipher。換言之，有 256 個 stream(基本上可以假設它們在 $GF(2)$ 中線性獨立)，其中 16 個 (可重複選) 與明文做 XOR 後會得到密文。那我們有的 $26 \times 8 = 208$ 個 XOR 的等式 (事實上，是有 $208 + 8 + 35 = 251$，這是因爲 flag 裡面每個都是 ascii<128 的，不過比賽時也沒注意到。想說 216 跟 208 沒差，就懶得把最後一個 byte 也拿進來用)。寫成矩陣的樣子就是

$$Ax = b, \quad in \quad \mathbb{Z}/2\mathbb{Z}$$

這裏 A, x, b 的維度是 $208 \times 256, 256 \times 1, 208 \times 1$。由於我們最終只要挑 16 個，隨便取 208 個包含它們全部的機率大概是

$$\left(\frac{208}{256}\right)^{16} \sim 0.036$$

取個 30 次就應該要中獎了吧！最後解得的 x 有 14 個 1，那就代表有兩個 stream 一樣，不會影響解密。

1.5 心得

其實比賽還有兩題很有趣的，分別是集合吧！地球防衛隊跟神奇的糊塗魔藥。前者是考 AES，大家都熟知 AES 主要有四個步驟：

1. Add Round Key
2. Sub-Bytes
3. Shift Rows
4. Mix Columns

那麼它就只做其中三個步驟，4 種情況都要我們破解它。不過那時候跟 AES 不太熟，同時又覺得另一題比較有趣...。後者則是給我們一個加密的計算機 f，它會算

$$f(Enc(a), Enc(b), op) = Enc(a\ op\ b)$$

其中 Enc 是一個用兩個 AES 做的 encryption，我們可以先當成 random number generator 來使用，op 可以爲 mul, pow, and, or。那目標非常明確，就是將加密的 2 冪都生出來即可，0 跟 1 都是比較簡單的，主要是利用 and 只會讓 Hamming distance 變小的性質，以及 python 內建 0 的 0 次方爲 1 的梗。事實上，2 的做法也差不多，我們也是一直去做 and 這個操作，只要確保跑出來不是 0 即可，如此會得到一個 2 冪，那要如何確保它是 2 呢？這裡就要用 or 只會增加 Hamming distance 的性質啦！只需要一直做 or，直到它 and 1 是 0，那這個數字高機率是 $2^{128} - 2$，最後只要做平方跟 and 得到 4，從而得以確認前面得到的數字是不是 2 的 encryption。

總之呢，這次出了不少 AES 相關的東西，並且也從 FreeRider 複習了一點線代，然後還看到 and 跟 or 可以這樣玩，也是蠻有收穫的。之後有空再把另外一題 crypto 給補完吧！

2 SECCON

就按照解題的順序來寫好了:P

2.1 XXX

因爲題目非常短，這邊就直接放 code

```
import os

flag = os.getenv("FLAG", "fake{fakeflag_blahblah}")
x = int.from_bytes(flag.encode(), "big")

p = random_prime(1 << int(x.bit_length() * 2.5))
Fp = GF(p)

params = []
while len(params) != 6:
    try:
        y = randint(2, x)
        a = randint(2, p-1)
        b = (y^2 - (x^3 + a*x)) % p

        EC = EllipticCurve(Fp, [a, b])
        EC(x, y)

        params.append([a, b])
    except ValueError:
        pass

print(p)
print(params)
```

不難注意到 p 相對 x, y 來說相當大，所以自然地就會想到 SVP，那我們要做的事情就相當清楚，要構造一些 vector，使得它們的線性組合長度不大。很直覺地想法是，

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \\ p & 0 & 0 & 0 & 0 & 0 \\ 0 & p & 0 & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 0 \\ 0 & 0 & 0 & 0 & 0 & p \end{bmatrix}$$

當權重是 $(x^3, x, 1, ?, ?, ?, ?, ?, ?)$ 時，會組合出 $(y_0^2, y_1^2, y_2^2, y_3^2, y_4^2, y_5^2)$。嗯，看起來很棒，只是跑不出答案...。這是因爲我們沒有去限制一下各個權重的範圍，那要怎麼做呢？除了把它當作 CVP 以外，也可以擴充這些向量。具體來說，要做以下的步驟

- 讓原本目標的 short vector 各分量的 order 一致 (叫它 v)，否則會被某個 row 主導。

- 第 i 個向量後面補上一個數字，大小大約爲第 i 個預期權重除以 v。

以這題來說，矩陣會變成

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & \frac{1}{x} & 0 & 0 \\ a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & 0 & x & 0 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 & 0 & 0 & x^2 \\ p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & p & 0 & 0 & 0 \end{bmatrix}$$

