❄️Winter notes for CTF🚩

Week 1

`Astrageldon`

2024-01-14

Contents

1 Crypto

1.1 Gröbner Bases Summary

在线性方程组中，通过高斯消元法可以从许多个多元方程中提取出一元方程，单独求解再进行回代即可得到解。高斯消元过程的精髓在于反复消去能够去除的首项，而类似的消元方法也可以应用在求解多元高次方程组上。当每两组多项式的线性组合都无法使首项在单项式序的意义下更小时，这一组多项式就是特殊的（类似于线性方程组中的行最简阶梯形）。于是我们有如下定义：

设 $g_1, \cdots, g_t \in \mathcal{K}[x_1, \cdots, x_n]$，$I = \langle g_1, \cdots, g_t \rangle$，$G = \{g_1, \cdots, g_t\}$ 是一组 Gröbner 基当且仅当

$$\forall f \in I, \quad \overline{f}^G = 0$$

或者等价地

$$\langle \mathrm{LT}(g_1), \cdots, \mathrm{LT}(g_t) \rangle = \langle \mathrm{LT}(I) \rangle$$

当给定的 G 不是 I 的 Gröbner 基时，可以通过 Buchberger 算法让 G 扩展成为 I 的 Gröbner 基。

1.1.1 An RSA Example

RSA 加密，已知

$$ed = 2\varphi(n) + 1, \quad e = 3, \quad \varphi^3(n) = a, \quad d^3 = b$$

那么可以通过构造如下的方程组解出 $\varphi(n)$：

$$\begin{aligned} f(x) = (2x+1)^3 - e^3 b &\equiv 0 \pmod{n} \\ g(x) = x^3 - a &\equiv 0 \pmod{n} \end{aligned}$$

```
#sage
from Crypto.Util.number import long_to_bytes

n =
a =
b =
e =

P.<x> = PolynomialRing(IntegerRing(), 1, order='lex')
b3 = int(b) * 27
f = (2*x+1)^3 - b3
g = x^3 - a
I = ideal(f,g)
gb = I.groebner_basis()
R = P.change_ring(IntegerModRing(n))
gb = [R(f) for f in gb if R(f)]
phi = (-gb[0][0]) % n
d = pow(e, -1, recovered_phi)
print(long_to_bytes(int(pow(c, d, n))))
```

1.1.2 An ECC Example

给定一组椭圆曲线 E 上的点 $\{(x_i, y_i)\}$，且已知 E 由有限域 $\mathbb{F}_p$ 上的方程 $y^2 = x^3 + ax + b$ 确定。欲求出 a, b, p，只需注意到

$$x_i a + b + kp + x_i^3 - y_i^2 = 0, \quad k \in \mathbb{Z} \tag{1}$$

可以看成是关于 a, b, p 的多项式，若用 a_0, b_0 表示值，用 a, b 表示形式变量，则上式化为

$$x_i(a - a_0) + (b - b_0) + k'p = 0, \quad k' \in \mathbb{Z}$$

因此，通过计算方程组 (1) 的 Gröbner 基 $\{a - a_0 + k_1 p, b - b_0 + k_2 p, p\}, k_1, k_2 \in \mathbb{Z}$，便可以求得 a_0, b_0, p。

实际操作时，计算出的 p 有可能会是真正的 p 的某一个倍数，但随着不同的点的个数增加，计算出 p 的可能性会迅速提高。

```
#sage
Points =

fs = []
P.<a,b> = PolynomialRing(ZZ)
for x, y in Points:
    f = x^3 + a*x + b - y^2
    fs.append(f)
I = Ideal(fs)
print(I.groebner_basis())
```

