What did you learn?

> Attacks

Number 33: How does the Bellcore attack work against RSA with CRT?

> 钟核攻击? 🐷

RSA-CRT (#21) 是一种加速 RSA 加密和解密的方法，但是如果实现 RSA 的硬件（或软件）不时地产生错误或故障，那么就可以利用其来进行攻击。也就是我们接下来要介绍的。

RSA 是基于大整数分解问题的公钥密码算法，对其攻击往往涉及对模数的因式分解，但是 Boneh、DeMillo 和 Lipton 找到一种攻击，可以避免直接分解模数。他们表明，错误的密码值可以被攻击者利用，进而危害安全性。

首先回忆一下 RSA-CRT：

本来是是计算 $S = x^d \ mod \ N$，现在首先计算 $S_1 = x^d \ mod \ p$ 和 $S2 = x^d \ mod \ q$，然后再通过 CRT 计算 S。

如果我们需要对同一消息 x 的进行两次签名, 一次签名正确 $S = x^d \ mod \ N$，另一个签名 $\hat{S}$ 将错误。 首先需要计算 S_1 和 $S2$，同样的，也需要计算 $\hat{S}_1$ 和 $\hat{S}2$。 假设仅在计算 $\hat{S}_1$ 时出现了错误，就会造成 $S_1 \neq \hat{S}_1 \ mod \ p$，但是 $S_2 = \hat{S}_2$。也就意味着 $S \neq \hat{S} \ mod \ p, S = \hat{S} \ mod \ q$。也就有了:

$$gcd(S - \hat{S}, N) = q$$

结果就是，仅用一个错误签名就成功分解了 N。

Number 34: Describe the Baby-Step/Giant-Step method for breaking DLPs

Baby-Step/Giant-Step 是由 Daniel Shanks 提出的一种方法，用于解决离散对数问题（Discrete Logarithm Problem，DLP）。

DLP

给定一个 阶为 n 循环群 G，一个生成元 g 和一个元素 h，DLP 就是为了找到一个整数 x，使得 $g^x = h$。这个问题是一个困难问题。

Baby-Step/Giant-Step

因为 n 是群的 order，所以对于一个 $0 \leq x \leq n$，我们可以将 x 写为:

$$x = i\lceil\sqrt{n}\rceil + j \tag{1}$$

其中 $0 \leq i, j \leq \lceil\sqrt{n}\rceil$。

因此 DLP 可以转化为:

$$h = g^{i\lceil\sqrt{n}\rceil + j} \ h(g^{-j}) = g^{i\lceil\sqrt{n}\rceil}$$

现在问题就找到一个满足等式的对儿 (i, j)。

一个方式就是预计算一张表 $g^{i\lceil\sqrt{n}\rceil}, 0 \leq i \leq \sqrt{n}$ 与 g^{-1} 然后就是通过迭代 j 计算 $h(g^{-1})^j$，直到找到一个匹配的值。 然后就可以通过 (1) 计算出 x。

此方法的时间和空间复杂度均为 $O(\sqrt{n})$，但是目前还构不成威胁。

Number 35: Give the rough idea of Pollard rho, Pollard "kangaroo" and parallel Pollard rho attacks on ECDLP.