這邊要注意到，我們其實不知道 x 是多少，不過用它的近似值 $x = p^{\frac{2}{5}}$ 即可。

2.2 oOoOoO

這題一樣 code 很短，

```
import signal
from Crypto.Util.number import long_to_bytes, bytes_to_long, getPrime
import random
from flag import flag

message = b""
for _ in range(128):
    message += b"o" if random.getrandbits(1) == 1 else b"O"

M = getPrime(len(message) * 5)
S = bytes_to_long(message) % M

print("M =", M)
print('S =', S)
print('MESSAGE =', message.upper().decode("utf-8"))

signal.alarm(600)
ans = input('message =').strip().encode()

if ans == message:
    print(flag)
else:
    print("  ")
```

bytes_to_long 簡單來說就是 256 進位，所以可以把題目想成是 128 個東西的 01 背包問題：首先，原本的背包有

$$79(256^{128} + 256^{127} + \cdots + 1)$$

而第 i 個 byte 是 o 的話，就是加上 $32 \cdot 256^{128-i}$，然後我們的目標是 S。不難發現，還沒模 M 時，每個物品的重量都差了 256 倍。因爲 M 有點大，因此可以期待模 M 後，兩兩

之間的差距還是不小，那麼用 LDA 就能找回原本的 message。具體的作法是構造矩陣

$$\begin{bmatrix} 2 & 0 & \cdots & 0 & a_0 \\ 0 & 2 & \cdots & 0 & a_1 \\ & \vdots & \cdots & \vdots & \\ 0 & 0 & \cdots & 2 & a_{n-1} \\ 1 & 1 & \cdots & 1 & S' \\ 0 & 0 & \cdots & 0 & M \end{bmatrix}$$

然後跑 LLL，最後看每一列的最後一個是否爲 0，以及其他分量是否爲正負 1。如果是的話，就能根據正負號來區分是否選了 a_i，從而拿到 flag。

2.3 CCC

可能題目名稱有疊字的都是 code 短的:D

```
from Crypto.Util.number import bytes_to_long, getPrime, getRandomInteger,
    isPrime
from secret import flag


def create_prime(p_bit_len, add_bit_len, a):
    p = getPrime(p_bit_len)
    p_bit_len2 = 2*p_bit_len // 3 + add_bit_len
    while True:
        b = getRandomInteger(p_bit_len2)
        _p = a * p
        q = _p**2 + 3*_p*b + 3*b**2
        if isPrime(q):
            return p, q


def encrypt(p_bit_len, add_bit_len, a, plain_text):
    p, q = create_prime(p_bit_len, add_bit_len, a)
    n = p*q
    e = 65537

    c = pow(plain_text, e, n)
    print(f"{n=}")
    print(f"{e=}")
    print(f"{c=}")
    print(f"{a=}")


if __name__ == "__main__":
    encrypt(1024, 9, 23, bytes_to_long(flag))
```

總之呢，這題就是 RSA 加密，但是使用了奇怪的方式生成質數。所以我們有：

$$an = ap((ap)^2 + 3(ap)b + 3b^2) = (ap)^3 + 3(ap)^2b + 3(ap)b^2 = (ap+b)^3 - b^3$$

本來是想說，b 不到 691bits，因此可以用類似 Fermat's factorization 的方式去解 $ap+b, b$。然後我就寫了一下的 code：

```
from output import *
from decimal import *

getcontext().prec = 1000

pb = int(Decimal(23*n) ** (Decimal(1)/Decimal(3))) + 1

for i in range(100000):
        _ = (pb**3) - (23*n)
        b = int(Decimal(_) ** (Decimal(1)/Decimal(3)))

        if b**3 == _:
                print("Found!")
                print(f"pb = {pb}")
                print(f"b = {b}")
                break
        pb += 1
        if i % 100 == 0:
                print(i)
```

但是這樣跑了好幾個小時後，也沒找到答案，因此我當下就覺得 $ap+b$ 跟 $\sqrt[3]{an}$ 是不是其實有點差距。後來想了想，an 大約有 3000bits，並且有 $ap+b$ 的近似，應該可以考慮

$$f(x,y) = (\lfloor \sqrt[3]{an} \rfloor + x)^3 - y^3 = an$$

它有相對小的根。沒錯！又是相對小，可以試試看 bivariate Coppersmith，這會 work 在我自己生成的測資，不過沒辦法拿到 flag...QQ。最後跟隊友討論後才知道，雖然有把 precision 調高，仍然有浮點數誤差，還是應該要檢查 $(b-1)^3, b^3, (b+1)^3$ 是否爲 _。