1.2 Invalid Curve Point Attack

众所周知，对于椭圆曲线 $E(\mathbb{F}_p, a, b)$ 以及两个点 P, Q 而言，改变 b 的值并不会改变 $P+Q$ 的结果。

现在我们有一个预言机 $\mathcal{O}$，它接受一个点的输入 P，并在 $E(\mathbb{F}_p, a, b)$ 上计算 sP，然后返回 sP 的笛卡尔 x 坐标。若 $\mathcal{O}$ 不验证输入的点 P 是否在曲线 $E(\mathbb{F}_p, a, b)$ 上而直接计算 sP，那么 $\mathcal{O}$ 的行为可以视作在任一条椭圆曲线 $E(\mathbb{F}_p, a, b')$ 上计算 sP，而椭圆曲线簇 $E(\mathbb{F}_p, a, b')$ 中很可能存在性质有缺陷的曲线，根据这种缺陷我们可以尝试恢复出 s。

已知 s 满足 $s^2 < u$，那么我们的攻击可以分为如下几步：

1. 在 $\mathbb{F}_p$ 中选择几个 $b_i, i = 1, 2, \cdots, m$，得到对应的 $E_i = E(\mathbb{F}_p, a, b_i)$，使得 $o_i = \#E_i$ 有较小的因子。
2. 从这些较小的因子中选出一些 $p_{ij}^{e_{ij}}, i = 1, 2, \cdots, m, j = 1, \cdots, n_i$，使得 $N = \prod_{i,j} p_{ij}^{e_{ij}} \geq u$。
3. 从每个 $E_i (n_i > 0)$ 中找到一个生成元 G_i，并且满足 $P_i = (o_i/N_i)G_i$ 的阶为 N_i，其中 $N_i = \prod_{j=1}^{n_i} p_{ij}^{e_{ij}}$。
4. 将所有得到的 P_i 输入 $\mathcal{O}$，得到一系列 $[s_i P]_x$，通过求解离散对数问题的算法可以快速计算出 s_i，并且有 $s \equiv s_i \pmod{N_i}$ 或者 $s \equiv -s_i \pmod{N_i}$，即 $s^2 \equiv s_i^2 \pmod{N_i}$。
5. 由于模数两两互质，根据中国剩余定理可以求出 $s^2 \bmod N$，也即 s^2。
6. 最后，通过 Tonelli-Shanks 算法在 $\mathbb{F}_N$ 上计算出 s^2 的平方根，即 $\pm s$。

现在，为了方便起见，假设 $\mathcal{O}$ 的返回值为点对 (x, y)（因此只需满足 $s < u$ 即可），$u = p$，对于 NIST192 的参数而言，攻击脚本示例如下：

```
#sage
def oracle(P: tuple) -> tuple:
    #...

# Finite field prime
p = 0xfffffffffffffffffffffffffffffffeffffffffffffffff
# Create a finite field of order p
FF = GF(p)
a = p - 3
# Curve parameters for the curve equation: y^2 = x^3 + a*x +b

# Define NIST 192-P
b192 = 0x64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1
n192 = 0xffffffffffffffffffffffff99def836146bc9b1b4d22831
P192 = EllipticCurve([FF(a), FF(b192)])

#   0 : 2^64 * 3 * 5 * 17 * 257 * 641 * 65537 * 274177    * 6700417 *
    67280421310721
# 170 : 73 * 35897 * 145069 * 188563 * 296041             * 749323 * 6286019 *
    6279866923899952450429 9

mods = []
vals = []

for b in [0, 170]:
```

```
    E = EllipticCurve([FF(a), FF(b)])
    G = E.gens()[0]
    factors = sage.rings.factorint.factor_trial_division(G.order(), 300000)
    G *= factors[-1][0]

    x, y = oracle(G.xy())
    H = E(x, y)

    # get dlog
    tmp = G.order()
    mods.append(tmp)
    vals.append(G.discrete_log(H,tmp))

s = CRT_list(vals, mods)
print(s)
```

1.3 Hidden Subset Sum Problem (HSSP)

1.3.1 Orthogonal Lattice

一个格（lattice）$\mathcal{L} \subset \mathbb{Z}^m$ 的正交格（orthogonal lattice）$\mathcal{L}^\perp$ 被定义为

$$\mathcal{L}^\perp = \{\boldsymbol{v} \in \mathbb{Z}^m : \forall \boldsymbol{b} \in \mathcal{L}, \langle \boldsymbol{v}, \boldsymbol{b} \rangle = 0\} = E_{\mathcal{L}}^\perp \cap \mathbb{Z}^m$$

