Survey: Lattice Reduction Attacks on RSA

1. 背景和动机

几十年来， 密码破译者一直在寻找方法破解 RSA。对 RSA 的攻击分为两类：对系统实施的攻击和数学攻击。在过去这些年，RSA 的数学密码分析已经被证明是困难的。但研究人员发现可以攻击 RSA 的宽松模型。宽松模型是指知道明文信息的“一部分”或知道素数 p,q 的“近似值”或私有指数“足够小”……在对这些宽松模型的攻击研究中格的约简技术被证明是密切相关的。

1995 年，Coppersmith 发表了一篇关于如何使用格的约简技术（如 LLL 算法）攻击 RSA 宽松模型的文章。

几年后，Howgrave-Graham 重新审视了 Coppersmith 的算法，并对其进行改进，使其更容易理解和应用。

2. RSA

要使用 RSA 进行加密，需要一个公钥来加密和一个私钥来解密

密钥的生成：首先生成两个素数 p 和 q（p 和 q 的大小应该相同），使用 p 和 q 得到模数 $N = p \times q$，随机选择一个整数 e (满足 $\gcd(e, \varphi(N)) = 1$, $\varphi(N)$为 N 的欧拉函数)，找到一个整数 d (使得 $ed \equiv 1 \bmod \varphi(N)$)。现在 (N, d) 作为私钥，(N, e) 作为私钥。

加密：$c = m^e \pmod N$

解密：$m = c^d \pmod N$

3. 格(Lattic)

简单描述：n 维空间中具有周期结构的点集合

第一种定义：$L(b_1, \cdots, b_n) = \{\sum x_i b_i \mid x_i \in \mathbb{Z}\}$，$b_i \in R^m$为线性无关的向量

第二种定义：$L(B) = \{Bx \mid x \in \mathbb{Z}^n\}$，$B$为 $m \times n$ 的矩阵

这两种定义本质上是相同的，B即$b_1, \cdots, b_n$为行向量组成的矩阵，B称为格L的基

一个格L可以由不同的基B所形成，如果一个格L的基B是约简基（即施密特正交化的矩

阵），那么定义这个格的行列式为：$det(L) = \prod \|\tilde{b}_i\|$

为了方便计算，我们使用满秩和上三角的格基，此时行列式即可通过对角项的乘积得到

3.1. Lenstra-Lenstra-Lovasz(LLL)

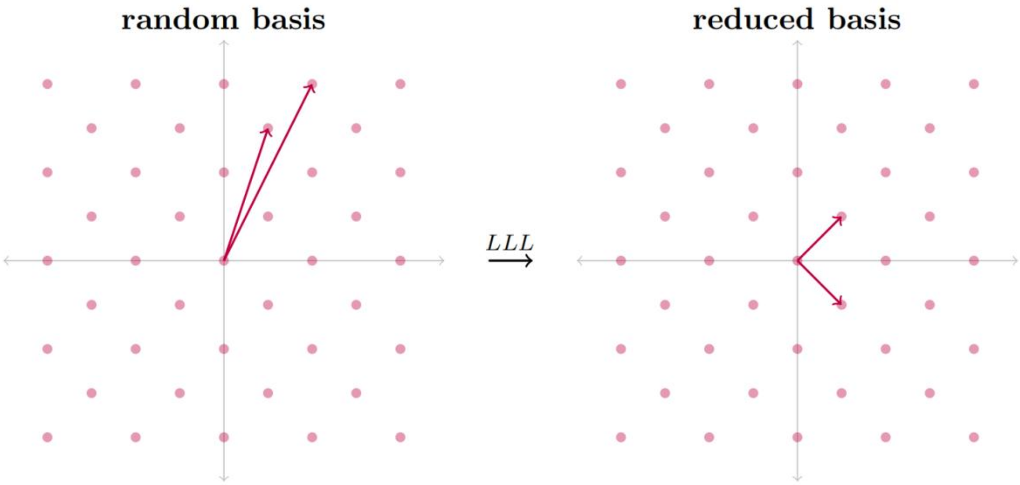
LLL 格基简约算法是一个在多项式时间内寻找约简基的算法

δ-LLL 算法应用于格L的基B，生成该格L的一个约简基$B' = \{b_1, \cdots, b_n\}$，该δ-LLL 约简基满足：

$$\forall 1 \le j < i \le n,\ 有\ |u_{i,j}| \le \frac{1}{2}$$

$$\forall 1 \le i < n,\ 有\ \delta \bullet \|\tilde{b}_i\|^2 \le \|u_{i+1,i} \bullet \tilde{b}_i + \tilde{b}_{i+1}\|^2$$

$$u_{i,j} = \frac{b_i \bullet \tilde{b}_j}{\tilde{b}_j \bullet \tilde{b}_j}\ 且\ \tilde{b}_1 = b_1$$



