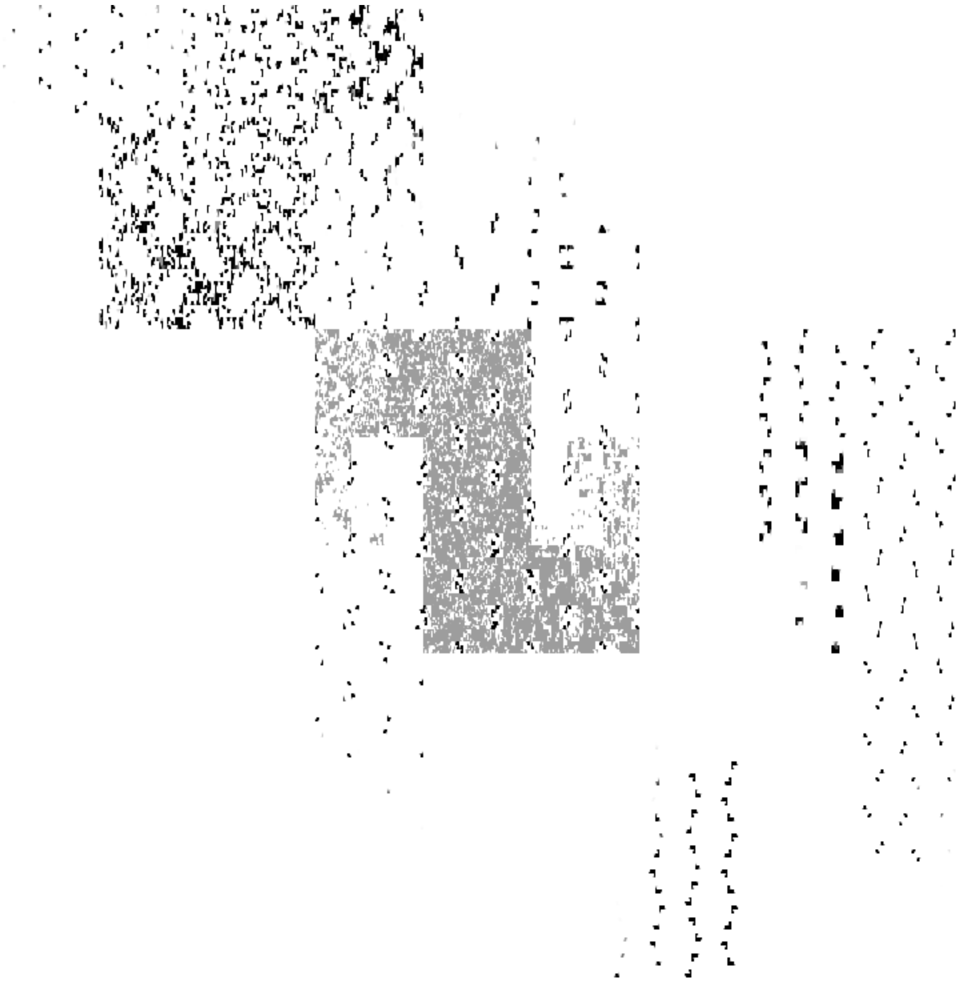
🌲 💐 Spring notes for CTF 🚩

Week 2

Astrageldon

2024-03-22



Contents

1 Crypto

1.1 Discrete Logarithm Problems in Matrix Groups

问题：给定有限域上的矩阵 $\mathbf{M} \in \mathbb{F}_q^{n\times n}$，$\mathbf{Y} = \mathbf{M}^x$，求 x。

此时我们有

$$\mathbf{Y} = \mathbf{M}^x \Rightarrow \det(\mathbf{Y}) \equiv \det(\mathbf{M})^x \pmod q$$

显然当 $\det(\mathbf{M}) \neq 0, 1$ 时（此时 $\mathbf{M} \in \mathrm{GL}_n(\mathbb{F}_q)\backslash\mathrm{SL}_n(\mathbb{F}_q)$），我们可以通过求解上述离散对数问题来获得 x。

当 $\mathbf{M}$ 不满足上述条件时，我们尝试将其转化为 Jordan 标准型：

$$\mathbf{M} = \mathbf{P}\mathbf{J}\mathbf{P}^{-1}$$

求出 $\mathbf{P}$ 与 $\mathbf{J}$ 后，注意到

$$\mathbf{J} = \mathbf{P}^{-1}\mathbf{Y}\mathbf{P}$$

并注意到每个 Jordan 块 $\mathbf{D}_k$ 都具有如下的性质：

$$\mathbf{D}_k = \begin{pmatrix} \lambda_k & 1 & & & \\ & \lambda_k & \ddots & & \\ & & \ddots & \ddots & \\ & & & \lambda_k & 1 \\ & & & & \lambda_k \end{pmatrix}, \quad \mathbf{D}_k^x = \begin{pmatrix} \lambda_k^x & x\lambda_k^{x-1} & * & \cdots & * \\ & \lambda_k^x & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & * \\ & & & \lambda_k^x & x\lambda_k^{x-1} \\ & & & & \lambda_k^x \end{pmatrix}$$