而 $\mathcal{L}$ 的补（completion）$\bar{\mathcal{L}}$ 被定义为

$$\bar{\mathcal{L}} = E_{\mathcal{L}} \cap \mathbb{Z}^m = (\mathcal{L}^\perp)^\perp$$

$\mathcal{L}$ 被称为是完备的（complete）$\Leftrightarrow \bar{\mathcal{L}} = \mathcal{L}$。
那么显然有

1. $\mathcal{L}$ 是 $\mathcal{L}^\perp$ 的一个满秩子格。
2. $\dim \mathcal{L} + \dim \mathcal{L}^\perp = m$。
3. $\det(\mathcal{L}^\perp) = \det(\bar{\mathcal{L}}) \leq \det(\mathcal{L})$。

1.3.2 Computing the Orthogonal Lattice

设 $\boldsymbol{U}_{r \times m}$ 是 $\mathcal{L}$ 的行向量 $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_r$ 组成的矩阵，那么对

$$\mathcal{L}_c(\boldsymbol{U}) = \left(\begin{array}{c|c} c \cdot \boldsymbol{U}^T & \boldsymbol{I}_m \end{array} \right), \quad c \in \mathbb{N}_+, \quad c > 2^{\frac{m}{2}} \cdot \sqrt{m} \cdot \|\boldsymbol{U}\|^{\frac{r}{m-r}} \geq 2^{\frac{m-1}{2}} \cdot \lambda_{m-r}(\mathcal{L}^\perp)$$

进行 LLL 格基规约，得到行向量 $\boldsymbol{b}_1, \cdots, \boldsymbol{b}_m \in \mathbb{Z}^{r+m}$，那么 LLL 约化后的 $\mathcal{L}^\perp$ 由行向量 $\pi(\boldsymbol{b}_1), \cdots, \pi(\boldsymbol{b}_{m-r})$ 组成，其中

$$\pi(v_1, \cdots, v_{m+r}) = (v_{r+1}, \cdots, v_{m+r}) \in \mathbb{Z}^m$$

该算法的时间复杂度大致（Heuristically）是

$$\mathcal{O}\left(m^5 \left(m + \frac{m}{m-r} \log \|\boldsymbol{U}\|^2 \right) \right)$$

其中 $\|\boldsymbol{U}\|$ 代表 $\max\limits_i \|\boldsymbol{a}_i\|$。
而在模环 $\mathbb{Z}_q$ 上，

$$\mathcal{L}_{c,q}(\boldsymbol{U}) = \left(\begin{array}{c|c} c \cdot \boldsymbol{U}^T & \boldsymbol{I}_m \\ \hline cq \cdot \boldsymbol{I}_r & \boldsymbol{O}_{r \times m} \end{array} \right)_{(m+r) \times (m+r)}$$

1.3.3 The Problem

隐藏子集和问题：M 是一个整数，$\alpha_1,\cdots,\alpha_n\in\mathbb{Z}_M$，$\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n\in\mathbb{Z}_2^m$，而 $\boldsymbol{h}=(h_1,\cdots,h_m)\in\mathbb{Z}^m$ 满足

$$\boldsymbol{h}\equiv\alpha_1\boldsymbol{x}_1+\cdots+\alpha_n\boldsymbol{x}_n\pmod M$$

现在，给定 M 与 $\boldsymbol{h}$，要求计算出 $\boldsymbol{\alpha}=(\alpha_1,\cdots,\alpha_n)$ 与全部的 $\boldsymbol{x}_i$，排列顺序可以不同。

