Weekly Notes for CTF 🚩

Week 2

Astrageldon

2023-11-30

1 Crypto

1.1 Miller-Rabin primality test

给定奇数 n，为了检验它是不是素数，我们可以选取若干个正数 a $(2 \le a \le n-2)$，计算 $a^{n-1} \pmod n$，由于当 n 为素数且 a 与 n 互质时该式结果为 1，如果有一个 a 使得该式不等于 1，那么 n 就必然不是素数，此时 a 被称作是“费马证人数”（Fermat witness for the compositeness of n）。然而，如果一个 a 能使该式结果为 1，但 n 却不是素数，那么这个 a 就被称作是“费马骗子数”（Fermat liar），n 被称作是“费马伪素数”（Fermat pseudoprime to base a）。

上述检验方法被称为费马素性检验（Fermat primality test）。然而这种检验方法对于卡迈克尔数（Carmichael number）是完全无效的。如果 n 是卡迈克尔数，那么 $a^{n-1} \equiv 1$ 对于所有满足 $\gcd(a, n) = 1$ 的 a 都成立，也即所有的 a 都是费马骗子数。我们考虑奇素数 p 具有的另外一个性质：

- 若 $x^2 \equiv 1 \pmod p$，则必有 $x \equiv 1 \pmod p$ 或者 $x \equiv -1 \pmod p$。

于是，如果 $a^{p-1} \equiv 1 \pmod{p}$，那么 $a^{\frac{p-1}{2}} \equiv 1$ 或 -1。我们令 $n-1 = 2^s d$，那么序列 $a^{2^0 d}, a^{2^1 d}, \cdots, a^{2^s d}$ 在模 p 的意义下以 1 结尾，并且要么前面的项全为 1，要么中间某一项为 $p-1$，之后全为 1。对于要检验的奇数 n 而言，如果某一个 a $(\gcd(a, n) = 1)$ 产生的序列满足这个特点，那么 n 就被称作是一个“strong probable prime to base a”，可以生硬地翻译为“对于 a 而言的强可能素数”；反之，a 产生的序列不满足这个特点的话，那么 n 就肯定是合数，a 被称作是一个“见证者”（witness for the compositeness of n）。

只要我们合理地选择底数 a，就可以将出错的概率降至很低。按照这里的表给出的底数组，选择 $2, 325, 9375, 28178, 450775, 9780504, 1795265022$ 这一串作为用来尝试的 a，可以保证 2^{64} 以下的数检验不出错（$\ast$）。而选择 $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41$ 则可以保证 $3317044064679887385961981$ 以下的数检验不出错。

然而，对于随机生成的 k 个不同的 a $(\gcd(a, n) = 1)$ 而言，n 是合数但判断是素数的概率上界是 4^{-k}（$\ast$），且一般情况下概率远低于该上界。即便我们不知道应该如何选择 a，我们也可以通过提高轮数 k 来降低出错的概率。

该算法被称作米勒-拉宾素性检验（Miller-Rabin primality test），最好的实现方法时间复杂度可以达到 $O(k\log^2 n\log\log n)=\tilde{O}(k\log^2 n)$，其中 k 是选取底数并检验的次数。

```python
def isPrime_mr(n, bases = [2, 325, 9375, 28178, 450775, 9780504, 1795265022]):
    if n == 2: return 1
    if n % 2 == 0: return 0
    s = 0
    while (n-1)%2**s == 0:
        s += 1
    s -= 1
    d = (n-1)//2**s
    for a in bases:
        if a % n == 0: continue
        x = pow(a,d,n)
        y = 1
        if x==1 or x==n-1: continue
        for r in range(s):
            y = pow(x,2,n)
            if y==1 and x!=1 and x!=n-1:
                return 0
            x = y
        if y != 1:
            return 0
    return 1
```

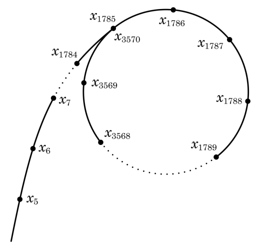
1.2 Integer factorization based on Miller-Rabin tests

接上一章，如果 $x = a^{2^r d}, x^2 \equiv 1 \pmod{n}$ 且 $\neg x \equiv \pm 1$，那么显然 $n \mid (x-1)(x+1)$ 且 $n \nmid x-1 \wedge n \nmid x+1$，于是，$A = \gcd(x-1, n), B = \gcd(x+1, n)$ 是 n 的非平凡因子并且 $\gcd(A, B) = 1, n = AB$。

这也就是说，如果 n 相对于 a 来说并不是“strong probable prime”，但是是一个“probable prime”，或者说是费马伪素数，我们就能用米勒-拉宾检验法找到 n 的一个非平凡因子 $\gcd(x-1, n)$，其中 $x = a^{2^r d}$。