我们据此可以很轻易地求出 x。

上述方法非常笨重，然而，改进的方法也相当自然。

众所周知

$$\lambda_i \text{ is an eigenvalue of } \mathbf{A} \Rightarrow \lambda_i^k \text{ is an eigenvalue of } \mathbf{A}^k$$

所以我们计算特征多项式

$$\chi_{\mathbf{M}}(\lambda) := \det(\lambda\mathbf{I} - \mathbf{M}) \quad \text{on } \mathbb{F}_q$$

的分裂域（splitting field）$\mathbb{F} = \mathbb{F}_{q^t}$，显然此时 $\chi_{\mathbf{M}}(\lambda)$ 与 $\chi_{\mathbf{Y}}(\lambda)$ 在 $\mathbb{F}$ 上均有 $\deg\chi_{\mathbf{M}} = \deg\chi_{\mathbf{Y}} = n$ 个根，而两组根之间显然存在幂次关系。

随手搓了一个脚本，急需完善大致能用，只不过需要注意的是 $\mathbb{F}_{q'}$ 上离散对数的计算速度可能取决于 $q' - 1 = p'^k - 1$ 的因子大小⋯⋯ 完整代码见 Github🔗 😊

`matrix_dlp.sage` *(part)*

```
def matrix_dlp(Y, M):
    assert M.base_ring() == Y.base_ring()
    Fq = M.base_ring()
```

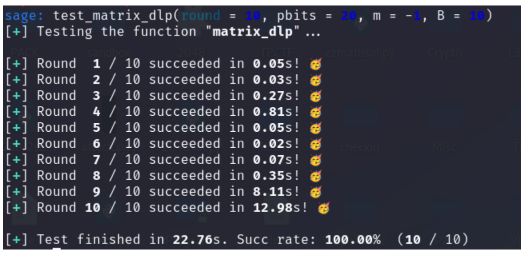
```
    assert Fq.is_field()
    assert Fq.is_finite()
    p = Fq.base().order()

    char_poly_M = M.characteristic_polynomial()
    F.<x> = char_poly_M.splitting_field()
    J_M = M.change_ring(F).jordan_form()
    J_Y = Y.change_ring(F).jordan_form()
    assert J_M.is_diagonal()
    assert J_Y.is_diagonal()
    M_eigenvals = sorted(set([J_M[i,i] for i in range(J_M.dimensions()[0])]), key
        = lambda entry: entry.polynomial().degree())
    Y_eigenvals = sorted(set([J_Y[i,i] for i in range(J_Y.dimensions()[0])]), key
        = lambda entry: entry.polynomial().degree())

    for entry_M in M_eigenvals:
        for entry_Y in Y_eigenvals:
            try:
                res = discrete_log(entry_Y, entry_M)
                if M**res == Y: return res
            except (ValueError, ZeroDivisionError):
                continue
    return None
```

简易测试:

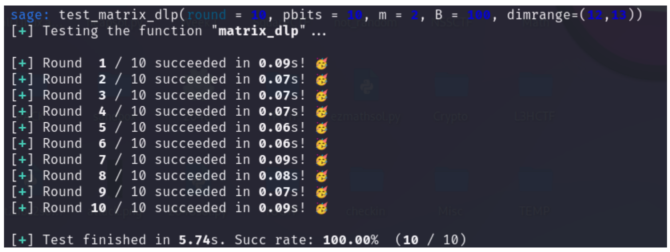
round = 10, pbits = 20, B = 10, m = -1, dimrange = (5,6)

round = 10, pbits = 10, B = 100, m = 2, dimrange = (12,13)

round = 10, pbits = 10, B = 100000, m = -1, dimrange=(10,11)



```
sage: test_matrix_dlp(round = 10, pbits = 20, m = -1, B = 10)
[+] Testing the function "matrix_dlp" ...

[+] Round  1 / 10 succeeded in 0.05s! 🥳
[+] Round  2 / 10 succeeded in 0.03s! 🥳
[+] Round  3 / 10 succeeded in 0.27s! 🥳
[+] Round  4 / 10 succeeded in 0.81s! 🥳
[+] Round  5 / 10 succeeded in 0.05s! 🥳
[+] Round  6 / 10 succeeded in 0.02s! 🥳
[+] Round  7 / 10 succeeded in 0.07s! 🥳
[+] Round  8 / 10 succeeded in 0.35s! 🥳
[+] Round  9 / 10 succeeded in 8.11s! 🥳
[+] Round 10 / 10 succeeded in 12.98s! 🥳

[+] Test finished in 22.76s. Succ rate: 100.00%  (10 / 10)
```



```
sage: test_matrix_dlp(round = 10, pbits = 10, m = 2, B = 100, dimrange=(12,13))
[+] Testing the function "matrix_dlp" ...

[+] Round  1 / 10 succeeded in 0.09s! 🥳
[+] Round  2 / 10 succeeded in 0.07s! 🥳
[+] Round  3 / 10 succeeded in 0.07s! 🥳
[+] Round  4 / 10 succeeded in 0.07s! 🥳
[+] Round  5 / 10 succeeded in 0.06s! 🥳
[+] Round  6 / 10 succeeded in 0.06s! 🥳
[+] Round  7 / 10 succeeded in 0.09s! 🥳
[+] Round  8 / 10 succeeded in 0.08s! 🥳
[+] Round  9 / 10 succeeded in 0.07s! 🥳
[+] Round 10 / 10 succeeded in 0.09s! 🥳

[+] Test finished in 5.74s. Succ rate: 100.00%  (10 / 10)
```

```
sage: test_matrix_dlp(round = 10, pbits = 10, m = -1, B = 100000, dimrange=(10,11))
[+] Testing the function "matrix_dlp" ...

[+] Round  1 / 10 succeeded in 17.29s! 🥳
[+] Round  2 / 10 succeeded in 0.04s! 🥳
[+] Round  3 / 10 succeeded in 66.08s! 🥳
[+] Round  4 / 10 succeeded in 0.14s! 🥳
[+] Round  5 / 10 succeeded in 0.08s! 🥳
[+] Round  6 / 10 succeeded in 0.04s! 🥳
[+] Round  7 / 10 succeeded in 0.04s! 🥳
[+] Round  8 / 10 succeeded in 0.05s! 🥳
[+] Round  9 / 10 succeeded in 0.05s! 🥳
[+] Round 10 / 10 succeeded in 0.05s! 🥳

[+] Test finished in 83.85s. Succ rate: 100.00%  (10 / 10)
```