1.3.4 Nguyen-Stern Attack

Step 1. 设 $\mathcal{L}_0=\Lambda_M^\perp(\boldsymbol{h})=\{\boldsymbol{u}\in\mathbb{Z}^m:\langle\boldsymbol{u},\boldsymbol{h}\rangle\equiv0\pmod M\}$，$\mathcal{L}_{\boldsymbol{x}}=\mathcal{L}(\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n)$，$\boldsymbol{p}_{\boldsymbol{u}}=(\langle\boldsymbol{u},\boldsymbol{x}_1\rangle,\cdots,\langle\boldsymbol{u},\boldsymbol{x}_n\rangle)$，那么显然，如果 $\|\boldsymbol{p}_{\boldsymbol{u}}\|<\lambda_1(\mathcal{L}_0)$，就一定有 $\boldsymbol{p}_{\boldsymbol{u}}=0$，即 $\boldsymbol{u}\in\mathcal{L}_{\boldsymbol{x}}^\perp$。

不妨设 $M\in\mathbb{P},h_1\neq0,\gcd(h_1,M)=1$，那么可以像这样构造 $\mathcal{L}_0$：

$$\mathcal{L}_0=\left(\begin{array}{c|c}M & \boldsymbol{O}_{1\times(m-1)}\\ \hline -h_1^{-1}[M]\cdot\boldsymbol{h}' & \boldsymbol{I}_{m-1}\end{array}\right)$$

其中 $\boldsymbol{h}'=(h_2,\cdots,h_m)\in\mathbb{Z}^{m-1}$。

对于 LLL 约化后的 $\mathcal{L}$ 的前 $m-n$ 个行向量 $\boldsymbol{u}_1,\cdots,\boldsymbol{u}_{m-n}$，当

$$\sqrt{mn}\cdot2^{\frac{m}{2}}\cdot\lambda_{m-n}\left(\mathcal{L}_{\boldsymbol{x}}^\perp\right)<\lambda_1\left(\Lambda_M^\perp(\boldsymbol{\alpha})\right)\tag{2}$$

时，$\boldsymbol{u}_1,\cdots,\boldsymbol{u}_{m-n}$ 构成 $\mathcal{L}_{\boldsymbol{x}}^\perp$ 的基向量。此外，

$$\lambda_1(\Lambda_M^\perp(\boldsymbol{\alpha}))\geq M^{1/n}/4\tag{3}$$

对于 $1/2$ 随机选择的 $\boldsymbol{\alpha}$ 成立。再对 $\lambda_{m-n}\left(\mathcal{L}_{\boldsymbol{x}}^\perp\right)$ 进行少量的变形，结合 (2), (3) 便得到：对于 $1/2$ 随机选择的 $\boldsymbol{\alpha}$，当 $m>n,\log M\geq2mn\log m,M\in\mathbb{P}$ 时，可以在多项式时间内计算出 $\bar{\mathcal{L}}_{\boldsymbol{x}}$。

Step 2. 在上一步中，我们得到了 $\bar{\mathcal{L}}_{\boldsymbol{x}}$ 的一组基 $(\boldsymbol{c}_1,\cdots,\boldsymbol{c}_n)$，现在不妨设 $\mathcal{L}_{\boldsymbol{x}}$ 时完备的，那么对其进行参数 $\beta=\omega(n/\log n)$，时间复杂度为 $2^{\Omega(n/\log n)}$ 的 BKZ 规约，再进行 $\mathcal{O}(n^3)$ 次某种（TL;DR）测试即可恢复出所有的 $\boldsymbol{x}_i$，其中“TL;DR”定义为“Too long, didn't read ;)”。

Step 3. 最后一步，求解线性方程组恢复出 $\boldsymbol{\alpha}$ 即可。

复杂度分析：TL;DR

1.3.5 Jean-S´ebastien Coron and Agnese Gini's Super Attack

TL;DR，但是上述攻击中的第二步从指数时间优化到了多项式时间。

1.4 Approximate Common Divisor Problem (ACD)

ACD 问题分为两种：GACD（General）和 PACD（Partial）。设 $\gamma > \eta > \rho$，一个 $\mathcal{GACD}(\gamma, \eta, \rho)$ 问题被定义为：给定一个 η-bit 奇数 p 与下述集合中的一些（关于 γ, η, ρ 的多项式个）样例，

$$\{a_i = pq_i + r_i : q_i \in \mathbb{Z} \cap (0, 2^\gamma/p), r_i \in \mathbb{Z} \cap (-2^\rho, 2^\rho)\}$$