3.2. LLL 中对我们有用的性质

LLL 可以得到最短向量问题(SVP)的近似值，这对我们很有用，因为我们可以把格基的行向量看作多项式的系数向量，通过 LLL 可以找到那些多项式（具有“足够小”的系数）的线性组合

LLL 算法输出的约简基向量v_i满足：

$$\|v_1\| \le \|v_2\| \le \cdots \le \|v_n\| \le 2^{\frac{n(n-1)}{4(n+1-i)}} \bullet det(L)^{\frac{1}{n+1-i}}$$

4. Known High Bits Message Attack

攻击条件:知道消息 m 的很大的一部分m_0，即$m = m_0 + x$，但是我们不知道x

我们的问题就转化为了求一个多项式方程的根:

$$f(x) = (m_0 + x)^e - c \text{ with } f(x_0) = 0 \ (\text{mod } N)$$

Coppersmith 说如果x_0和e“足够小”, 我们就能够在多项式时间内求解这个多项式方程。x_0和e需要满足的要求:

定理: 令N为一个未知因数分解的数字，但存在一个除数b（满足$b \geq N^{\beta}, 0 < \beta \leq 1$)

令$f(x)$为δ维一元多项式，令$c \geq 1$, 那么我们就能在 $\mathbb{O}(c\delta^5 log^9(N))$ 时间内求解下面方程的全部解x_0

$$f(x) = 0 \ (\text{mod } b) \text{ with } |x_0| \leq c \bullet N^{\frac{\beta^2}{\delta}}$$

把该定理应用在我们要求解的多项式方程上，就是令 $c = 1, \beta = 1$，则$|x_0| \leq N^{\frac{1}{e}}$

5. Coppersmith 具体流程

我们知道在整数环上求解多项式是困难的，但在整数上求解多项式是可以实现的，因此 Coppersmith 的直觉是寻找这样一个多项式:

$$f(x_0) = 0 \ (\text{mod } N) \text{ with } |x_0| < X$$

$$\downarrow$$

$$g(x_0) = 0 \text{ over } \mathbb{Z}$$

但是如何通过整数环上的多项式f找到与其有同根的多项式 g 呢?

Howgrave-Graham 定理：令$g(x)$是具有n项的单变元多项式，m是一个正整数。如果满足:

(1) $g(x_0) \equiv 0 \text{mod} N^m, |x_0| < X$

(2) $\|g(xX)\| < \frac{N^m}{\sqrt{n}}$

则有$g(x_0) = 0$ 在整数上成立

根据 Howgrave-Graham 定理，我们便可以通过组合以x_0为根的多项式f_i来找到这个多项

式 g

LLL 约简有两条性质对我们很有用：

-仅在基向量上做整线性组合

-输出的最短向量有界

第一条允许我们构建一个在模N^m下仍然以x_0为根的多项式：

$$g(x_0) = \sum_{i=1}^{n} a_i \cdot f_i(x_0) = 0 \pmod{N^m} \quad a_i \in \mathbb{Z}$$

第二条使得我们构建的多项式可以满足 Howgrave-Graham 定理的第二点要求

有了这两条性质，现在我们只需要以$f_i(xX)$为行向量创建格基，LLL 输出的最短向量b_1

作为系数向量，$b_1 f_i(x_0)$即为我们要找的多项式 g

现在让我们来回顾一下整体流程

$$f(x_0) = 0 \pmod N \text{ with } |x_0| < X$$

$$\downarrow$$

$$\text{generate } f_i \text{ s.t. } f_i(x_0) = 0 \pmod{N^m}$$

$$\downarrow$$

$$B = \begin{pmatrix} f_i(xX) \\ \vdots \\ f_n(xX) \end{pmatrix}$$

$$\downarrow \text{LLL}$$

$$B' = \begin{pmatrix} b_1 = g(xX) \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\downarrow$$