本章将讨论空间开销更小的算法，但是复杂度还是 $O(\sqrt{n})$。

> 说实话看到最后也没发现和 EC 有什么关系

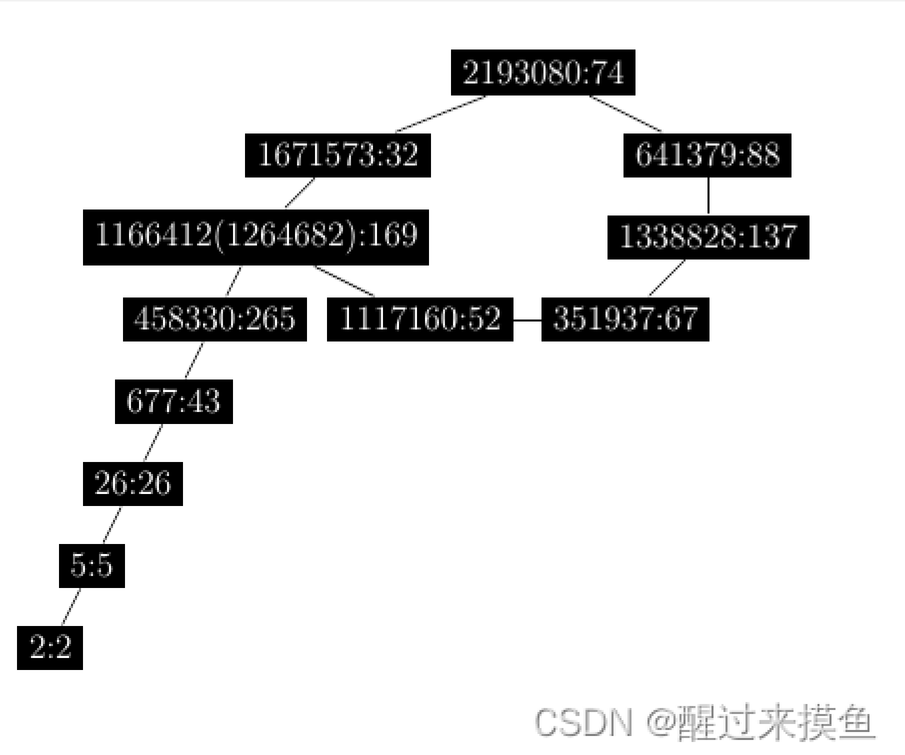
1. Pollard's Rho Algorithm

> 这个算法本来是用于因数分解的

令 $f : S \to S$ 为一个集合 S(大小为n) 到自身的随机映射，计算 $x_{i+1} = f(x_i)$ ，那么我们把 $x_0, x_1, x_2, \cdots$ 称为一个确定随机游走（deterministic random walk）

> 一般选 $f(x) = (x^2 + C)\ mod\ n$

因为 S 是有限的，所以必然会有 $x_i = x_j$，其中 $i < j$，也就形成了所谓的环。或者说，形成了一个碰撞

> 形状就跟符号 ρ 一样，x_0 看做左下角起点，总会形成一个环的 e.g.
> $x_0 = 2, f(x) = (x^2 + 2)\ mod\ 2206637 = 317 \times 6961$
>
> 
>

为了找到这样一个碰撞，我们可以使用 Floyd's cycle-finding algorithm，也就是快慢指针，即计算：

$$(x_{i+1}, x_{2i+2}) = (f(x_i), f(f(x_{i+1})))$$

总会找到一个 $x_m = x_{2m}$，其中 $m = O(\sqrt{n})$。

对于 DLP，我们将 S 分为三部分 S_1, S_2, S_3，其中 $1 \in S_2$，并定义以下随机游走：

$$x_{i+1} = f(x_i) = h \cdot x_i, x_i \in S_1 \ x_{i+1} = f(x_i) = x_i^2, x_i \in S_2 \ x_{i+1} = f(x_i) = g \cdot x_i, x_i \in S_3$$

我们实际上跟踪的是 (x_i, a_i, b_i)，其中：

```
\left{
\begin{aligned}
a_{i+1}=a_i,x_i\in S_1\
a_{i+1}=2a_i\ mod\ n, x_i\in S_2\
a_{i+1}=a_i+1\ mod\ n,x_i\in S_3
\end{aligned}
\right.\
\left{
\begin{aligned}
b_{i+1}=b_i+1\ mod\ n,x_i\in S_1\
b_{i+1}=2b_i\ mod\ n, x_i\in S_2\
b_{i+1}=b_i,x_i\in S_3\
\end{aligned}
\right.
```

我们从一个三元组 $(x_0, a_0, b_0) = (1, 0, 0)$ 开始，对于所有 i, **总有**！！

$$\log_g(x_i) = a_i + b_i \log_g(h) = a_i + b_i x$$

> $\log_g(h \cdot x_i) = a_i + (b_i + 1)\log_g(h) \ \ \log_g(x_i^2) = 2a_i + 2b_i \log_g(h)$
> $\log_g(g \cdot x_i) = a_i + 1 + b_i \log_g(h)$