1.3 Integer factorization based on Pollard's rho algorithm

设 $n = pq$，p, q 是 n 的两个非平凡因子。考虑一个伪随机数生成器：$g(x) = (x^2 + c) \pmod n$。选定初值 x_0，可以根据 $x_{k+1} = g(x_k)$ 得到序列 $\{x_k\}$ 和对 p 取模后的序列 $\{x_k \mod p\}$。如果由 g 生成的序列足够随机，那么根据生日问题，第一个重复出现的下标的期望不超过 $(1 + e^{-\frac{1}{2}})\sqrt{k} + 2 = O(\sqrt{k})$，$k = n$ 或 p，于是，很大概率上序列 $\{x_k \mod p\}$ 出现重复比序列 $\{x_k\}$ 要早，但最终都会进入一个循环，描绘在纸上就是一个"ρ"的形状。



在循环处，我们期望找到 k_1, k_2 使得 $|x_{k_1} - x_{k_2}| \equiv 0 \pmod p$ 但 $|x_{k_1} - x_{k_2}| \neq 0$。这样以来 $\gcd(|x_{k_1} - x_{k_2}|, n)$ 就是 n 的一个因子了。可以使用 Floyd 判环算法（龟兔赛跑）来检测出现的循环，而由于不知道 p 的值，可以用 $\gcd(|x_{k_1} - x_{k_2}|, n) > 1$ 作为找到环的标志。

```python
import random
from Crypto.Util.number import *

def _factor_update(factors, d):
    factors.update({d: factors.get(d,0) + 1})

def _pollard_rho(n, factors):
    if n <= 1: return
    if n%2 == 0:
        _factor_update(factors, 2)
        return _pollard_rho(n//2, factors)
    if isPrime(n): return _factor_update(factors, n)
    while True:
        c = random.randint(2, n-1)
```

```
        f = lambda x: x**2 + c
        x = y = 2
        d = 1
        while d == 1:
            x = f(x) % n
            y = f(f(y)) % n
            d = gcd((x - y) % n, n)
        if d != n:
            if not isPrime(d): return _pollard_rho(n, factors)
            _factor_update(factors, d)
            return _pollard_rho(n//d, factors)

def pollard_rho(n):
    if n <= 1: raise ValueError("Number to be factored must be >= 2 :(")
    factors = dict()
    _pollard_rho(n,factors)
    return list(factors.items())
```

1.4 Discrete logarithm problems based on the design philosophy of Pollard's rho algorithm

上一章说到 Pollard's rho 算法在一个 rho 形有向图上寻找碰撞的 x_{k_1} 与 x_{k_2}，一般保持 $k_2 = 2k_1$，也即寻找碰撞的 x_i 与 x_{2i}。本章也将采用类似的思想来寻找离散对数问题的解。

设 $G =< \alpha >$ 是一个 n 阶循环群，$\beta \in G$，要找到 γ 使得 $0 \leq \gamma \geq n-1$ 且 $\alpha^{\gamma} = \beta$。

现在，将 G 划分为三个不相交的部分 $G = S_1 \bigcup S_2 \bigcup S_3$，且保持三个子集的元素个数大致相同。

定义映射 $f : G \times \mathbb{Z}_n \times \mathbb{Z}_n \to G \times \mathbb{Z}_n \times \mathbb{Z}_n$：

$$f(x,a,b) = \begin{cases} (\beta x, a, b+1), & x \in S_1 \\ (x^2, 2a, 2b), & x \in S_2 \\ (\alpha x, a+1, b), & x \in S_3 \end{cases}$$

我们可以发现，如果有一组 (x,a,b) 满足 $x = \alpha^a \beta^b$，那么 $f(x,a,b)$ 仍然满足这个性质。

像上一章那样，选定初始值 $(x_0, a_0, b_0) = (1,0,0)$，得到递推关系：$f(x_{i+1}, a_{i+1}, b_{i+1}) = (x_i, a_i, b_i)$。

由于 $x_i \in G$，序列 x_i 最终一定会进入一个循环，这个循环的长度期望是 $\sqrt{\frac{\pi n}{8}}$ (?)。为了得到有意义的循环，我们不应忘记 $1 \notin S_2$，否则 $f(1,0,0) = (1,0,0)$。我们希望得到 $x_i = x_{2i}$，于是有 $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$，代入 $\alpha^{\gamma} = \beta$ 得到：

$$\gamma(b_{2i} - b_i) \equiv a_i - a_{2i} \pmod{n}$$

这是一个线性丢番图方程，欲见详细解法请戳这里。若 $\gcd(n, b_{2i} - b_i) \nmid a_i - a_{2i}$ 则该方程无解，反之总可以找到一个 γ 是该方程的解，从而是该离散对数问题的解。

参数设置合理的话，该算法的时间复杂度取决于循环的长度，即 $O(\sqrt{n})$，结合 Pohlig-Hellman 算法的话时间复杂度则为 $(\sqrt{p})$，其中 p 为 n 的最大素因子。不过该算法并不保证一定能找到一个 γ，所以个人感觉效率不如大步小步算法（?）。

😈😈😈 就这样吧，接下来就该折腾计导了 😈