2.4 Sign Wars

最後是腦袋撞到，在賽中沒解出的題目... 因爲 code 比較長，所以這邊就簡單講一下：

1. 首先未知的東西有 4 個 msg1, msg2, flag1, flag2
2. 它將在 P-384 curve 上做 ECDSA，也就是回傳簽章 (r,s) 滿足

$$s = k^{-1}(z+rd) \pmod{n}$$

3. 對於第一個 msg1，先將它轉成整數 z_1，然後生成 80 個簽章，其中 d 是 flag1，然後 k 的生成方式是

$$k = k_3 \cdot 2^{256} + z_1 \cdot 2^{128} + k_1$$

而 $k1, k2$ 是使用 random.getrandombits(128) 得到

4. 接著來簽第二個 msg2，一樣先轉成整數 z_2，但是這次只簽三個，使用的 d 是 flag2，然後 k 是 random.getrandombits(384)。

我一開始是先看第二個 msg2 怎麼簽的，然後覺得沒有問題，畢竟是最正常的 ECDSA。那要怎麼拿到 flag 呢？肯定要預測 random 的輸出嘛！那一定是用 MT19937 嘛！所以目標非常明確，就是從 80 個詭異的簽章，透過 Lattice 之類的，得到 d 並推出所有的 k。

一般來說，這種題目都會轉成 HNP 來做，但是這樣的話，我們必須有 z_1 得值，不然沒辦法知道 rd 的 MSB 是多少。一連串的鬼打牆後，還是不知道怎麼轉成單變數的不等式問題，我始終忘記多邊數的不等式一樣能夠當成 CVP 來解... 每次遇到這種題目都會忘記有個很強的人已經寫好 code 給大家用了：

https://github.com/rkm0959/Inequality_Solving_with_CVP

我們改寫一下上面的式子：

$$s^{-1}z_1 + s^{-1}rd = k = k_3 \cdot 2^{256} + z_1 \cdot 2^{128} + k_1$$

移項後可得，

$$(s^{-1} - 2^{128}) \cdot z_1 - 2^{256} \cdot k_3 + s^{-1}rd = k_1$$

2.5 心得

Crypto 的部分其實有 6 題，但 pppp 是熱身題，本來想寫完其他題再回去看，但中途就被隊友處理掉了；另一方面，cerberus 一點開就看到 AES-PCBC mode，想說有點不熟，估計要花不少時間弄懂，於是放著先看其他疊字題，然後也被隊友處理掉了... 總之呢，這次的密碼學很乾淨簡短，客觀來說難度普通，很適合複習 Lattice，也讓我發現自己對於多變量不等式相關的問題，仍是非常之不熟悉，才會常常不知道如何用 CVP 來求解，聖誕大餐可能就邊吃飯邊配這個 respository 研究。

3 SCTF

這次的題目還行，不過好像是拿之前 CTF 的題目湊出來的 QQ

3.1 cubic

總之呢，它生成兩個 1024-bits 的質數 p, q，同時會給你

$$e \equiv d^{-1} \pmod{pq}$$

這很明顯用 Wiener 可以解出來。接著它把 flag 拆成兩半 f_1, f_2，然後在詭異的曲線上計算 $e \cdot (f_1, f_2)$。看起來非常之眼熟，沒錯，就是 pbCTF 中出現的東西：

https://eprint.iacr.org/2021/1160.pdf

order 是 $\prod_{p|n}(p^2+p+1)$，所以直接求模逆元，便能解回 f_1, f_2

3.2 ChristmasZone

先來介紹這題在幹嘛：

1. 用 LCG 生成係數 $c_0, c_1, \ldots, c_5$，換言之，(我們只會知道 p)

   $$c_{i+1} \equiv a \cdot c_i + b \pmod{p}$$

2. 根據係數建構多項式 P，

   $$P(x) = \sum_{i=0}^{5} c_i x^i$$

   然後告訴我們 $P(1), P(2), P(3)$ 跟 $P(4)$。

3. 計算 $P(flag) = f = 256^k \cdot f_1 + f_2$，這裏 f_1, f_2 長度差不多。

4. 隨機生成兩個 512-bits 的質數 p, q，然後在 $\mathbb{Z}_n[i]$ 上做計算：

   $$(v_1 + i \cdot v_2) \equiv (f_1 + i \cdot f_2)^{65537} \pmod{n = pq}$$

5. 找個 400-bits 的質數 d，算出模逆元

   $$e \equiv d^{-1} \pmod{(p^2+p+1)(q^2+q+1)}$$

   最後告訴我們 n, e, v_1, v_2 的值。

此題非常明顯是由四個不相關的東西拼起來的，我們就依序來拆解：

1. 注意到 $(p^2+p+1)(q^2+q+1)$ 遠比 d 還大，所以就聯想到 small private exponent 的攻擊方式，常見的是 Wiener 跟 Boneh-Duree。我們有