通过利用 Floyd's algorithm 我们可以找到一个碰撞，即 $x_m = x_{2m}$，也就是 $a_m + b_m x = a_{2m} + b_{2m} x$，也就有：$x = \frac{a_{2m} - a_m}{b_m - b_{2m}} \ mod \ n$ 如果 $x_0, x_1, x_2, \cdots$ 是由随机映射产生的，那么上述算法将在期望时间 $O(\sqrt{n})$ 内找到离散对数，即：

$$\log_g(h) = x$$

2. Pollard's Kangaroo Method

跟 Rho 算法很想，而且更适用于我们知道 x 的范围的情况，i.e. $x \in [a, \cdots, b]$。

令 $w = b - a$ 为 x 所在区间的跨度，并定义一个非递增的数列 $S = s_0, \cdots, s_{k-1}$，均值（mean）m 大约为 $N = \sqrt{w}$ 我们经常选择 $s_i = 2^i$（因此 $m = \frac{2^k}{k}$）以及 $k \approx \frac{1}{2}\log_2(w)$，群则被分为 k 个部分 $S_i, i \in [k-1]$，我们定义一个确定随机游走：$x_{i+1} = x_i \cdot g^{s_j}, if \ x_i \in S_j$。

现在我们计算确定性随机游走： 首先从 $g_0 = g^b$开始，然后计算 $g_{i+1} = g_i \cdot g^{s_j}$，同时设置 $c_0 = b, c_{i+1} = c_i + s_j \ (mod \ q)$，最后存储 g_N, 发现有如下关系：

$$c_N = \log_g(g_N)$$

> q就是 G 的素数阶，原文没说，但是不可能不是

接下来算第二个随机游走，但是起点变为了未知点 x： 令 $h_0 = h = g^x$，并计算 $h_{i+1} = h_i \cdot g^{s_j}$，同时设置 $d_0 = 0, d_{i+1} = d_i + s_j \ (mod \ q)$，发现有如下关系：

$$\log_g(h_i) = x + d_i$$

综上，我们可以发现，如果 g_i 和 h_i 的路径相遇，那么 h_i 会沿着 g_i 继续走下去，所以我们最终会找到一个 $h_M = g_N$，也就是：

$$c_N = \log_g(g_N) = \log_g(h_M) = x + d_M$$

如果没有发生碰撞，那么就可以通过增加 N 来继续搜索，直到发生碰撞，这个算法的期望时间复杂度为 $O(\sqrt{w})$，而存储空间是恒定的。

> 整个算法都是**没有**被证明的。这只是一个经验性的算法。也就是说，袋鼠算法甚至有可能无法解决某个DLP问题。不过据原作者'claim'，算法表现良好。另外这个算法还是被广为应用，let's believe it works well... （有点丑陋

3. Parallel Pollard Rho

当我们使用基于随机游走的技术来解决 DLP 时，我们经常使用并行版本。 $h = g^x$, group G, prime order q

我们首先确定一个易于计算的函数 $H : G \to 1, \cdots k$，然后定义一组乘数 $m_i = g^{a_i} h^{b_i}$，其中 $a_i, b_i \overset{\$}{\in} [0, \cdots q-1]$

> k 通常为 20

为了能够让随机游走起步，我们需要随机选取 $s_0, t_0 \in [0, \cdots, q-1]$，然后计算 $g_0 = g^{s_0} h^{t_0}$，并在三元组 (g_i, s_i, t_i) 上定义随机游走：

$$g_{i+1} = g_i \cdot m_{H(g_i)} \ s_{i+1} = s_i + a_{H(g_i)} (\ mod\ q) \ t_{i+1} = t_i + b_{H(g_i)} (\ mod\ q)$$

这样一来，可以确定如下关系：

$$g_i = g^{s_i} h^{t_i}$$

当我们有 m 个处理器时，就可以从不同的起点开始，使用相同的算法，当两个处理器(或同一处理器)遇到同一个之前出现的元素时，就可以得到方程 $g^{s_i} h^{t_i} = g^{s_j} h^{t_j}$，然后就可以计算 x：

$$x = \frac{s_i - s_j}{t_j - t_i} \ mod\ q$$

> 自己推导的

预计在 $O(\sqrt{\pi q/2}/m)$ 次迭代后，一个碰撞能够被发现，DLP也随之被解决，但是这也意味着需要存储每个处理器产生的所有结果（毕竟要比对的嘛），需要巨量的存储空间 $O(\sqrt{\pi q/2})$