那么 p 是多少。

而 $\mathcal{PACD}(\gamma, \eta, \rho)$ 问题则在给出上述样例的基础上增加一条 $a_0 = pq_0$。

1.4.1 The First OL Attack

令

$$\mathcal{L}_1(\alpha) = \begin{pmatrix} \alpha a_1 & 1 & & \\ \alpha a_2 & & 1 & \\ \vdots & & & \ddots \\ \alpha a_n & & & 1 \end{pmatrix}_{n \times (n+1)}$$

其中 α 是一个足够大的数，那么从 LLL 或 BKZ 规约后得到的较短向量中筛选出 $|\langle \boldsymbol{u}, \boldsymbol{r} \rangle| < p$ 的向量 $\boldsymbol{u}$，便可以得到正交于 $\boldsymbol{a}, \boldsymbol{r}$ 的向量，并且 $\boldsymbol{u}$ 根据线性性还正交于 $\boldsymbol{q}$，找到 $n-2$ 个这样的向量，张成的空间记作 V，那么 $V^\perp$ 包含 $\boldsymbol{a}, \boldsymbol{r}$，又因为 $\boldsymbol{r}$ 是一个短向量，故对 $V^\perp$ 格基规约以后可以得到 $\boldsymbol{r}$，从而得到 $\boldsymbol{q}, p$。

优化：将第一列改为 $\left(\lfloor \frac{a_1}{\alpha} \rfloor, \cdots, \lfloor \frac{a_n}{\alpha} \rfloor\right)^T$，以在规约时获得更好的时空复杂度。

1.4.2 The Second OL Attack

令

$$\mathcal{L}_2(\alpha) = \begin{pmatrix} a_1 & 2^\rho & & \\ a_2 & & 2^\rho & \\ \vdots & & & \ddots \\ a_n & & & 2^\rho \end{pmatrix}_{n \times (n+1)}$$

规约后得到的 $n-1$ 个较短向量 $\boldsymbol{t} = \left(\sum_{i=1}^n u_i a_i, u_1 2^\rho, \cdots, u_n 2^\rho\right)$，大概率满足

$$\sum_{i=1}^n u_i a_i = \sum_{i=1}^n u_i r_i$$

换句话说，$\boldsymbol{u}$ 与 $\boldsymbol{q}$ 正交，故求得的核空间中应只有一个向量 $\frac{1}{g}\boldsymbol{q}$，其中 $g = \gcd(\boldsymbol{q})$。

2^ρ 起到的作用是平衡目标向量，即，使规约出来的向量中的第一项尽可能小。如果不进行平衡，那么可能导致一些向量前一项过大，后面的项过小。

1.4.3 The Third OL Attack

令

$$\mathcal{L}_3 = \begin{pmatrix} 1 & & & a_1 \\ & 1 & & a_2 \\ & & \ddots & \vdots \\ & & 1 & a_{n-1} \\ & & & a_n \end{pmatrix}_{n\times n}$$

这样，$\boldsymbol{q}$ 的求解思路与第二类 OL 攻击类似。

1.4.4 The Third OL Attack on PACD

对于 PACD 问题第三种格可以改进为

$$\mathcal{L} = \begin{pmatrix} 1 & & & \frac{-a_1}{a_n}[a_0] \\ & 1 & & \frac{-a_2}{a_n}[a_0] \\ & & \ddots & \vdots \\ & & 1 & \frac{-a_{n-1}}{a_n}[a_0] \\ & & & a_0 \end{pmatrix}_{n\times n}$$

1.4.5 Applications

Maybe next time :)