$$g(x_0) = 0 \pmod{N^m}$$
$$\|g(xX)\| < \frac{N^m}{\sqrt{n}}$$

$$\downarrow$$

$$g(x_0) = 0 \text{ over } \mathbb{Z}$$

6. 实验验证

使用 sage 实现，因为 sage 有打包好的 LLL 函数，在数论领域应用起来很方便

RSA 参数信息

```
# RSA公私钥生成
length_N = 1024
unknow_bit = 200
e = 3
p = next_prime(2^int(round(length_N/2)))
q = next_prime(p)
N = p*q
ZmodN = Zmod(N);

# 使用私钥加密信息
K = ZZ.random_element(0, 2^unknow_bit)
Kdigits = K.digits(2)
M = [0]*unknow_bit + [1]*(length_N-unknow_bit);    #信息由未知信息和已知信息组成，未知信息即我们要破解的部分
for i in range(len(Kdigits)):
    M[i] = Kdigits[i]
M = ZZ(M, 2)
C = ZmodN(M)^e
```

关键函数代码（input：整数环上的多项式方程，output：该方程的根）

```
def Coppersmith(pol, N, beta, mm, tt, XX):
    dd = pol.degree()
    nn = dd * mm + tt

    polZ = pol.change_ring(ZZ)
    x = polZ.parent().gen()

    gg = []
    for ii in range(mm):
        for jj in range(dd):
            gg.append((x * XX)**jj * N**(mm - ii) * polZ(x * XX)**ii)
    for ii in range(tt):
        gg.append((x * XX)**ii * polZ(x * XX)**mm)

    # 创建格基B
    BB = Matrix(ZZ, nn)
    for ii in range(nn):
        for jj in range(ii+1):
            BB[ii, jj] = gg[ii][jj]
    print("fi(xX)创建的格基为:")
    print_LB(BB, N^mm)

    # LLL
    BB = BB.LLL()

    # 使用最短向量构整数上的多项式
    new_pol = 0
    for ii in range(nn):
        new_pol += x**ii * BB[0, ii] / XX**ii

    roots = []
    potential_roots = new_pol.roots()
    for root in potential_roots:
        roots.append(ZZ(root[0]))

    return roots
```

实验结果（使用在线 sage 运行）

Share

```
fi(xX)创建的格基为:
00 X 0 0 0 0 0 0 0 0 ~
01 0 X 0 0 0 0 0 0 0 ~
02 0 0 X 0 0 0 0 0 0 ~
03 X X X X 0 0 0 0 0
04 0 X X X X 0 0 0 0
05 0 0 X X X X 0 0 0
06 X X X X X X X 0 0
07 0 X X X X X X X 0
08 0 0 X X X X X X X
原未知信息x0: 662698525180008765767004265500287434577394358589571238402007
计算得到的未知信息x0: [662698525180008765767004265500287434577394358589571238402007]
耗时: 0.0627145767211914 s
```

Help | Powered by SageMath

7. 总结

论文中还有一个 Håstad 提出的 RSA 广播攻击。就是一个用户使用同一个加密指数 e 加密了同一个密文 m，并发送给了其他 e 个用户，此种情况就可以进行广播攻击，攻击原理比较简单，只需要使用中国剩余定理即可，例如:

$$c_1 = m^3 \bmod n_1$$
$$c_2 = m^3 \bmod n_2$$
$$c_3 = m^3 \bmod n_3$$

则 $m^3 \equiv C \bmod n_1 n_2 n_3$，对 C 开三次方根即可得到 m 的值
原理很直观，就不进行实验验证了

Copppersmith 的想法很巧妙，先将 RSA 的信息破解问题转化为求解多项式方程，再利用格理论来求解多项式方程，应用起来速度也很快。
不过这些方法都是用来攻击 RSA 宽松模型的，在现实应用中很容易避免产生这些问题，因此 RSA 公钥系统还是很安全的。
格理论除了用来攻击这些 RSA 宽松模型，在密码学中还有很多其他应用。格理论的潜力还有待挖掘，其抗量子性也让越来越多人重视起来。