如何降存储？ 存储可以降到任何需要的值。看不懂看不懂，原文贴下面了： Moreover the storage can be reduced to any required value as follows: We define a function d on the group, $d : G \to 0, 1$ such that $d(g) = 1$ around $1/2^t$ of the time. The function d is often defined by returning $d(g) = 1$ if a certain subset of t of the bits representing g are set to zero for example. The elements in G for which $d(g) = 1$ will be called *distinguished*. It is only the distinguished group elements which are now transmitted back to the central server, which means that we expect the deterministic random walks to continue another 2^t steps before a collision is detected between two deterministic random walks. Hence, the computing time now becomes $O(\sqrt{\pi q/2}/m + 2^t)$ and storage becomes $O(\sqrt{\pi q/2}/2^t)$. Thus, storage can be reduced to any manageable amount, at the expense of a little extra computation.

尝试理解：（应该没错）：

1. 这个 $d(g)=1$ 中的 g 并不是指上面生成元那个 g，而是一个通用符号，就像 $f(x)$ 中的 x 一样
2. 基于 1，这样我们把输入的 g 转化为 bit 表示，然后取其中的 t 个 位置（这些位置事先确定的），如果这 t 个 bit 全为 0，那么 $d(g)=1$，否则 $d(g)=0$。这样一来 $\Pr[d(g)=1]=1/2^t$，而满足这些条件的元素我们就称为 *distinguished*。
3. 只存储可以可区分的群元素（注意没说不计算），这样一个元素被发现发生碰撞时，就已经做过了 2^t 次计算

> 有种我已经把 2^t 次计算的功下了，找到碰撞那是必然的感觉

4. 这样时间就涨到 $O(\sqrt{\pi q/2}/m+2^t)$, 但是存储直接降到 $O(\sqrt{\pi q/2}/2^t)$

Number 36: Index Calculus Algorithm

What is the objective? index calculus attack 还是尝试解决 DLP。思路： 将目标值转为一些元素的幂的乘积，这些值的对数是已知的，最后利用对数定律求出目标值

How does it work? 如果已知 $L_i=\log_g(x_i)$，且可将未知的 h 写为 $h=x_1^{a_1}\cdots x_r^{a_r}$，那么就可以得出：

$$\log_g(h)=a_1L_1+\cdots+a_rL_r$$

Terms:

- "offline computation" / "precomputation" : 每个群只需要进行一次的操作
- "online computation" / "everytime" : 每次需要解决 DLP 时的操作

Steps:

1. Choose a Factor Base *(Precomputation, basically free)* 就是选择一组底数（factor base） $b_0=g,b_1,\cdots,b_r\in G$，也就是上述的 x_i 但是选多少？ 这个我们操作的组有关系，r 小，线上操作就多，r 大，线下操作就多，这是一个 trade-off 怎么选？ 一般选 -1 和 前 r 个素数，这往往会使线上计算更加高效

   > 就是 $-1,2,3,5,7,11,\cdots,p_r$

2. Find relations between the DLPs of the Factor Base elements. *(Precomputation, expensive but **very parallel**)* 该步骤目的是找到这些关系

   $$g^k\ mod\ q=(-1)^{e_0}2^{e_1}3^{e_2}\cdots p_r^{e_r}$$

   通过取对数，可以转换为线性关系，最后找到 r 个这样的关系（说白了就是取 r 次 k） 此步骤有点玄学

   > 原文：Using whatever techniques we can (generally just taking arbitrary products and hoping to get lucky!)

   r 越大，这里操作也就越多，和上面对应起来了 但是每个 g^k 都是独立的，所以可以让每个进程负责一个，即很容易实现并行计算

3. Solve the Factor Base DLPs *(Precomputation, relatively cheap)* 我们对这 $r+1$ 个关系（还有一个 $\log_g(g)=1$）,最后可以用一个矩阵求解器得到 $r+1$ 个 等价关系