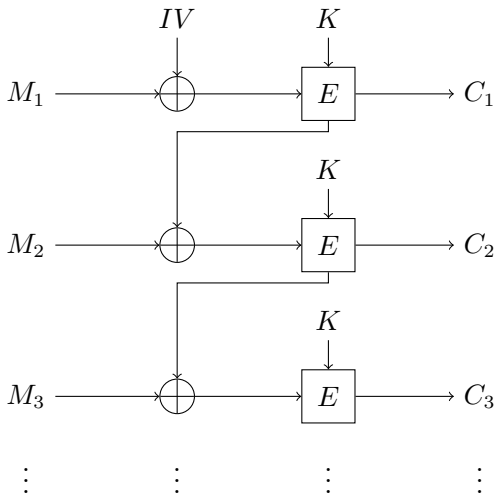
1.5 Padding Oracle Attack

给定一个预言机 $\mathcal{O}$，将用户输入的 IV+密文进行 CBC 模式解密，并尝试判断解密出来的明文是否符合某种 Padding 规则。比如对于一个不超过八字节的块而言，规定 Padding 的规则为：

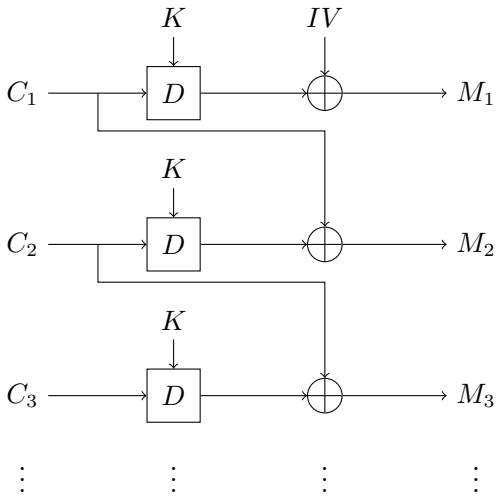
```
** ** ** ** ** ** ** **          ** ** ** ** ** ** ** ** 08 08 08 08 08 08 08 08
** ** ** ** ** ** **             ** ** ** ** ** ** ** 01
** ** ** ** ** **                ** ** ** ** ** ** 02 02
** ** ** ** **                   ** ** ** ** ** 03 03 03
** ** ** **                      ** ** ** ** 04 04 04 04
** ** **                         ** ** ** 05 05 05 05 05
** **                            ** ** 06 06 06 06 06 06
**                               ** 07 07 07 07 07 07 07
```

若解密出的明文不满足这种 Padding 规则，则返回一种结果（如 0），反之返回另一种结果（如 1）。

CBC 种模式中会在密文 C 的开头引入初始化向量 IV 来增强密文的随机性，加密过程如下图



解密过程则为



根据上述原理我们可以轻易地写出攻击脚本。

```python
def oracle(payload: bytes) -> bool:
    #...

BYTE_NB = 8

def poc(IV, encrypted):
    block_number = len(encrypted)//BYTE_NB
    decrypted = bytes()
    # Go through each block
    for i in range(block_number, 0, -1):
        current_encrypted_block = encrypted[(i-1)*BYTE_NB:(i)*BYTE_NB]
        # At the first encrypted block, use the initialization vector if it is known
        if(i == 1):
            previous_encrypted_block = bytearray(IV)
        else:
            previous_encrypted_block = encrypted[(i-2)*BYTE_NB:(i-1)*BYTE_NB]
        bruteforce_block = previous_encrypted_block
        current_decrypted_block = bytearray(IV)
        padding = 0
        # Go through each byte of the block
        for j in range(BYTE_NB, 0, -1):
            padding += 1
            # Bruteforce byte value
            for value in range(0,256):
                bruteforce_block = bytearray(bruteforce_block)
                bruteforce_block[j-1] = (bruteforce_block[j-1] + 1) % 256
                joined_encrypted_block = bytes(bruteforce_block) + current_encrypted_block
                # Ask the oracle
                if(oracle(joined_encrypted_block)):
                    current_decrypted_block[-padding] = bruteforce_block[-padding] ^ previous_encrypted_block[-padding] ^ padding
                    # Prepare newly found byte values
                    for k in range(1, padding+1):
                        bruteforce_block[-k] = padding+1 ^ current_decrypted_block[-k] ^ previous_encrypted_block[-k]
                    break
        decrypted = bytes(current_decrypted_block) + bytes(decrypted)
    return decrypted[:-decrypted[-1]]  # Padding removal
```