1.2 Cryptanalysis of the Cyclic Redundancy Check

给定如下的 CRC 函数：

```python
def crc(msg,IN,OUT,POLY):
    crc256 = IN
    for b in msg:
        crc256 ^= b
        for _ in range(8):
            crc256 = (crc256 >> 1) ^ (POLY & -(crc256 & 1))
    return (crc256 ^ OUT).to_bytes(32,'big')
```

那么显然该函数具有多重线性性，即：

$$\begin{aligned}&\mathrm{crc}(\mathrm{msg}_1 \oplus \mathrm{msg}_2, \mathrm{IN}_1 \oplus \mathrm{IN}_2, \mathrm{OUT}_1 \oplus \mathrm{OUT}_2, \mathrm{POLY})\\ =\ &\mathrm{crc}(\mathrm{msg}_1, \mathrm{IN}_1, \mathrm{OUT}_1, \mathrm{POLY}) \oplus \mathrm{crc}(\mathrm{msg}_2, \mathrm{IN}_2, \mathrm{OUT}_2, \mathrm{POLY})\end{aligned}$$

据此，我们可以将取模之前的运算做如下改写：

$$\mathrm{crc}(\mathbf{m}, 0, 0, 0) = \mathbf{A}\mathbf{m}$$

其中 $\mathbf{A} \in \mathbb{F}_2^{256\times 256}$，$\mathbf{m} \in \mathbb{F}_2^{256}$，据此，我们只要求出标准正交基赋给 $\mathrm{crc}(\cdot, 0, 0, 0)$ 的所有值就可以推算出 $\mathbf{A}$，此后解密的过程便极其容易：

$$\mathbf{A} = \begin{pmatrix} | & \cdots & | \\ \mathrm{crc}(\mathbf{e}_1, 0, 0, 0) & \cdots & \mathrm{crc}(\mathbf{e}_{256}, 0, 0, 0) \\ | & \cdots & | \end{pmatrix}_{256\times 256}, \qquad \mathbf{m} = \mathbf{A}^{-1}\mathrm{crc}(\mathbf{m}, 0, 0, 0)$$

我们还可以用多项式的形式来表示上述函数，也即：

$$\mathrm{crc}\big(M(x), 0, 0, G(x)\big) = M(x) \cdot x^{256} \bmod G(x)$$

$$\mathrm{crc}\big(M(x), A(x), B(x), G(x)\big) = \big(M(x) + A(x)\big) \cdot x^{256} + B(x) \bmod G(x)$$

其中 $M(x), G(x), A(x), B(x) \in \mathbb{F}_2[x]$，$\deg M < 256$，$\deg G = 256$。
相比于矩阵表示，它在外观上长得更加简洁。

现在假设我们需要求出模多项式 $G(x) = x^{256} + g(x)$ $(\deg g < 256)$，那么我们需要一对满足下式的输入 $M_1(x), M_2(x)$：

$$M_1(x) + M_2(x) = 1$$

从而

$$\begin{aligned}C_1(x) + C_2(x) &\equiv \big(M_1(x) + M_2(x)\big) \cdot x^{256} \pmod{G(x)}\\ &\equiv g(x) \pmod{G(x)}\end{aligned}$$

如果有一个 CRC Oracle 能够告诉我们 $C_1(x)$ 与 $C_2(x)$ 的值，那么我们就能得到 $G(x)$，如果还知道另外两个（长度不超过 256 比特）消息的加密异或值 $C_3(x) + C_4(x)$，那么通过在 $\mathbb{F}_2[x]/(G(x))$ 上除去 x^{256}，便可以得到两个消息的异或值 $M_3(x) + M_4(x)$。