   > 也就是找到了 $L_i=\log_g(x_i)$

4. Write h as a product of factor base elements *(Online, expensive but very parallel)* 该步骤目的是尝试找到一个 y 和 一系列 a_i，和离线阶段得到的关系组成以下的等式：

$$hg^y = b_1^{a_1} \cdots b_r^{a_r}$$

答案也就呼之欲出：

$$\log_g(h) = -y + \sum_{i=1}^{r} a_i L_i$$

问题来到了：如何找到这个 y？

> 是的，只能试，我们取的因子基数是小素数，所以试起来相对比较容易

Conclusion: Index Calculus算法通过离散对数转化为和的形式找出离散对数的结果。它通过建立一个已知的表（因子基数库）来解决这个问题，然后找到一个与目标相关的等式将目标写成这种形式。 该算法不得不说一声通用，但是不可能每个阶段都很轻松

> r 大了，离线阶段算的就多；r 小了，线上阶段算的就多（y 得多试好多次）

Number 37: The Number Field Sieve

> Continue the mathematical attacks with the ***NFS*** algorithm.

数域筛法 (the Number Field Sieve, NFS) 是目前已知最有效的因式分解算法。 它的运行时间取决于要分解数字的大小，而不是其因子的大小

> 啊？不是后者更好吗？

NAF 基于同余平方理论（congruent squares）：如果 $x^2 = y^2 \ mod \ N$，那么 $gcd(x-y, N)$ 就是 N 的一个非平凡因子。

1. 选择两个多项式 f_1 和 f_2
    1. 本原 (monic)，不可约（irreducible）
    2. degree 分别为 d_1 和 d_2
    3. 令 $m \in Z$ 为一个共同根，即 $f_1(m) = f_2(m) = 0(mod \ N)$
    4. 令$\theta_1, \theta_2 \in C$ 分别为 f_1 和 f_2 的复数根 (complex roots)
2. 构建两个代数数域 (algebraic number fields) $Z[\theta_i] = Q[\theta_i], i = 1, 2$

    > 这其实也是俩数环（number ring），乘法操作就是多项式乘法 Q 好像没有用到？

3. 定义将 θ_i 映射到 m 的同态映射 $\phi_i : Z[\theta_i] \to Z_N$

    > 也就是说，$\phi_i(\theta_i) = m$

4. NAF 算法目标是从两个数环中找到两个平方 γ_1^2 和 γ_2^2，使得

    $$\gamma_1^2 = \prod_{(a,b)\in S} (a - b \cdot \theta_1) \ \gamma_2^2 = \prod_{(a,b)\in S} (a - b \cdot \theta_2)$$

    其中$\gamma_1 \in Z[\theta_1], \gamma_2 \in Z[\theta_2]$ S 就是一个互质整数对 (a, b) 的有限集合
5. 为了找到这么一个集合，我们对 $a - b \cdot \theta_i$ 进行筛选，观察其在一些代数基数上是否平滑

> 说得很抽象啊 In order to find such a set S, we will sieve the elements of the form $a-b\cdot\theta_i$ for pairs of (a,b) such that $a-b\cdot\theta_i$ is smooth over some algebraic factorbase. 找到 S 的速度决定了算法的效率

6. 接下里就是提取 γ_i^2 的平方根 γ_i

> 可以使用 Couveignes [1] and Montgomery [2] 算法 [1] Couveignes, Jean-Marc. "Computing a square root for the number field sieve." The development of the number field sieve. Springer Berlin Heidelberg, 1993. 95-102. [2] Montgomery, Peter L. "Square roots of products of algebraic numbers." Mathematics of Computation (1993): 567-571. APA

7. 一旦求出平方根，就可以利用同态求得

$$\phi_1(\gamma_1)^2=\phi_2(\gamma_2)^2\ mod\ N$$

从而得到 $gcd(\phi_1(\gamma_1)-\phi_2(\gamma_2),N)\neq 1,N$

> 不怎么看懂但是大受震撼