$$ed-1=k(p^2+p+1)(q^2+q+1)\to k(p^2+p+1)(q^2+q+1)+1\equiv 0 \pmod{e}$$

繼續化簡會得到

$$k(n^2-n+1+(n+1)(p+q)+(p+q)^2)+1\equiv 0 \pmod{e}$$

也就是要找多項式

$$P(x,y)=x(n^2-n+1+(n+1)y+y^2)+1$$

是否在 $\mathbb{Z}_e$ 下有小的根。這用 Coppersmith 即可得到 $p+q$，從而解出 p,q。

2. 接著要解密複數上的 RSA，基本上就是找到 ϕ 使得：

$$a^{\phi(p)}\equiv 1 \pmod{p}$$

那就跟尤拉定理的證明方式雷同，我們想要知道哪些形如 $a+bi$ 與 p 互質，也就會牽扯到 p 在 $\mathbb{Z}[i]$ 下是否能夠分解。大家都熟知複數乘法跟 -1 是 p 的二次剩餘若且唯若 p 是 $4k+1$ 型的質數，結合 $\mathbb{Z}[i]$ 是 UFD，這樣就能夠推出，

$$\phi(p)=\begin{cases}(p-1)^2 & p\equiv 1 \pmod{1}\\ p^2-1 & \equiv 3 \pmod{4}\end{cases}$$

因此，我們可以將 f_1,f_2 找回來啦！

3. 最後要找出多項式的係數，然而 6 個未知數，卻只有 4 個等式。理論上能用 LCG 的關係式，各種暴力運算之類的。這邊我是用 Groebner basis 進行降維打擊。

```
P.<a, b, c> = PolynomialRing(GF(p))

f = [c]
for i in range(5):
        f += [a * f[-1] + b]

G = []
for i in range(4):
        g = 0
        for j in range(6):
                g += f[j] * vs[i][0] ** j
        G += [g - vs[i][1]]

B = Ideal(G).groebner_basis()
print(B)
```

3.3 ciruit map

我們拿到了一大包檔案，根據裡面的檔案名稱，可知考點是 Garbled circuit。這裏就簡略地介紹這個東西是想要幹嘛：

1. 首先要生成 boolean function，會用一個 circuit 來表示。
2. 其中一位玩家，叫作 Alice 好了，會嘗試去混淆這個電路。方法是對於每個 gate，每個 wire 上的 $0, 1$ 都先換成是 random number。舉例來說，如果我們現在有個 AND gate，Alice 會生成隨機數字 a_0, a_1, b_0, b_1 跟 c_0, c_1。接著計算

   $$Enc_{a_i,b_j}(c_{i\&j})$$

   如此就得到混淆的 output。那麼對每個 gate 都做，便完成一個混淆的 circuit。
3. Alice 會將每個 gate 的 output 以及自己選擇的 input 都告訴 Bob，然後換 Bob 要選擇自己的 input，但只有 Alice 知道 input 的 label，並且 Bob 也不想讓 Alice 知道自己的選擇，於是要透過 Oblivious Transfer 來完成，這邊就先略過，只要先記得 OT 可以滿足他們的需求。所以最後 Alice 跟 Bob 都能夠計算出共同的 output label。

回到正題，我們拿到了一個 Garbled circuit 的 output，目標是將每個 wire 上 random number 都找出來，如此便能找回加密 flag 的 OTP。那 Garbled circuit 到底有什麼問題呢？理論上是沒有，只是題目提供的程式碼中，Enc 函數的做法是做兩次 AES，加上 Alice 生成的 random number 是 24-bits 的。不難想到要用 MITM。拿到該題的 input labels 後，把所有可能都丟回去 evaluate 一次，就可以知道每個 wire 的 output 了。

3.4 心得

這次學到最多的是 Garbled circuit，雖說之前就知道 OT 是什麼，也知道可以應用在類似 zero-knowledge 的東西上，但這倒是頭一次好好地把一個完整的應用給看懂。此外，題目提供的 code 有點醜，於是也花了不少時間重新改寫，感覺做了不少事，即使事實上並沒有 XD。另一方面，還考了 small private exponent 相關的攻擊，雖然我還是沒有很清楚 Boneh-Duree 的細節，可能要等到哪天超級閒再研究...。然後研究 complex 的 RSA，這部分就還好之前有修過代數，所以讀起來蠻順的，不過印象就不太深刻，所以在 EOF 的時候，花了一堆時間還是沒想到 easyRSA 的 level2，看來要整理一下學過的東西，讓自己比較好尋找了 QQ。